Style Report Enterprise Edition

Welcome to Style Report Enterprise Edition!

Style Report is a report 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 Report to be agile, robust, and easy to use.

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

Creator 1. *Connecting to Your Data*

Creator 2. *Transforming and Mashing Up Data*

Creator 3. *Creating a Report*

User 4. *Using the Portal*

User 5. *Interacting with Reports*

Advanced 6. *Managing the Server*

Style Report 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 begin, first create a *connection* to your data. This connection is called a *datasource*. You can 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', 'Style Intelligence', or 'Style Report' program group. This starts Style Studio, the desktop development tool.

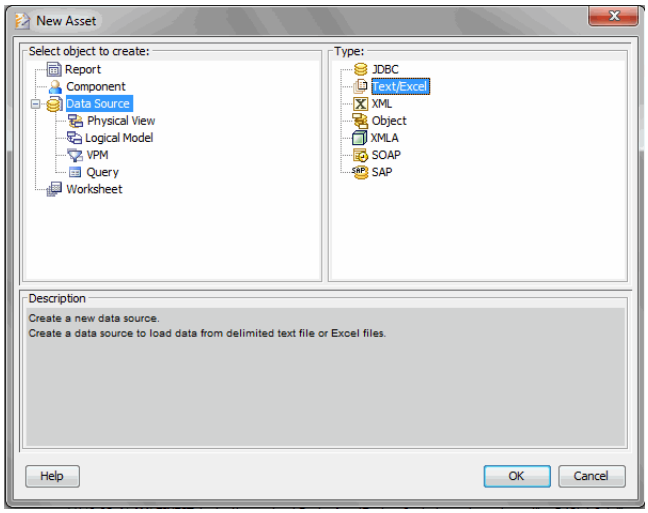


2. Press the 'New' button in the Style Studio toolbar. This opens the 'New Asset' dialog box.



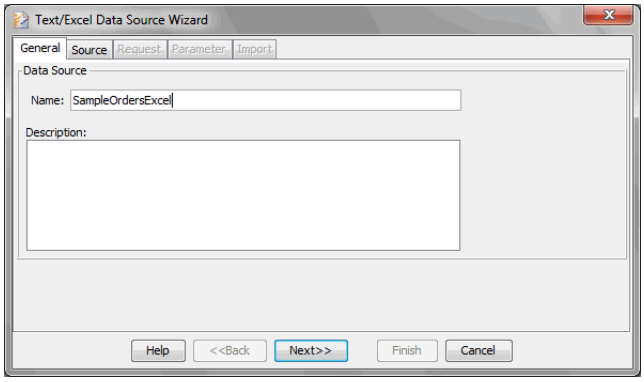
3. In the left panel of the 'New Asset' dialog box, press the 'Data Source' node on the tree.

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



This opens the Text Data Source Wizard.

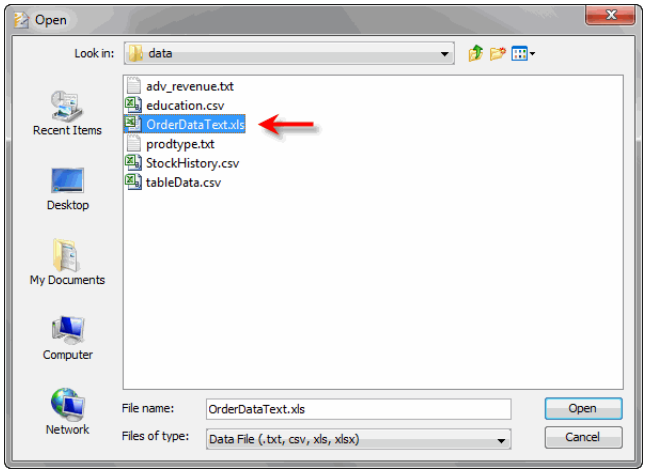
5. Enter "SampleOrdersExcel" in the 'Name' field. This is the name that will be used to identify the data source. Press 'Next'.



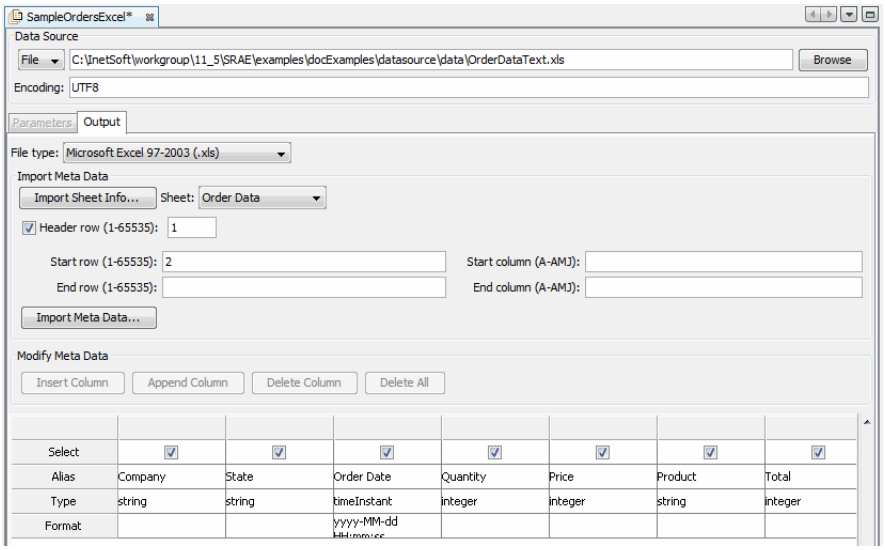
6. Press the 'Browse' button and navigate to the following directory in the installation folder:

   `{Installation Folder}\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. JDBC drivers are available for most databases.

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*:

Amazon Redshift	*Apache Hadoop and Hadoop Hive*	*Cloudera*
Derby	*EMC Greenplum*	*HP Vertica*
IBM DB2	*IBM Informix*	*IBM Netezza*
Ingres	*LucidDB*	*Microsoft Access*
Microsoft SQL Server	*Microsoft SQL Server Express*	*MySQL*
Oracle	*Oracle Business Intelligence Enterprise Edition (OBIEE)*	*Oracle JD Edwards*
Oracle PeopleSoft	*PostgreSQL*	*SAP Hana*
SAP SQL Anywhere	*Siebel CRM*	*SQLite*
Teradata		

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', 'Style Intelligence', or 'Style Report' 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.



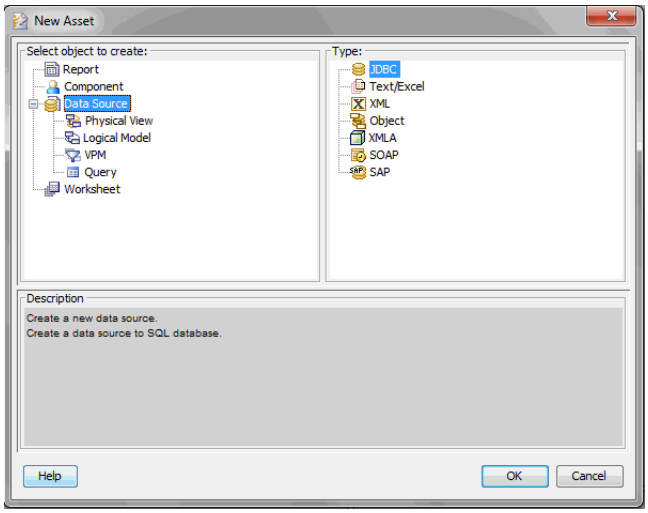
4. Press the ‘New’ button in the Style Studio toolbar. This opens the ‘New Asset’ dialog box.



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

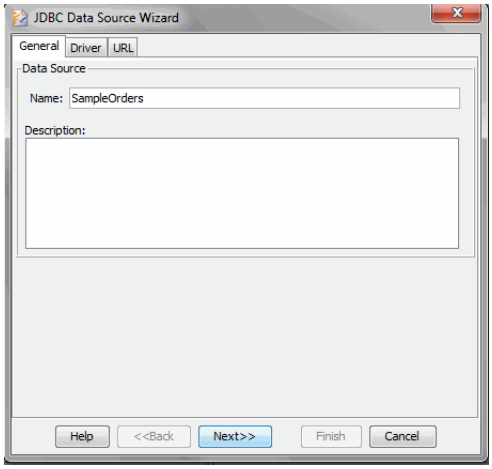
Most relational databases (Oracle, SQLServer, etc.) are “JDBC” databases, which means they provide a Java driver that facilitates connections to the database.

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

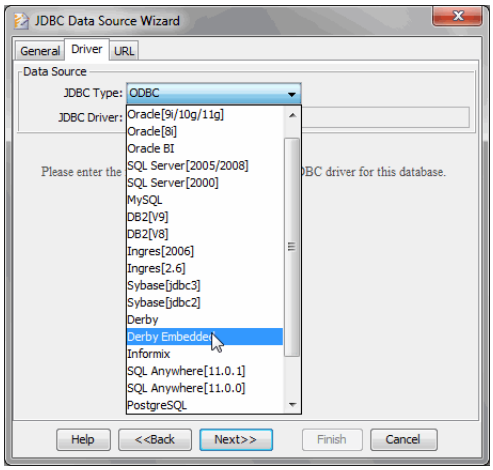


This opens the JDBC Data Source Wizard.

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.



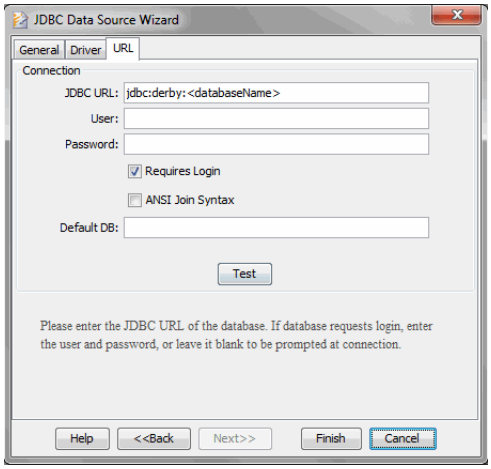
9. Press ‘Next’ to continue to the ‘JDBC URL’ page.

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

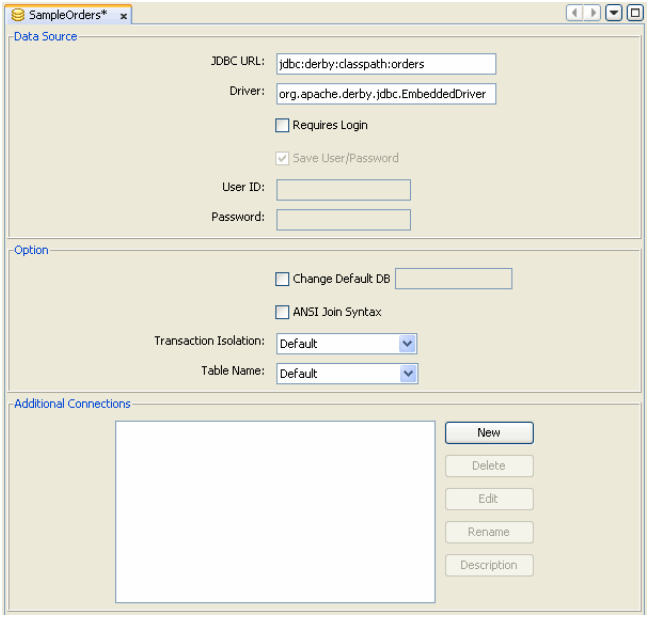
    ```
    jdbc:derby:classpath:orders
    ```

    This is the connection URL for the sample Derby database.

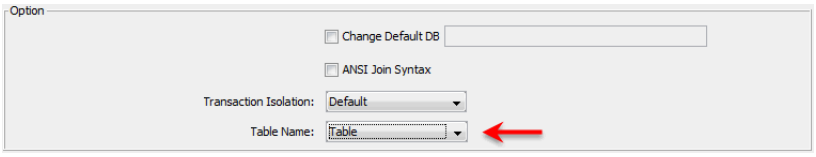
11. Uncheck the ‘Requires Login’ option.



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



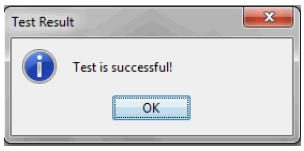
13. If you are configuring an ODBC datasource such as Microsoft Access, set the ‘Table Name’ option to ‘Table’.



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*
– *Oracle Hyperion and Hyperion Essbase*
– *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.

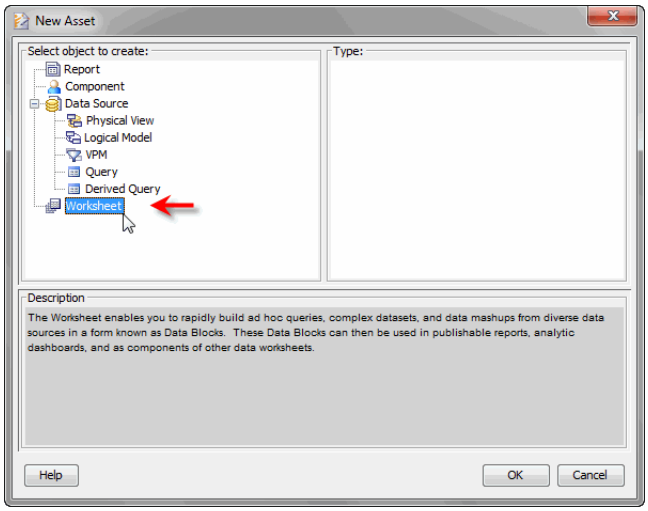
1. From the Windows *Start* Menu, select the 'Style Studio' shortcut in the 'Style Scope', 'Style Intelligence, or 'Style Report' program group. This starts Style Studio, the desktop development tool.



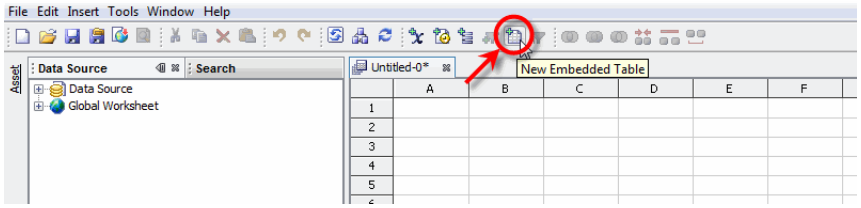
2. Press the 'New' button in the Style Studio toolbar. This opens the 'New Asset' dialog box.

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

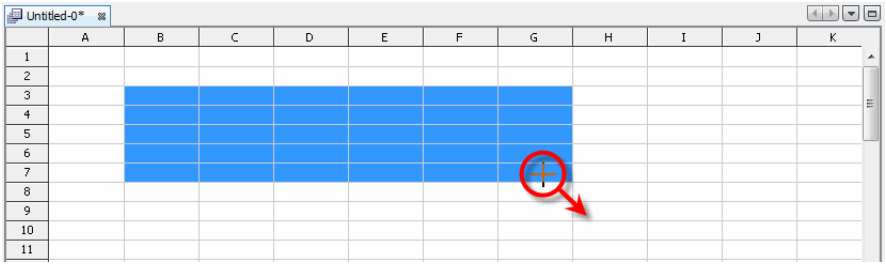
3. In the left panel, select 'Worksheet', and press 'OK'. This opens a new Data Worksheet.



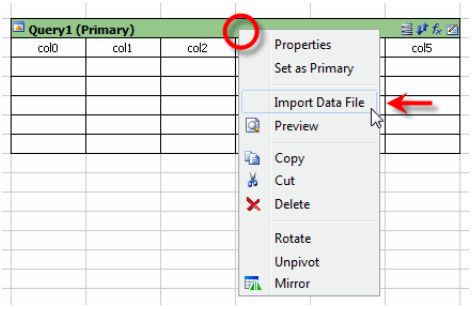
4. Press the 'New Embedded Table' button on the toolbar. This changes the standard pointer into a crosshairs pointer.



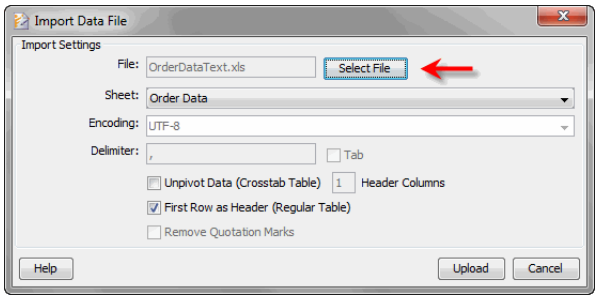
5. Drag the crosshairs pointer on the Worksheet grid to create a new table. (It does not matter how tall or wide you make the table.)



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



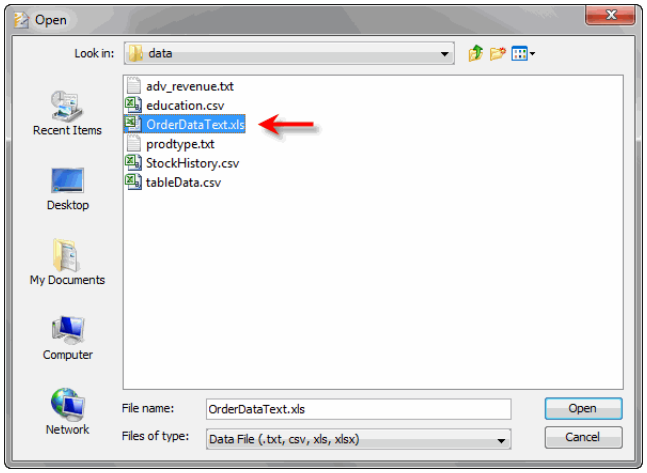
7. Press the ‘Select File’ button.



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

   `{Installation Folder}\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.



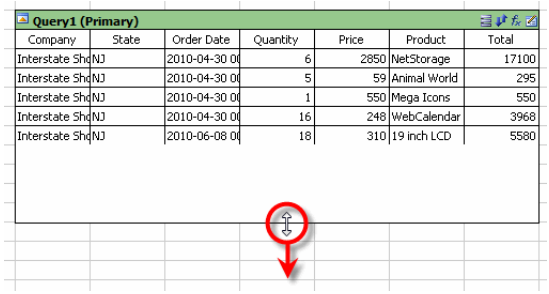
Query1 (Primary)

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.

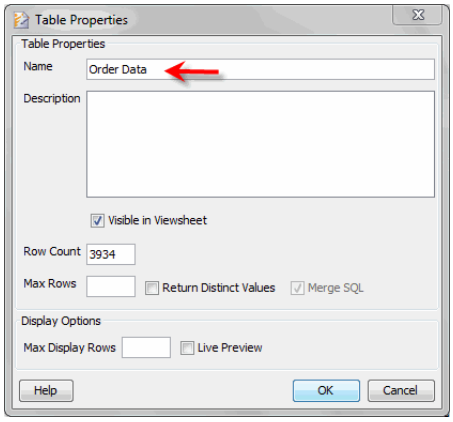


Query1 (Primary)

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

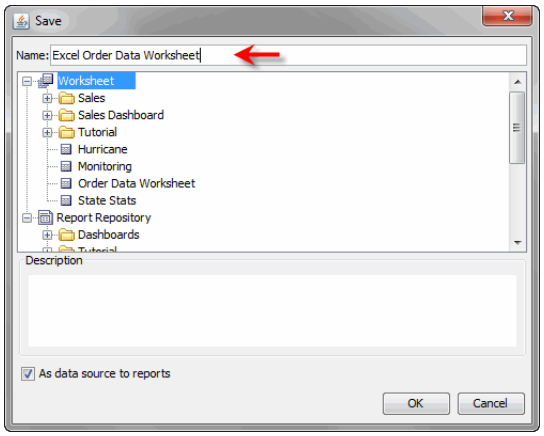
12. Right-click on the table, and select ‘Properties’ from the context menu. This opens the ‘Table Properties’ dialog box.

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

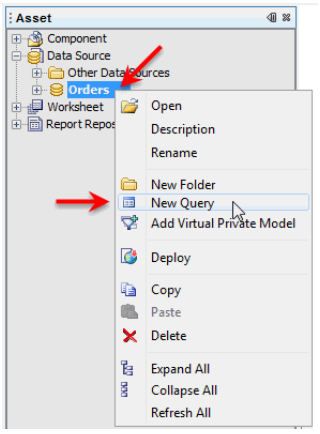
After you have connected to your data (see *Connecting to Text or Excel Data* and *Connecting to a Database*), you can create a query to retrieve your data.

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.

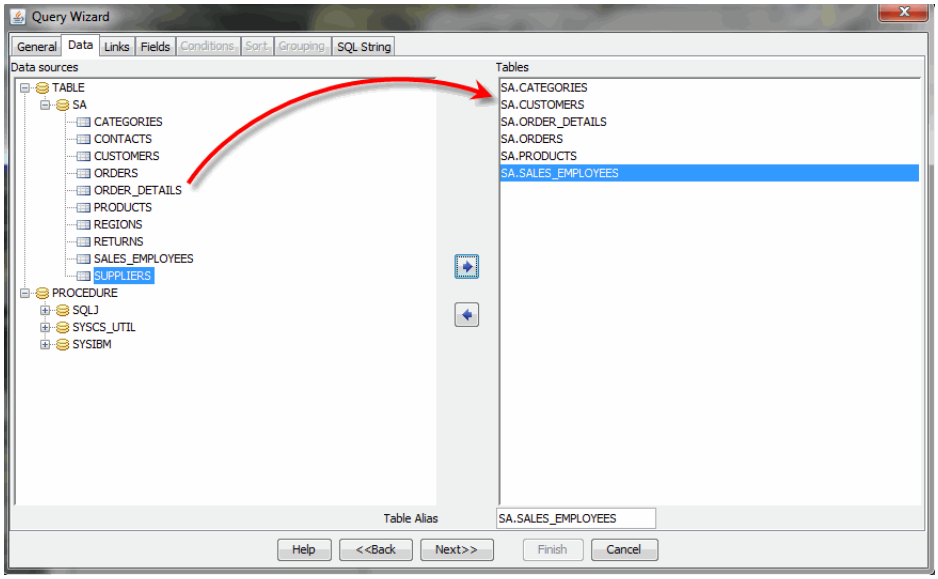
2. Right-click on the data source that you wish to query, and select 'New Query' from the context menu. This opens the Query Wizard.



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.

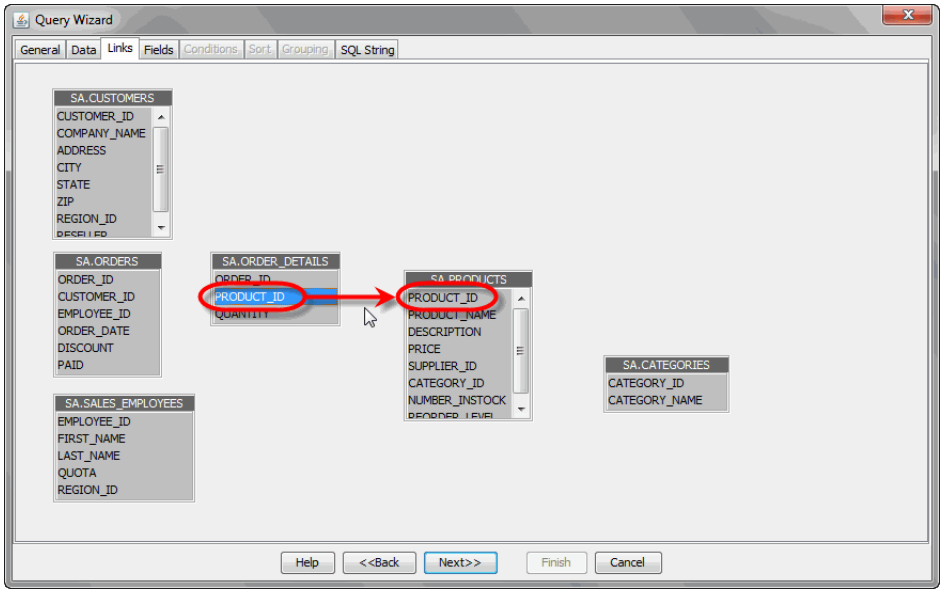
You can also double-click fields to add them, or use the arrow buttons.

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.

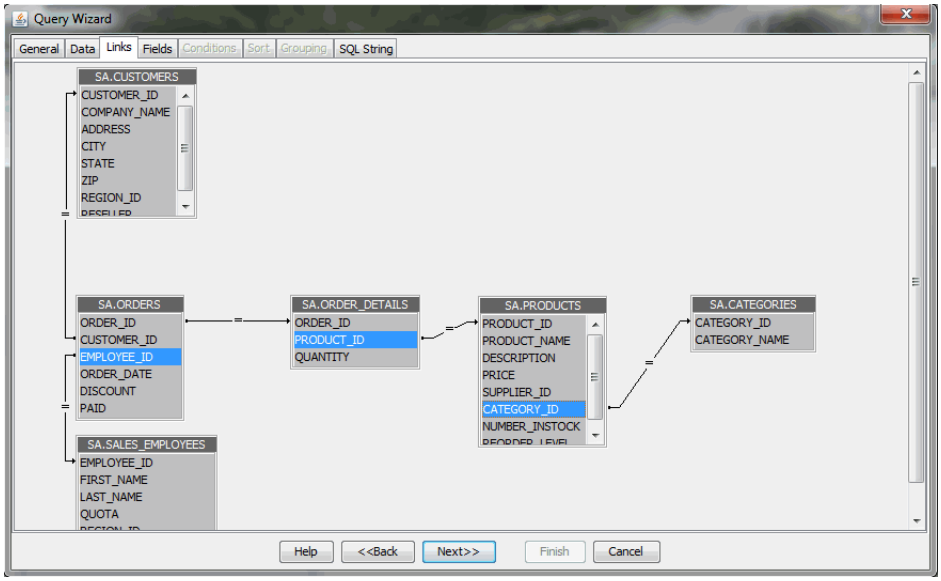


To manually enter SQL rather than using the Wizard, press the **SQL String** tab. See *Creating a Query from SQL*.

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.

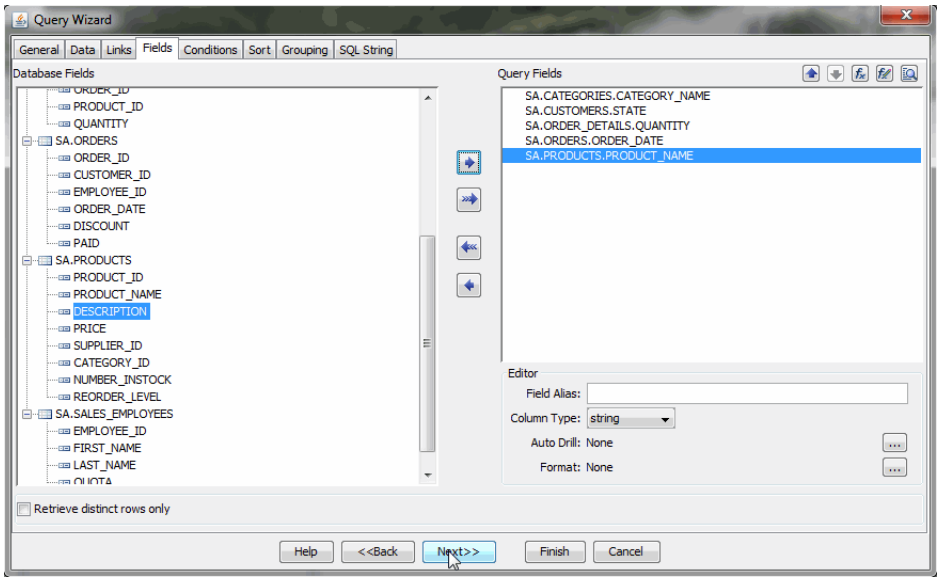


When you have joined all the tables, press ‘Next’ to advance to the next tab.

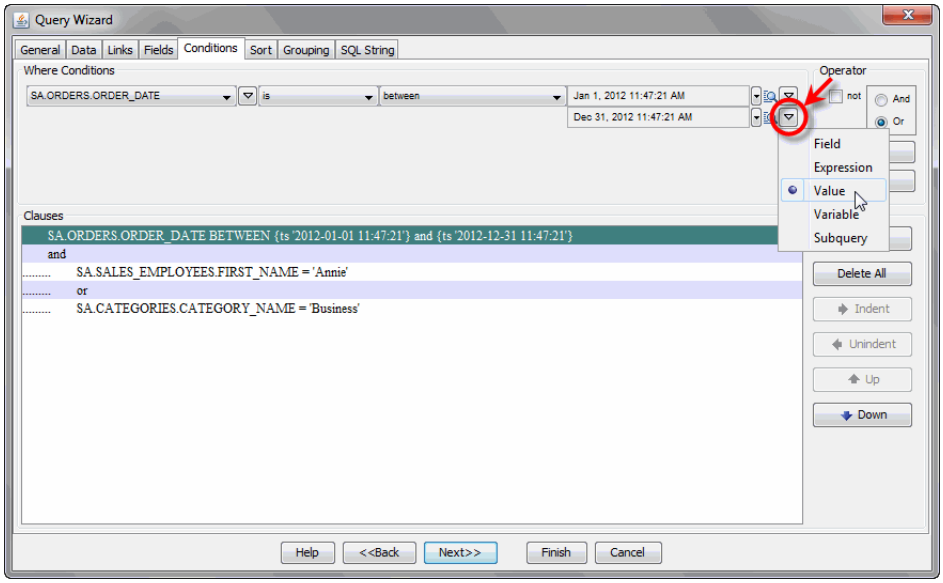


You can also double-click fields to add them, or use the arrow buttons.

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.

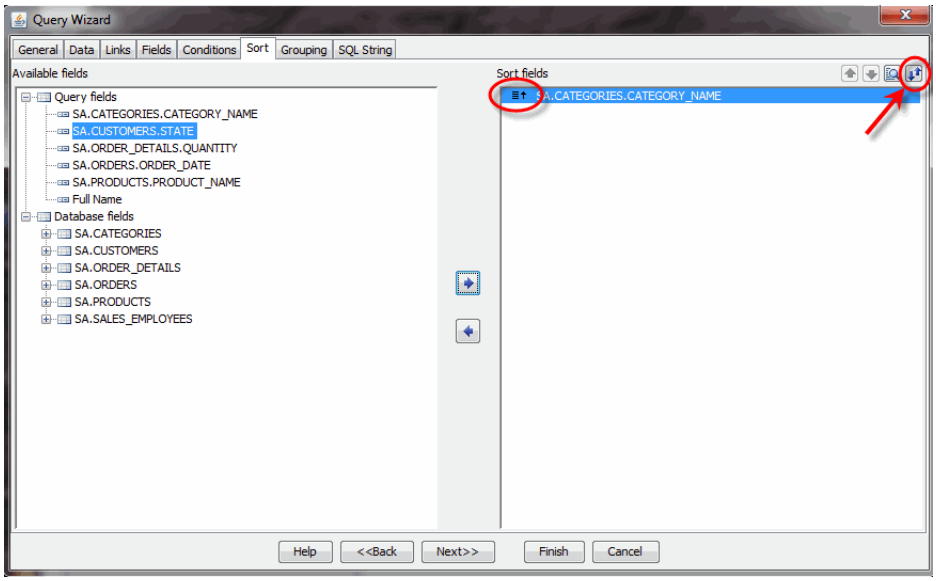


Press ‘Next’ to proceed to the **Sort** tab.

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.

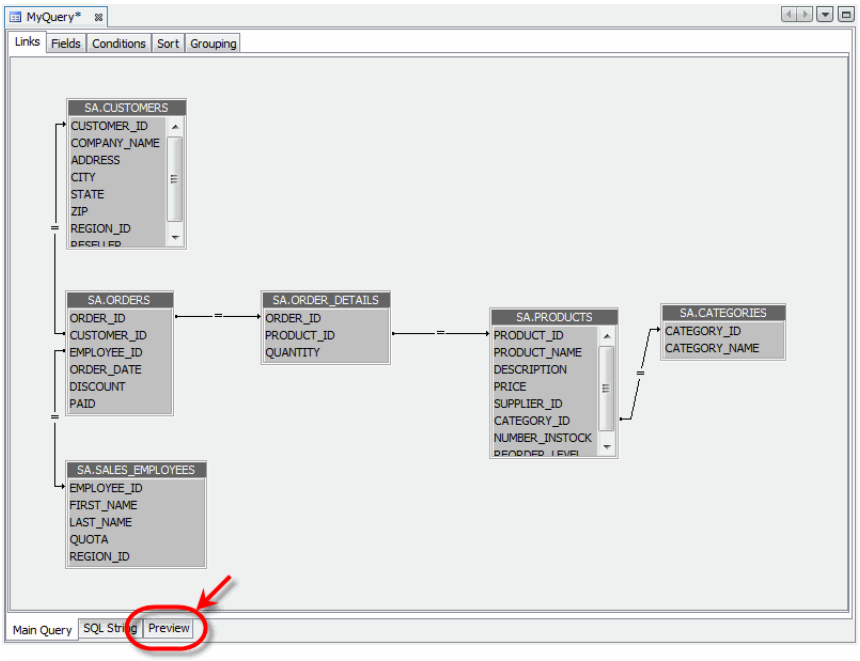
9. Select a field in the ‘Sort Fields’ panel, and press the ‘Order’ button above the panel. Press once to sort in descending order. Press again to sort in ascending order.



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.



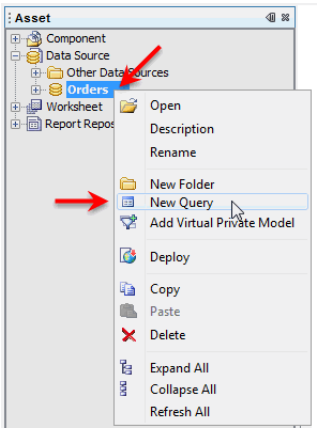
Creating a Query from SQL

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.

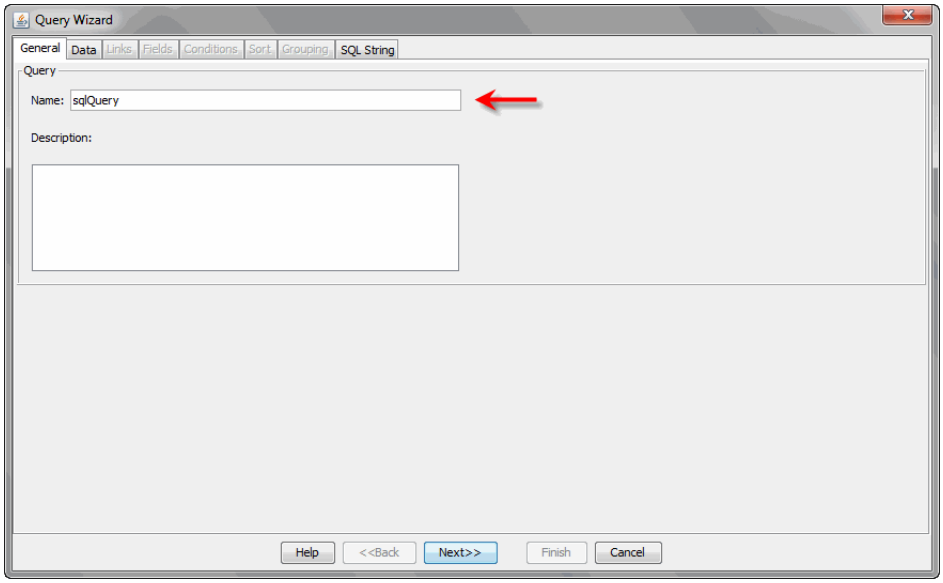
If you would like to use an existing SQL query string, follow the steps below:

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

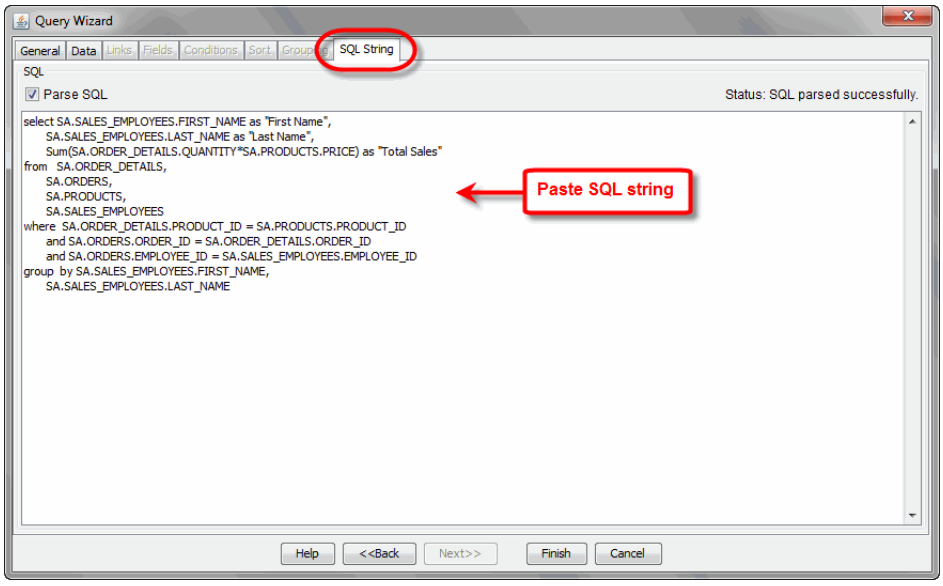
2. Right-click on the data source that you wish to query, and select 'New Query' from the context menu. This opens the Query Wizard.



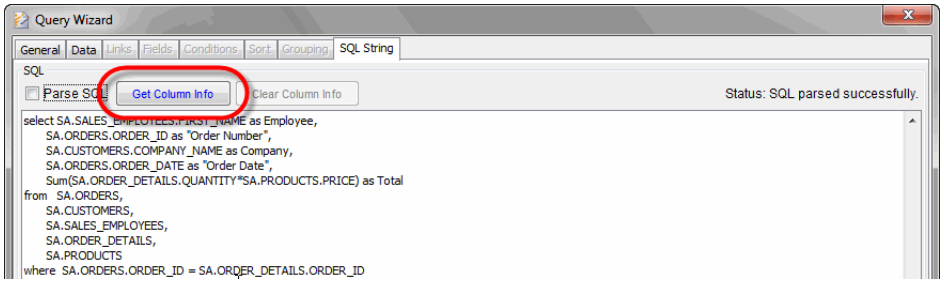
3. Under the **General** tab of the Query Wizard, enter a name for the query and (optional) description.



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.



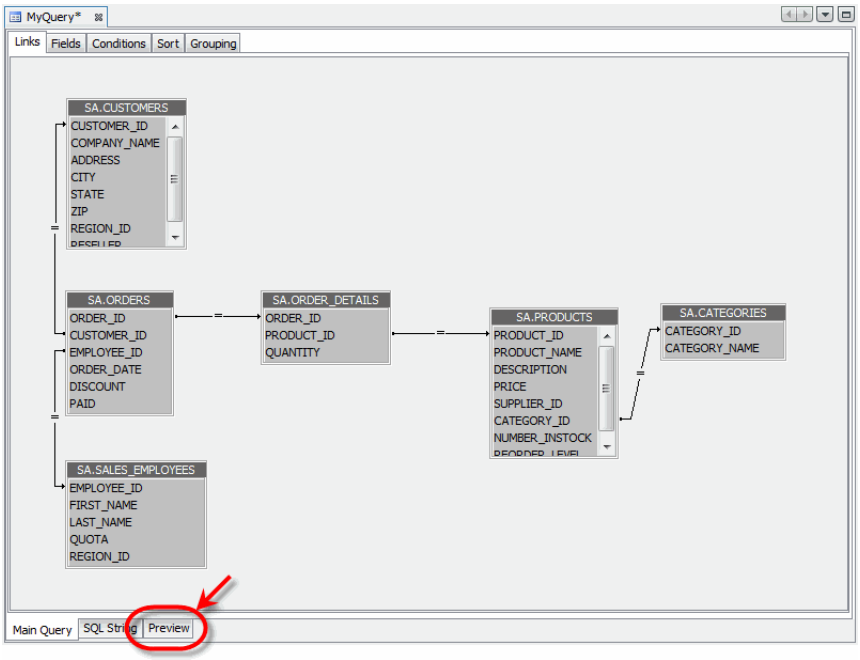
If you uncheck the 'Parse SQL' option, press the 'Get Column Info' button to acquire the necessary meta-data.



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.

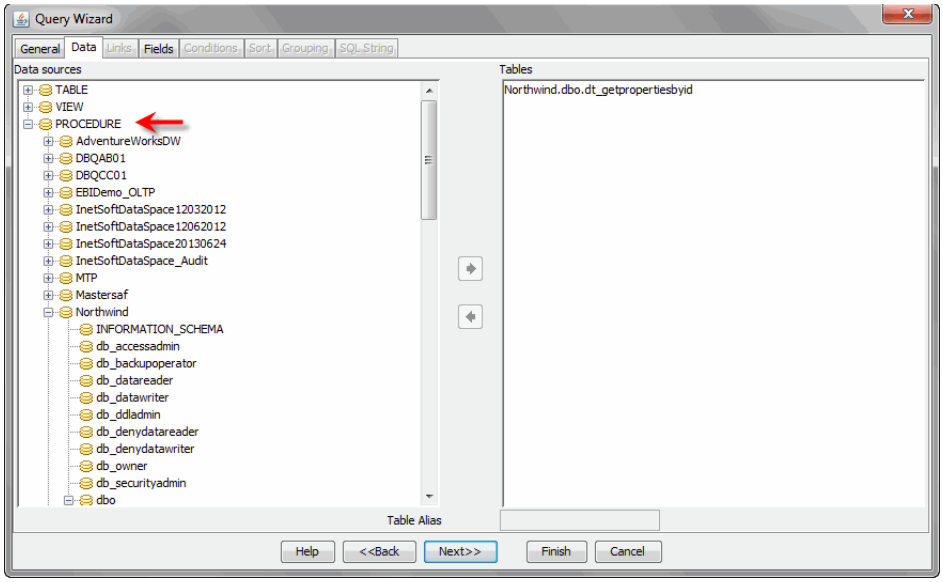
7. To execute the query and preview the result set, select the **Preview** tab at the bottom of the query definition screen.



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:

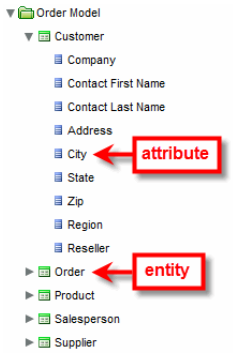
- Only one dataset is allowed in a stored procedure.
- In Oracle, only one cursor is allowed and it must be the last parameter in the stored procedure.
- In Oracle, only standard SQL types should be used in the parameters (user-defined types are not allowed).
- In PostgreSQL, create a free-form SQL query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

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

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

Creating a Data Model (Optional)

A *data model* is an (optional) simplified 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’.



Data models can greatly simplify the development of assets that rely on complex datasets. For information on how to create a data model, see *Relational Databases* in *Advanced Data Modeling*.

Transforming and Mashing Up Data

If your data source returns data that requires more sophisticated manipulation (e.g., filtering, grouping, sorting), or if you need to mashup data from multiple data sources, use a Data Worksheet. The following sections explain the features that Data Worksheets provide for transformation and mashup.

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

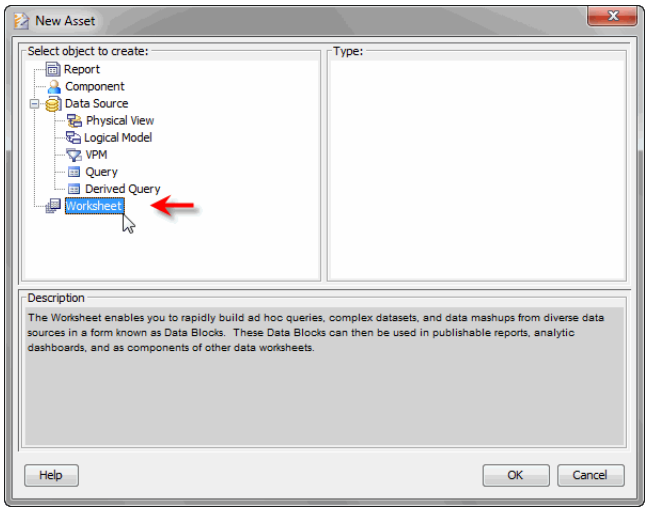
To create a new Data Worksheet in Style Studio, follow the steps below:

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

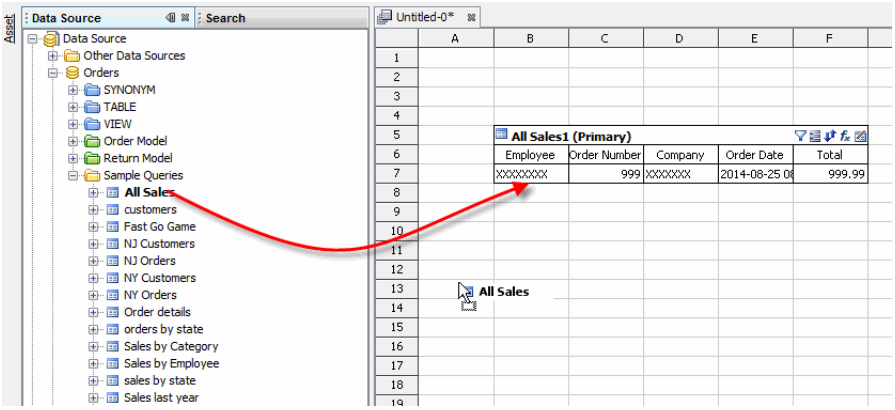


2. Press the 'New' button in the Style Studio toolbar. This opens the 'New Asset' dialog box.

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

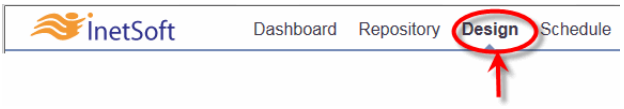
Creating a Data Block, in *Advanced Data Mashup*, for full information on hot to create data bocks.

Creating a Worksheet in the User Portal

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

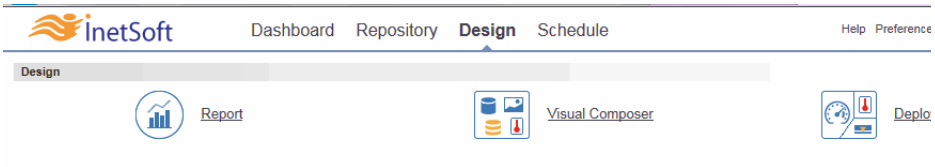
See *Using the Portal* for more information about the User Portal.

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.



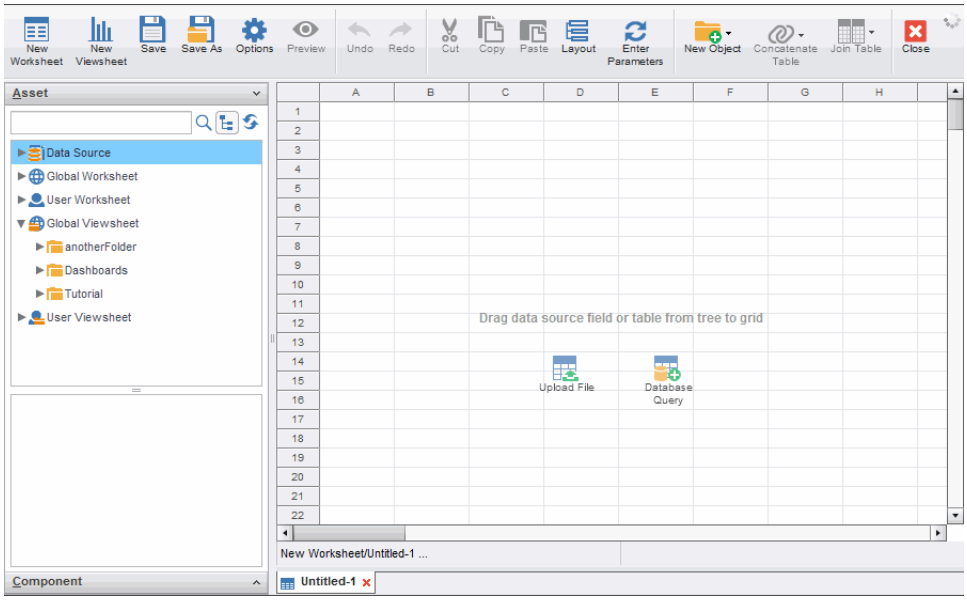

```
Note: Style Report EE does not provide access to Visual Composer.
See Creating a Worksheet in Style Studio.
```

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

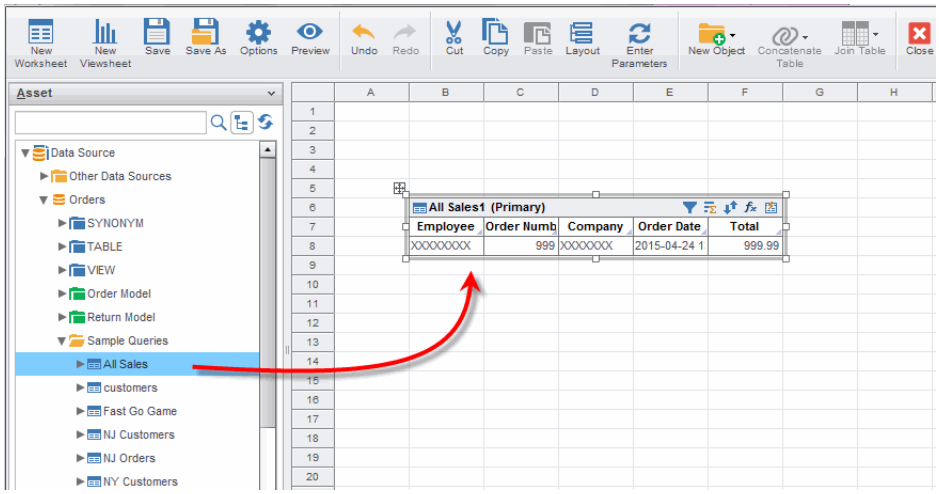




4. Press the 'New Worksheet' button in the Visual Composer toolbar. This opens a blank Worksheet named 'Untitled'.



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

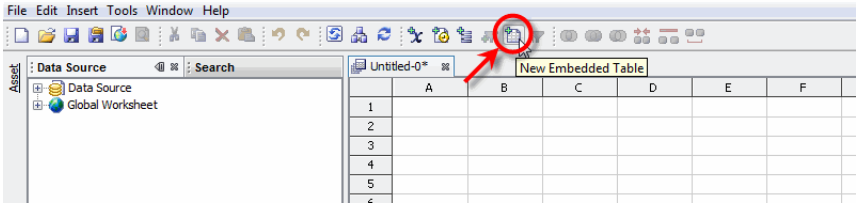
Importing a Data File

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.

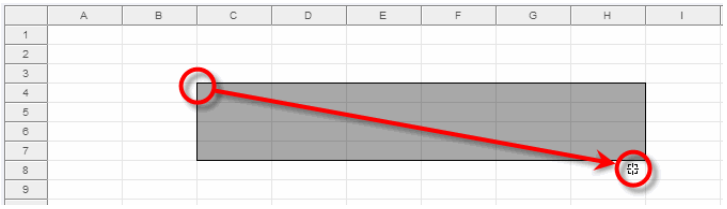
Follow the steps below to import the sample Excel file into a Data Worksheet.

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

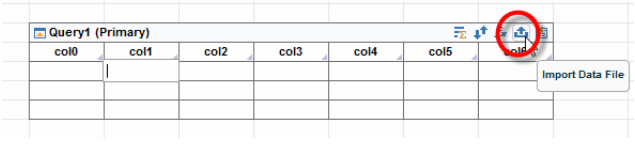
2. Press the 'New Object' button on the toolbar, and select 'Embedded Table'. This changes the standard pointer into a crosshairs pointer.



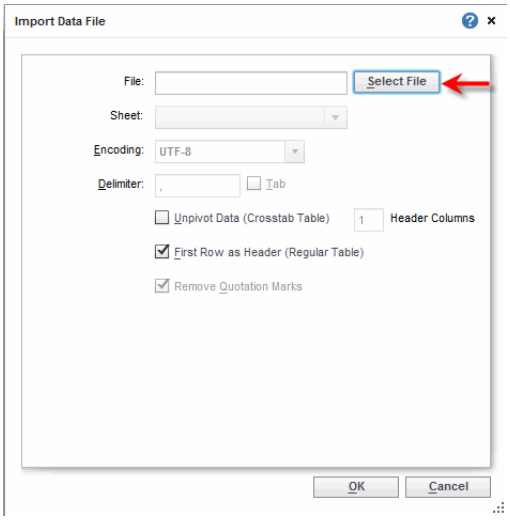
3. Drag the crosshairs pointer on the Worksheet grid to create a new table. (It does not matter how tall or wide you make the table.)



4. Press the 'Import Data File' button in the data block title bar. This opens the 'Import Data File' dialog box.



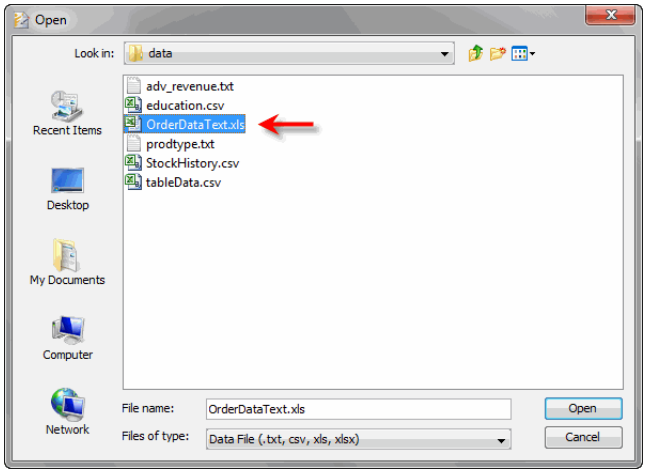
5. Press the 'Select File' button.



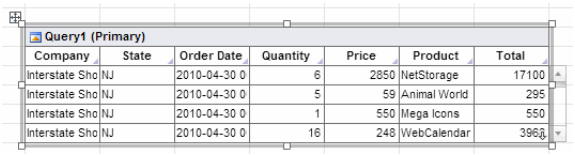
6. Select the Excel file you want to import.

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

   `{Installation Folder}\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.



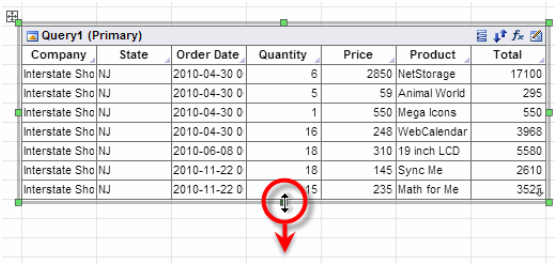
Query1 (Primary)

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**.

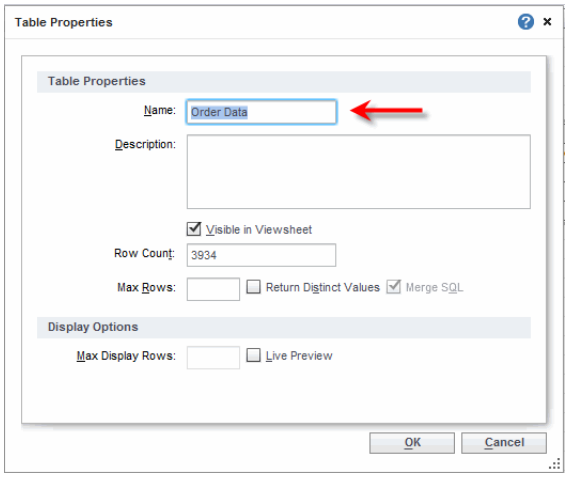
8. To display more rows in the table (also called a “data block”), drag down the bottom border of the table.



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

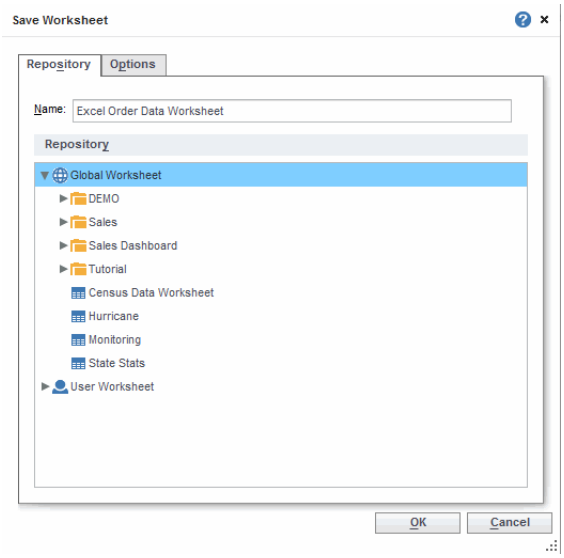
9. Right-click on the table, and select ‘Properties’ from the context menu. This opens the ‘Table Properties’ dialog box.

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.

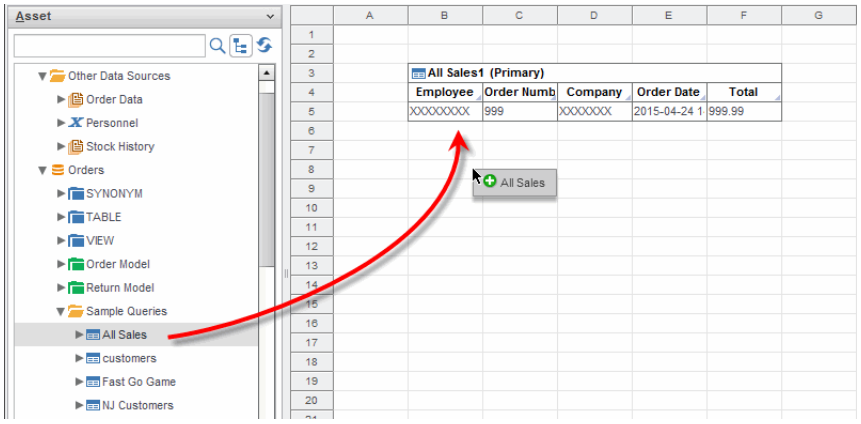
Creating a Report, for information on creating a report based on a Data Worksheet.

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.

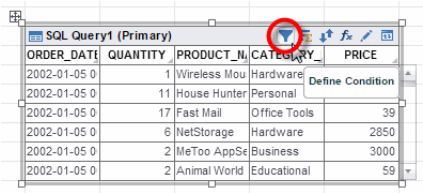
1. Create a new Data Worksheet. See *Creating a Data Worksheet*.

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.

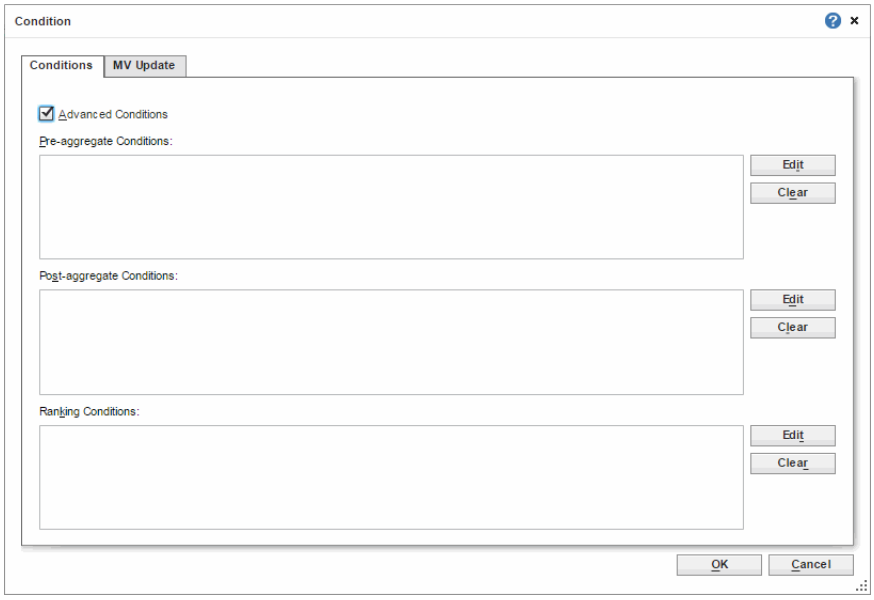




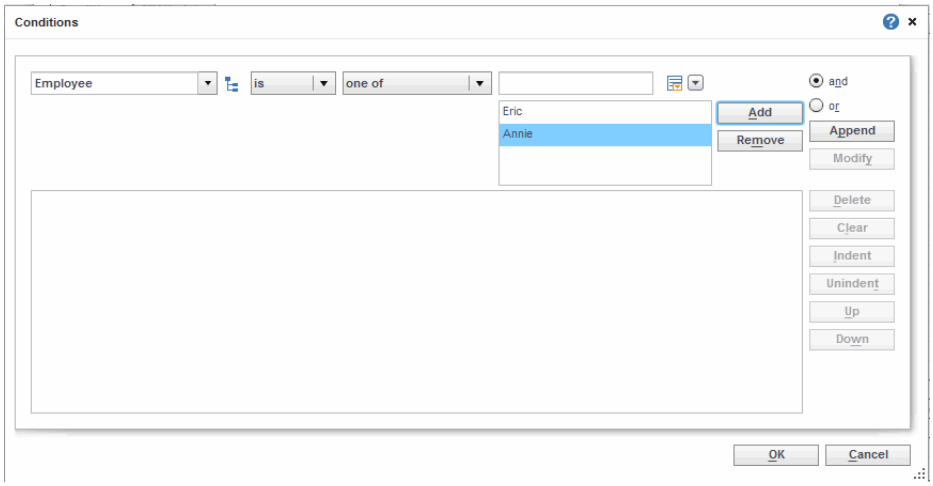
3. Press the 'Define Condition' button on the title bar of a data block. This opens the 'Condition' dialog box.



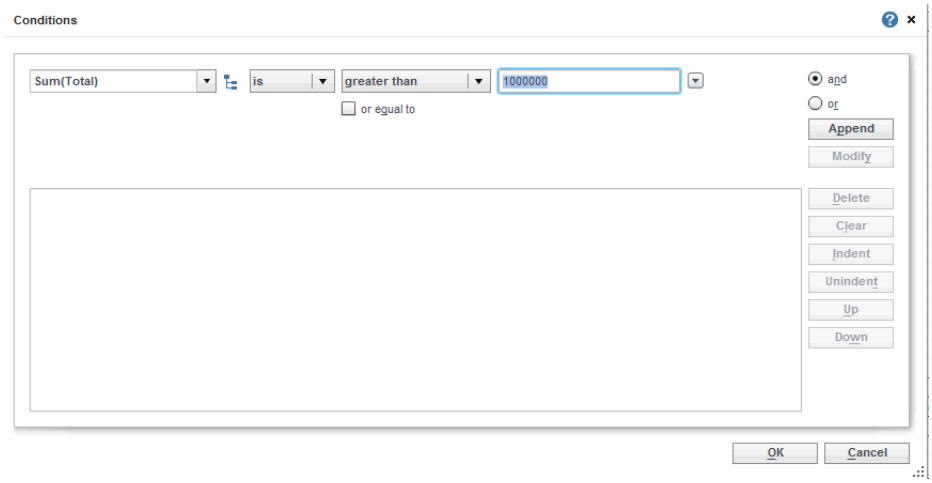
4. Select the 'Advanced Conditions' check box.



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.

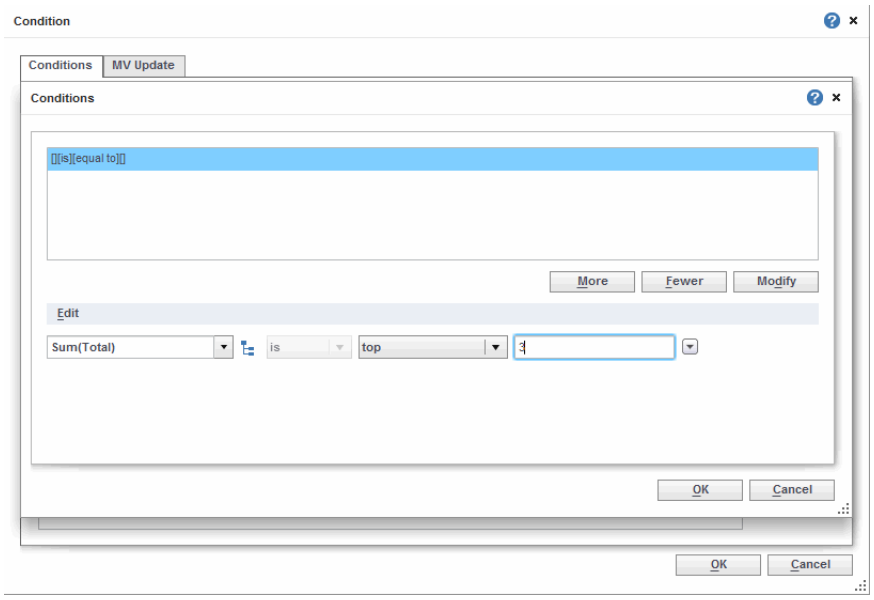


6. To create a **post-aggregate condition** (a condition that applies to summarized data following grouping and aggregation), press the 'Edit' button next to the 'Post-aggregate Conditions' panel. Use the menus to specify the desired condition. Then press 'Append' to add the condition and 'OK' to close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

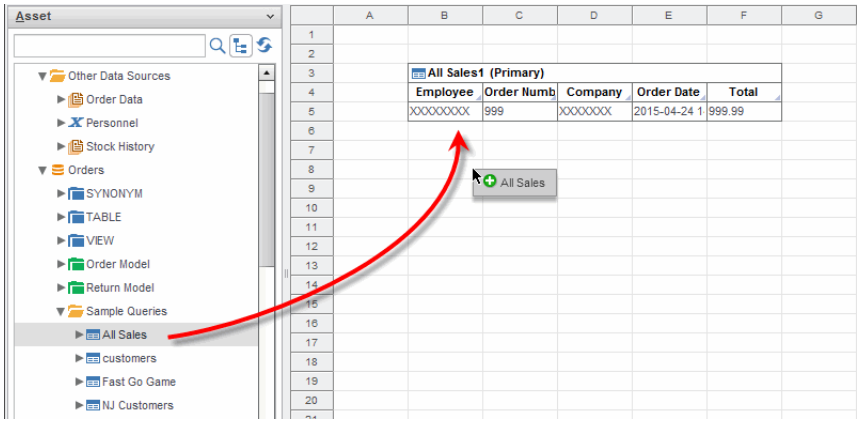
Filtering, in *Advanced Data Mashup* for full information on filtering data blocks.

Grouping and Aggregating Data

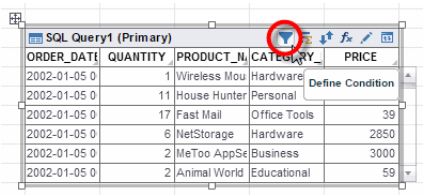
To group and aggregate data, follow the steps below.

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

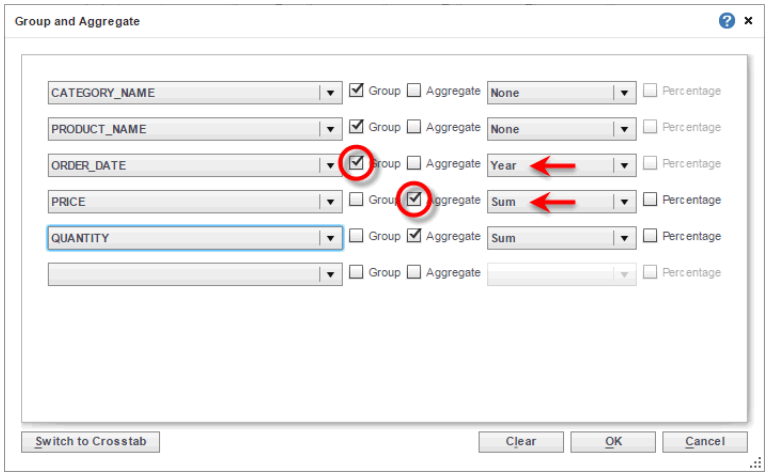
2. From the 'Data Source' listing in the **Asset** panel, drag in the query you want to group and aggregate. This creates a new data block in the Data Worksheet.



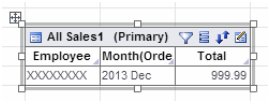
3. Press the 'Group' button on the title bar of a data block. This opens the 'Aggregate' dialog box.



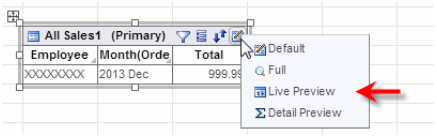
4. Use the menus to select grouping fields and summary fields. For date-type fields, select the desired level of date grouping ('Year', 'Month', etc.). For summary fields, select the desired method of aggregation ('Sum', 'Average', etc.).



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.

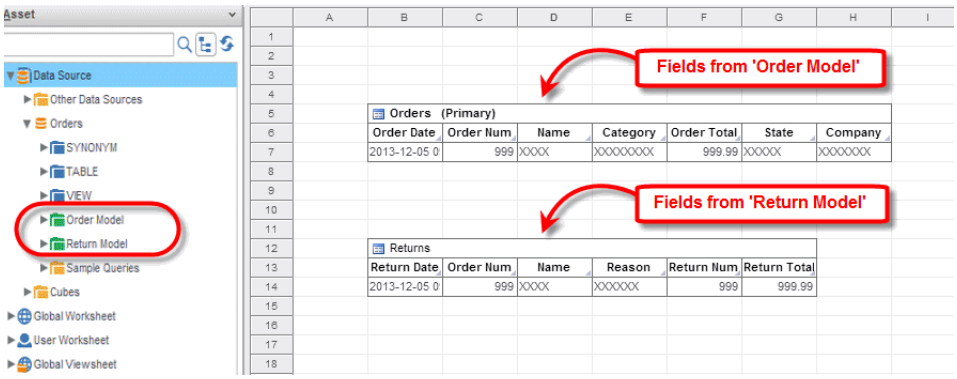
To join tables together using an inner join, outer join, or inequality join, follow the steps below:

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.

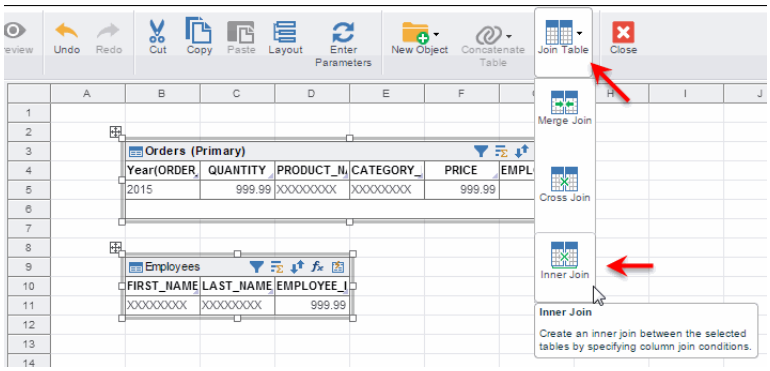


You can do the same mash-up operations with any data blocks, whether they are derived from database tables, queries, data models, or Data Worksheets, and whether they are from the same data source or from different data sources.

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.)

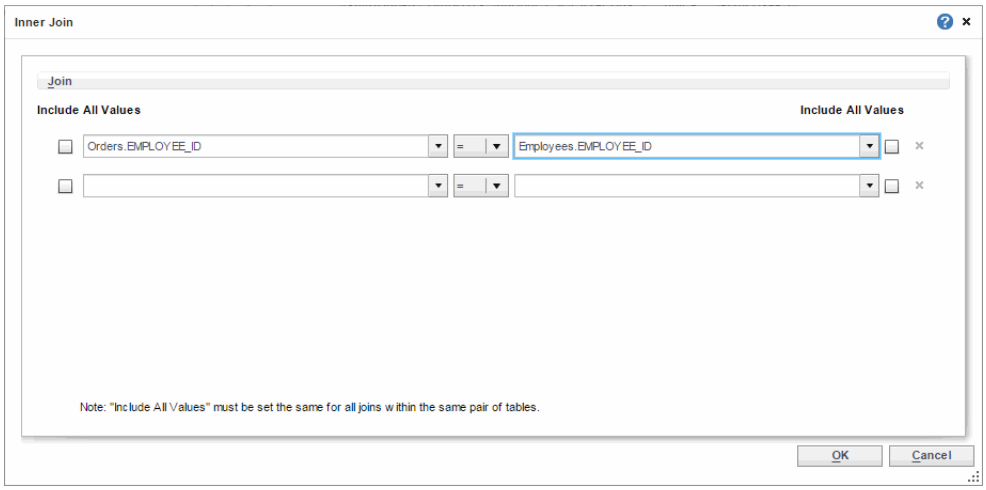
You can also create a join by dragging columns. See *Creating a Inner Join by Dragging Columns* in *Advanced Data Mashup*.

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.

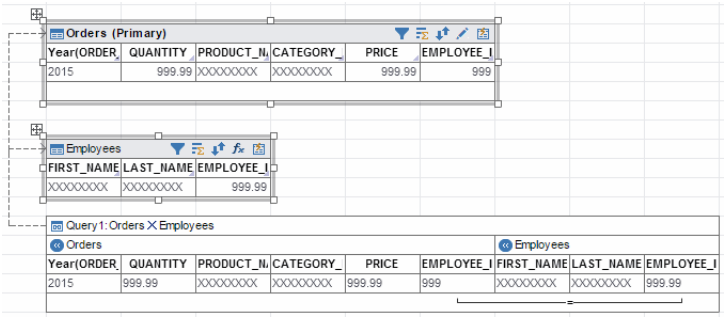
```
Orders.Order Num = Returns.Order Num
```

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
```

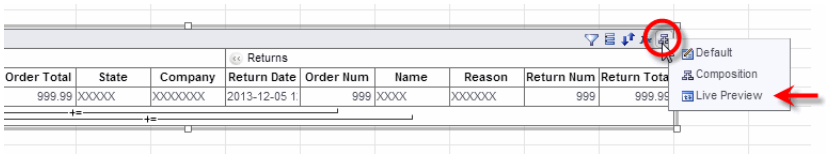
6. Select the 'Include All Values' option next to the left-side 'Orders' data block fields. This creates a *left outer join*, which includes all values from the 'Orders' table in the result, even if these orders have no matches in the 'Returns' data block.

7. Press 'OK' to create the joined data block.

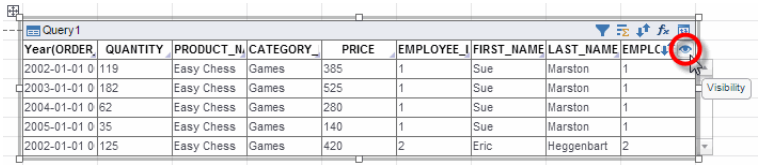


The resulting data block is displayed in 'Composition View', which shows the original 'Orders' and 'Returns' tables and how they are joined.

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

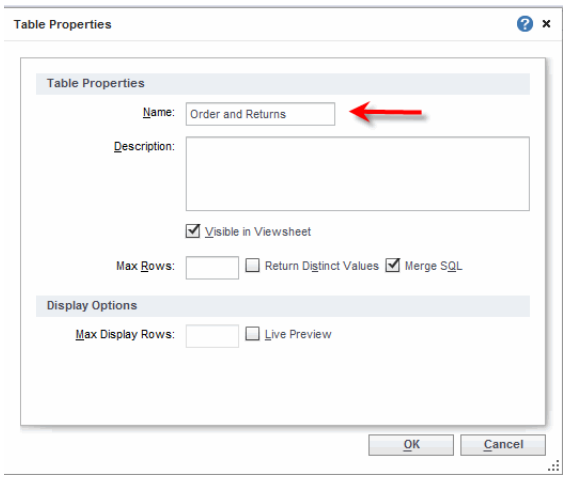


9. To remove duplicate columns (e.g., the join columns), press the 'Visibility' icon in the column header.

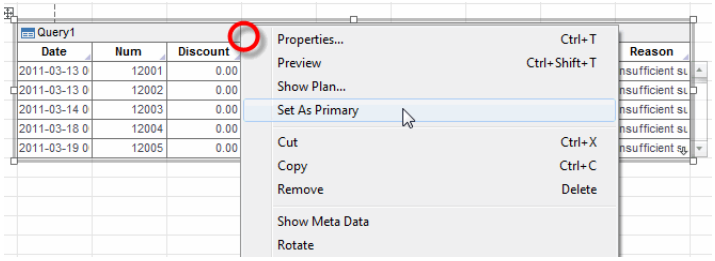


10. To change column names, double click the column header and edit the text.

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

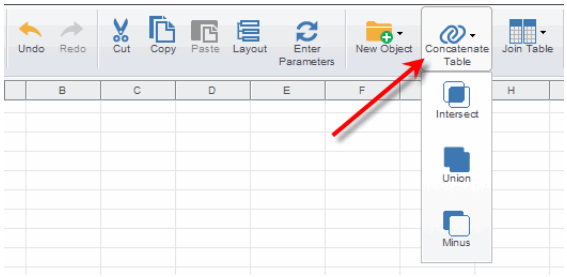
Union, intersection, and difference operations are known as table *concatenations*. Tables must have the same number of columns in order to participate in concatenations, and corresponding columns of each table must have the same data type.

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.

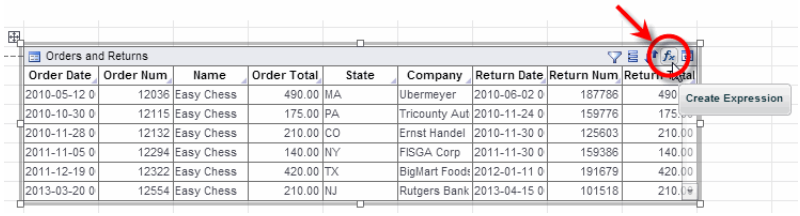
Creating an Expression

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.

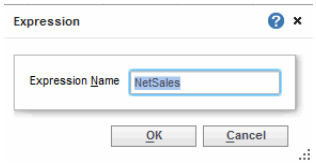
To add an expression column to a data block, follow the steps below:

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.



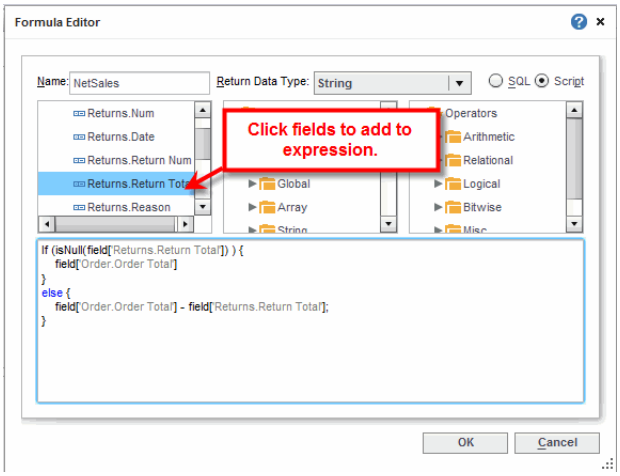
3. In the 'Input' dialog box, enter the name of the new column. Press 'OK' to open the Expression Editor.



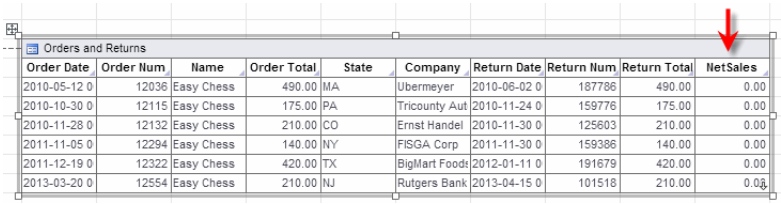
4. At the top of the Expression Editor, select either 'SQL' or 'Script'. Select 'SQL' (if available) to write the expression using your database's SQL syntax. Select 'Script' to write the expression using JavaScript.

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

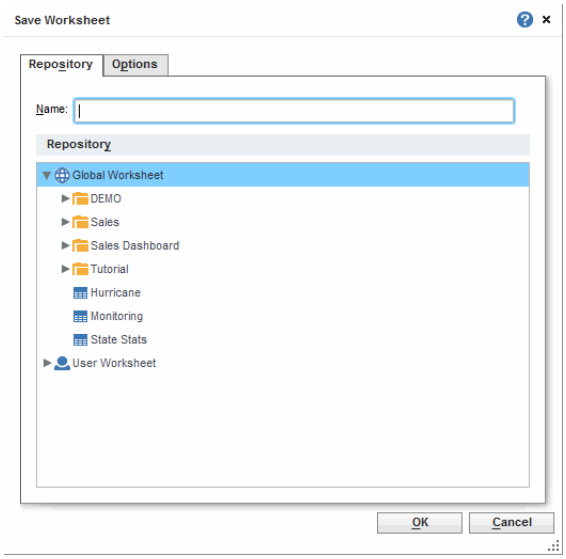
Saving a Data Worksheet

To save a Data Worksheet, follow these steps:

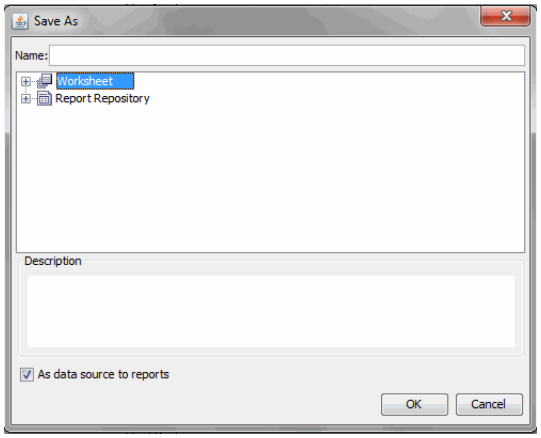


1. Press the ‘Save’ button in the toolbar, or press Ctrl-S on the keyboard. This opens the ‘Save Worksheet’ dialog box.

   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.

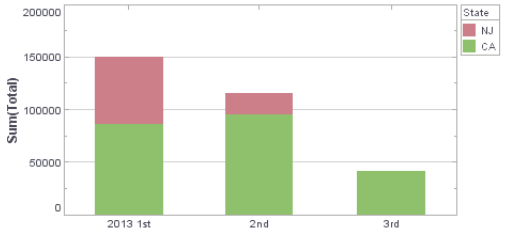
Data Worksheets are stored in the *asset.dat* file or *asset.dat.d* directory in the working repository.

Creating a Report

Reports are designed for pixel-perfect paginated output, and provide certain built-in interactivity such as filtering and sorting. To build a report, you add components such as text, tables, and charts to the report page. The report engine will automatically generate as many pages as need to display all of the data. Reports can retrieve data from queries, data models, or Data Worksheets.

The following sections illustrate how to create a simple sales report such as the one below.

Sample Sales Report



Order Date	Company	State	Product	Quantity	Price	Total
2013 1st	Computer Tech	CA	Sync Me	10	$145	$1,450
	Computer Tech	CA	WebCalendar	11	$248	$2,728
	Computer Tech	CA	Wireless Keyboard	17	$40	$680
	Computer Tech	CA	Fancy Menus	19	$49	$931
	Computer Tech	CA	Animal World	6	$59	$354
	Computer Tech	CA	NetStorage	8	$2,850	$22,800
	Computer Tech	**Company Total:**		71	$3,391	$28,943
	Daycare One	NJ	Math for Me	6	$235	$1,410
	Daycare One	NJ	Info Folder	18	$399	$7,182
	Daycare One	NJ	Combat Hero	14	$29	$406
	Daycare One	NJ	19 inch LCD	9	$310	$2,790
	Daycare One	NJ	Wireless Keyboard	12	$40	$480
	Daycare One	NJ	Mega Icons	14	$550	$7,700
	Daycare One	NJ	Barbie's Fashion	17	$79	$1,343
	Daycare One	NJ	Green Planet	16	$75	$1,200
	Daycare One	NJ	NetStorage	9	$2,850	$25,650
	Daycare One	NJ	17 Inch LCD	11	$210	$2,310
	Daycare One	NJ	Fancy Menus	20	$49	$980

Creating a Blank Report

Follow the steps below to create a new blank report:

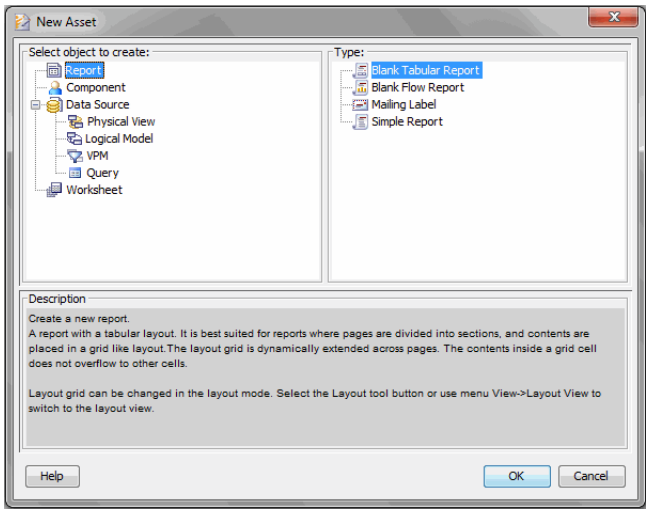
1. Start Style Studio from the Windows *Start* menu. (See *Using Style Studio* for information about configuring Style Studio.)



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

Most reports use a tabular rather than flow layout.

3. Select the 'Report' node in the left panel, then select 'Blank Tabular Report' in the right panel.



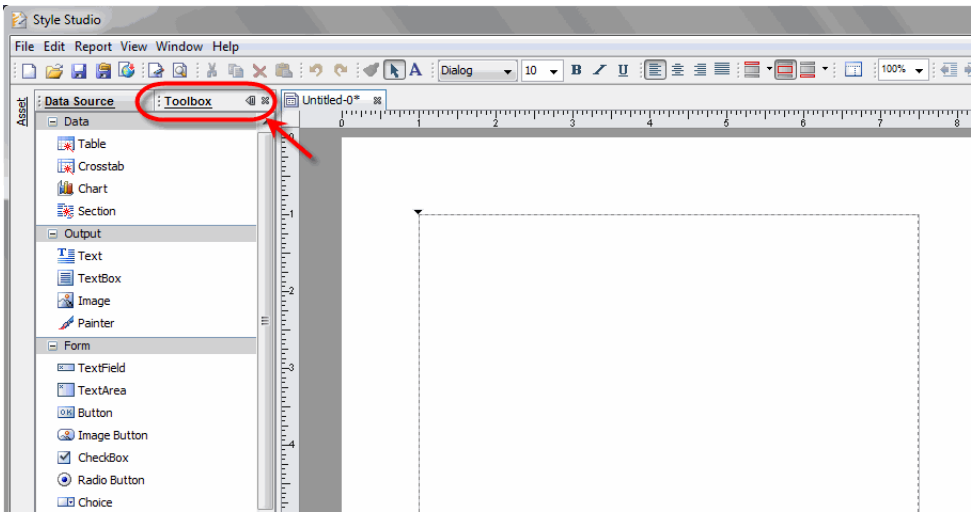
This opens a new blank report for editing. Continue to *Adding a Text Element*.

See Also

Creating a New Blank Report, in *Advanced Report Design*.

Adding a Text Element

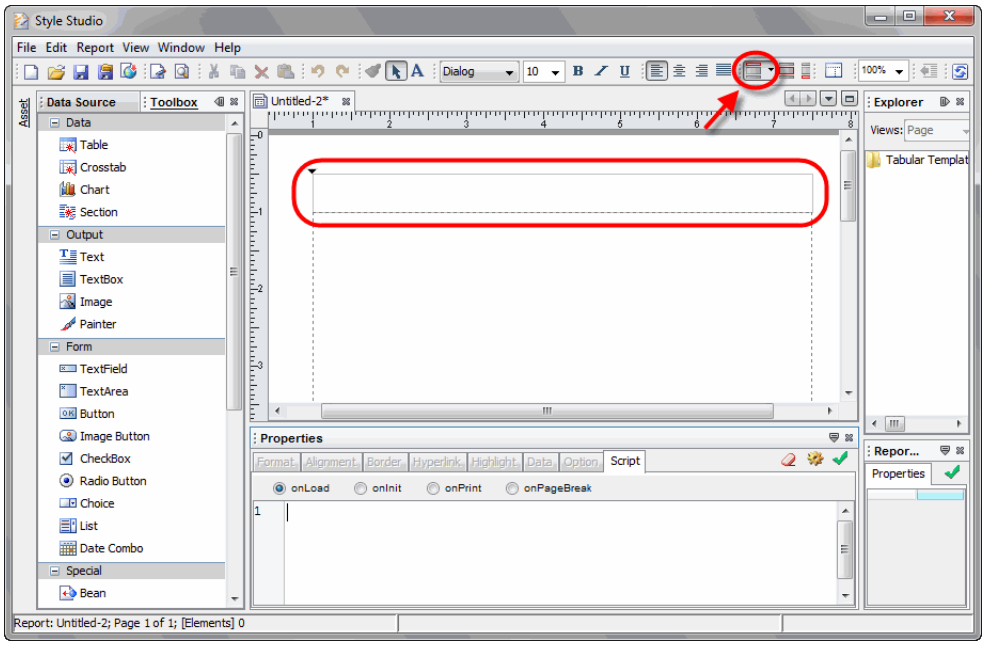
To add an element to a report, click the corresponding element in the **Toolbox** panel. (You can also drag the element into the report page itself.)



In the following steps, you will use Text elements to add a title and page numbering to the report:



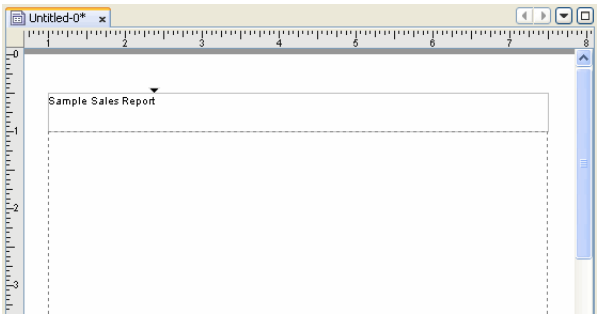
1. Press the 'Header' button in the top toolbar to select the header. The insertion point indicator (black triangle) appears at the top left edge of the header.





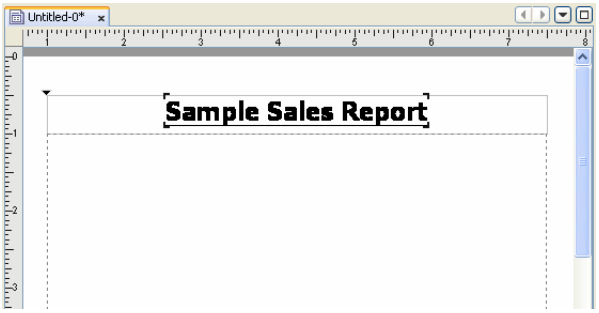
2. Click the ‘Text’ element in the **Toolbox** panel. A new Text element appears in the header of the report. (You can also drag the Text element into the Header region.)

3. In the Text field, type “Sample Sales Report”. Click outside of the element to finish editing.



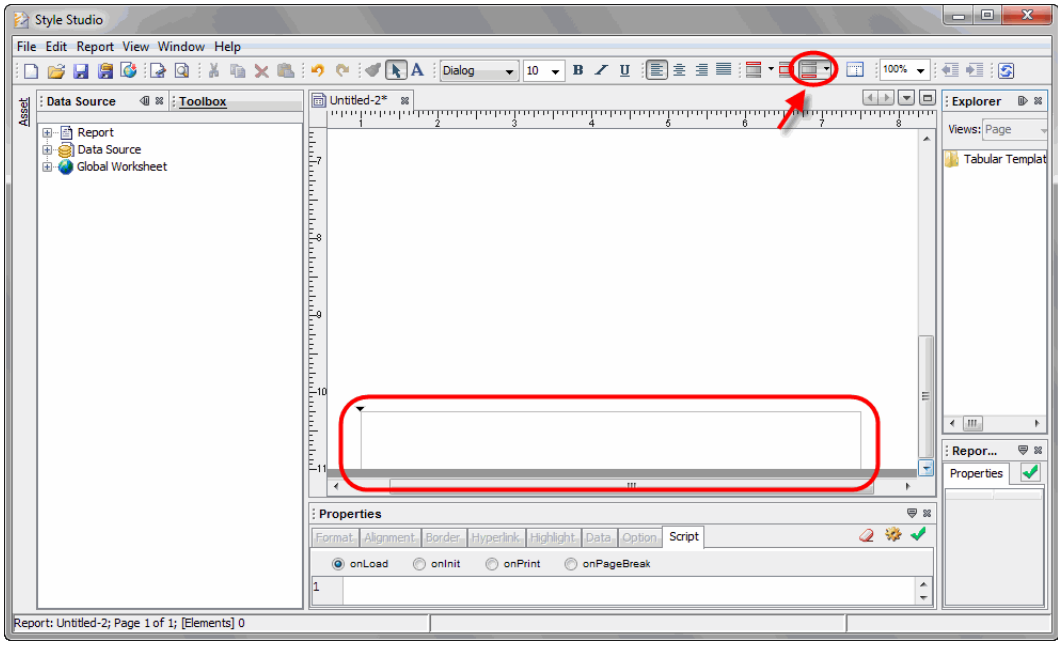


4. Click on the new Text element to select it. Use the controls on the toolbar to change the font to Tahoma 24-point Bold Underline. Then click the ‘Center’ button to center-align the Text element on the page.

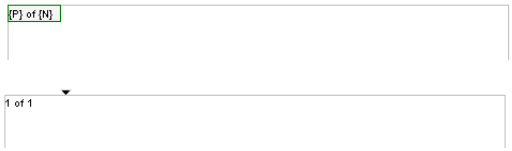




1. Press the ‘Footer’ button in the top toolbar. The insertion point indicator appears at the top left edge of the footer.

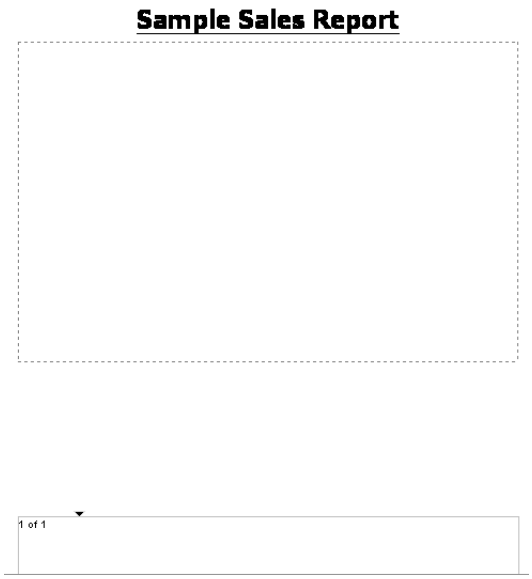


2. Click the 'Text' element in the **Toolbox** panel to add a new text element to the footer. In the text field, type "`{P} of {N}`".



The `{P}` and `{N}` tags designate the current page number and the total number of pages, respectively. The actual page numbers are created by the report engine when the report is generated. (These special tags can only be used in a report header or footer.)

3. Click anywhere outside of the footer to deselect it. The footer is now complete.



Continue to *Adding a Table*.

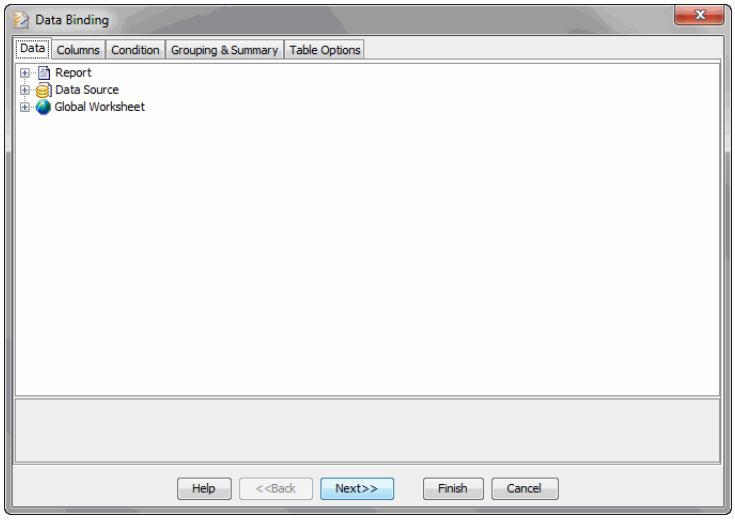
See Also
Text Element, in *Advanced Report Design*.
Headers and Footers, in *Advanced Report Design*.

Adding a Table

You will now add a new Table element. The table will display several columns of a data block in the “Excel Order Data Worksheet”. To create this Worksheet, see *Importing a Data File* in *Connecting to Your Data*.

Follow the steps below:

1. Click once on the ‘Table’ element in the **Toolbox** panel. This opens the Data Binding Wizard.

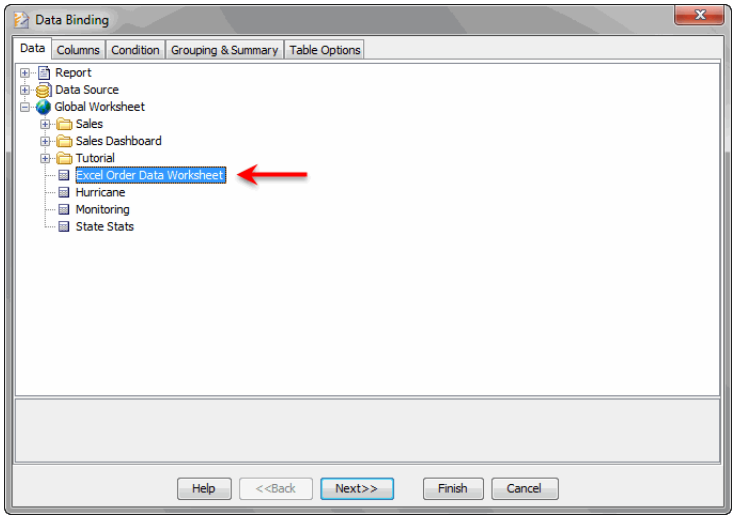


`The Data Binding Wizard is only for initial set-up. It cannot be accessed after you have created the Table.`

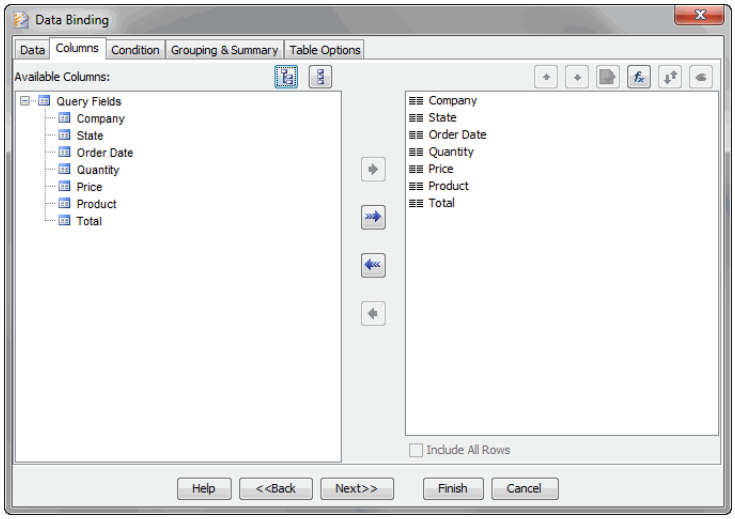
The Data Binding Wizard helps you to initially set up your table. When you complete the Wizard you will be able to continue making modifications. *Data binding* is the process of associating a Worksheet or other data source with a report element. When the report is generated, the data retrieved from the data source is placed into the element.

To create the ‘Excel Order Data Worksheet’ see *Importing a Data File* in *Connecting to Your Data*.

2. Under the **Data** tab, expand the ‘Global Worksheet’ node. Select the ‘Excel Order Data Worksheet’.



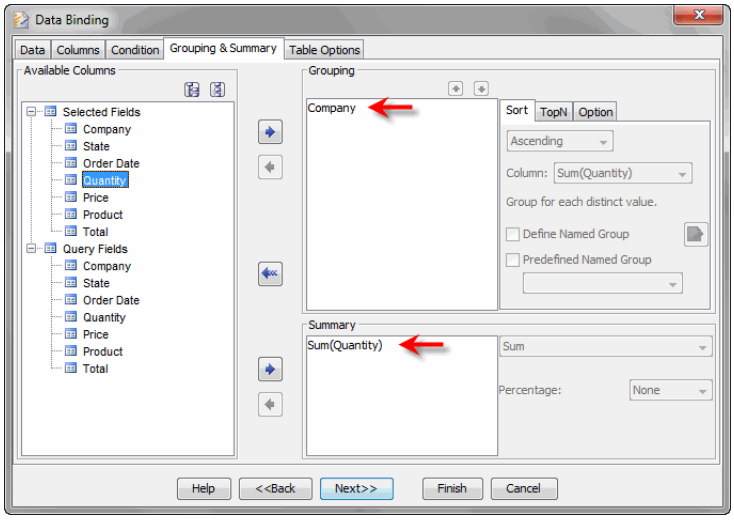
3. Under the **Columns** tab, verify that all columns are present in the right panel. If they are not, then drag the desired columns from the 'Available Columns' panel to the right panel.



4. Create groups based on the 'Company' field and summarize the 'Quantity' for each company. Follow the steps below:

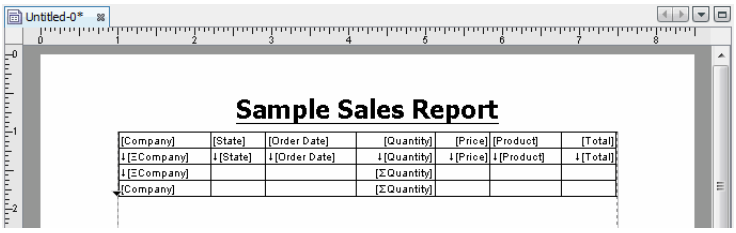
   a. Select the **Grouping and Summary** tab.

   b. From the 'Available Columns' panel, drag the 'Company' field to the 'Grouping' panel. Drag the 'Quantity' field to the 'Summary' panel.

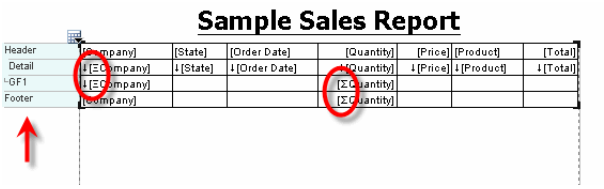


You can add additional grouping and summary fields to the 'Grouping' and 'Summary' panels in the Wizard. However, it is also important to understand how grouping can be added to an existing Table, so in this example you will add the additional grouping after the Table has been created.

5. Press 'Finish' to create the new table.



6. Click on the table and observe the grouping structure.

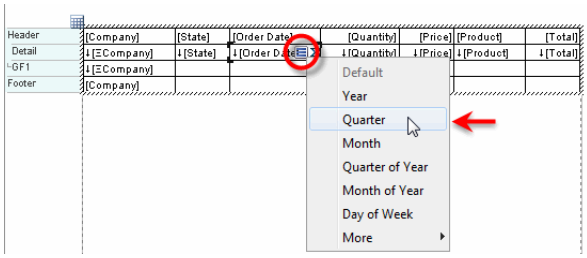


The Table displays only *meta-data* by default in design view, not actual data. In *Previewing a Report*, you will see how to display data.

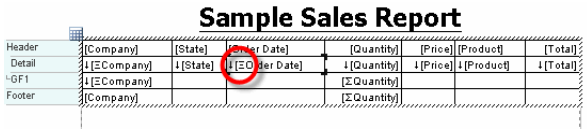
The 'Header' row contains the columns header labels. The 'Detail' row will expand to display all of the detail records grouped by 'Company'. (Note the grouping symbol on the 'Company' cell.) The 'GF1' row will display the summarized 'Quantity' for each group. (Note the summarization symbol on the 'Quantity' cell.) The 'Footer' row will display the grand summary for the entire table.

7. In the Data Binding Wizard you previously grouped on ‘Company’ and summarized by ‘Quantity’. Now group the records also by ‘Order Date’ (quarter), and summarize by ‘Total’. Follow the steps below:

   a. Move the mouse over the ‘Order Date’ cell in the ‘Detail’ row, and press the ‘Group’ button. In the menu, select ‘Quarter’.



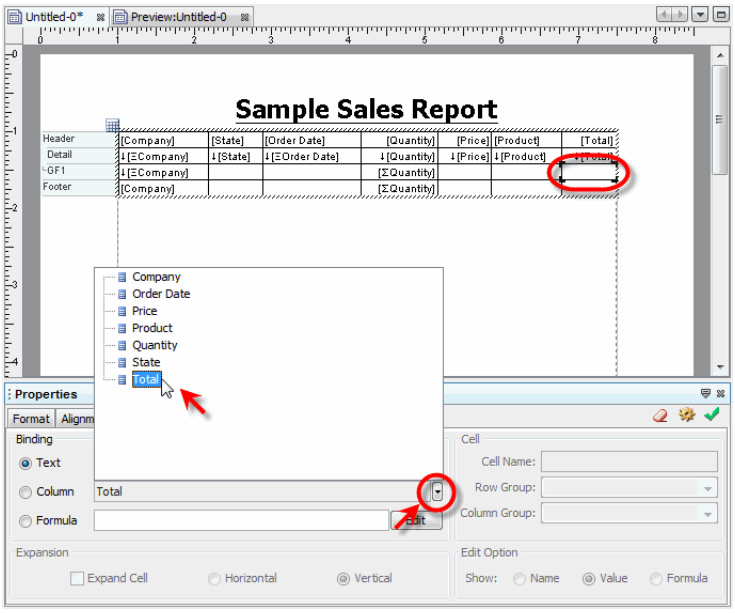
   This designates ‘Order Date’ as a grouping field, and the cell now displays the grouping symbol.



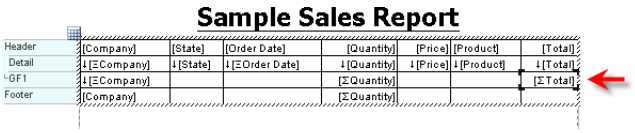
   b. Click on the empty cell in the ‘GF1’ row of the ‘Total’ column to select the cell.

   c. In the **Properties** panel at the bottom of Style Studio, select the **Data** tab.

   d. Expand the menu next to the ‘Column’ option in the ‘Binding’ panel, and select ‘Total’.

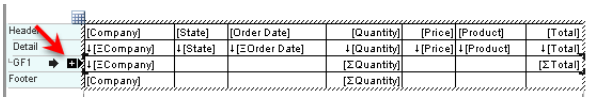


This specifies that the ‘Total’ field (as well as the ‘Quantity’ field) should be summarized for each individual company. The ‘GF1’ band corresponds to “group 1” which is the ‘Company’ grouping.

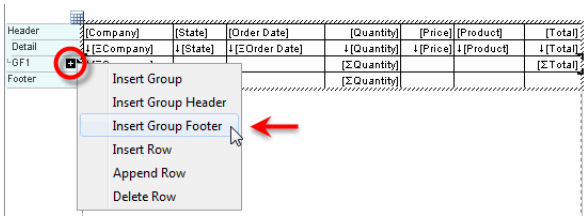


However, in this case it makes sense to summarize the ‘Quantity’ and ‘Total’ also for each date grouping, that is, for each quarter. To do this, you will add a new grouping footer in the following steps.

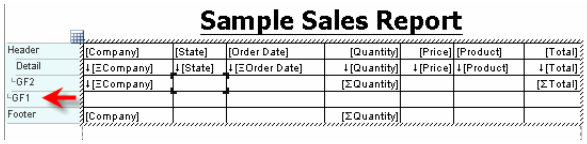
e. Click in the left margin of the table next to the ‘GF1’ row. This displays a small ‘+’ icon next to the row.



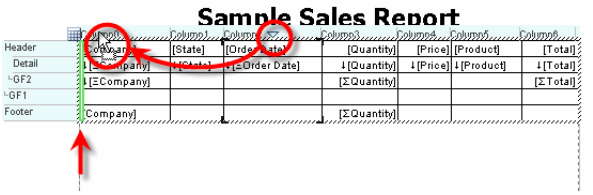
f. Click the ‘+’ icon and select ‘Insert Group Footer’ from the menu.



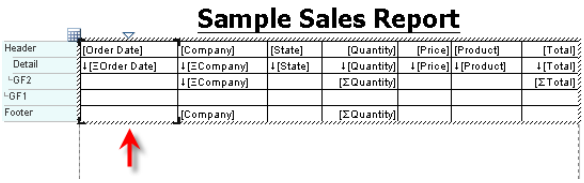
This adds a new group footer called ‘GF1’. The previous ‘GF1’ row is now called ‘GF2’, indicating that there are two levels of grouping.



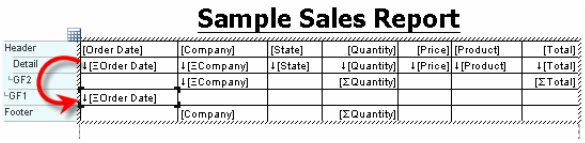
8. Move the ‘Order Date’ column to the far left of the table, since it makes sense to use ‘Order Date’ as the outer grouping level. To do this, click immediately above the column header, and drag the column to the far left of the table.



The green bar shows you where the column will be placed.



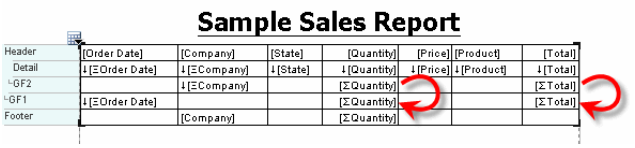
9. Click on the ‘Order Date’ field in the ‘Detail’ row to select it. Right-click on the cell and select ‘Copy’ from the context menu. Now click on the empty cell in the ‘GF1’ row of the ‘Order Date’ column to select it. Right-click and select ‘Paste’ from the context menu.



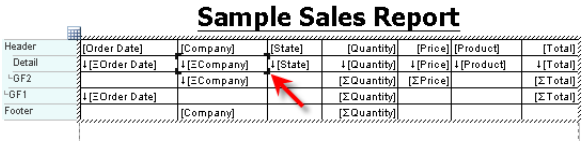
This copies the ‘Order Date’ field into the ‘GF1’ row so that the date group will appear in the summary row.

You can also make copies by using Ctrl-C and Ctrl-V on the keyboard or by holding down the Ctrl key while dragging on a field.

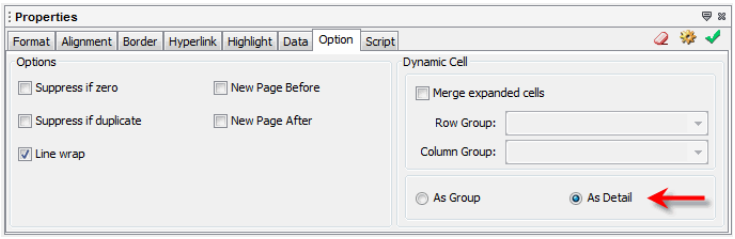
10. Repeat the above steps to copy the ‘Quantity’, ‘Price’, and ‘Total’ fields into the ‘GF1’ row to display these values in the date summary row.



11. Click to select the ‘Company’ cell in the ‘Detail’ row.



12. In the bottom **Properties** panel, select the **Option** tab. In the ‘Dynamic Cell’ panel, select ‘As Detail’.



This will cause the ‘Company’ field values to be displayed on each detail row (rather than just once at the start of the group).

You are now finished with the data binding of the table. You will add formatting the table later. First, proceed to *Saving a Report* and *Previewing a Report*.

See Also

Creating a Table Using the Data Binding Wizard, in *Advanced Report Design*.
Creating a Table Using Freehand Operations, in *Advanced Report Design*.
Grouping Data in a Table, in *Advanced Report Design*.

Saving a Report

See *Configuring Style Studio* for information on choosing a repository.

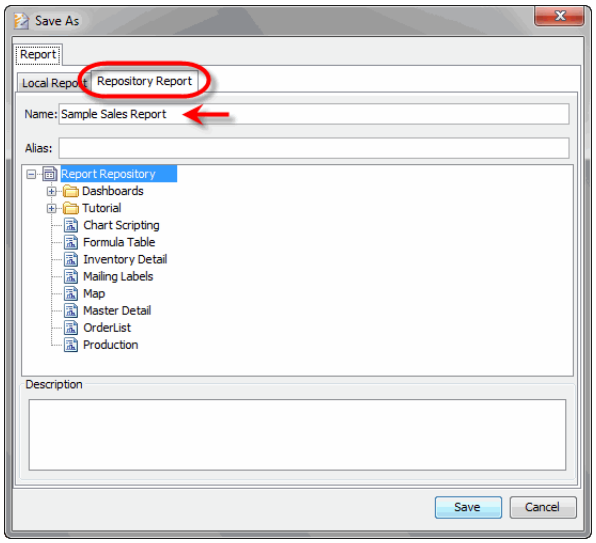
You will now save the report into the current working repository. Style Studio allows you to set up multiple repositories. The default working repository for Style Studio is the same as the default server repository (*sree/WEB-INF/classes*). This arrangement means that reports you save to the default repository will be immediately available to the server as well.

Follow the steps below to save the report into the working repository.



1. Press the 'Save' button on the toolbar to open the 'Save As' dialog box.

2. Select the **Repository Report** tab and enter "Sample Sales Report" as the 'File name'.



3. Press 'Save' to save the report.

When you save a report, you are saving the report page layout information and data binding information. No data is saved with the report. Data is drawn live from the data source when the report is generated by the user. Continue to *Previewing a Report*.

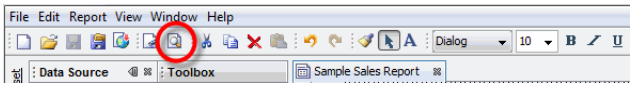
See Also

Saving a Report, in *Advanced Report Design*, for additional information.

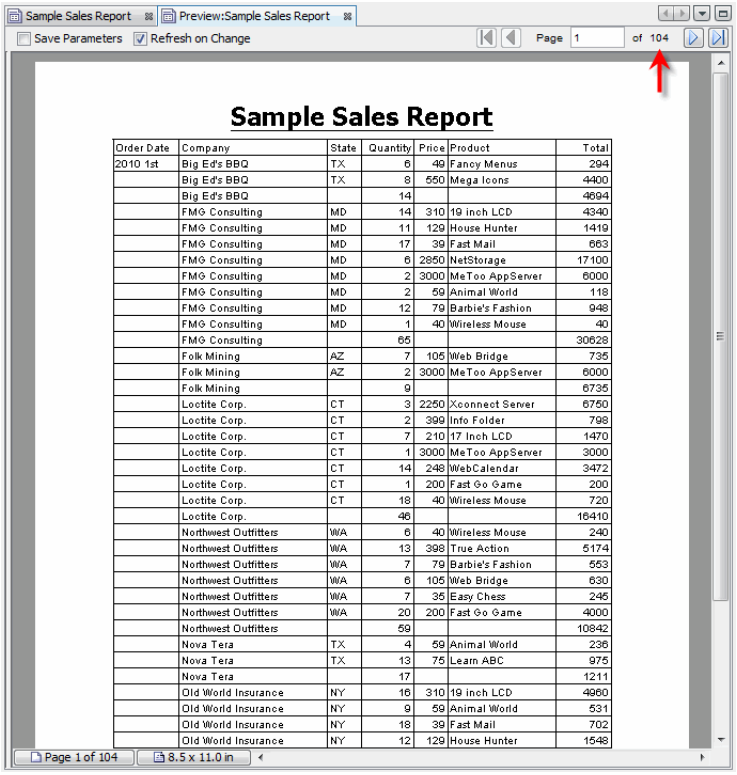
Previewing a Report

There are two ways to preview data in a report:

1. Press the ‘Preview’ button in the top toolbar.



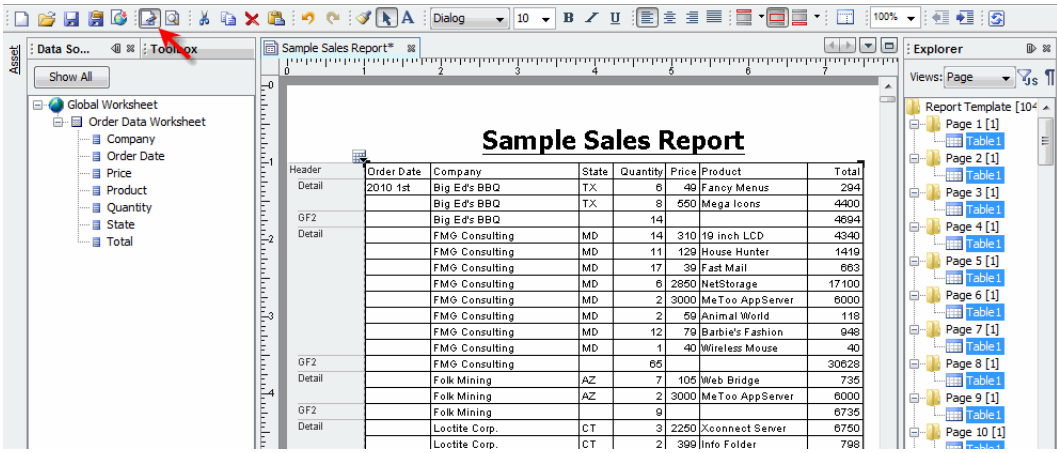
2. This previews the report in a new tab.



Order Date	Company	State	Quantity	Price	Product	Total
2010 1st	Big Ed's BBQ	TX	6	49	Fancy Menus	294
	Big Ed's BBQ	TX	8	550	Mega Icons	4400
	Big Ed's BBQ		14			4694
	FMG Consulting	MD	14	310	19 inch LCD	4340
	FMG Consulting	MD	11	129	House Hunter	1419
	FMG Consulting	MD	17	39	Fast Mail	663
	FMG Consulting	MD	6	2850	NetStorage	17100
	FMG Consulting	MD	2	3000	MeToo AppServer	6000
	FMG Consulting	MD	2	59	Animal World	118
	FMG Consulting	MD	12	79	Barbie's Fashion	948
	FMG Consulting	MD	1	40	Wireless Mouse	40
	FMG Consulting		65			30628
	Folk Mining	AZ	7	105	Web Bridge	735
	Folk Mining	AZ	2	3000	MeToo AppServer	6000
	Folk Mining		9			6735
	Loctite Corp.	CT	3	2250	Xconnect Server	6750
	Loctite Corp.	CT	2	399	Info Folder	798
	Loctite Corp.	CT	7	210	17 Inch LCD	1470
	Loctite Corp.	CT	1	3000	MeToo AppServer	3000
	Loctite Corp.	CT	14	248	WebCalendar	3472
	Loctite Corp.	CT	1	200	Fast Go Game	200
	Loctite Corp.	CT	18	40	Wireless Mouse	720
	Loctite Corp.		46			16410
	Northwest Outfitters	WA	6	40	Wireless Mouse	240
	Northwest Outfitters	WA	13	398	True Action	5174
	Northwest Outfitters	WA	7	79	Barbie's Fashion	553
	Northwest Outfitters	WA	6	105	Web Bridge	630
	Northwest Outfitters	WA	7	35	Easy Chess	245
	Northwest Outfitters	WA	20	200	Fast Go Game	4000
	Northwest Outfitters		59			10842
	Nova Tera	TX	4	59	Animal World	236
	Nova Tera	TX	13	75	Learn ABC	975
	Nova Tera		17			1211
	Old World Insurance	NY	16	310	19 inch LCD	4960
	Old World Insurance	NY	9	59	Animal World	531
	Old World Insurance	NY	18	39	Fast Mail	702
	Old World Insurance	NY	12	129	House Hunter	1548

Note that this report is over 100 pages long. (Paging controls are in the top-right corner.) Observe also the grouping on the ‘Company’ and ‘Order Date’ columns.

3. Return to the report design tab. Press the ‘Live Edit’ button in the toolbar. This populates the report with data in the design tab.



In ‘Live Edit’ mode you can continue to make modifications to report while the data is displayed. To turn off ‘Live Edit’ mode, simply press the ‘Live Edit’ button again.

The Table is functionally complete but doesn’t look very attractive. Continue to *Formatting a Table* to improve the appearance of the Table.

See Also

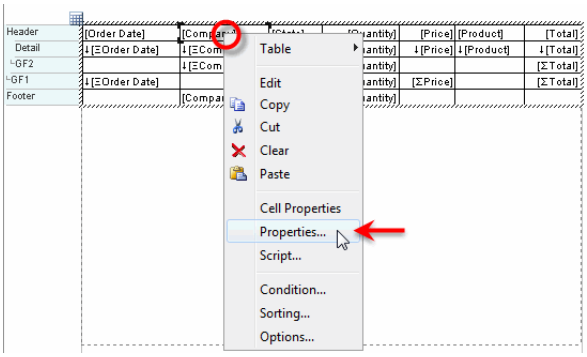
Displaying a Report in Live Edit View, in *Advanced Report Design*, for more information.

Displaying a Report in the Preview Tab, in *Advanced Report Design*, for more information.

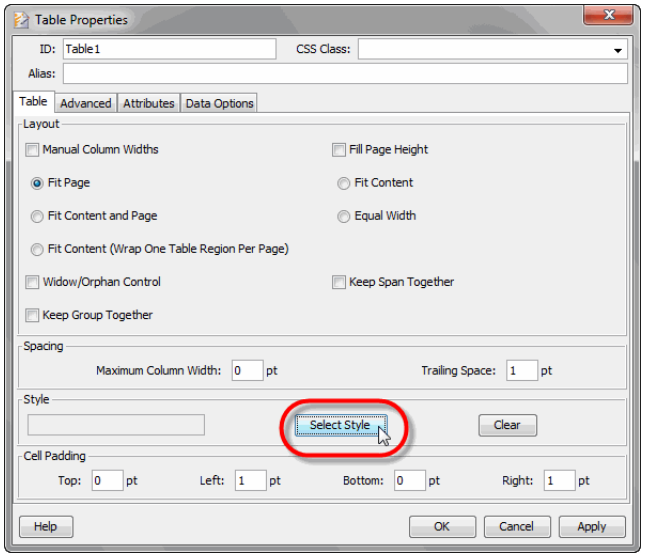
Formatting a Table

You will now proceed to format the table to improve the presentation. Follow the steps below.

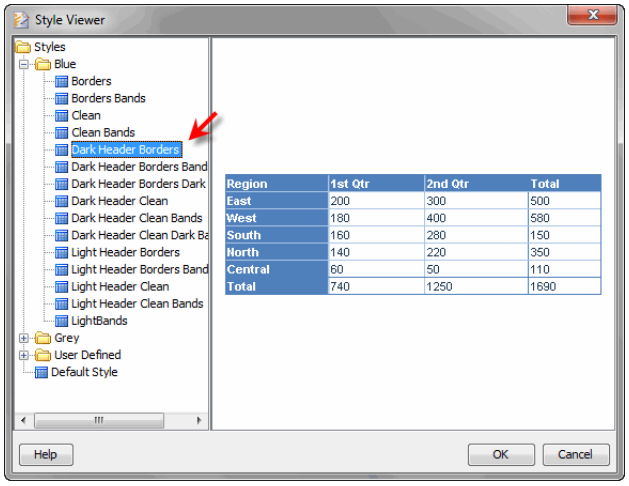
1. Right-click anywhere on the Table (or in the margin to the left of the Table), and select ‘Properties’ from the context menu. This opens the ‘Table Properties’ dialog box.



2. Press the ‘Select Style’ button to open the Style Viewer.

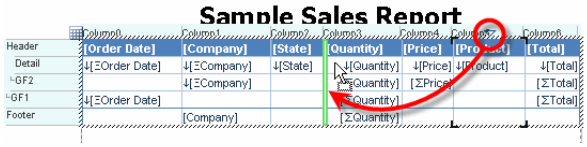


3. Select the ‘Dark Header Borders’ style and press ‘OK’.

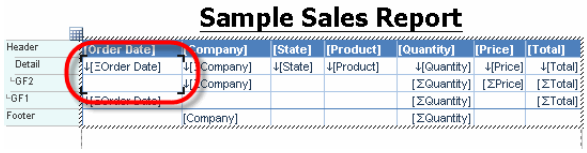


Press ‘OK’ again to dismiss the ‘Table Properties’ dialog box.

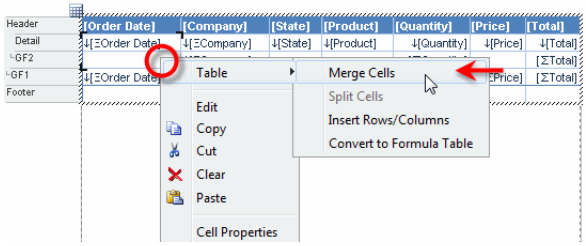
4. Move the ‘Product’ column to the left of the ‘Quantity’ column by dragging above the column (as you did earlier for the ‘Order Date’ column).



5. Click to select the ‘Order Date’ cell in the ‘Detail’ row. Then Shift-click to select also the empty cell in the ‘GF2’ row below it.

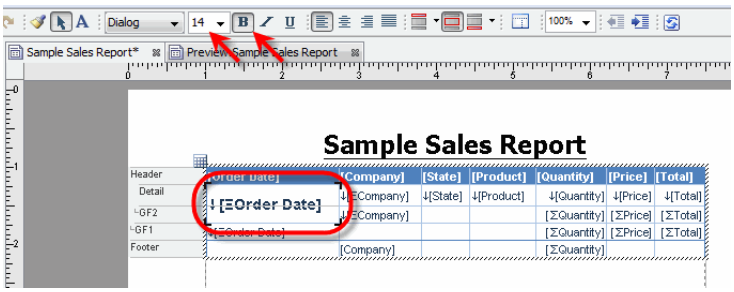


6. Right-click on either of the selected cells, and select ‘Table’ > ‘Merge Cells’ from the context menu.



This combines the two cells so that the date groupings (quarters) will not be interrupted by border lines.

7. Change the font of the date values to be large and bold. To do this, click to select the merged ‘Order Date’ cell, and use the toolbar font buttons.

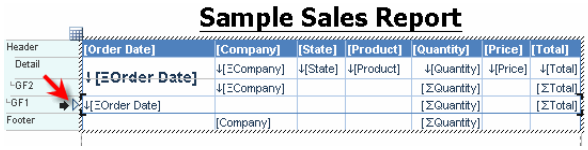


8. Preview the report to see the effect of these modifications.

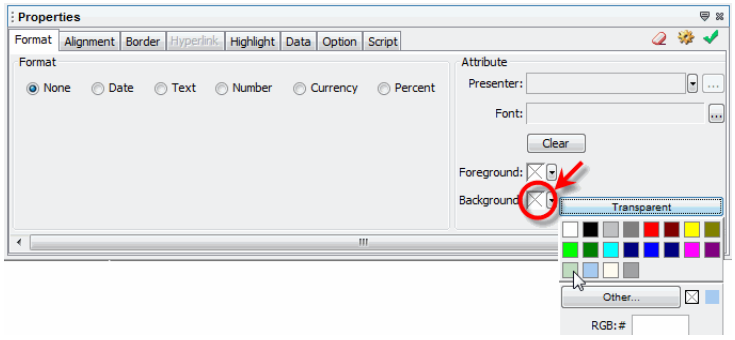
   Keep the **Preview** tab open so that you can switch back and forth between the design view and the preview. The **Preview** tab will automatically refresh to show changes you make in the design view.

9. Format the ‘Order Date’ summary row so that it is more visible. Follow the steps below:

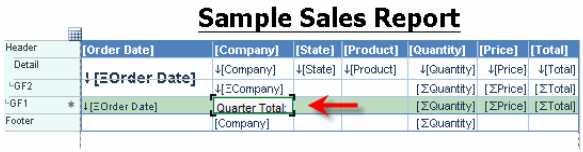
   a. Click in the left margin next to the ‘GF1’ row. This selects the entire summary row.



   b. In the bottom **Properties** panel, select the **Format** tab. From the ‘Background’ menu, select a light color.

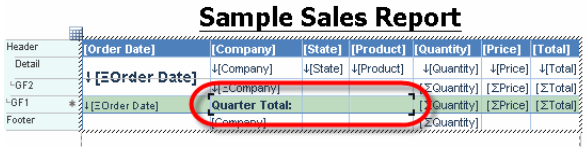


c. Click the cell to the right of the 'Order Date' cell in the 'GF1' row to select it. Type the following text into the cell: "Quarter Total:".



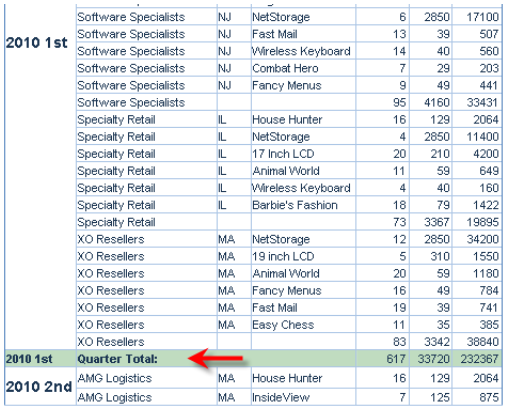
d. Use the toolbar font controls to make the text for the 'Order Date' cell and the 'Quarter Total' cell 11pt and bold.

e. Select the 'Quarter Total' cell and the two adjacent cells to the right.



Right-click on any selected cell, and select 'Table' > 'Merge Cells'.

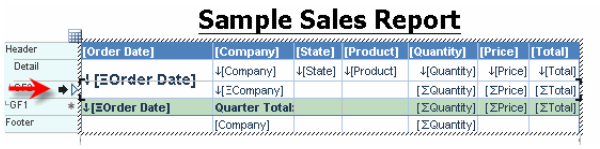
f. Preview the report and observe how the 'GF1' row is now displayed. (You may need to page forward in the report to find an example of the 'GF1' row.)



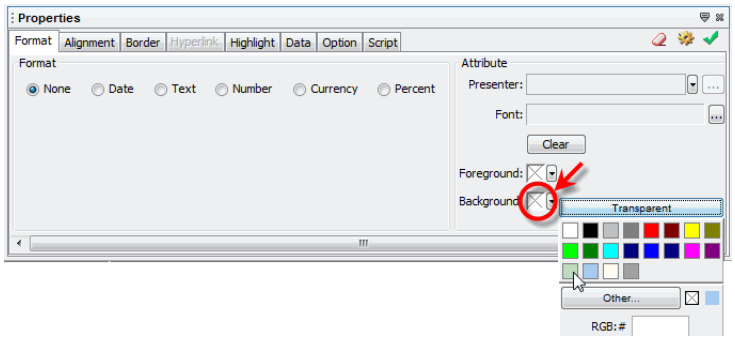
2010 1st	Software Specialists	NJ	NetStorage	6	2850	17100
	Software Specialists	NJ	Fast Mail	13	39	507
	Software Specialists	NJ	Wireless Keyboard	14	40	560
	Software Specialists	NJ	Combat Hero	7	29	203
	Software Specialists	NJ	Fancy Menus	9	49	441
	Software Specialists			95	4160	33431
	Specialty Retail	IL	House Hunter	16	129	2064
	Specialty Retail	IL	NetStorage	4	2850	11400
	Specialty Retail	IL	17 Inch LCD	20	210	4200
	Specialty Retail	IL	Animal World	11	59	649
	Specialty Retail	IL	Wireless Keyboard	4	40	160
	Specialty Retail	IL	Barbie's Fashion	18	79	1422
	Specialty Retail			73	3367	19895
	XO Resellers	MA	NetStorage	12	2850	34200
	XO Resellers	MA	19 inch LCD	5	310	1550
	XO Resellers	MA	Animal World	20	59	1180
	XO Resellers	MA	Fancy Menus	16	49	784
	XO Resellers	MA	Fast Mail	19	39	741
	XO Resellers	MA	Easy Chess	11	35	385
	XO Resellers			83	3342	38840
2010 1st	Quarter Total:			617	33720	232367
2010 2nd	AMG Logistics	MA	House Hunter	16	129	2064
	AMG Logistics	MA	InsideView	7	125	875

10. Format the 'Company' summary row so that it is more visible. Follow the steps below:

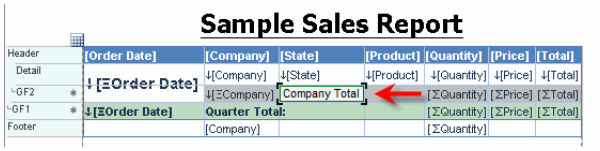
a. Click in the left margin next to the 'GF2' row. This selects the entire summary row.



b. In the bottom **Properties** panel, select the **Format** tab. From the 'Background' menu, select a light color.

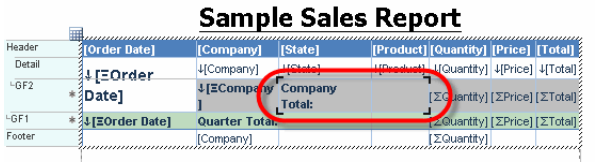


c. Click the cell to the right of the 'Company' cell in the 'GF2' row to select it. Type the following text into the cell: "Company Total:".



d. Use the toolbar font controls to make the text for the 'Company' cell and the 'Company Total' cell 11pt and bold.

e. Select the 'Company Total' cell and the adjacent cell to the right.



Right-click on any selected cell, and select 'Table' > 'Merge Cells'.

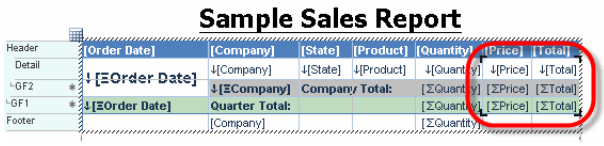
f. Preview the report and observe how the 'GF2' row is now displayed.

Sample Sales Report

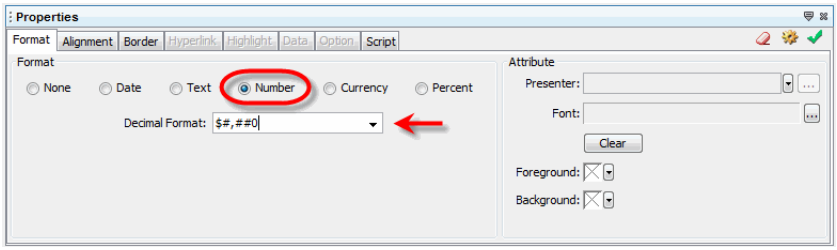
Order Date	Company	State	Product	Quantity	Price	Total
	Big Ed's BBQ	TX	Fancy Menus	6	49	294
	Big Ed's BBQ	TX	Mega Icons	8	550	4400
	Big Ed's BBQ	**Company Total:**		14	599	4694
	FMG Consulting	MD	19 inch LCD	14	310	4340
	FMG Consulting	MD	House Hunter	11	129	1419
	FMG Consulting	MD	Fast Mail	17	39	663
	FMG Consulting	MD	NetStorage	6	2850	17100
	FMG Consulting	MD	MeToo AppServer	2	3000	6000
	FMG Consulting	MD	Animal World	2	59	118
	FMG Consulting	MD	Barbie's Fashion	12	79	948
	FMG Consulting	MD	Wireless Mouse	1	40	40
	FMG Consulting	**Company Total:**		65	6506	30628
	Folk Mining	AZ	Web Bridge	7	105	735
	Folk Mining	AZ	MeToo AppServer	2	3000	6000
	Folk Mining	**Company Total:**		9	3105	6735
	Loctite Corp.	CT	Xconnect Server	3	2250	6750
	Loctite Corp.	CT	Info Folder	2	399	798
	Loctite Corp.	CT	17 Inch LCD	7	210	1470
	Loctite Corp.	CT	MeToo AppServer	1	3000	3000
2010 1st	Loctite Corp.	CT	WebCalendar	14	248	3472

11. Apply appropriate numerical formats to the measure columns. To do this, follow the steps below:

    a. Shift-click to select all of the ‘Price’ and ‘Total’ cells.

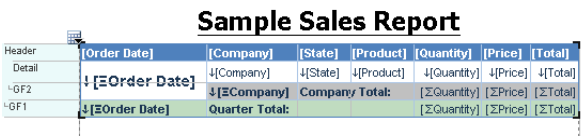


    b. Select the **Format** tab in the bottom **Properties** panel. Select the ‘Number’ format, and enter “`$#,##0`” in the ‘Decimal Format’ field. Press the green ‘Apply’ button.



    This format is the same as the ‘Currency’ format option, but omits the zeros to the right of the decimal point.

12. Delete the global ‘Footer’ row. To do this, click in the left margin next to the ‘Footer’ row to select the entire row. Press the ‘+’ icon in the margin, and select ‘Delete Row’ from the menu.

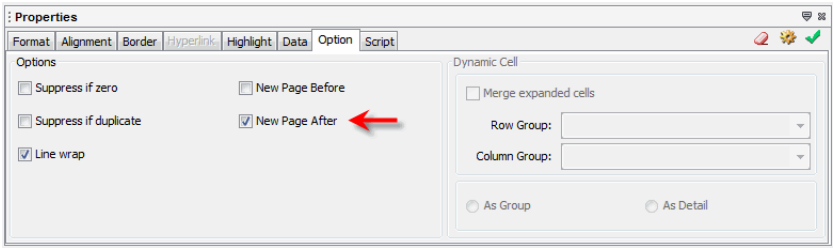


13. Insert a page break following each date group (i.e., quarter). Follow the steps below:

    a. Click in the left margin next to the 'GF1' row to select the entire row.

    b. Select the **Option** tab in the bottom **Properties** panel.

    c. Select 'New Page After' and press the green 'Apply' button.



14. Preview the report again. Page through the report and observe the page breaks following each date group.

The Table formatting is now complete. Continue to *Filtering a Table*.

See Also
Table Formatting, in *Advanced Report Design*.
Splitting and Merging Table Cells, in *Advanced Report Design*.
Selecting a Cell, Row, or Column of a Table, in *Advanced Report Design*.

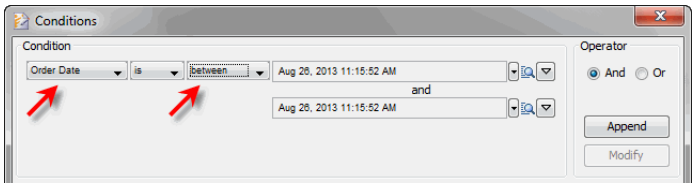
Filtering a Table

The table you created in the previous sections displays data from the full range of dates in the database, making it rather long. It might be helpful to focus the results by allowing the user to specify the range of dates and set of companies they want to see in the table.

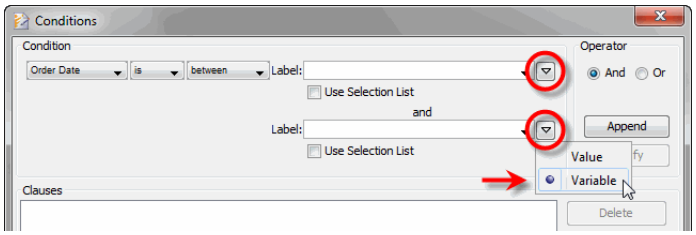
To do this, add a parameterized filter to the Table. Follow the steps below:

1. Right-click on the Table and select ‘Condition’. This opens the ‘Conditions’ dialog box.

2. From the left-most menu, select ‘Order Date’. From the middle menu select ‘between’.



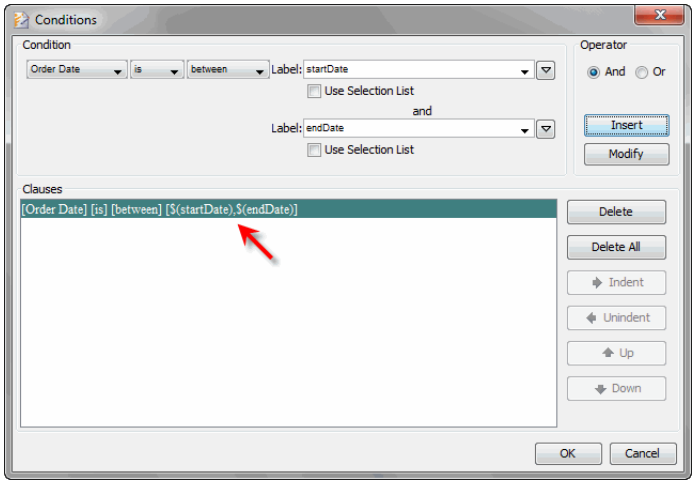
3. For each of the two right pop-up menus, select ‘Variable’.



You don’t have to use “startDate” and “endDate”. You can use any variable names you prefer.

4. In the top ‘Label’ field, enter “startDate”. In the bottom ‘Label’ field, enter “endDate”. These are the names of variables that will hold the value of the starting date and ending date that the user specifies.

5. Press the ‘Append’ button to add the clause to the condition. Note that the ‘startDate’ and ‘endDate’ terms appear in the condition as “`$(startDate)`” and “`$(endDate)`”, which indicates that these are parameters whose values will be supplied at a later time.

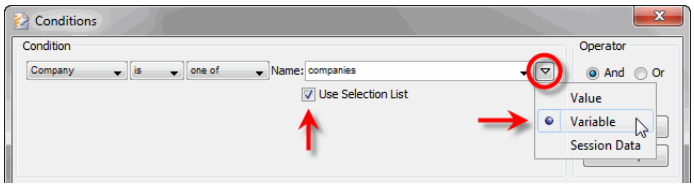


6. Add a second clause to the condition to allow the user to select a set of companies. Follow the steps below:

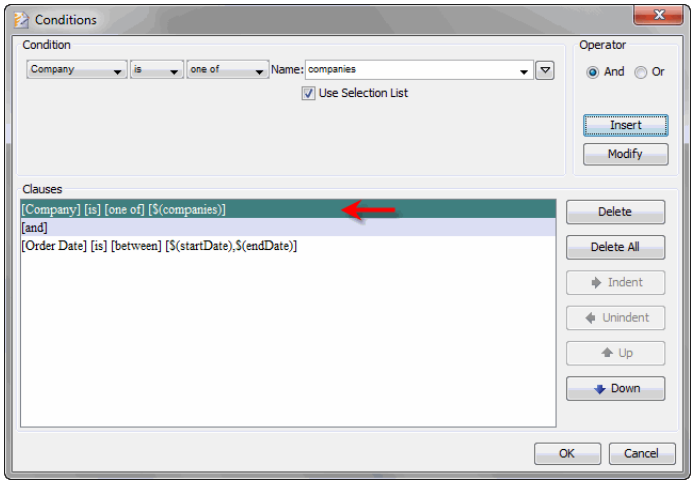
   a. From the left-most menu select ‘Company’. From the middle menu select ‘one of’.

   b. From the right-most pop-up menu, select ‘Variable’. In the ‘Label’ field, enter “companies”.

   c. Check the ‘Use Selection List’ option. (This will present the user with a menu from which the can select a set of companies.)



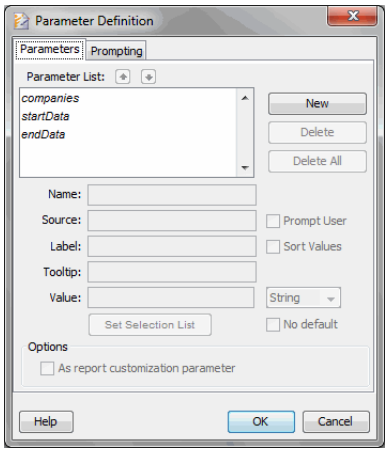
   d. Press ‘Insert’ to add the clause to the condition.



The condition now contains two clauses joined by a conjunction (AND) operator.

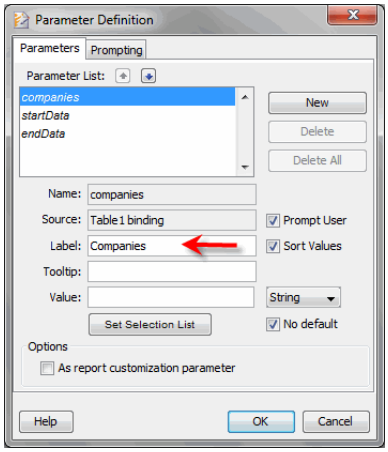
7. Press the ‘OK’ button to finalize the condition and close the dialog box.

8. From the Style Studio ‘Report’ menu, select ‘Parameter...’ This opens the ‘Parameter Definition’ dialog box. This is the central location to manage all report parameters.



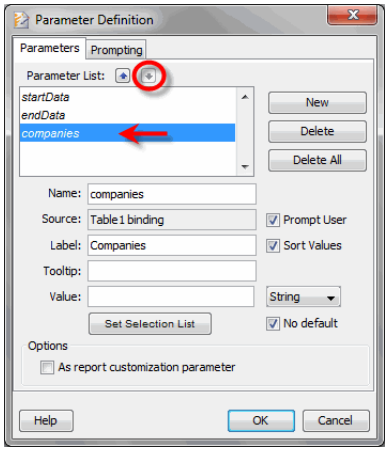
Note that the filter parameters you have defined are listed in the dialog box.

9. In the ‘Parameter Definition’ dialog box, click the ‘companies’ parameter to select it. In the ‘Label’ field, change the label to “Companies:”.



10. Repeat the above step to change the ‘startDate’ label to “Starting Date:” and the ‘endDate’ label to ‘Ending Date:’.

11. Click again on the ‘companies’ parameter, and press the “down arrow” above the list to move the ‘companies’ parameter to the bottom of the list.

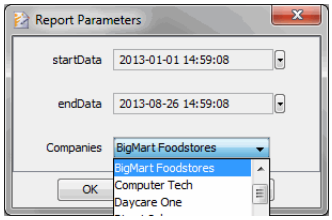


This specifies that the input for ‘companies’ will appear below the inputs for the other two parameters.

12. Press ‘OK’ to close the ‘Parameter Definition’ dialog box.

13. Press the ‘Preview’ button on the toolbar to create a fresh preview. Note that Style Studio now prompts you to enter values for the parameters.

14. Enter a ‘Starting Date’ and ‘Ending Date’ by using the pop-up calendar controls, and select a list of ‘Companies’ to view. (Shift-click to select multiple companies.)



15. Press 'OK' to run the report with these parameter values. Note that the Table is now considerably shorter, displaying data only for the date range and companies that you selected.

Sample Sales Report

Order Date	Company	State	Product	Quantity	Price	Total
	Computer Tech	CA	Sync Me	10	$145	$1,450
	Computer Tech	CA	WebCalendar	11	$248	$2,728
	Computer Tech	CA	Wireless Keyboard	17	$40	$680
	Computer Tech	CA	Fancy Menus	19	$49	$931
	Computer Tech	CA	Animal World	6	$59	$354
	Computer Tech	CA	NetStorage	8	$2,850	$22,800
	Computer Tech	**Company Total:**		71	$3,391	$28,943
	Daycare One	NJ	Math for Me	6	$235	$1,410
	Daycare One	NJ	Info Folder	18	$399	$7,182
	Daycare One	NJ	Combat Hero	14	$29	$406
	Daycare One	NJ	19 inch LCD	9	$310	$2,790
	Daycare One	NJ	Wireless Keyboard	12	$40	$480
	Daycare One	NJ	Mega Icons	14	$550	$7,700
	Daycare One	NJ	Barbie's Fashion	17	$79	$1,343
	Daycare One	NJ	Green Planet	16	$75	$1,200
	Daycare One	NJ	NetStorage	9	$2,850	$25,650
2013 1st	Daycare One	NJ	17 Inch LCD	11	$210	$2,310
	Daycare One	NJ	Fancy Menus	20	$49	$980
	Daycare One	NJ	NetStorage	1	$2,850	$2,850
	Daycare One	NJ	Fast Mail	18	$39	$702
	Daycare One	NJ	Animal World	1	$59	$59
	Daycare One	NJ	Barbie's Fashion	5	$79	$395
	Daycare One	NJ	Info Folder	8	$399	$3,192
	Daycare One	NJ	19 inch LCD	16	$310	$4,960
	Daycare One	**Company Total:**		195	$8,562	$63,609

You have now completed the Table. Continue to *Adding a Chart*.

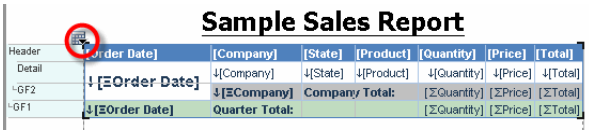
See Also

Filtering Data in a Table, in *Advanced Report Design*.

Adding a Chart

You will now add a chart to the report that you created in previous sections. The chart will display the 'Total' by quarter and state. Follow the steps below:

1. Click once on the Table to select it. This places the insertion point (the black triangle) above the Table.



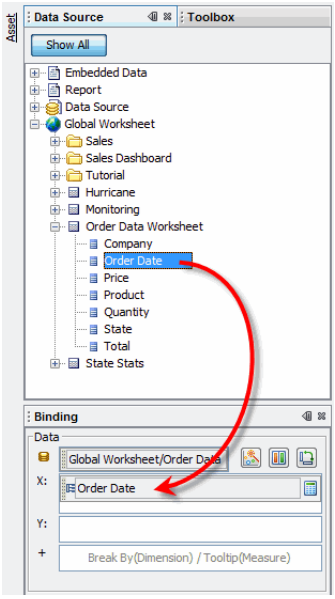
   The insertion point indicates where the next component will be placed.

2. In the **Toolbox** panel, click on the Chart element to add a Chart above the Table. This also opens the Chart Editor in the left panel. (If you do not see the Chart Editor, click on the Chart to select it.)

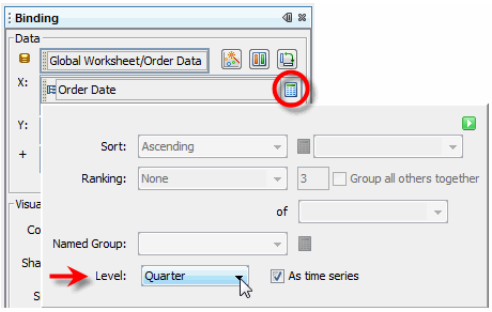
The 'Excel Order Data Worksheet' was created in *Importing a Data File*.

3. In the **Data Source** panel of the Chart Editor, expand the 'Global Worksheet' folder and expand the 'Excel Order Data Worksheet' node.

4. Drag the 'Order Date' field from the **Data Source** panel to the 'X' region of the **Binding** panel.

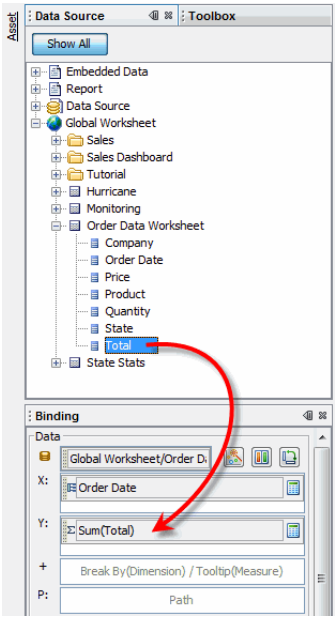


5. Press the ‘Edit Measure’ button next to the ‘Order Date’ field in the Chart Editor, and set the ‘Level’ to ‘Quarter’. Then press the green ‘Apply’ button.

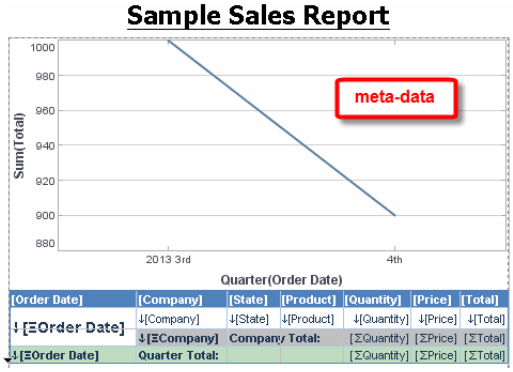


This changes the date grouping on the X-axis to from year to quarter.

6. Drag the ‘Total’ field from the **Data Source** panel to the ‘Y’ region of the **Binding** panel This places the ‘Total’ field onto the chart as a measure.

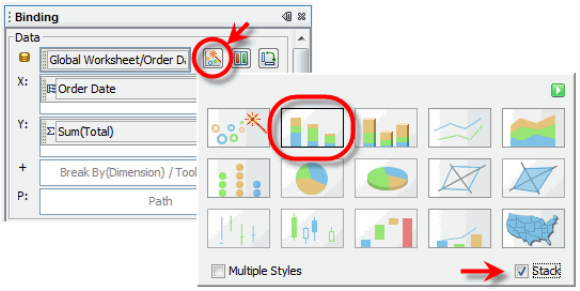


Note that, like the Table, the Chart displays meta-data by default.



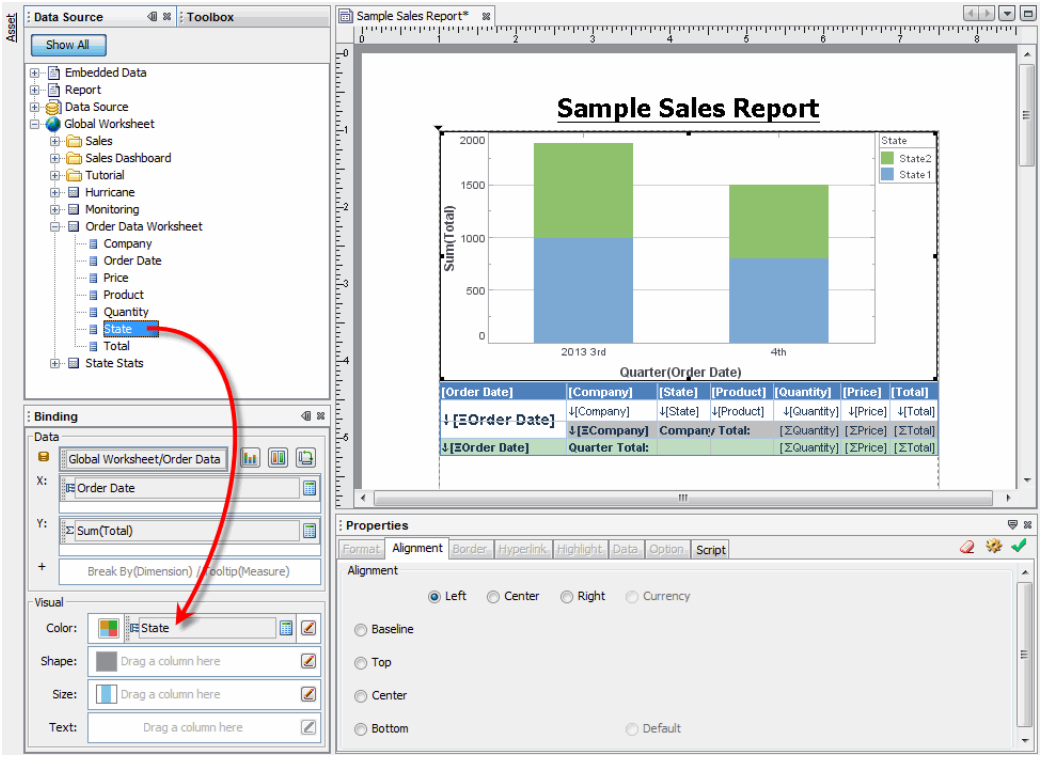
To display actual data, either preview the report or press the ‘Live Edit’ button in the toolbar.

7. Change the Chart style to a bar-type chart. To do this, press the ‘Select Chart Style’ button in the **Binding** panel. Select the ‘Bar’ chart type and enable the ‘Stack’ option at the bottom of the panel.

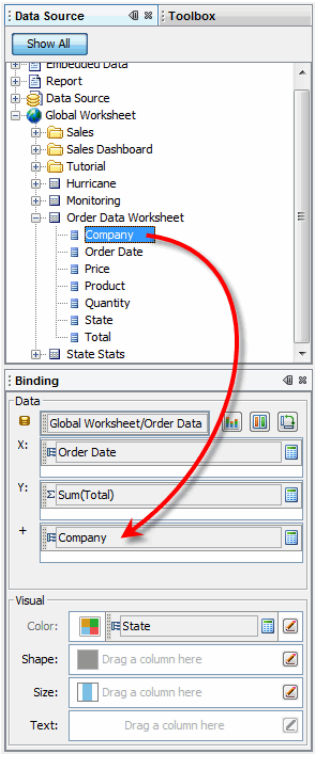


Press the green ‘Apply’ button.

8. Break out the data by state using color. To do this, drag the ‘State’ field from the **Data Source** panel to the ‘Color’ region in the **Visual** panel.

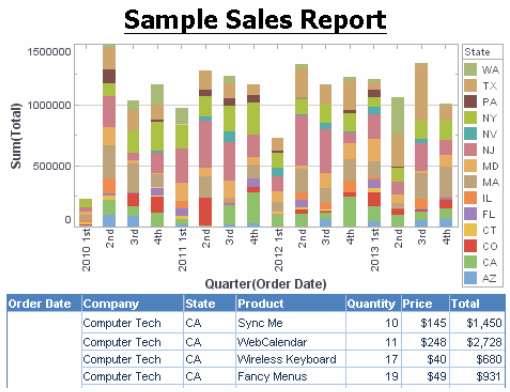


9. Use tooltips to distinguish the data from different companies: To do this, drag the 'Company' field from the **Data Source** panel to the 'Break By' region in the **Visual** panel.



Tooltips will be active when the report is viewed in the User Portal.

10. Preview the report to see the Chart with its data.



Order Date	Company	State	Product	Quantity	Price	Total
	Computer Tech	CA	Sync Me	10	$145	$1,450
	Computer Tech	CA	WebCalendar	11	$248	$2,728
	Computer Tech	CA	Wireless Keyboard	17	$40	$680
	Computer Tech	CA	Fancy Menus	19	$49	$931

When you create a new preview, you will be prompted again to enter the parameters used by the table filter condition you created in *Filtering a Table*. Note that the values that you enter at the parameter prompt do not yet have any effect on the chart.

In the next section you will add a filtering condition to the chart so that the 'Start Date', 'End Date', and 'Companies' parameters will filter the Chart as well as the Table. Continue to *Filtering a Chart*.

See Also
Adding a Chart Element, in *Advanced Report Design*.

Filtering a Chart

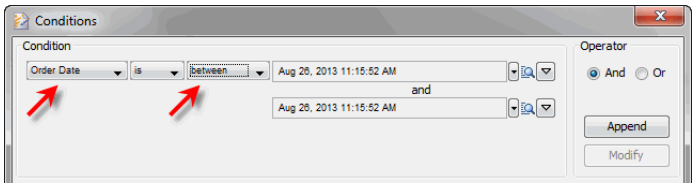
In this section, you will add the same filtering conditions to the Chart that you already added to the Table in *Filtering a Table*. This way, when the user enters values for the parameters, these values will be used to filter both the Table and the Chart.

Note: Since the Table and Chart are using the same condition, you could instead have added a single condition to the Worksheet data block.

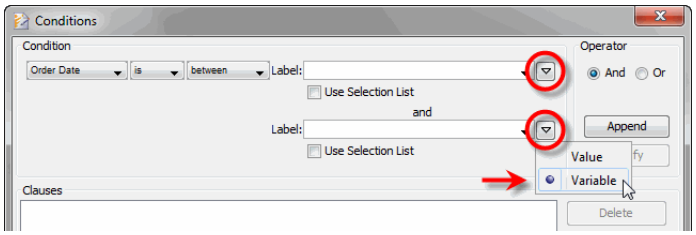
Follow the steps below to add a filtering condition to the Chart:

1. Right-click on the Chart and select ‘Condition’. This opens the ‘Conditions’ dialog box.

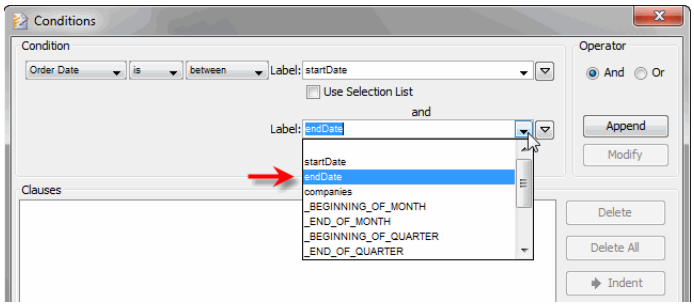
2. From the left-most menu, select ‘Order Date’. From the middle menu select ‘between’.



3. For each of the two right pop-up menus, select ‘Variable’.



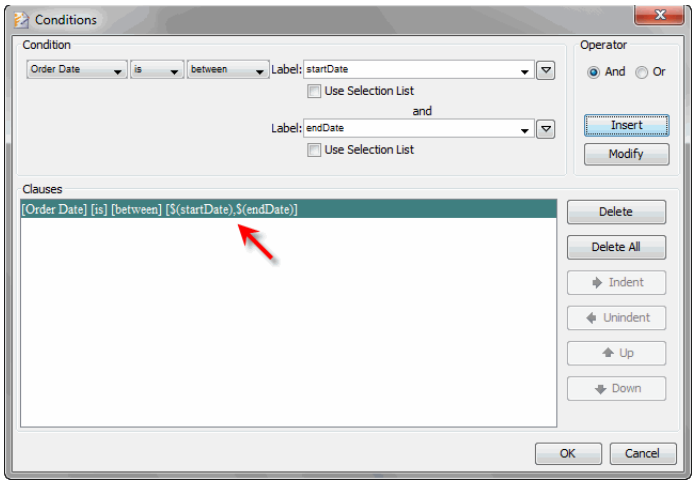
4. Expand the top ‘Label’ menu, and select “startDate”. Expand the bottom ‘Label’ menu, and select “endDate”.



These are the same variable names that you used in the Table condition. **This is important!** You want to prompt the user *only once* for each unique parameter. If you use different variable names

(even different capitalizations) in the Chart condition and the Table condition, the user will be prompted separately for each uniquely-spelled variable.

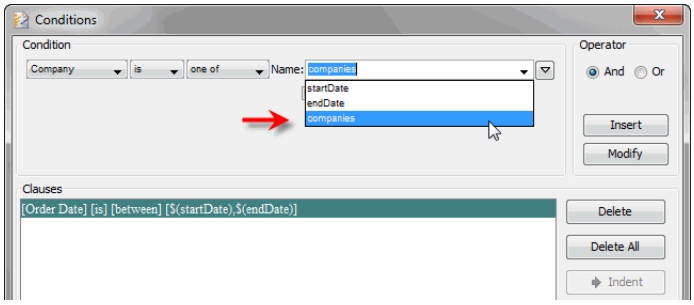
5. Press the 'Append' button to add the clause to the condition.



6. Add a second clause to the condition to allow the user to select a set of companies. Follow the steps below:

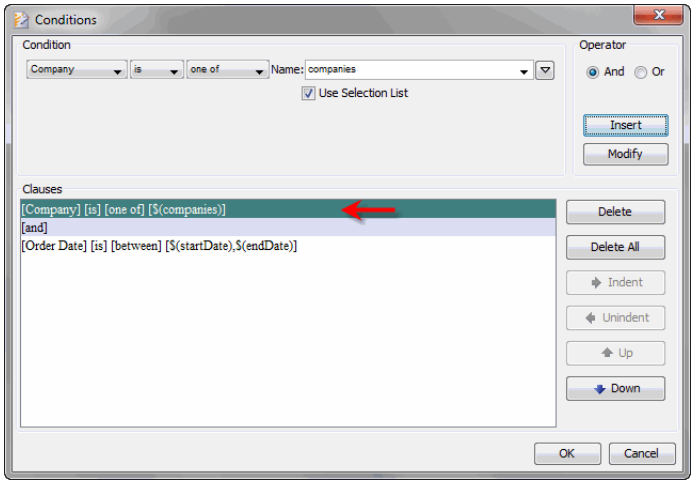
   a. From the left-most menu select 'Company'. From the middle menu select 'one of'.

   b. From the right-most pop-up menu, select 'Variable'. Expand the the 'Label' menu, and select 'companies'.



   c. Check the 'Use Selection List' option. (This will present the user with a menu from which the can select a set of companies.)

   d. Press 'Insert' to add the clause to the condition.

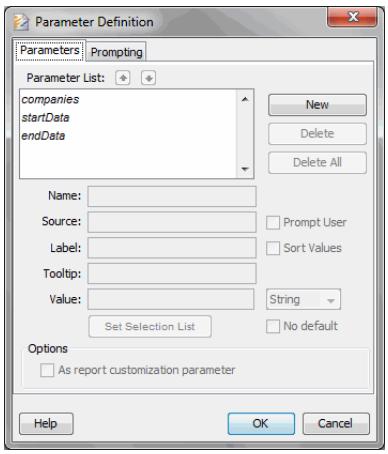


The condition now contains two clauses that are joined by a conjunction (AND) operator.

7. Press the ‘OK’ button to finalize the condition and close the dialog box.

8. From the Style Studio ‘Report’ menu, select ‘Parameter’ This opens the ‘Parameter Definition’ dialog box.

   Observe that only one set of parameters is listed, even though these parameters are used to filter both the Table and the Chart. The user will be prompted for only one set of parameters.



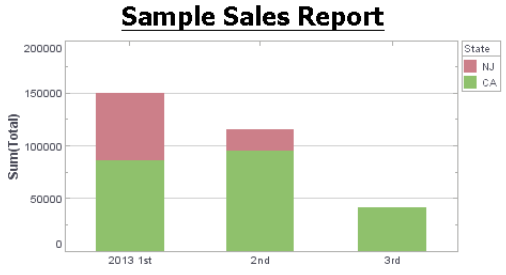
Since you have edited the ‘Label’ property for these parameters previously, press ‘Cancel’ to dismiss the dialog box.

9. Press the ‘Save’ button in the toolbar to save the report.

10. Press the 'Preview' button on the toolbar to create a fresh preview. Style Studio prompts you to enter values for the parameters.

11. Enter a 'Starting Date' and 'Ending Date' by using the pop-up calendar controls, and select a list of 'Companies' to view. (Shift-click to select multiple companies.)

12. Press 'OK' to run the report with these parameter values.



Order Date	Company	State	Product	Quantity	Price	Total
2013 1st	Computer Tech	CA	Sync Me	10	$145	$1,450
	Computer Tech	CA	WebCalendar	11	$248	$2,728
	Computer Tech	CA	Wireless Keyboard	17	$40	$680
	Computer Tech	CA	Fancy Menus	19	$49	$931
	Computer Tech	CA	Animal World	6	$59	$354
	Computer Tech	CA	NetStorage	8	$2,850	$22,800
	Computer Tech	**Company Total:**		71	$3,391	$26,943
	Daycare One	NJ	Math for Me	6	$235	$1,410
	Daycare One	NJ	Info Folder	18	$399	$7,182
	Daycare One	NJ	Combat Hero	14	$29	$406
	Daycare One	NJ	19 inch LCD	9	$310	$2,790
	Daycare One	NJ	Wireless Keyboard	12	$40	$480
	Daycare One	NJ	Mega Icons	14	$550	$7,700
	Daycare One	NJ	Barbie's Fashion	17	$79	$1,343
	Daycare One	NJ	Green Planet	16	$75	$1,200
	Daycare One	NJ	NetStorage	9	$2,850	$25,650
	Daycare One	NJ	17 Inch LCD	11	$210	$2,310
	Daycare One	NJ	Fancy Menus	20	$49	$980

Note that the both the Chart and the Table are filtered to display only data for the date range and companies that you selected.

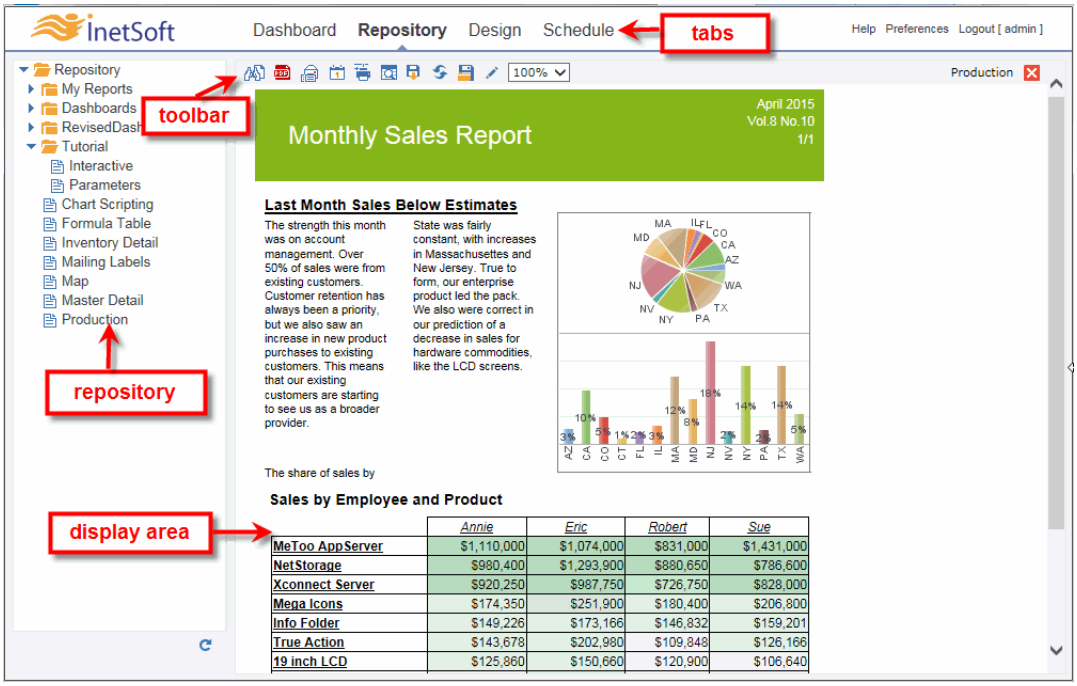
See Also

Filtering Data in a Chart, in *Advanced Report Design*, for additional information.

Using the Portal

The User Portal provides features for accessing and managing existing 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

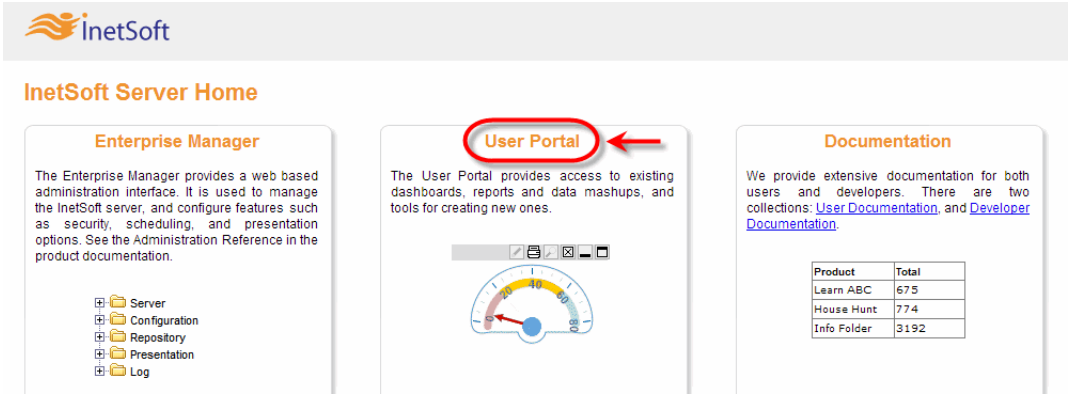
The Style Scope, Style Intelligence, or Style Report server must be running in order to access features of the User Portal and Enterprise Manager.

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 by selecting 'Style Scope Server', 'Style Intelligence Server' or 'Style Report 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.

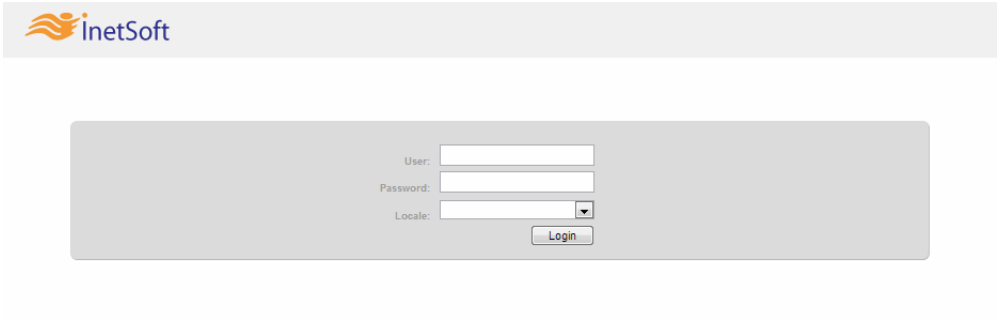


Logging into the Portal

If the server is already 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.

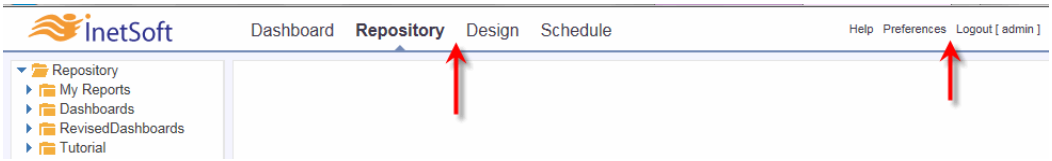


After you log in, select the desired Portal tab for the features you want to access or press the 'Help' link to get information about using the Portal. See *Accessing Portal Functions* for more information about the available features.

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 assets. The options available will depend on what permissions you have been granted by the administrator.

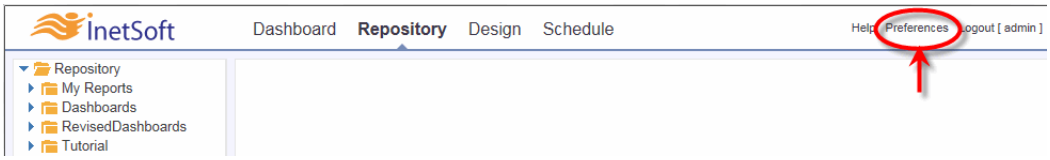


The table below explains the tabs and links that appear at the top of the Portal.

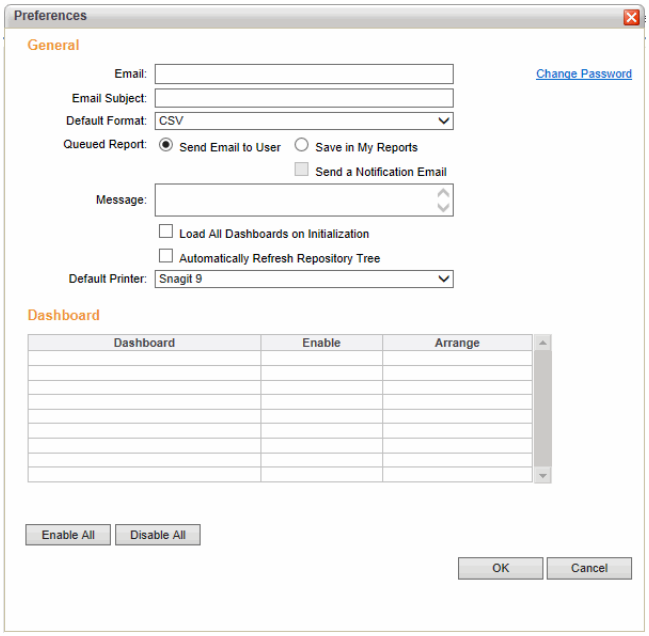
TAB	**FUNCTION**
Dashboard	Access your important dashboards. See *Deploying a Dashboard* for information on configuring dashboards.
Repository	View all available dashboards and reports in the repository. See *Visualizing with Dashboards* and *Interacting with Reports* for more details on using dashboards and reports.
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.

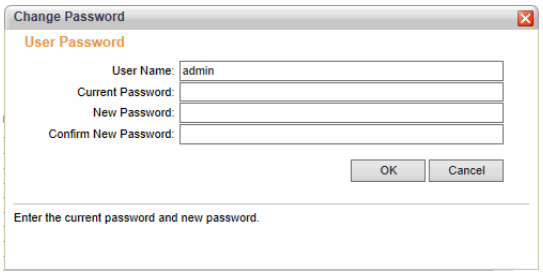


This opens the 'Preferences' dialog box.



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.

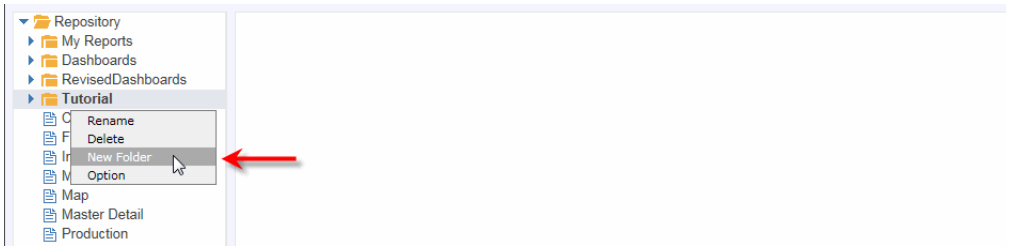
See Also

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

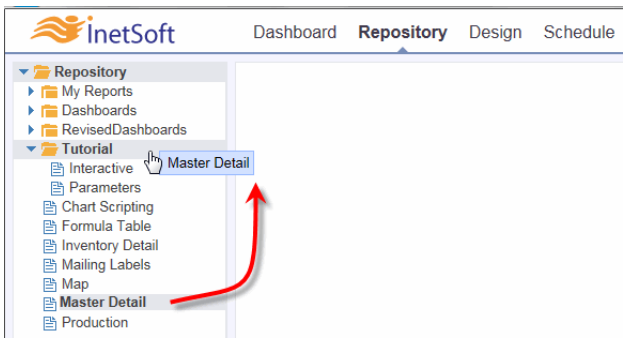
To create a new folder in the Repository, right-click on the folder that you want to contain the new folder, and select ‘New Folder’. This opens the ‘Create Folder’ page.



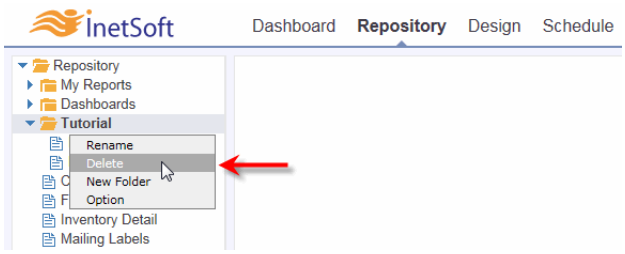
The ‘Alias’ is the name displayed in the Repository.

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.

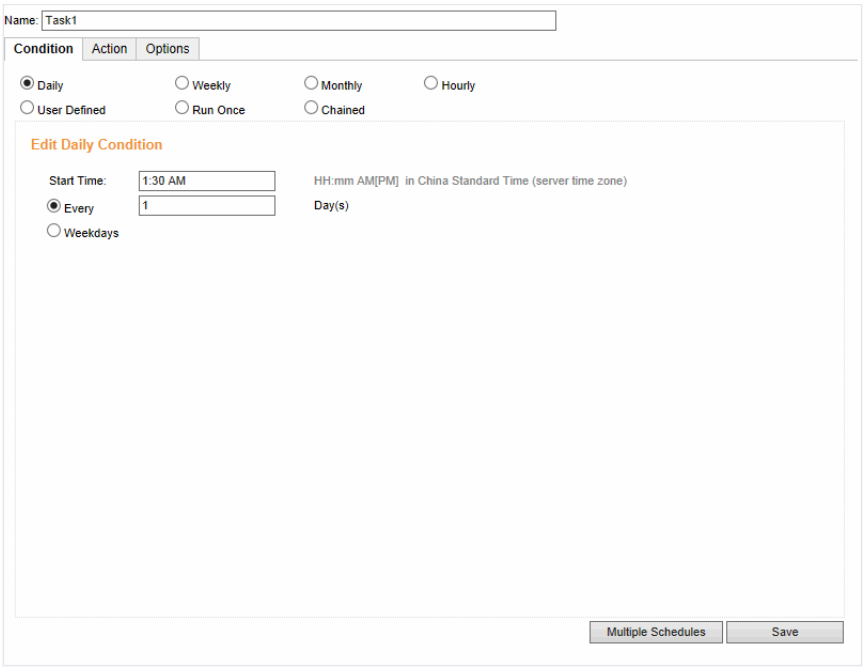
Scheduling Dashboards and Reports

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.

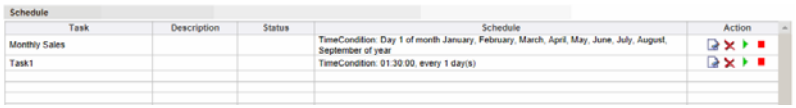


   Using the tabs at the bottom of the Editor, specify when the task executes, what actions it performs, and additional options. See *Setting Schedule Conditions*, *Setting Schedule Actions*, and *Setting Schedule Options* for more information.

3. Press 'Close' to exit the Editor.

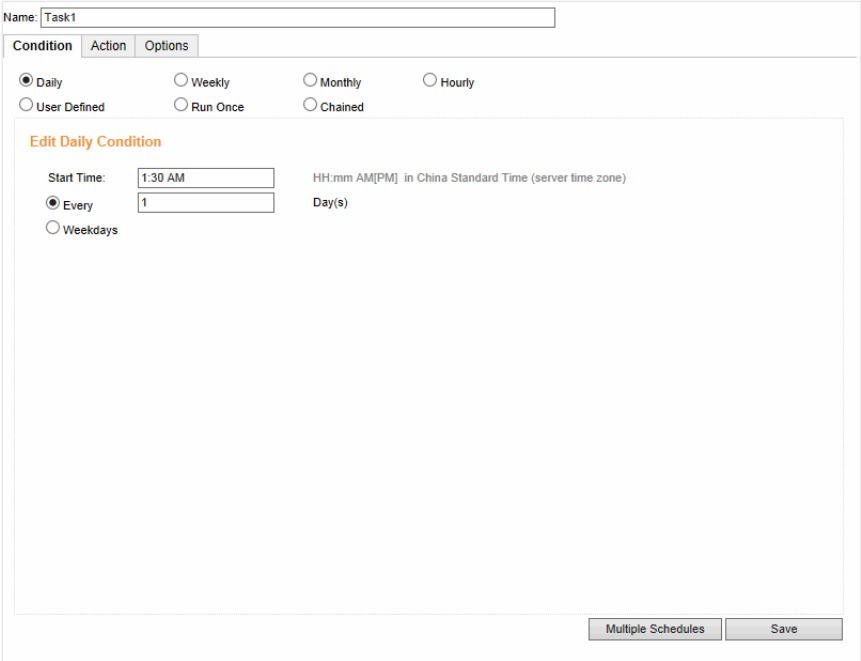


To modify an existing task, press the task's 'Edit' button in the 'Schedule' table. This opens the Editor as described above.



Setting Schedule Conditions

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



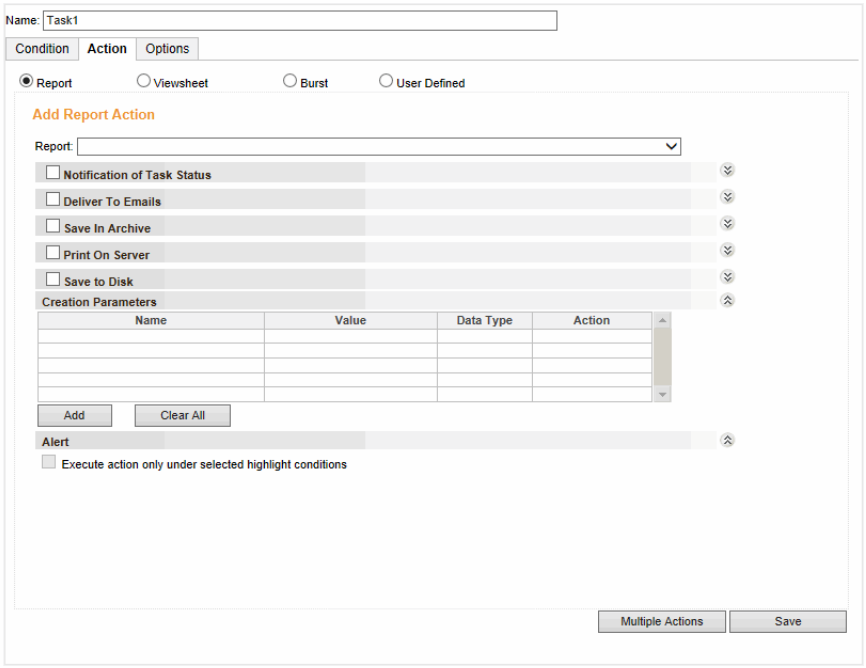
There are six types of conditions: Daily, Weekly, Monthly, Run Once, Chained, and User Defined.

- **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 Server	Save the report on the file system or FTP server in any of the following formats: PDF, HTML, Excel, PowerPoint, RTF, SVG, CSV, Text, XML. To save to file system, 'Path' should specify a valid absolute path, including filename. (The filename extension is added automatically.) To save to FTP server, 'Path' should specify a valid URL in the following format: `ftp://<username>:<password>@<hostname>:<port>/<filename>`, where `password` and `port` are optional. 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.

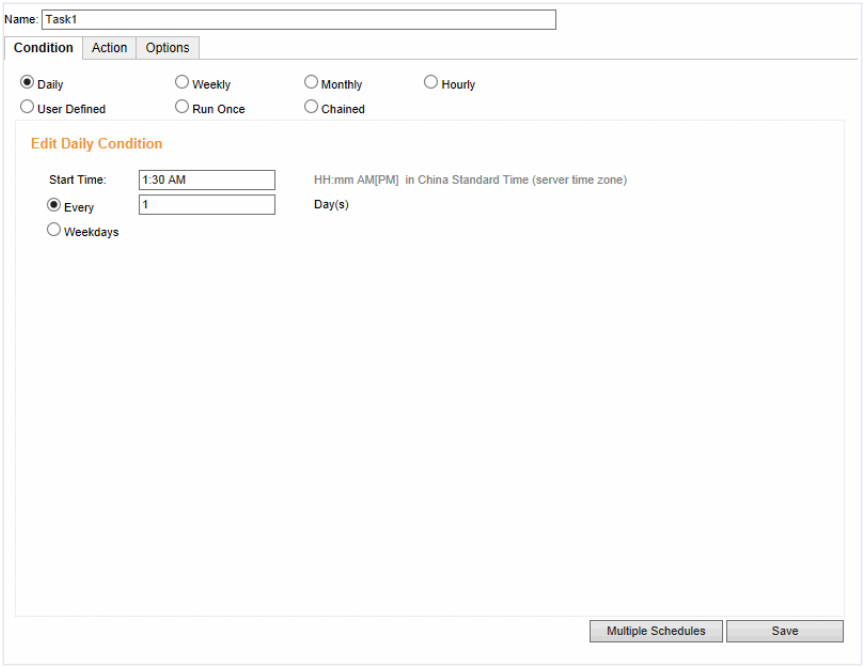
Setting an Alert

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.

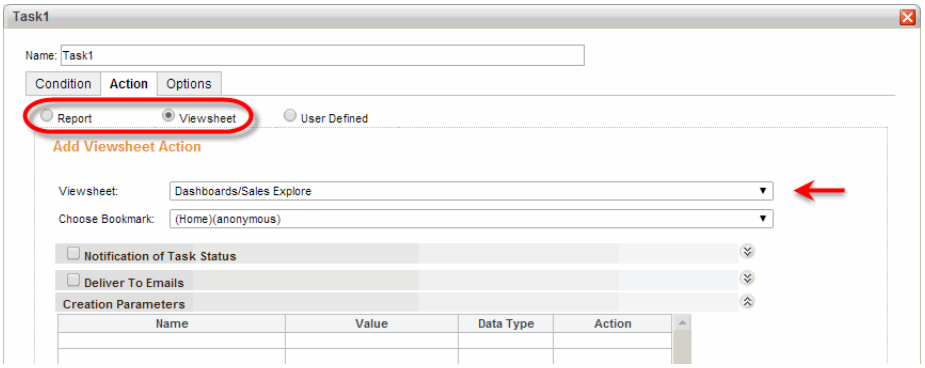
2. Press the ‘New Task’ button below the Schedule table. This creates a new task with the name ‘Task1’, and opens the Editor.



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.

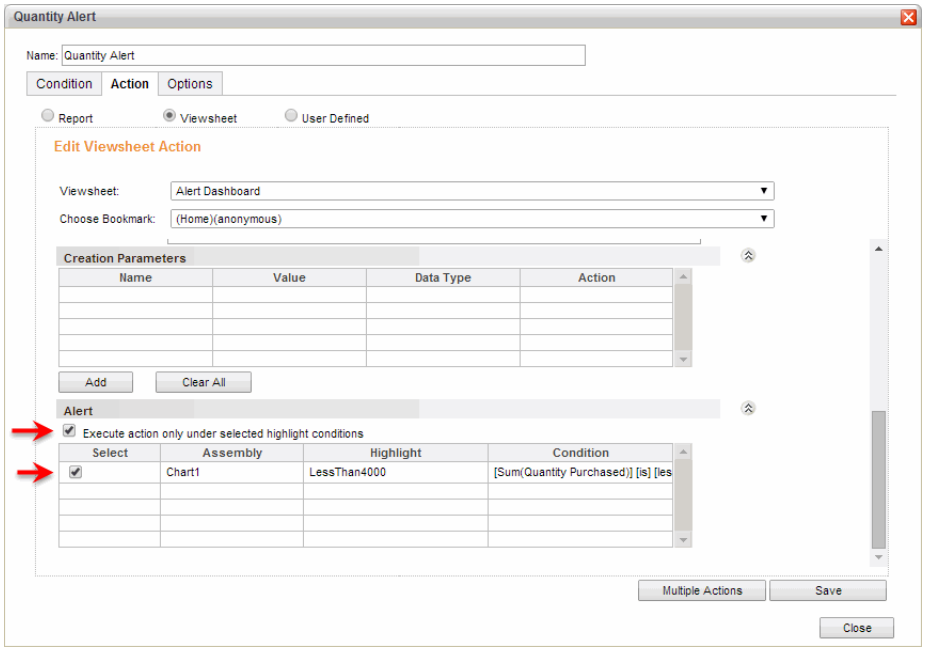


6. Select the actions that you want to perform if the alert is triggered. Generally, you will at least select the ‘Deliver To Emails’ option. (See *Setting Schedule Actions* for more about the available actions.) Enter a ‘Subject’ and ‘Message’ appropriate for the nature of the alert.

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

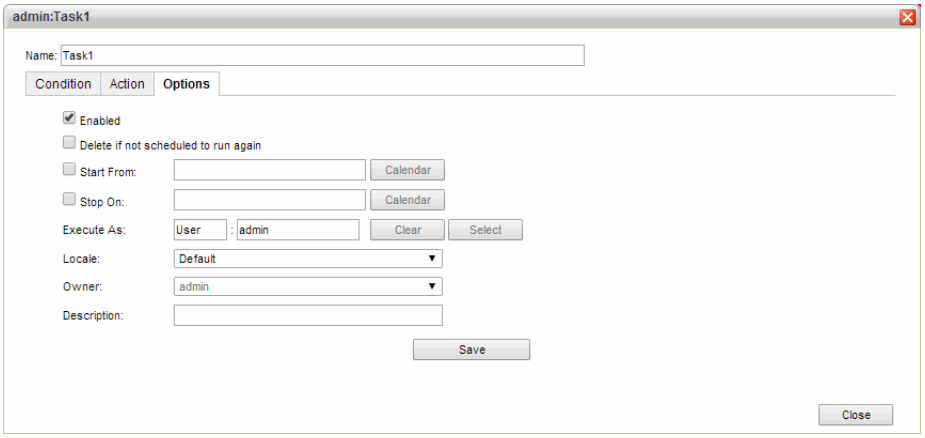
Highlights, in *Advanced Dashboard Design*, for information on creating Viewsheet highlights.

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

Gauge 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.

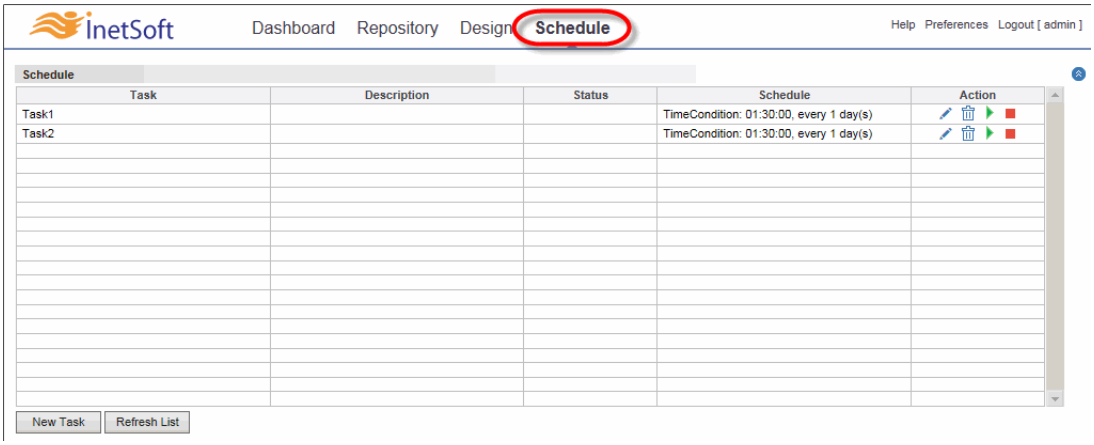


- **Enabled**: Temporarily enables or disables a task.

- **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.

Interacting with Reports

The following sections explain the basics of using reports.

Running a Report

To run a report in the User Portal, follow the steps below:

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

2. In the Portal, select the **Repository** tab.

3. In the left panel of the Portal, expand the 'Repository' tree.

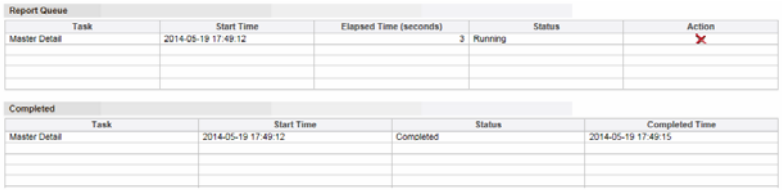
4. Click on the desired report in the Repository Tree. Small reports will load immediately, while large reports may display a loading screen.



Note: If the report is almost finished generating when you press 'Add to Queue', it will not be added to the cue, but will rather display normally in the Portal.

   You can either wait for the report to load or press 'Add to Queue' to queue the report for later viewing. To cancel the report, press 'Cancel'.

5. (Optional) If you choose to queue the report, you can check the report's status under the **Schedule** tab of the User Portal.



When a queued report has finished processing, a copy of the report is either emailed to you or saved into your 'My Reports' folder, based on the preferences that you have specified. See *Configuring Portal Preferences* for more information about queued report settings.

Report Toolbar

The figure below shows a sample toolbar.



Different reports may provide different sets of buttons from the list below.

BUTTON	FUNCTION
	Go to first page.
	Go to previous page.
	Go to next page.
	Go to last page.
Page 1 of 3	Go to specified page (page number in the field).
	Searches within the current report.
	View/Save-As PDF.
	Refresh the current report. (This differs from the browser Reload function, as 'Refresh' regenerates the report on the server.) For a "pre-generated" report, the 'Refresh' option re-executes the pre-generation cycle, saving the updated report.
	Export report.
	Returns to the parent report from a drill-down report.
	Mail report.
	Print report using a local printer via PDF. (May not be available on certain platforms. See below.)
	Server-side printing.
	Default printing. Prints to the default printer configured in the 'Preferences' dialog box.
	Customize report parameters.
	Save the report. This overwrites the existing template. Unsaved changes are denoted by an '*' next to this button.
	Save As. This allows you to copy the template or save an archived copy.
	Pop-up menu. This provides a customizable drop-down list.
	Report Explorer.
	Close Report.

The ‘Print’ button in the browser toolbar only prints the single page that is currently displayed in the Portal.

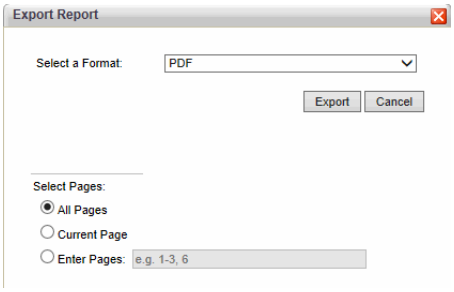
Note that there is both a ‘Server Print’ and a regular ‘Print’ button. The regular ‘Print’ button will load a PDF in the background so that every page can be printed. Most browsers will launch the Adobe Reader plug-in to complete the printing operation. This client-side print feature is only available on platforms that fully support Adobe Reader.

Exporting a Report

To export a report, follow the steps below:



1. Press the ‘Export’ button in the toolbar. This opens the ‘Export Report’ dialog box.



2. In the ‘Select a Format’ menu, choose the desired export format.

3. (Optional) For PDF and HTML formats, select the page range that you want to export.

4. Press ‘Export’.

Reports can be exported to local files in any of the following formats:

Table 1. Export formats

FORMAT	DESCRIPTION
PDF	Portable Document Format.
Excel (match exact layout)	Microsoft Excel worksheet preserving the report layout.
Excel (no pagination)	Same as ‘Excel (match exact layout)’ except that page breaks are eliminated for easier data manipulation.
Excel (best data editing)	Microsoft Excel file with two sheets. The version of the report on the first sheet is the same as the ‘Excel (match exact layout)’ version, while the version of the report on the second sheet attempts to preserve the report layout without using any cell spanning.
RTF (editable document)	Rich Text Format, readable and editable by most word processors.
RTF (match exact layout)	Same as RTF except presentation is more similar to the actual report. Block layout may make data manipulation more difficult in some cases.
HTML (match exact layout)	Standard HTML, preserving the report layout.

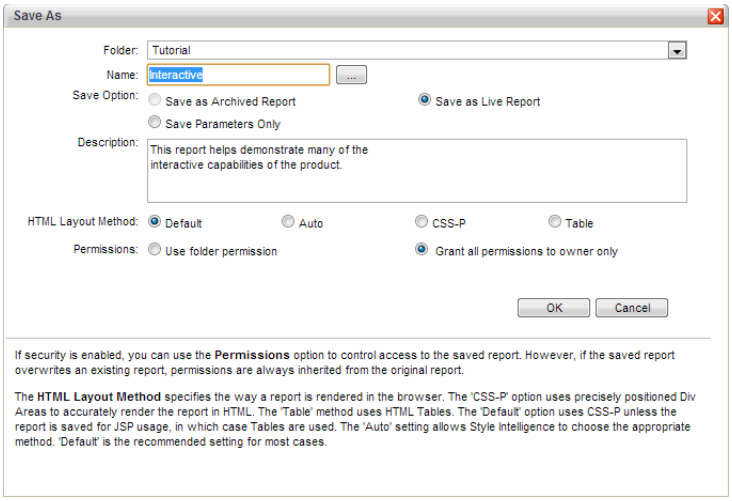
HTML (no pagination)	Same as HTML (match exact layout) except page breaks are eliminated for easier data manipulation.
HTML (bundled as zip)	Creates a zip file containing the HTML report as well as all associated image files.
HTML (no pagination, bundled as zip)	Same as HTML (bundled as zip) except that the page breaks are eliminated to produce a single page HTML report for easier data manipulation.
CSV	Delimited text file, readable by most spreadsheets. Only table data is exported.
SVG	Scalable Vector Graphics, an XML-based language for web graphics.
Text	Text file representing the report as closely as possible.
PowerPoint	Microsoft PowerPoint presentation.
XML	Well-formed XML file, intended for data analysis, including data from Text elements, Tables, Charts, and Sections. Crosstab and Chart data are represented in flattened form. Section data appears without group or band delimitation.

Saving a Report

To save a report under a different name or location in the repository, or to create an archived version of the report, follow the steps below:



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




```
You must possess “write” permission for a folder in order to save
the report in that location.
```

2. In the ‘Folder’ menu, select the repository folder into which you want to save the report.

3. In the ‘Name’ field, enter the desired name for the saved report.

4. Select a desired ‘Save Option’. The following three choices are available:

   a. Select ‘Save as Archived Report’ to save the report *together with* its current data. The archived report will subsequently always display this same dataset, and will never be updated with any new data. An archived report represents a “snapshot” of the data at a particular moment in time.

   b. Select ‘Save as Live Report’ to save the report *without* archiving its data. A *live report* is a report that draws its data from the data source at runtime and always displays “fresh” data. (This is the default way that reports are saved.)

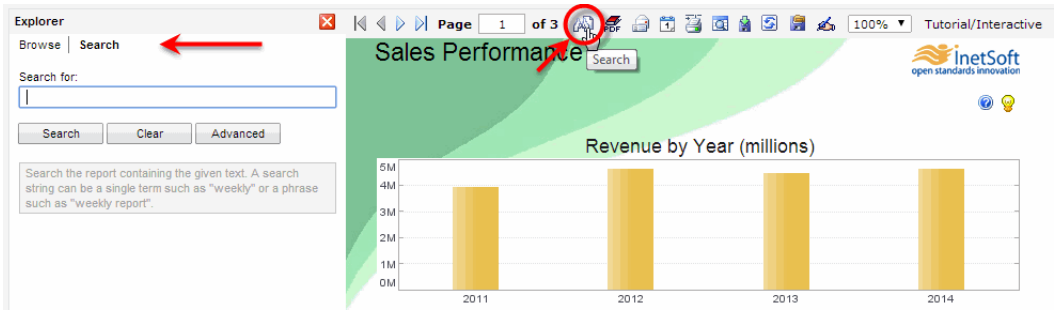
   c. Select the ‘Save Parameters Only’ option to save the values of all input parameters that you have previously entered. The

report will apply these same parameters (to fresh data) the next time it runs.

5. Press ‘OK’ to close the dialog box and save the report.

Searching a Report

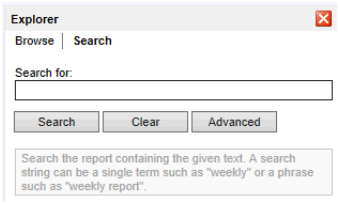
To search within a report, press the ‘Search’ button on the report toolbar. This opens the **Search** tab in the left-side ‘Explorer’ panel.



For example, follow the steps below to search the sample ‘Interactive’ report.

1. In the User Portal, open the ‘Tutorial’ > ‘Interactive’ report.

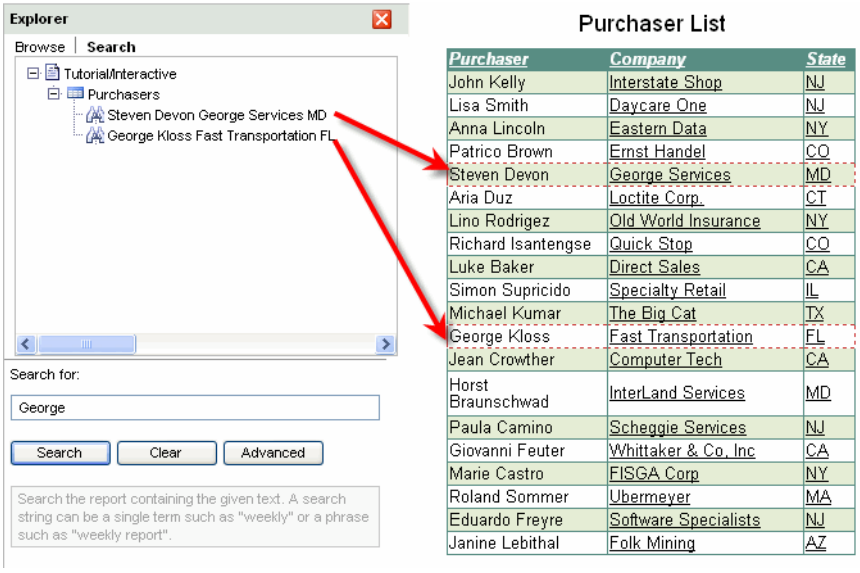
2. Press the ‘Search’ button in the report toolbar to open the **Search** tab of the ‘Explorer’ panel.



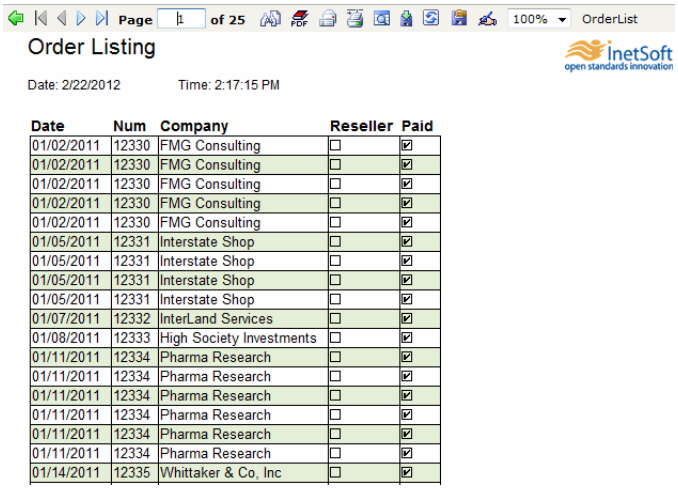
3. Enter “George” in the ‘Search for’ box, and press ‘Search’ to find all occurrences in the report.

   The **Search** panel lists the locations of the searched term, and all results are outlined in a red dotted line in the report.

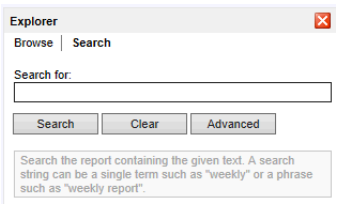
4. Click an instance in the search list to jump to the appropriate page in the report.

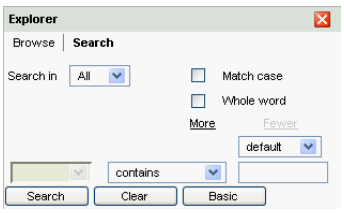


5. Now do an ‘Advanced Search’: First, click on the bar for 2011 in the ‘Revenue by Year’ chart. This drills down to the orders for 2011.

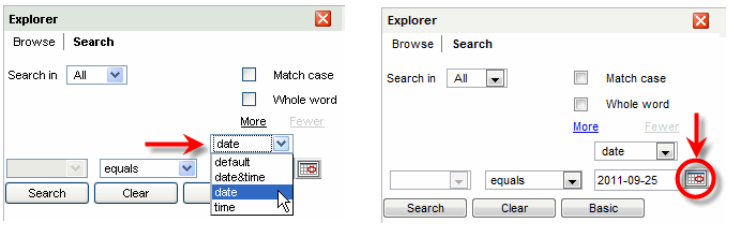


6. Press the ‘Search’ button to open the **Search** panel, then press the ‘Advanced’ button to open the advanced search options.

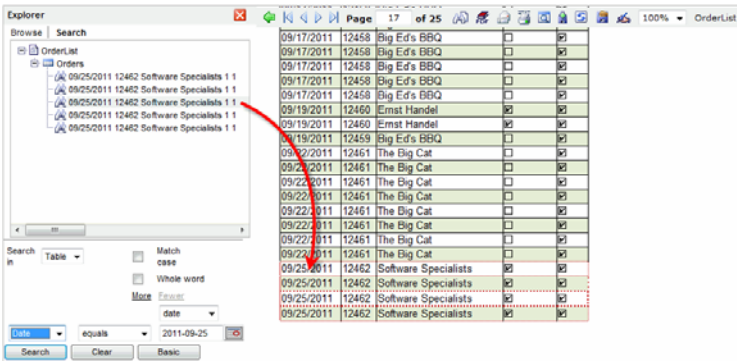




7. Search for a specific date. Select the ‘Date’ option in the ‘Type’ menu. Press the ‘Calendar’ button below the menu, and select the following date: September 25, 2011.

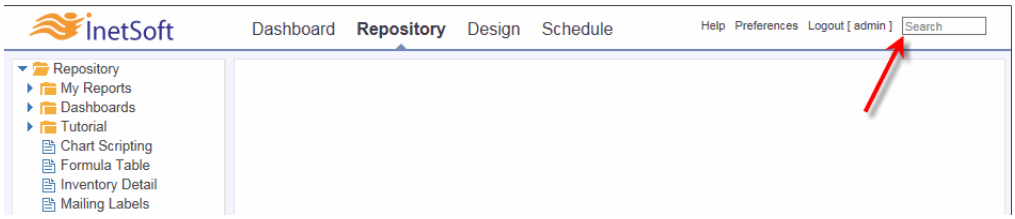


8. Press the ‘Search’ button to list the results Click an entry in the list to jump to the corresponding location in the report.



Searching Archived Reports

To search through archived reports, enter a term in ‘Search’ box at the top of the Portal, and press ‘Enter’ on the keyboard.



This displays a list of search results in the left panel. Click on the desired archived report to open it.

Scheduling a Report

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

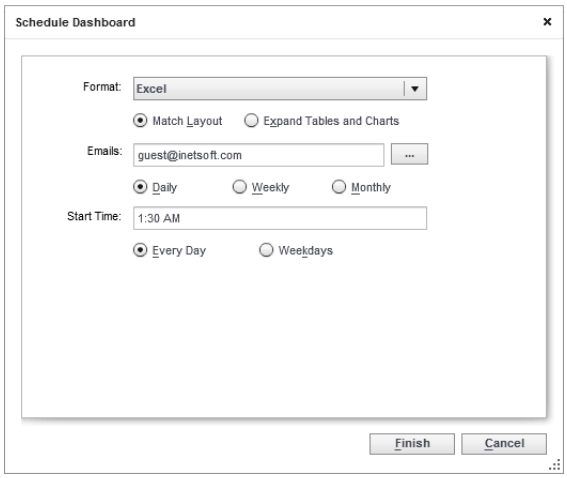


1. Press the ‘Schedule’ button on the toolbar. This opens the ‘Schedule Report’ dialog box.

2. In the ‘Format’ menu, select the format in which the report should be delivered.

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

4. Select a ‘Daily’, ‘Weekly’, or ‘Monthly’ option and enter the desired date specifications for when the report should be generated.



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

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

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.

Sorting a Table

You can sort a table with the Report Explorer tool. The following example illustrates this approach:

1. In the User Portal, open the ‘Tutorial’ > ‘Interactive’ report.

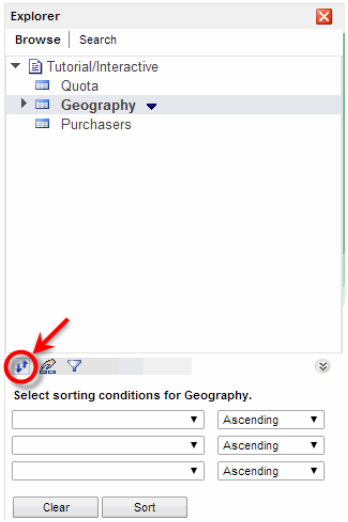


2. Press the ‘Report Explorer’ button in the report toolbar. This opens the ‘Explorer’ panel.

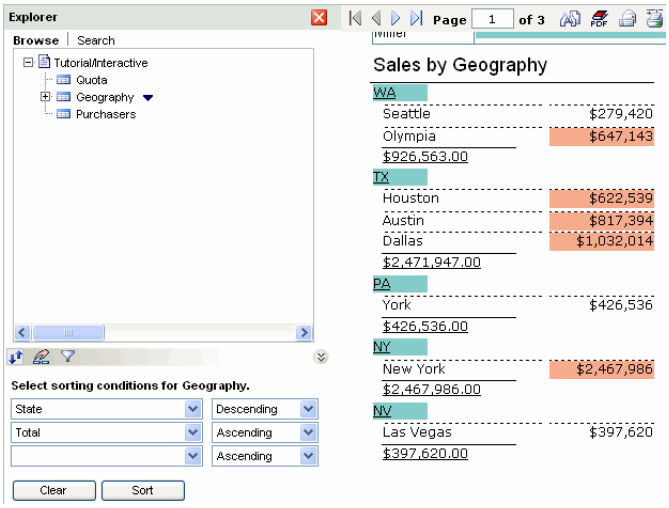
   Report Explorer is a tool that allows you to modify a report for purposes of viewing, emailing, archiving, or exporting. Report Explorer does not *change* the original report, but only manipulates the way the information in the report is displayed. Note that complex formula tables cannot be manipulated within Report Explorer, and Section elements do not permit column rearrangement and/or hiding.



3. Select the ‘Geography’ element in the tree, and press the ‘Show Sorting Options’ button below.

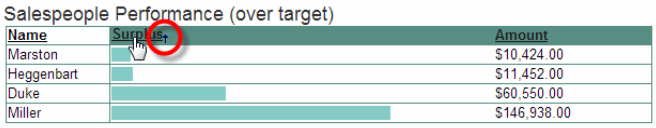


4. Select the fields you want to sort from the provided menus, and indicate whether each field should be sorted in ‘Ascending’ or ‘Descending’ order.



5. Press the ‘Sort’ button to implement the specified sort order.

Note that if the report designer has enabled the ‘Sort On Header’ feature for the report, you can also sort a table by clicking a column header. Click once to sort the data in ascending order. Click again on the same column to sort the data in descending order.



A small up-arrow or down-arrow indicates the current sort order.

Filtering a Report Element

You can filter report elements using the Report Explorer. The following example illustrates this approach.

1. In the User Portal, open the 'Tutorial' > 'Interactive' report.

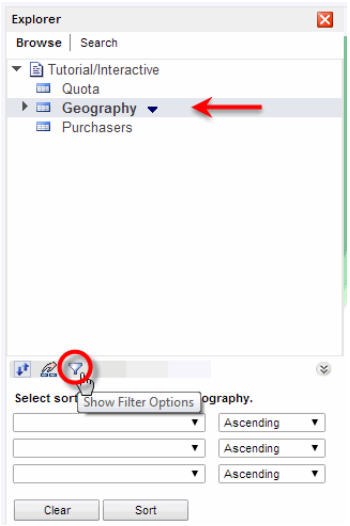


2. Press the 'Report Explorer' button in the report toolbar. This opens the 'Explorer' panel.

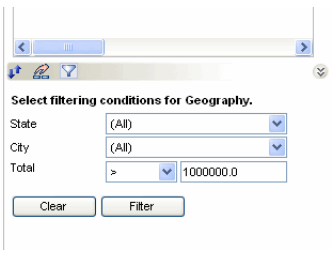
   Report Explorer is a tool that allows you to modify a report for purposes of viewing, emailing, archiving, or exporting. Report Explorer does not *change* the original report, but only manipulates the way the information in the report is displayed. Note that complex formula tables cannot be manipulated within Report Explorer, and Section elements do not permit column rearrangement and/or hiding.

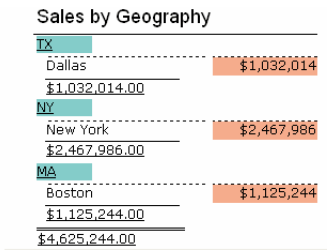


3. Select the 'Geography' element in the tree, and press the 'Show Filter Options' button below.



4. In the menu next to 'Total', select '>' and type "1000000" in the text box. Press 'Filter'.





The ‘Geography’ table is filtered to display only sales greater than $1,000,000.

Hiding Table Columns

You can hide columns of a table using the Report Explorer. The example below illustrates this approach.

1. In the User Portal, open the 'Tutorial' > 'Interactive' report.

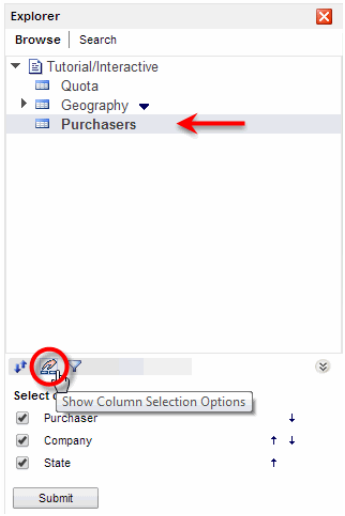


2. Press the 'Report Explorer' button in the report toolbar. This opens the 'Explorer' panel.

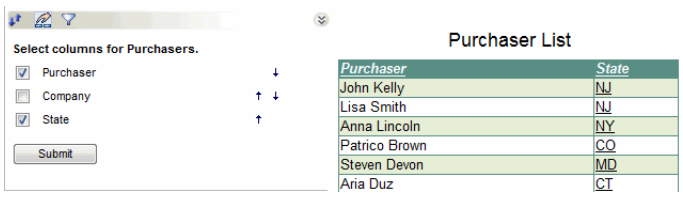
   Report Explorer is a tool that allows you to modify a report for purposes of viewing, emailing, archiving, or exporting. Report Explorer does not *change* the original report, but only manipulates the way the information in the report is displayed. Note that complex formula tables cannot be manipulated within Report Explorer, and Section elements do not permit column rearrangement and/or hiding.



3. Select the 'Purchasers' element in the 'Explorer' panel. Press the 'Show Column Selection Options' button below.



4. Uncheck 'Company'. Press 'Submit' to generate the report.



Note that the 'Company' column has been removed from the table.

Hiding Table Rows

You can hide detail rows of a grouped table using the Report Explorer. The example below illustrates this approach.

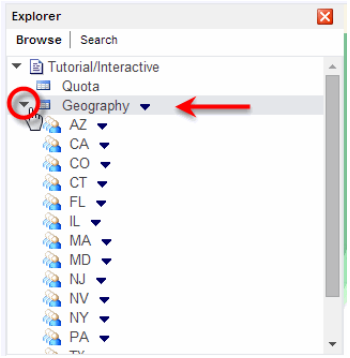
1. In the User Portal, open the 'Tutorial' > 'Interactive' report.



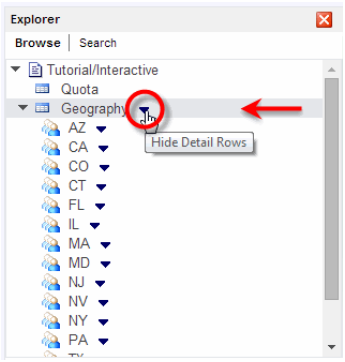
2. Press the 'Report Explorer' button in the report toolbar. This opens the 'Explorer' panel.

   Report Explorer is a tool that allows you to modify a report for purposes of viewing, emailing, archiving, or exporting. Report Explorer does not *change* the original report, but only manipulates the way the information in the report is displayed. Note that complex formula tables cannot be manipulated within Report Explorer, and Section elements do not permit column rearrangement and/or hiding.

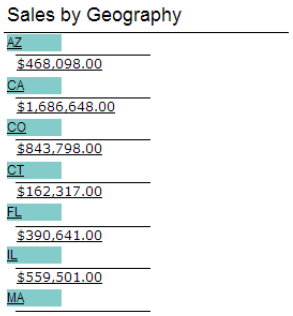
3. Expand the 'Geography' node in the 'Explorer' tree.



4. Press the 'Hide Detail Rows' button next to 'Geography'.



This hides the detail rows in the 'Sales by Geography' table.


Sales by Geography
AZ
$468,098.00
CA
$1,686,648.00
CO
$843,798.00
CT
$162,317.00
FL
$390,641.00
IL
$559,501.00
MA

Managing the Server

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

The following sections discuss several key aspects of managing the server, including configuring security, scheduling tasks, and deploying assets.

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, Style Scope, or Style Report 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.

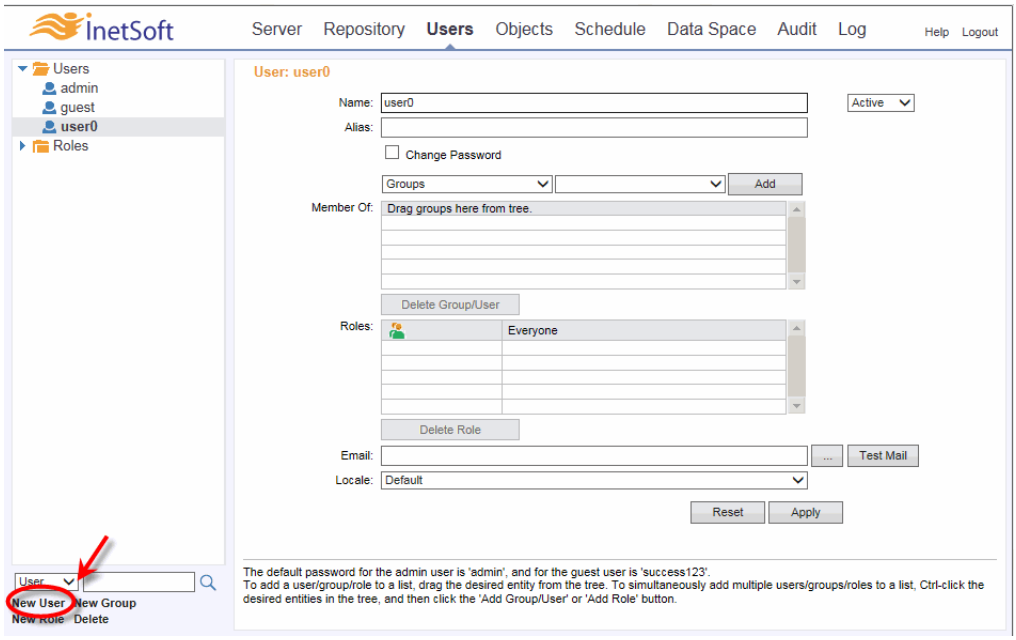
- *Creating a User*
- *Creating a Group*
- *Creating a Role*

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.

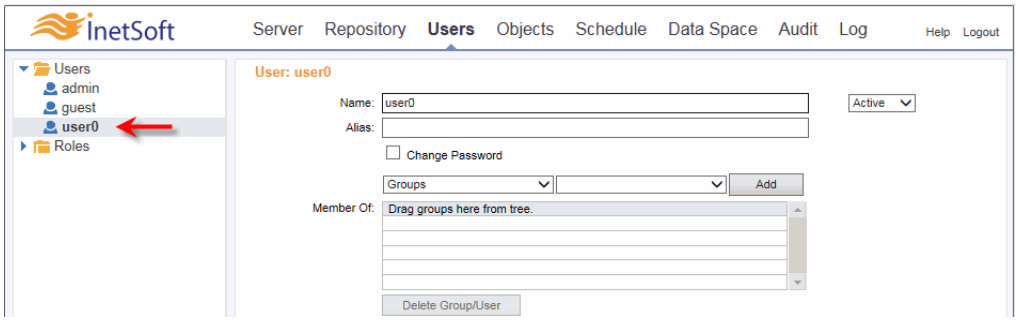
5. Drag groups from the left-side tree into the ‘Member Of’ box to assign the user to the designated groups. (Alternatively, select ‘Groups’ from the menu above the ‘Member Of’ box, choose the desired group, and press ‘Add’.)

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.)

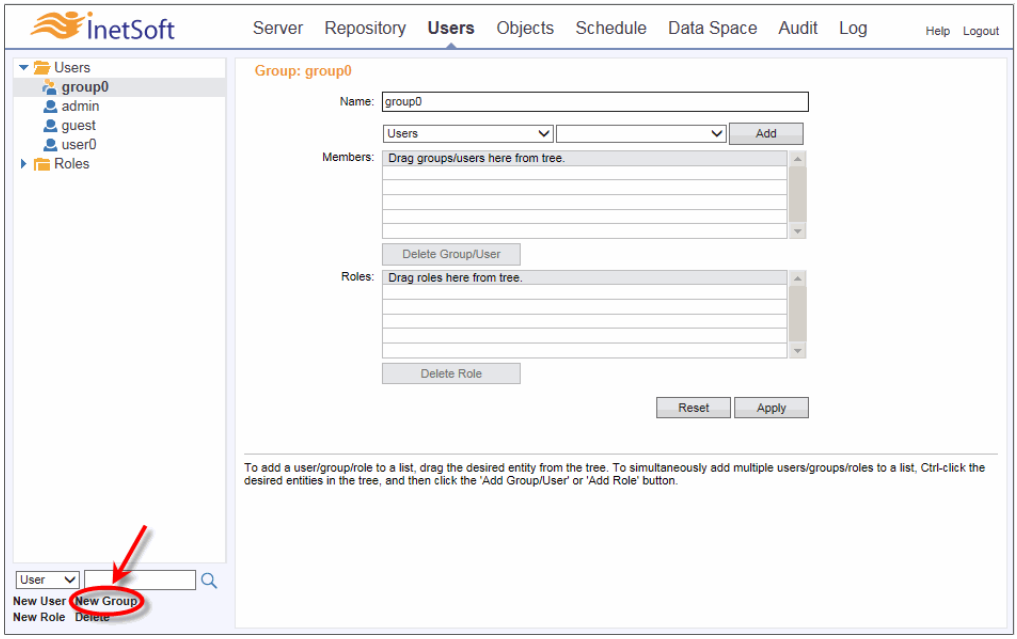


Creating a Group

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.).



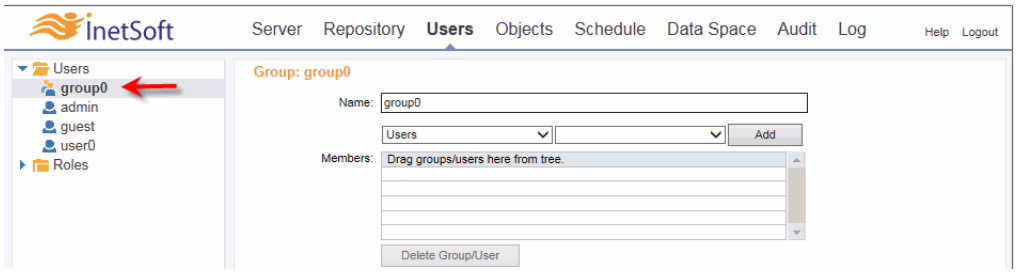
2. Type in the 'Name' field to change the name of the group.

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’.



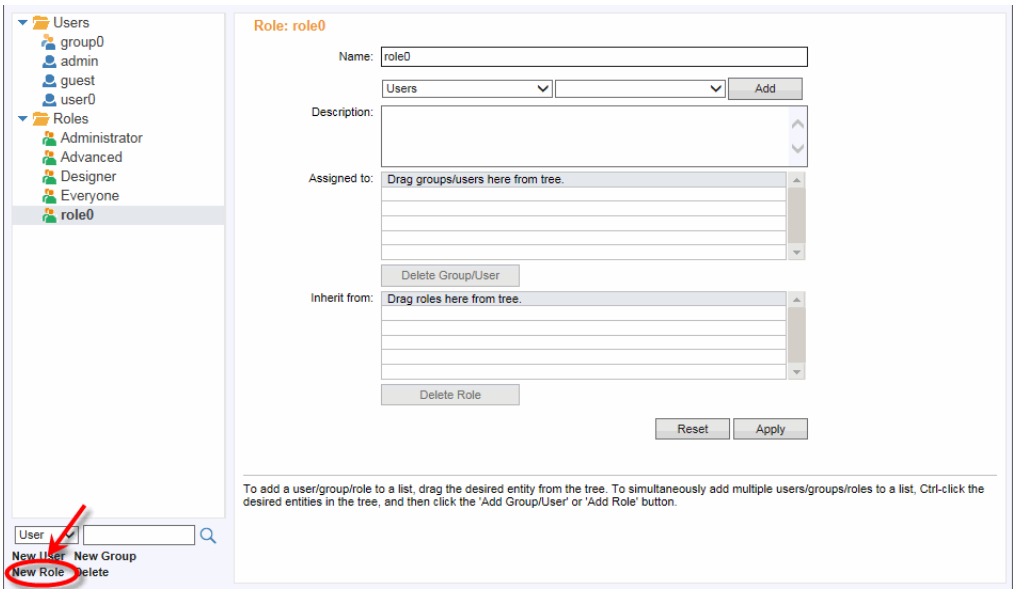
Creating a Role

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.).



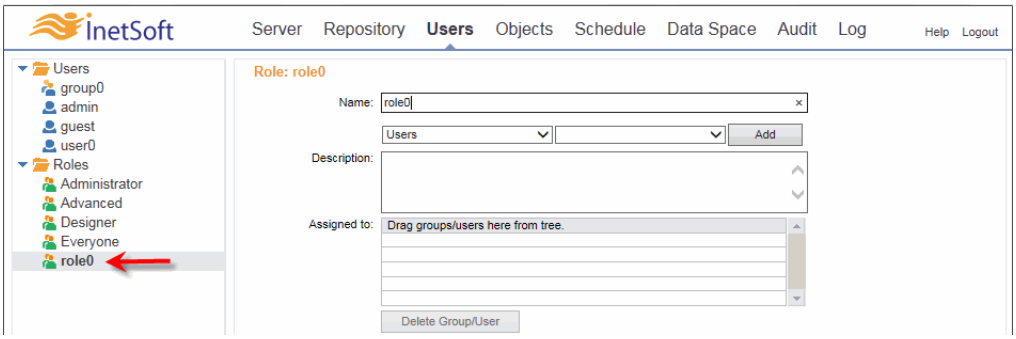
2. Type in the ‘Name’ field to change the name of the role.

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 **Repository** 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*.

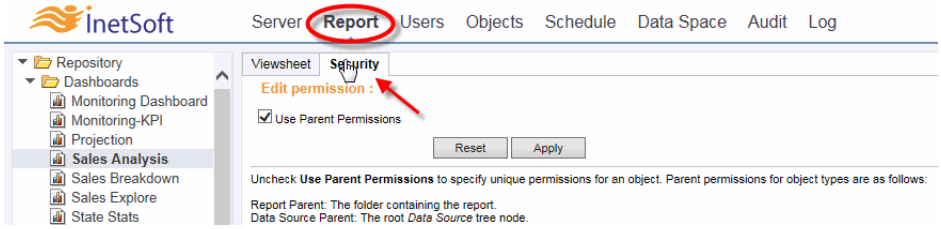
Setting Repository Permissions

Repository folders are *logical* groupings, and do not represent file system directories.

To set permissions for dashboards, reports, and repository folders, select the **Repository** 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.

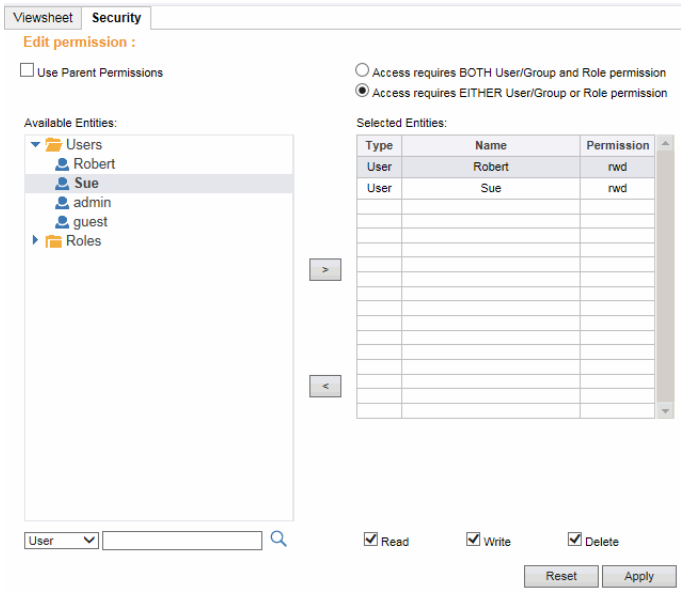
To set the permissions for an individual dashboard, report, or folder, follow the steps below:

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.

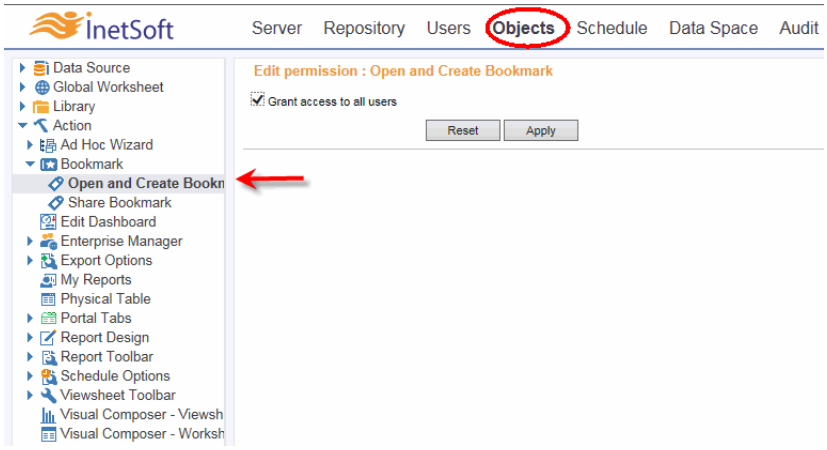
See Also

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:

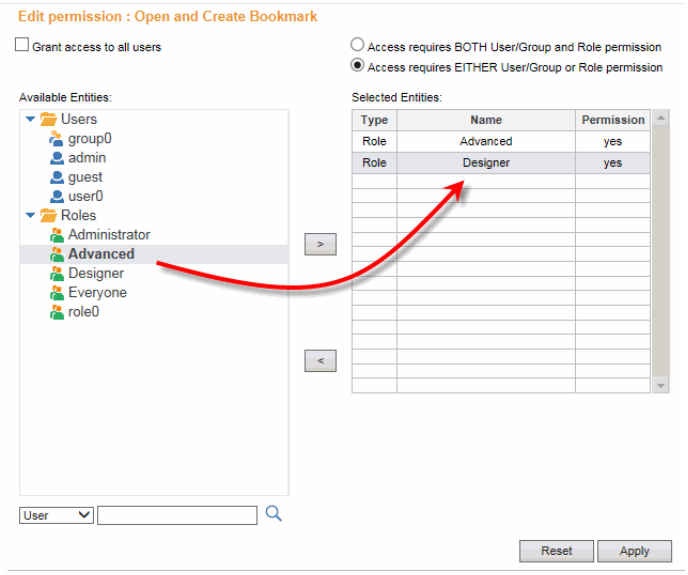
1. Select the **Objects** tab in Enterprise Manager, and select the 'Bookmark' item in the tree.



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

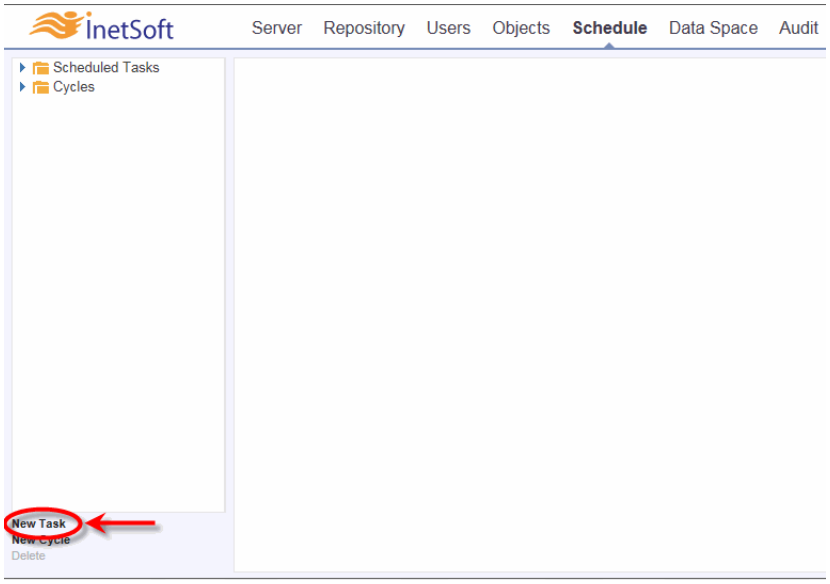
Style Intelligence, Style Scope, and Style Report include a Scheduler application that allows you to automatically generate and email dashboards, and to define cycles for materialized views. The following sections explain how to use the Scheduler.

Creating a Scheduled Task

Users can also create scheduled tasks in the Portal. See *Scheduling Dashboards and Reports*.

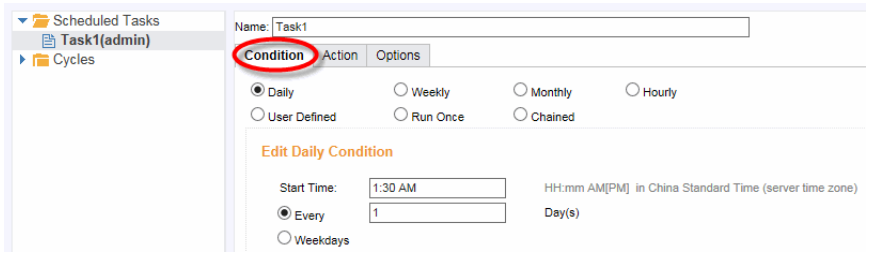
To create a scheduled task, follow the steps below:

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.).

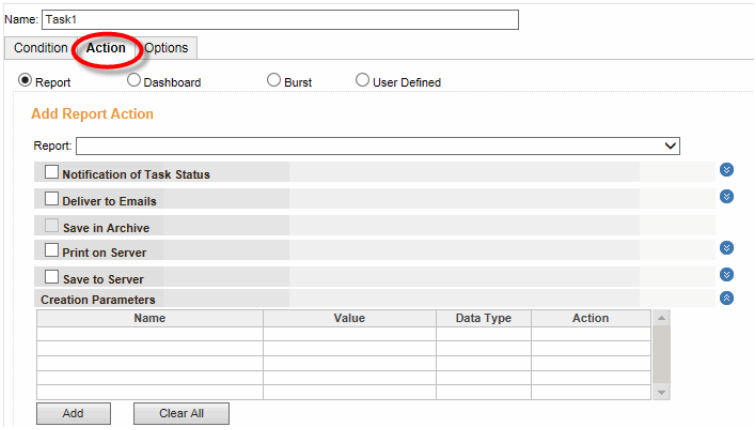


2. Enter a name for the scheduled task in the 'Name' field.

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.

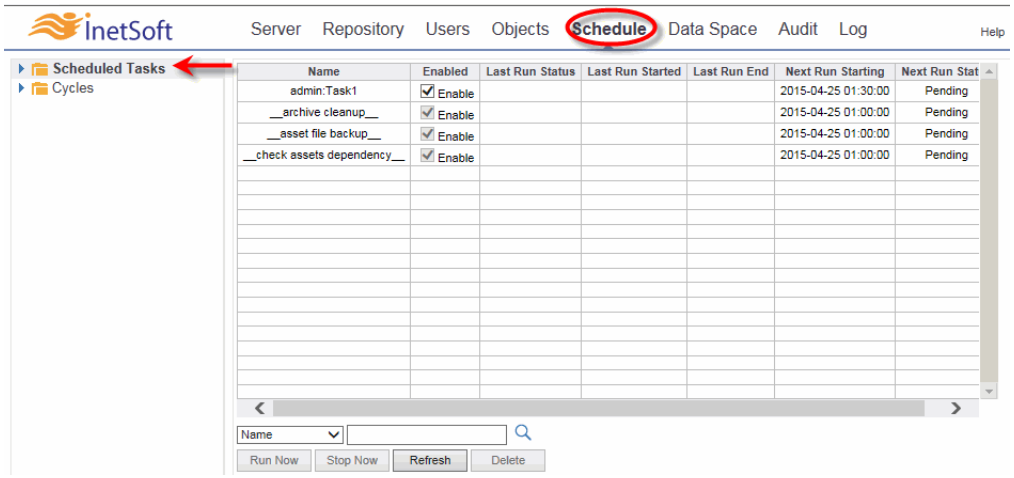


5. Press the 'Save' button at the bottom to save the task.

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.

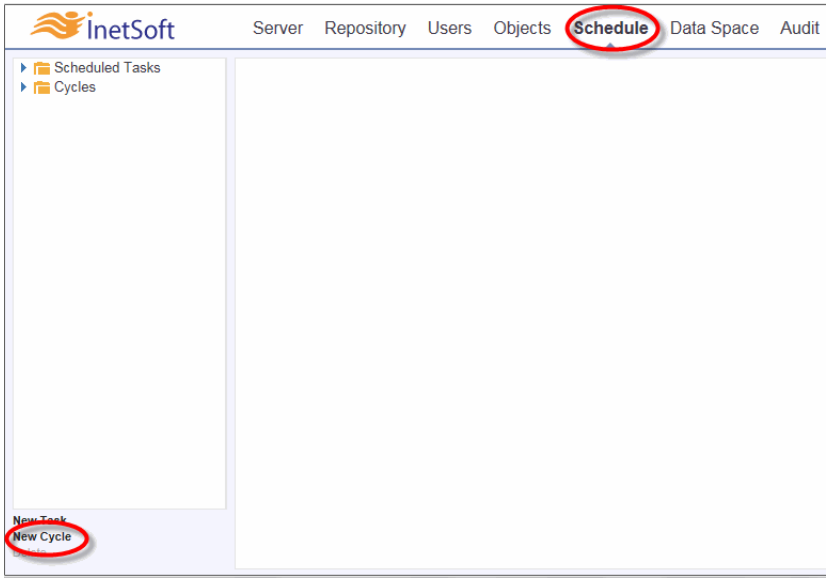
4. To halt a task in progress, press the ‘Stop Now’ button.

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.)

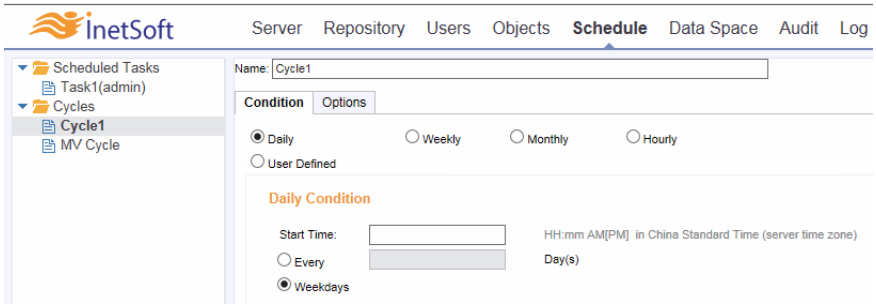
To create a new cycle, follow the steps below:

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

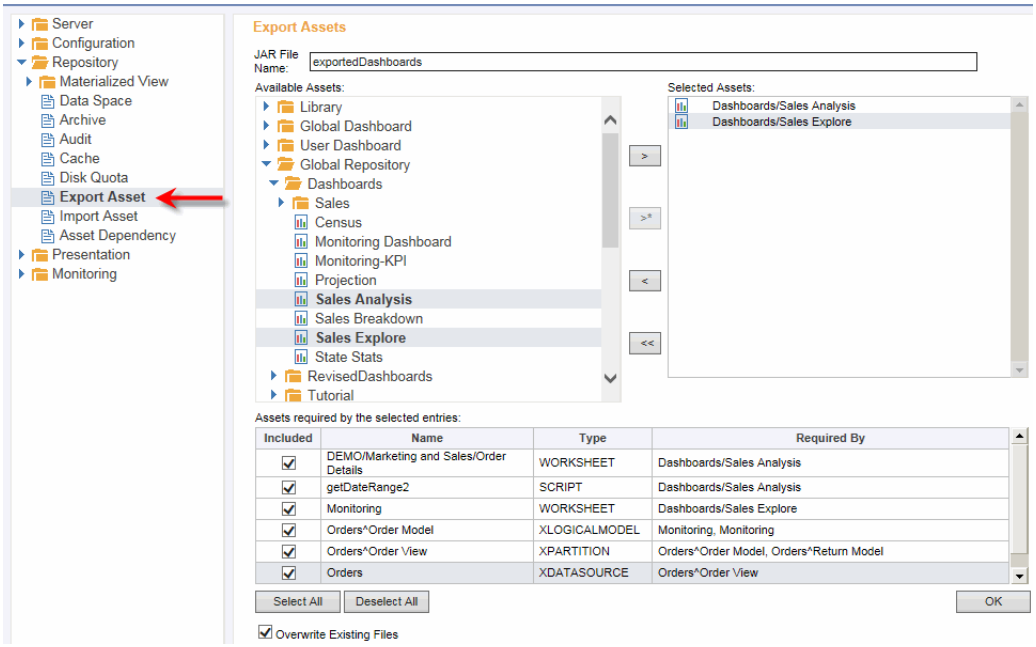
To export a set of assets into an archive (JAR) file, follow the steps below:

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.

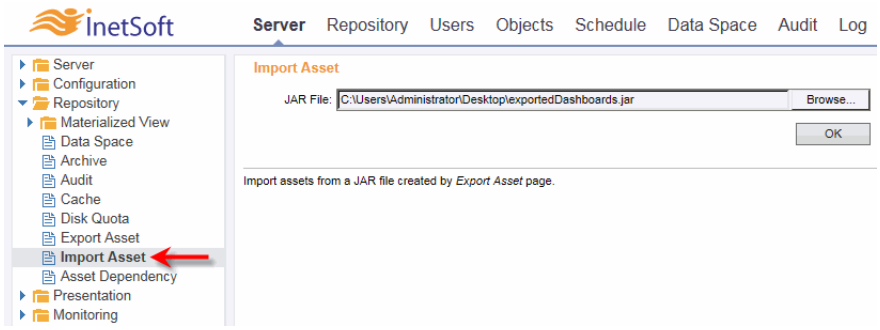
See *Importing Assets* for instructions on how to import the assets from the JAR file into another server environment.

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:

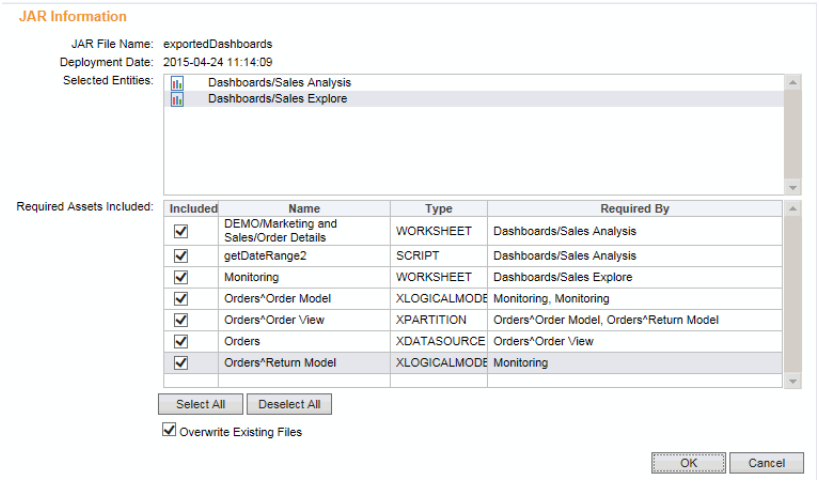
1. In Enterprise Manager, open the 'Repository' > 'Import Asset' page under the **Server** tab.

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

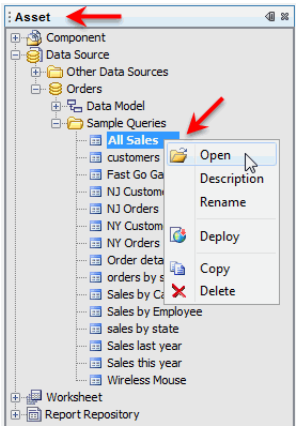
To access an existing data source, query, model, or VPM, follow the steps below:

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.

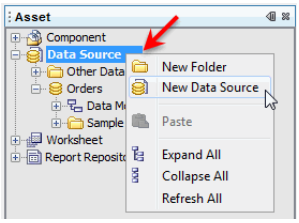
To create a new data source, query, model, or VPM, follow the steps below:

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 The Data Modeling Toolbar

The data modeling options on the toolbar may change slightly depending on the data source type.

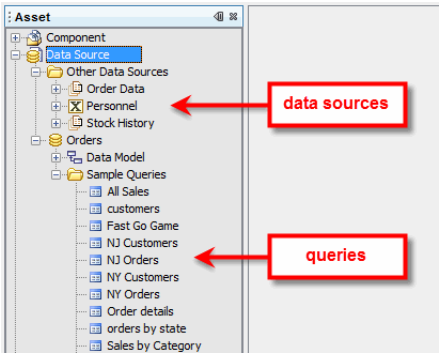
Table 1. Data Modeling Toolbar Buttons

	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).

2.3 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.

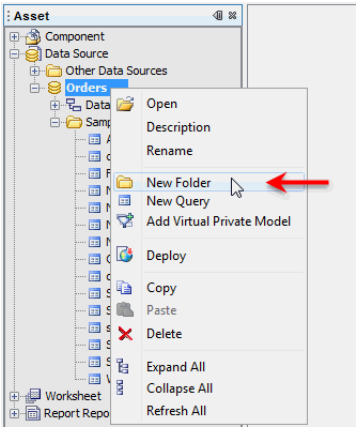
	*Relational Databases* (JDBC: Oracle, DB2, MySql, SQL Server, Sybase, Access, etc.)
	*XML Data Source*
	*Generic Multidimensional Data Source* (XMLA)
	*Text Data Source* (CSV, Tab-delimited)
	*Web Service Data Source* (SOAP)
	*Java Object Data Source*
	*SAP Data Source*

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.

2.3.1 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'.

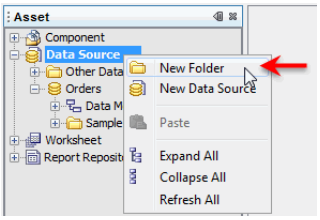
3. Drag the desired queries into the new folder.

2.3.2 Grouping Data Sources Using Folders

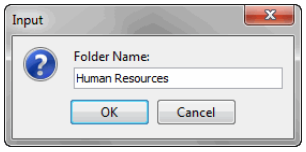
You can organize data sources into folders. This allows you to assign permissions to data sources on a folder-by-folder basis. (See *Component/ Object Permissions* in *Advanced Server Administration* for details on assigning permissions.)

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.

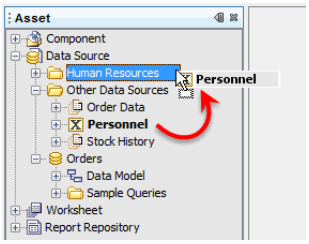


2. When prompted, enter a name for the new folder, and press 'OK'.



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.

2.3.3 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.

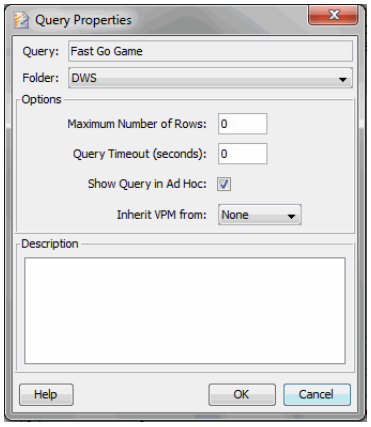
2.4 Advanced Toolbar Buttons

When you open a query for editing, the toolbar provides several query-specific buttons.

2.4.1 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.

2.4.2 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:

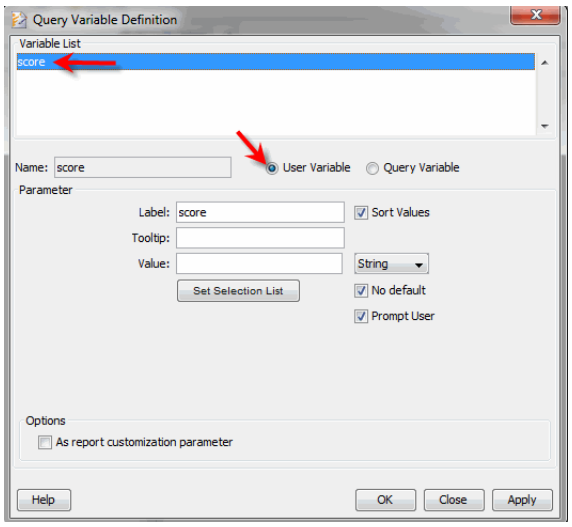
1. Open the desired query for editing. (Double-click the query in the **Asset** panel.)



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).



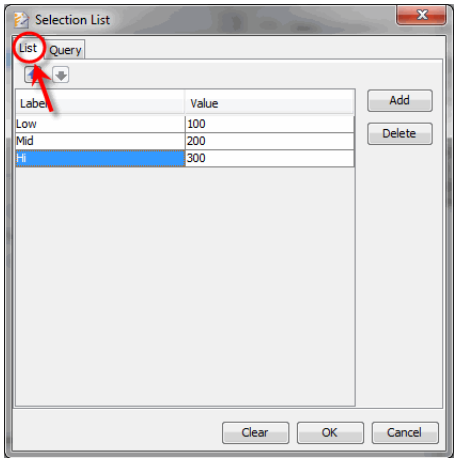
5. Click ‘Set Selection List’ to open the ‘Selection List’ dialog box. (The button’s label is displayed in green if the selected parameter already has a list associated.)

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.)



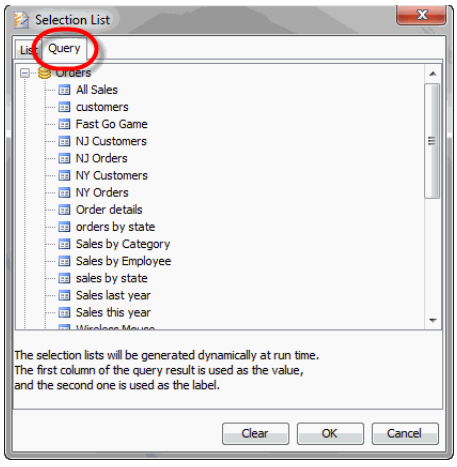
7. To acquire parameter labels/values from a different query at runtime, follow these steps:

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:

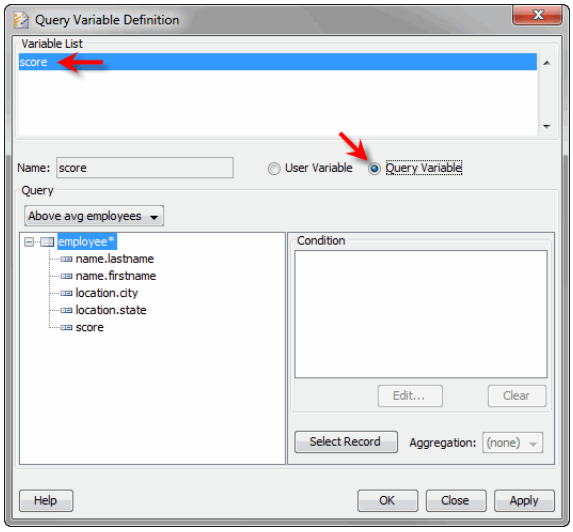
1. Open the desired query for editing. (Double-click the query in the **Asset** panel.)



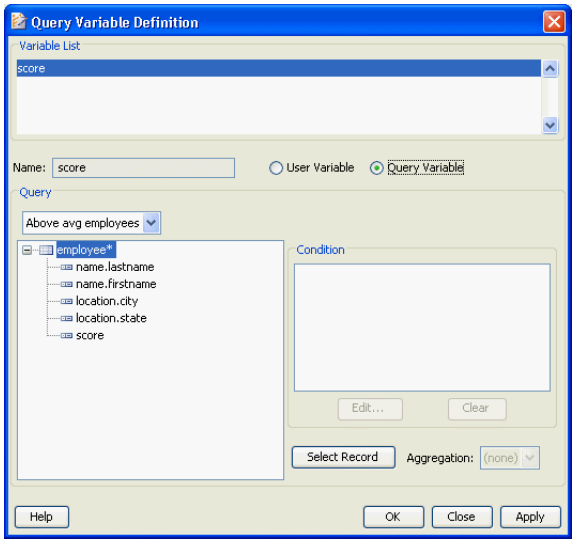
2. Click the 'Parameters' button in the toolbar. This opens the 'Query Variable Definition' dialog box. The 'Variable List' panel at the top displays any query parameters that have been specified in the query condition.

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.

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.

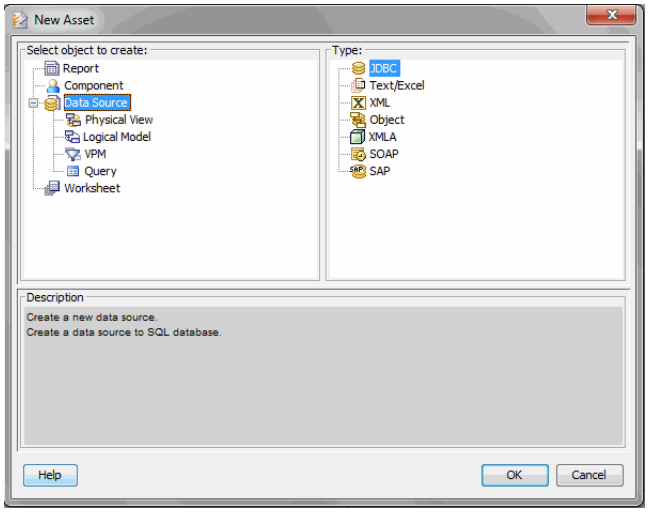


1. Click the 'New' button in the Style Studio toolbar. This opens the 'New Asset' dialog box.

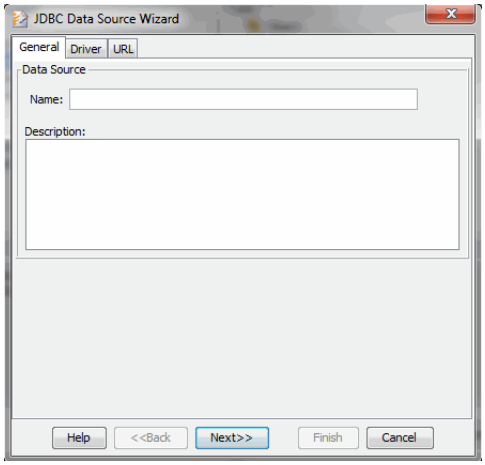


2. In the left panel of the 'New Asset' dialog box, click the 'Data Source' node on the tree.

3. In the right panel, select 'JDBC', and click 'OK'.



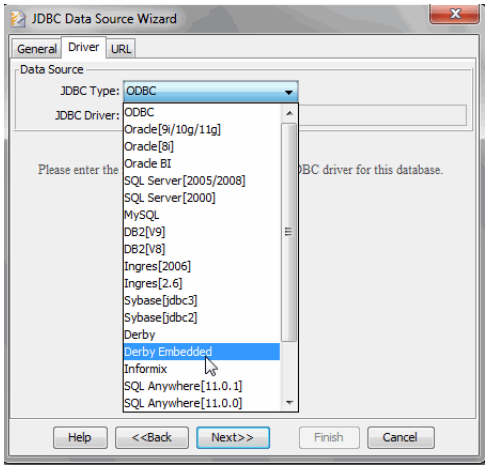
This opens the ‘JDBC Data Source Wizard’.



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.



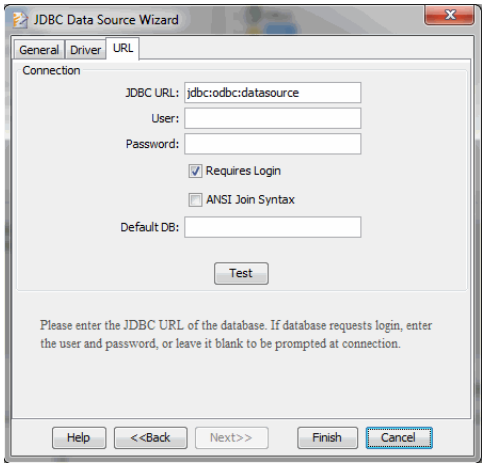
6. On the **URL** tab, enter the required connection information:

   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.

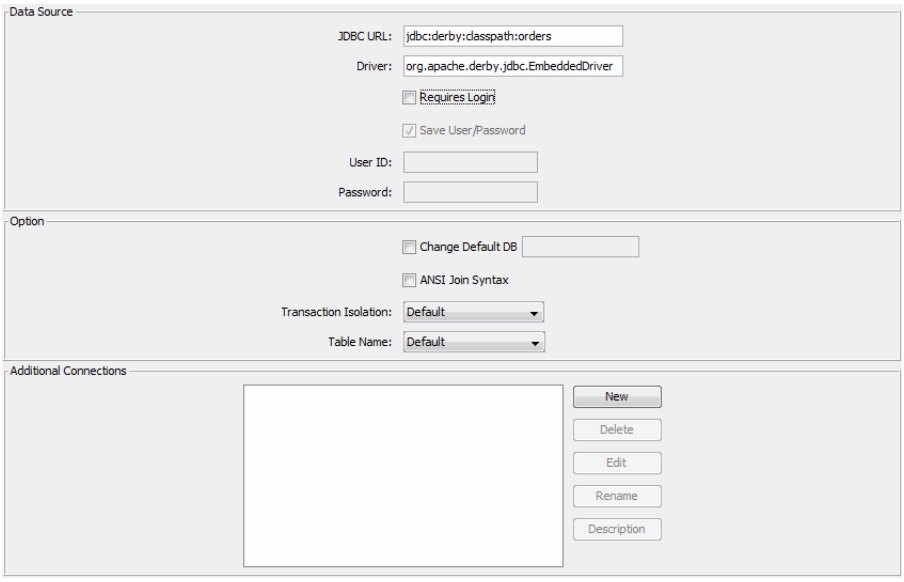
   c. (Optional) Select ‘ANSI Join Syntax’ to specify that queries should use the ANSI standard.

   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. Press the ‘Save’ button in the Style Studio toolbar to save the new data source to the *asset.dat* file or *asset.dat.d* directory.

The new data source is now listed under the ‘Data Source’ node of the **Asset** panel.

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:

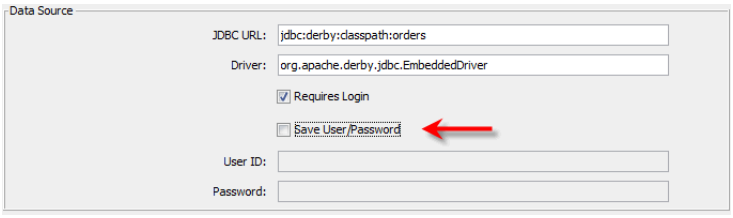
- Test for parameter stored as JVM system variable.
- Test for parameter stored as report parameter in the *sree.properties* file.
- Test for parameter passed within URL string.

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.

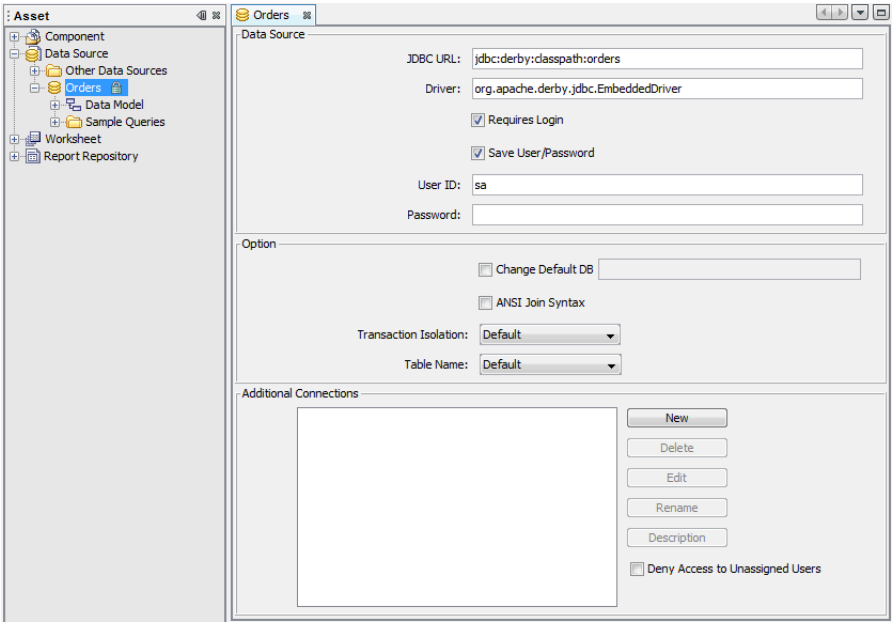
3.1.4 Adding Additional Connections

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.

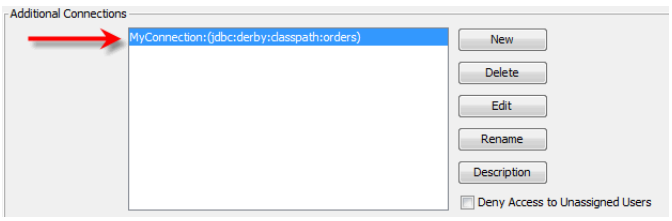
To add a new connection, follow the steps below:

1. From the 'Data Source' node of the **Asset** panel, open (double-click) the data source for which you want to create additional connections.

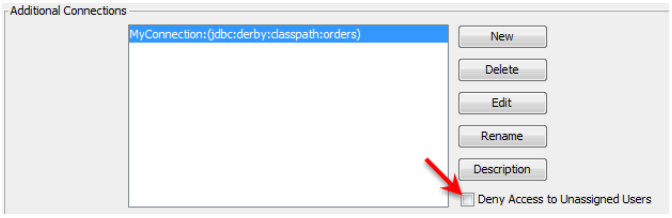
2. In the 'Additional Connections' panel of the data source definition page, press 'New'. This will start the 'New Data Source' wizard.



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.



4. (Optional) Select ‘Deny Access to Unassigned Users’ to prevent users who do not have explicit connection privileges from obtaining any data connection. Deselect this option to permit unassigned users to connect to the default connection.



   To assign connection permissions to users and groups, see *Component/Object Permissions*.

5. Press ‘Save’ on the toolbar to save the changes to the data source.

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, press the ‘Rename’ and ‘Description’ buttons, respectively. Press ‘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.

Amazon Redshift

Amazon Redshift is based on PostgreSQL. Download the appropriate JDBC driver from the PostgreSQL website. Then follow the procedure described in *Connecting to a Database*. The ‘JDBC URL’ in the JDBC Data Source Wizard has the following form:

```
jdbc:postgresql://{host}:{port}/{databaseName}
```

Note that Amazon Redshift does not recognize the JDBC `maxRows` parameter. Instead, specify a `LIMIT` clause to restrict the result set or use an `OFFSET` clause to skip to a specific starting point in the result set.

Apache Hadoop and Hadoop Hive

Download the appropriate JDBC driver from the Hive Driver website. (Consult your Hive database administrator for instruction on which drivers to use.) Then follow the procedure described in *Connecting to a Database*. For Apache Hive the 'JDBC URL' in the JDBC Data Source Wizard has the following form:

```
jdbc:hive://{host}:{port}/{databaseName}
```

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.

Cloudera

Cloudera is based on Apache Hive. Download the appropriate JDBC driver from the Hive Driver website. Then follow the procedure described in *Connecting to a Database*. The 'JDBC URL' in the JDBC Data Source Wizard has the following form:

```
jdbc:hive://{host}:{port}/{databaseName}
```

Derby

The examples in the documentation use the sample Derby database. See *Connecting to a Database* for more information.

EMC Greenplum

EMC Greenplum is based on PostgreSQL. Download the appropriate JDBC driver from the PostgreSQL website. Then follow the procedure described in *Connecting to a Database*. The 'JDBC URL' in the JDBC Data Source Wizard has the following form:

```
"jdbc:postgresql://{host}:{port}/<schema>?
kerberosServerName=postgres&jaasApplicationName=pgjdbc&
user={user/password}"
```

FileMaker

Locate the `fmjdbc.jar` JDBC driver in your FileMaker installation. Then follow the procedure described in *Connecting to a Database*. For FileMaker, the 'JDBC URL' in the JDBC Data Source Wizard has the following form:

```
jdbc:filemaker://{host}/{databaseName}
```

HP Vertica

Download the appropriate JDBC driver from the Vertica website. Then follow the procedure described in *Connecting to a Database*. For HP Vertica the 'JDBC URL' in the JDBC Data Source Wizard has the following form:

```
jdbc:vertica://{host}:{port}/{databaseName}
```

IBM 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://{host}:{port}/{databaseName}
```

IBM Informix

Download the appropriate JDBC driver from the IBM website. Then follow the procedure described in *Connecting to a Database*. For Informix the 'JDBC URL' in the JDBC Data Source Wizard has the following form:

```
jdbc:informix-sqli://{host}:{port}/{databaseName}
```

IBM Netezza

Download the appropriate JDBC driver from the IBM website. Then follow the procedure described in *Connecting to a Database*. For Netezza the 'JDBC URL' in the JDBC Data Source Wizard has the following form:

```
jdbc:netezza://{host}:{port}/{databaseName}
```

Ingres

Download the appropriate JDBC driver from the Actian website. Then follow the procedure described in *Connecting to a Database*. For Ingres the 'JDBC URL' in the JDBC Data Source Wizard has the following form:

```
jdbc:ingres://{host}:{port}/{databaseName}
```

LucidDB

Follow the procedure described in *Connecting to a Database*. For LucidDB the 'JDBC URL' in the JDBC Data Source Wizard has the following form:

```
jdbc:luciddb:http://{host}:{port}
```

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.

Microsoft 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://{host}:{port}
```

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.

Microsoft 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://{host};instanceName=SQLEXPRESS
```

Use your SQL Server login credentials rather than Windows domain credentials.

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://{host}:{port}/{databaseName}
```

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.

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:@{host}:{port}:{service ID}
```

Oracle Business Intelligence Enterprise Edition (OBIEE)

Locate the `bijdbc.jar` JDBC driver in `{MiddlewareHome}\orahome\bifoundation\jdbc\lib`. Then follow the procedure described in *Connecting to a Database*. For OBIEE, the 'JDBC URL' in the JDBC Data Source Wizard has the following form:

```
jdbc:oraclebi://{host}:{port}/
```

Oracle JD Edwards

Download the `e1dadriver.jar` JDBC driver from the Oracle website. Then follow the procedure described in *Connecting to a Database*. For JD Edwards the 'JDBC URL' in the JDBC Data Source Wizard has the following form:

```
jdbc:oracle:enterpriseone://{host}:{port}/{ERP Environment}
```

The JDBC driver implements JD Edwards EnterpriseOne connection modes using JDBj, which is a Java data-access API. (The JDBC driver parses SQL statements and transforms them into JDBj operations.)

Oracle PeopleSoft

Create a direct connection to the back-end database. See the appropriate section below:

- *IBM DB2*
- *Oracle*
- *Microsoft SQL Server*

PostgreSQL

Download the appropriate JDBC driver from the PostgreSQL website. Then follow the procedure described in *Connecting to a Database*. For

PostgreSQL, the 'JDBC URL' in the JDBC Data Source Wizard has the following form:

```
jdbc:postgresql://{host}:{port}/{databaseName}
```

SAP Hana

Download the appropriate driver from the SAP website. Then follow the procedure described in *Connecting to a Database*. For SAP Hana, the 'JDBC URL' in the JDBC Data Source Wizard has the following form:

```
jdbc:sapdb://{host}/{database}
```

See Also
SAP Data Source, for connection to other SAP data sources.

SAP SQL Anywhere

Download the appropriate driver from the SAP website. Then follow the procedure described in *Connecting to a Database*. For SQL Anywhere, the 'JDBC URL' in the JDBC Data Source Wizard has the following form:

```
jdbc:sqlanywhere:<connectionString>
```

See Also
SAP Data Source, for connection to other SAP data sources.

Siebel CRM

Create a direct connection to the back-end database. See the appropriate section below:

- *IBM DB2*
- *Oracle*

SAP Hana

Download the appropriate driver from the SAP website. Then follow the procedure described in *Connecting to a Database*. For SAP Hana, the 'JDBC URL' in the JDBC Data Source Wizard has the following form:

```
jdbc:sapdb://{host}/{databaseName}
```

See Also
SAP Data Source, for connection to other SAP data sources.

SQLite

Download the JDBC driver from the SQLite download website. Then follow the procedure described in *Connecting to a Database*. For SQLite, the 'JDBC URL' in the JDBC Data Source Wizard has the following form:

```
jdbc:sqlite:{databaseName}
```

Teradata

Download the appropriate driver from the Teradata website. Then follow the procedure described in *Connecting to a Database*. For Teradata, the 'JDBC URL' in the JDBC Data Source Wizard has the following form:

```
jdbc:teradata://{host}/DBS_PORT={port}/DATABASE=
{databaseName}/CHARSET=UTF8,COMPAT_DBS=true
```

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.

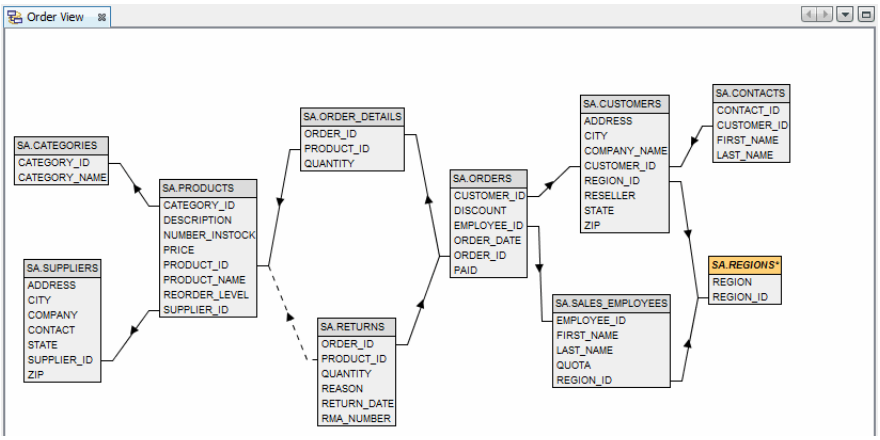
```
Data models are not available for the 'Hadoop Hive' JDBC type.
```

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*

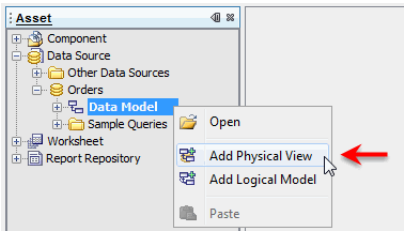
3. *Adding an Embedded SQL View*
4. *Defining Join Relationships*
5. *Resolving Loop Traps*

The following sections explain these steps in greater detail.

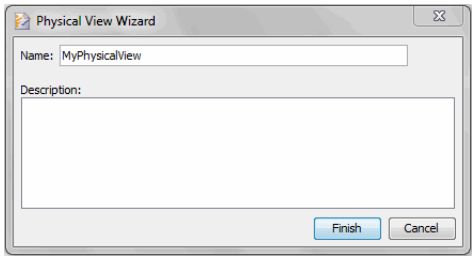
Defining a New Physical View

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.

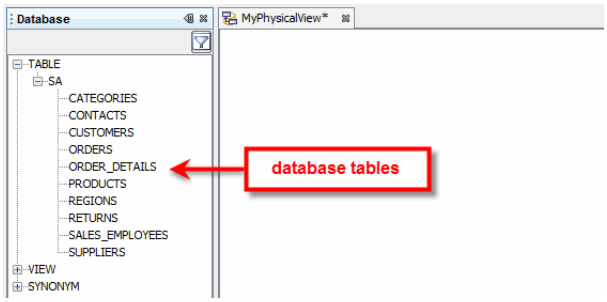


4. In the 'Physical View Wizard' dialog box, enter a name and description for the new view.



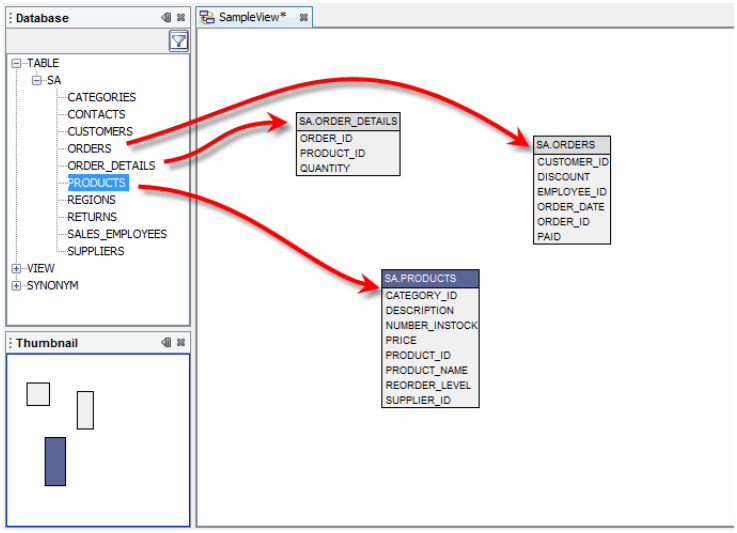
5. Click 'Finish' to close the dialog box.

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.



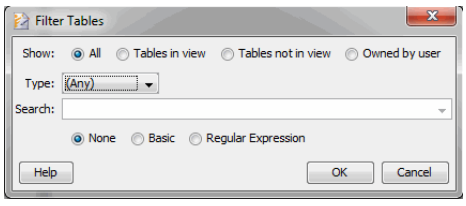
Adding Database Tables to the Physical View

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.

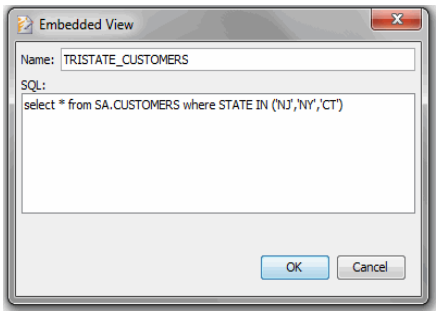
```
Note: An embedded view may result in a different execution plan
than a database view. You should test both options against your
database.
```

To create an embedded view, follow the steps below.

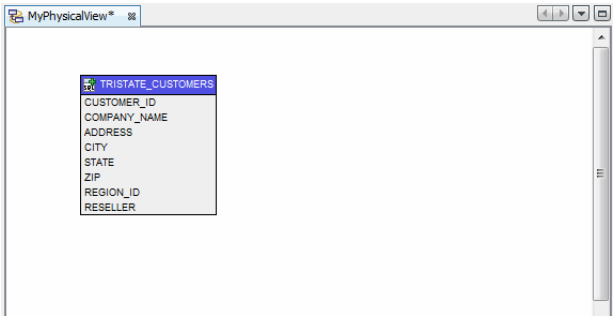
1. Press the 'New View' button in the toolbar.



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.

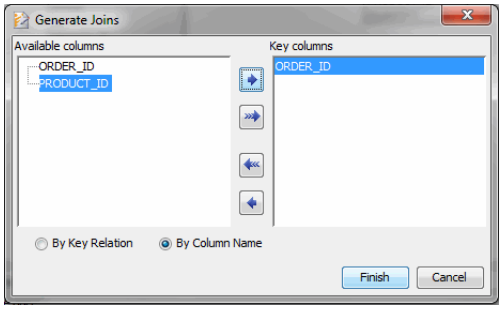
See Also
Defining Expression Attributes, for information on creating formula columns.

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.



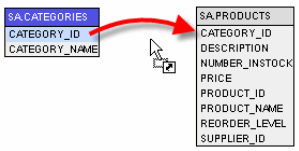
2. Select 'By Key Relation' to join columns by using primary key/ foreign key definitions, or 'By Column Name' to join columns by using common column names.

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

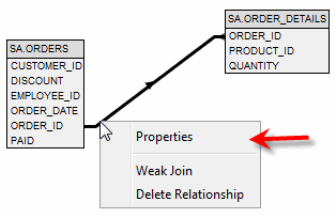
Defining the Join Type, for information on modifying a join.

Defining the Join Type

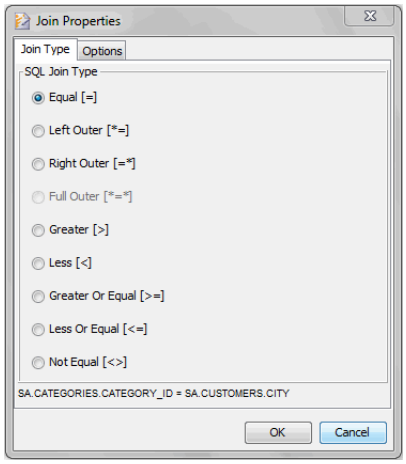
The default join type is an inner join. To change the join type, follow the steps below:

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

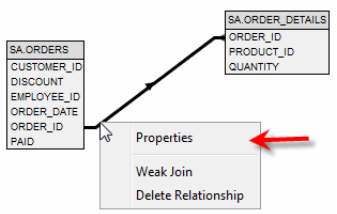
In many cases, the position of a join within a SQL query can greatly affect query performance. To determine the positioning of a particular join within the SQL query generated by the physical view, you can specify a priority for the join.

```
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:

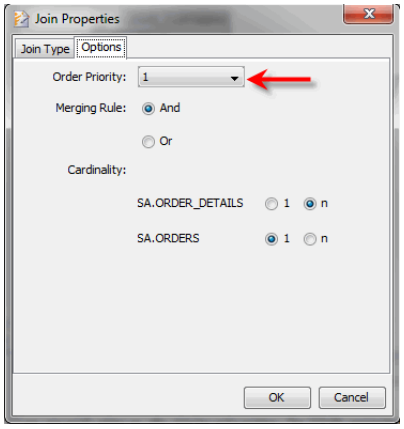
1. Click on the desired join line to select the join.

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.

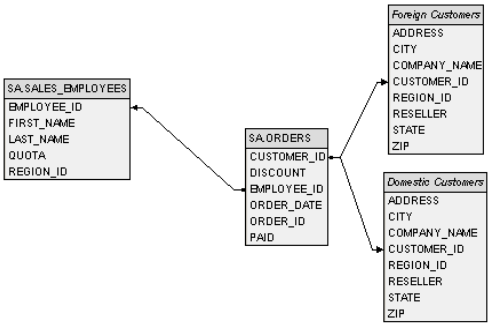


Changing the Join Operator

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

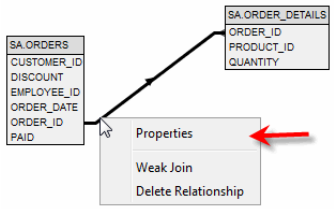
The `CUSTOMER_ID` field in the ‘Orders’ table can reference the `CUSTOMER_ID` field in either the ‘Domestic Customers’ or the ‘Foreign Customers’ table. In this case you need to a logical OR between the “Order -> Domestic Customers” join and the “Order -> Foreign Customers” join because they are mutually exclusive.



To specify the logical operator for a join, follow these steps:

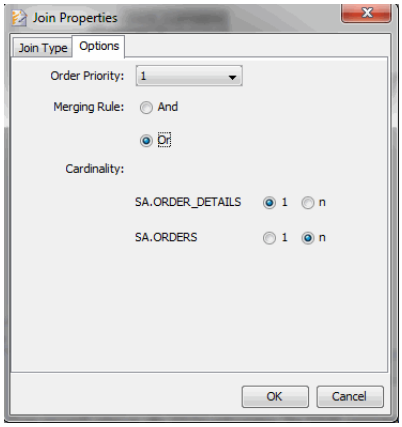
1. Click on the desired join line to select the join.

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’.

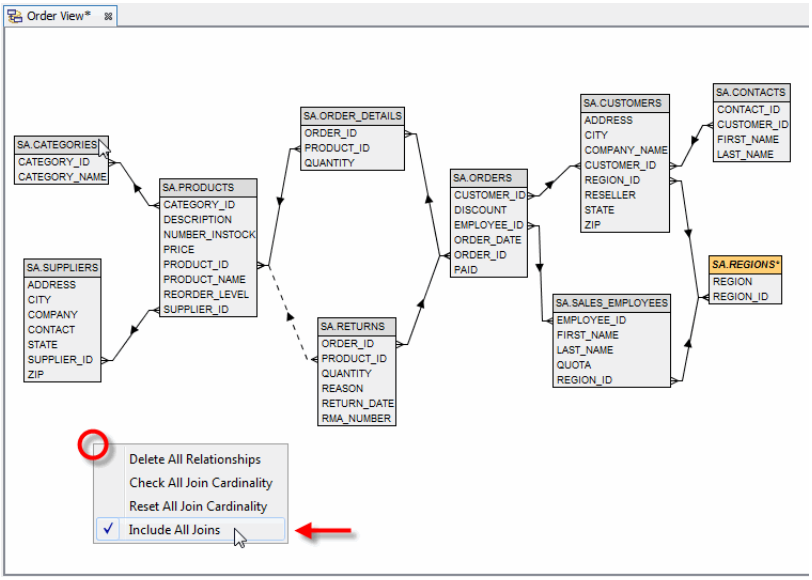


The ‘Merging Rule’ determines how the join is integrated together with other joins between the same tables. The ‘And’ option indicates that the join is applied conjunctively (i.e., the join condition must be satisfied even if another join condition is already

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

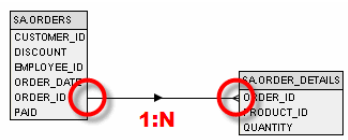
Generally, when Style Intelligence generates a query from a physical view, only the necessary joins between requested tables are included in the SQL string. If you want to include all of the physical view joins in the SQL queries generated from the physical view, right-click in an empty area of the physical 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.

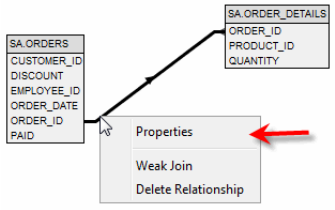
The automatically-detected cardinality for a join is indicated by a fan icon on the join link. However, if you need to override the automatic cardinality detection, you can do this by manually specifying the relationship between tables associated by a join.



To manually specify the cardinality relationship for a join, follow the steps below:

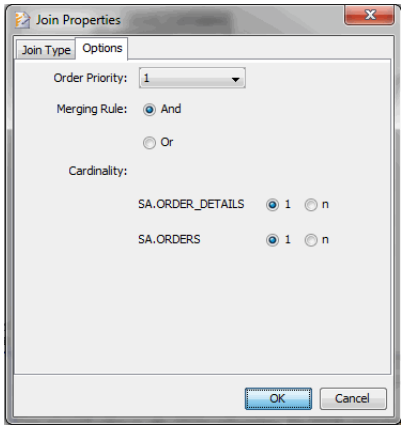
1. Click on the desired join line to select the join.

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.

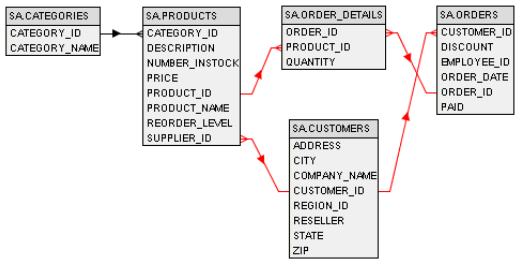
```
An aliased table is indicated by a colored header. To display the
name of the table that was originally aliased, hover the mouse over
the alias table header.
```

- *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

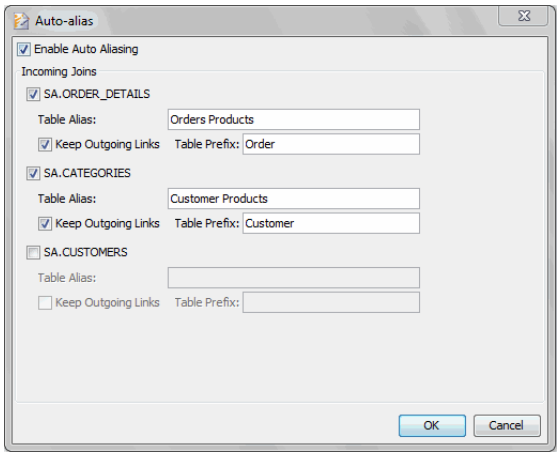
the table will be created. One copy corresponds to the incoming join from the 'ORDER_DETAILS' table, and the other copy corresponds to the incoming join from the 'CUSTOMERS'. By splitting the 'PRODUCTS' table into two aliases, the join cycle is eliminated. (Note, however, that if only the 'PRODUCTS' 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.

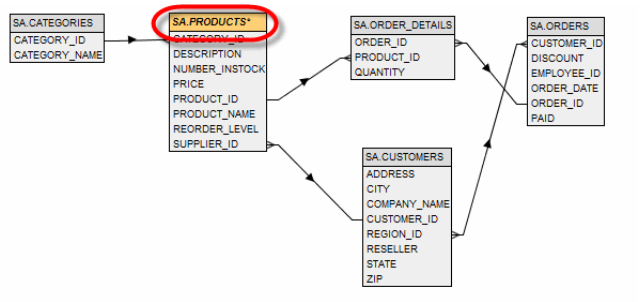
7. In the 'Table Alias' field for the 'CUSTOMERS' table, enter "Customer Products".

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.

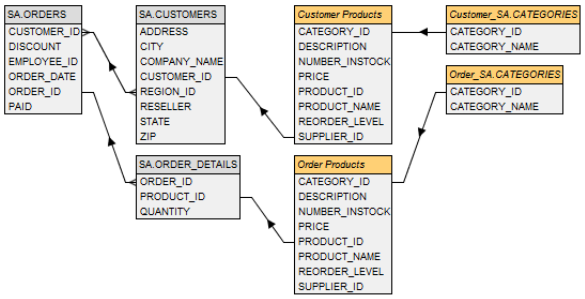


Note: In the diagram, an *italicized* table name and colored header indicate that the table is aliased. To view the full effect of aliasing, however, you must preview the physical view (below).

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.

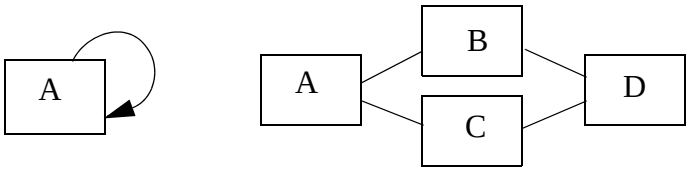
For loop traps, Style Intelligence highlights the offending joins in the physical view using red color. Loop traps are not corrected automatically, and you should select the appropriate method for resolving loops based on your knowledge of the schema. See *Resolving Loop Traps* for full details.

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.

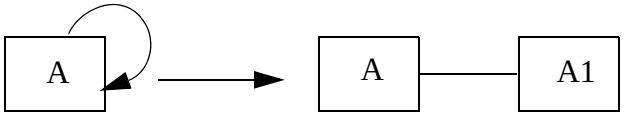
Loop Traps

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.

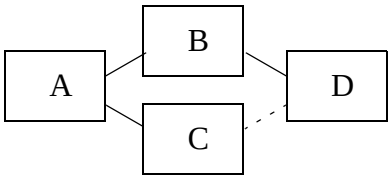


For example, when an “Employee” table contains a “manager id” is joined to itself (i.e., to indicate an employee’s manager), this produces a self-join loop. You must resolve the self-join to allow end users to find employees and their respective managers. Self-join loops are most easily resolved by creating 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.

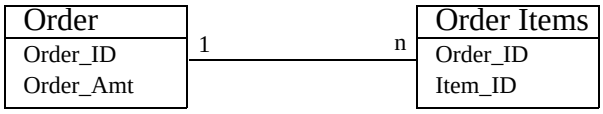
Multi-table loops can be addressed using aliasing or auto-aliasing. Additionally, these loops can be addressed by using a weak join which will not be included in the join path of the resultant query. (See *Resolving Loop Traps* for details.)



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.

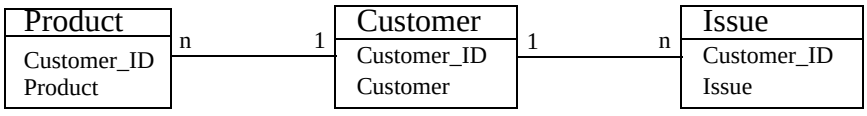
The *fan trap* is evidence of an inherent deficiency in this particular schema, wherein the schema is not providing the order amount on the item-level of granularity. If this aggregation is desired, the best option is to enhance the schema so that the order amount is broken down and recorded in the ‘Order 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*.

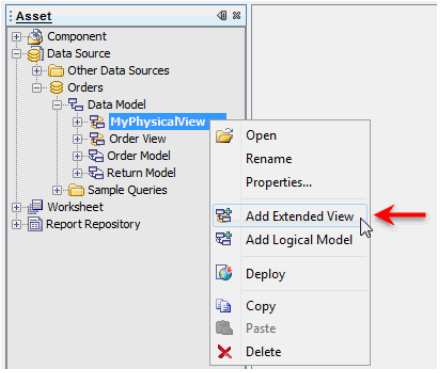
3.3.4 Extending a Physical View

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.

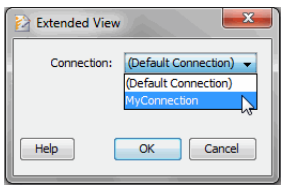
1. Open the **Asset** panel, and expand the 'Data Source' node.

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.



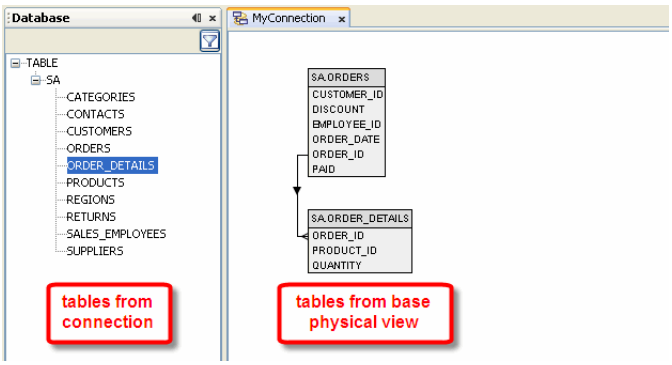
Note: An extended physical view is associated with one (and only one) connection. Likewise, a connection can be associated with only one extended physical view.

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:

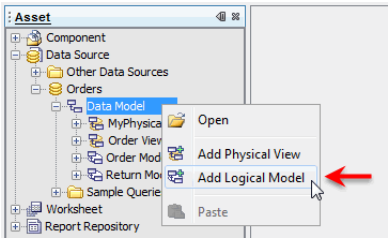
1. *Defining a New Logical Model*
2. *Adding Entities and Attributes*
3. *Modifying Entities and Attributes*
4. *Defining Expression Attributes*

The following sections explain these steps in greater detail.

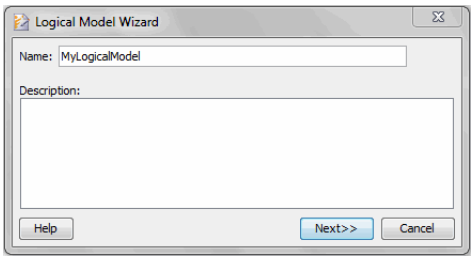
Defining a New Logical Model

To define a new logical view, follow the steps below:

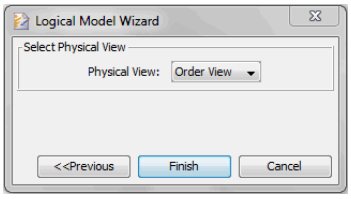
1. Click the 'Asset' button to expand the **Asset** panel.
2. Expand the 'Data Source' node, and then expand the relational database data source for which you want to create the data model.
3. Right click the 'Data Model' node, and select 'Add Logical Model' from the context menu. This open the 'Logical 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.

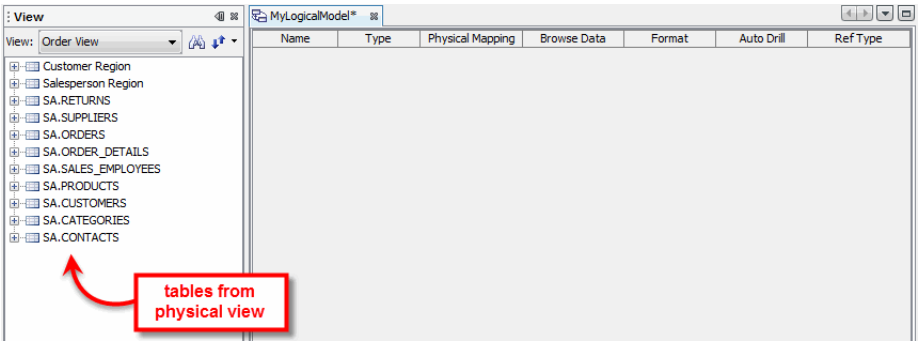


5. From the 'Physical View' menu, select the physical view on which the model should be based. (The model will have access to the tables of the underlying physical view.)



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.

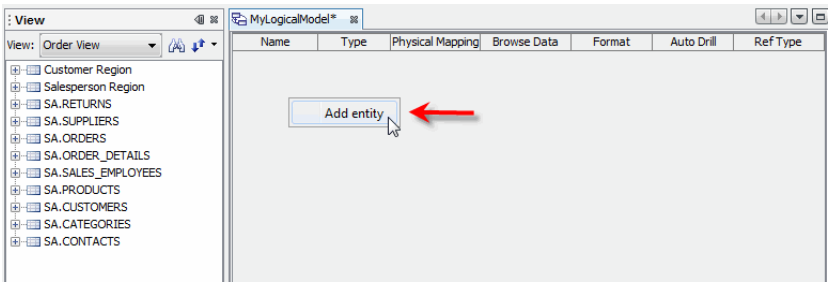


Adding Entities and Attributes

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:

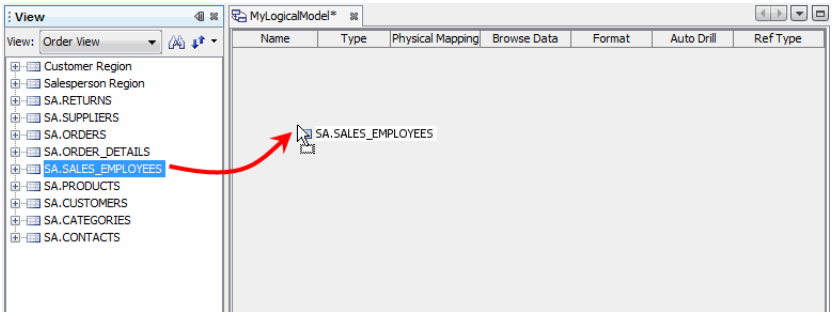
1. Right-click an empty region of the logical model panel, and select ‘New Entity’ from the context menu. This opens the ‘Entity Wizard’, where you can enter a name for the new entity.



Note that an entity name cannot contain any of the following characters:

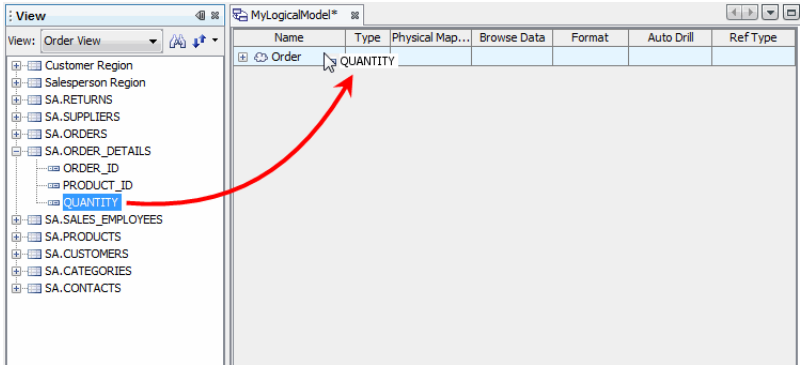
```
‘ ! ^ = [ ] \ / : " ' < > ?
```

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*.

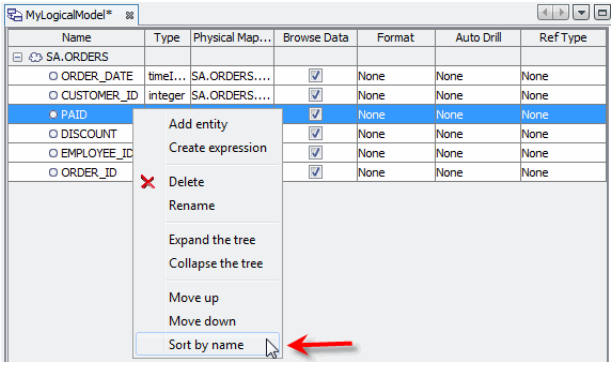
To add an attribute to an entity, drag a field from the **View** panel onto the desired entity. (Expand the entity node to see the component attributes.)



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.




```
Entity names cannot contain characters from the following set: ` !
^ = [ ] \ / : " ' < > ?. In addition to these, attribute names also
cannot contain characters from the following set: . & { }.
```

- **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.
- **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

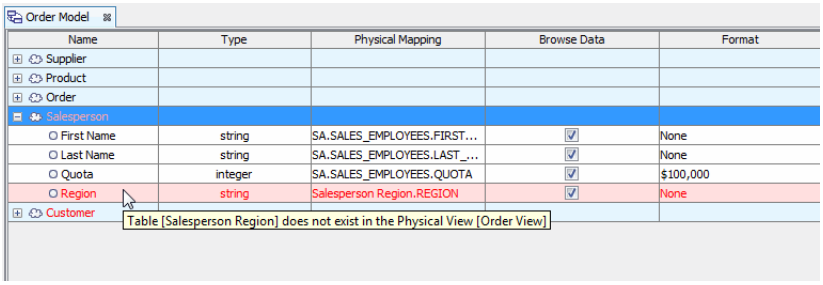
If you make changes to the underlying database schema or to the physical view (for example, if you remove tables or columns), the mappings in the logical model may no longer be valid. When Style Studio detects a validity

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.

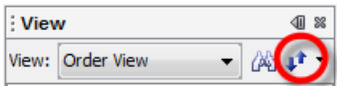
2. In the logical model, hover the mouse over the ‘Name’ field or ‘Physical Mapping’ field of the red-highlighted attribute. This displays a message explaining the cause of the mapping problem.



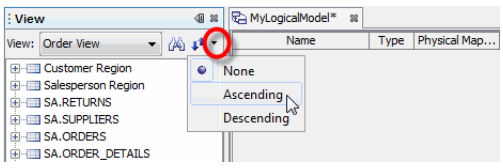
3. When you have determined the source of the problem, adjust the database schema, physical view, or logical model to restore a valid mapping.

Sorting and Filtering a Physical View

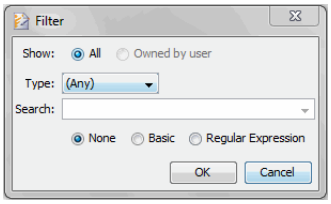
When you add table fields from a physical view to a logical model, it is often helpful to sort the physical view tables by name. To do this, click the ‘Sort Tables’ button on the **View** panel.



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.



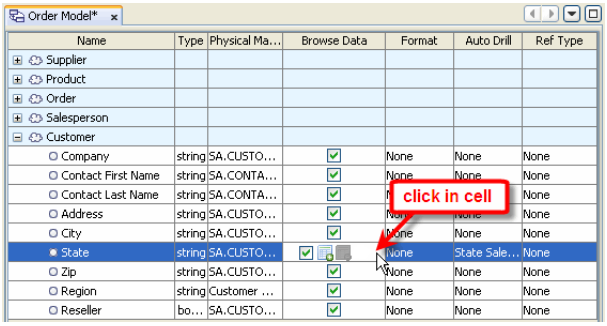
Select ‘Owned by user’ to display only those tables that belong to the user’s schema. Use the ‘Type’ menu to restrict the list to either ‘Tables’ or ‘Views’.

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.

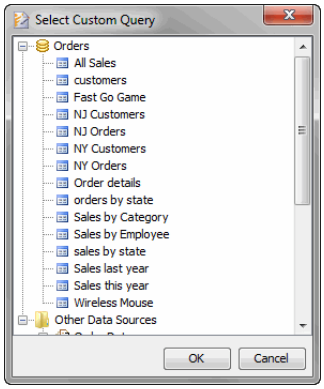
Customizing the Browse Data List

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.

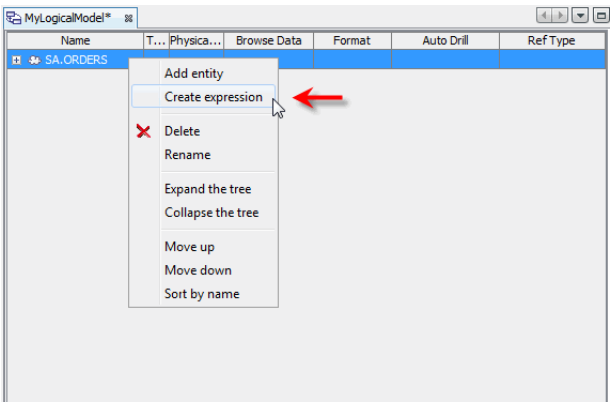
The selected query will now be used to supply the list of values that appears in the 'Browse Data' list for the corresponding attribute.

Defining Expression Attributes

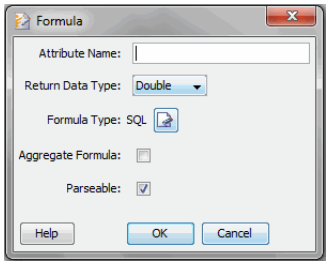
In addition to creating model attributes by directly mapping fields of the physical view, you can also define *expression attributes*. Expression attributes are attributes that are *derived* from table columns. You can use derived attributes in the same way as any other model attribute, e.g., in reports, 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

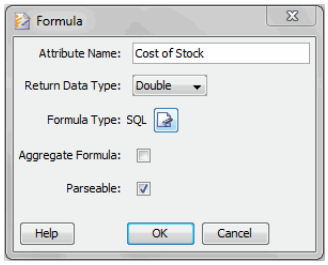
In this example, you will add a new expression attribute to the sample Order Model. The attribute will represent the total price of a particular product's stock (i.e., product price * number in stock).

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']
```

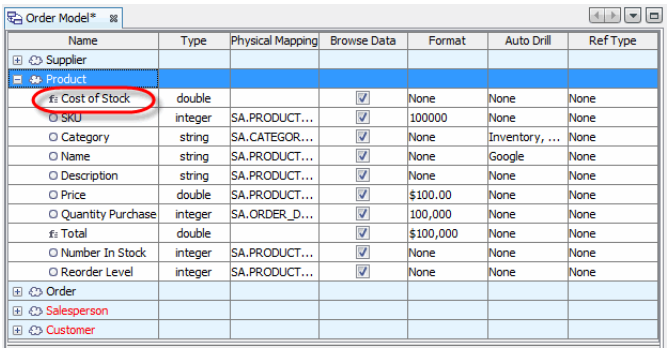
Note: You can double-click the field names in the ‘Properties’ list to add them to the expression (without typing).



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.

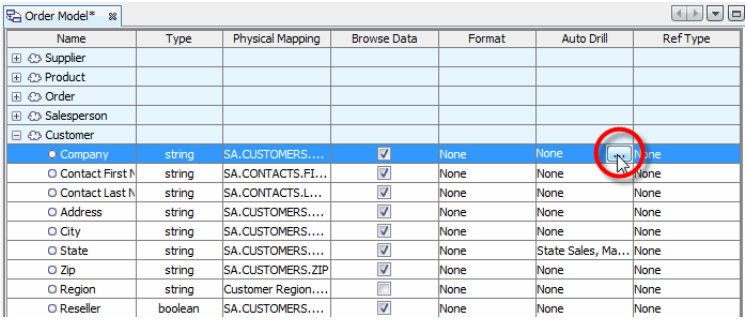
Walkthrough

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.

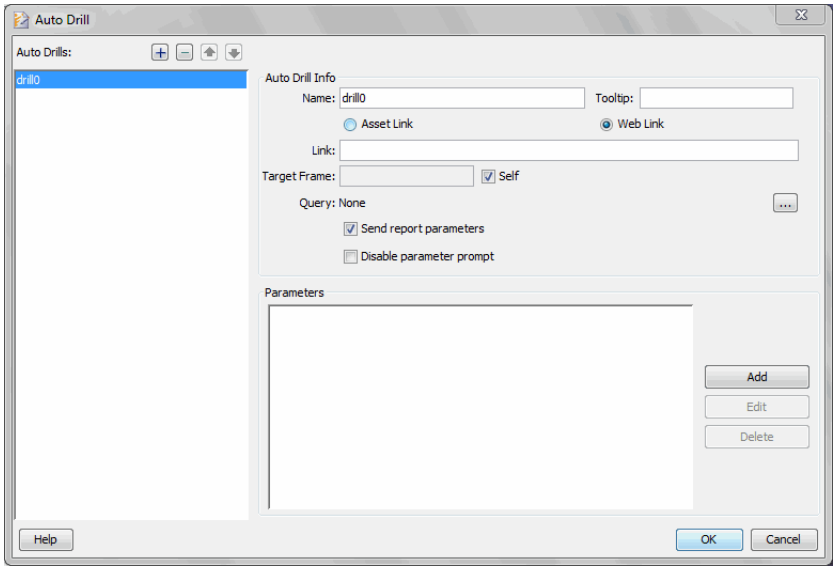
2. Double-click on the 'Order Model' to open the model for editing.

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.





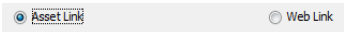
4. In the 'Auto Drill' window, click the 'New Drill' button to create a new auto-drill scheme. The new drill is named 'drill0' by default. (If you add more than one drill to a given attribute, the end-user will be presented with a menu of hyperlink choices corresponding to the different drills you have 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* file (for reports) and *asset.dat* file or *asset.dat.d* directory (for Viewsheets), 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.

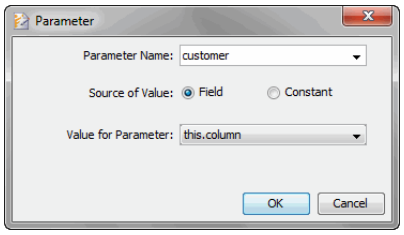
8. (Optional) If you want the target report to be displayed in a particular browser window, enter the name of the desired window in the ‘Target Frame’ field. (The specified name becomes the DOM ‘name’ attribute.) To specify that the target report should open in the User Portal frame, select the ‘Self’ 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.



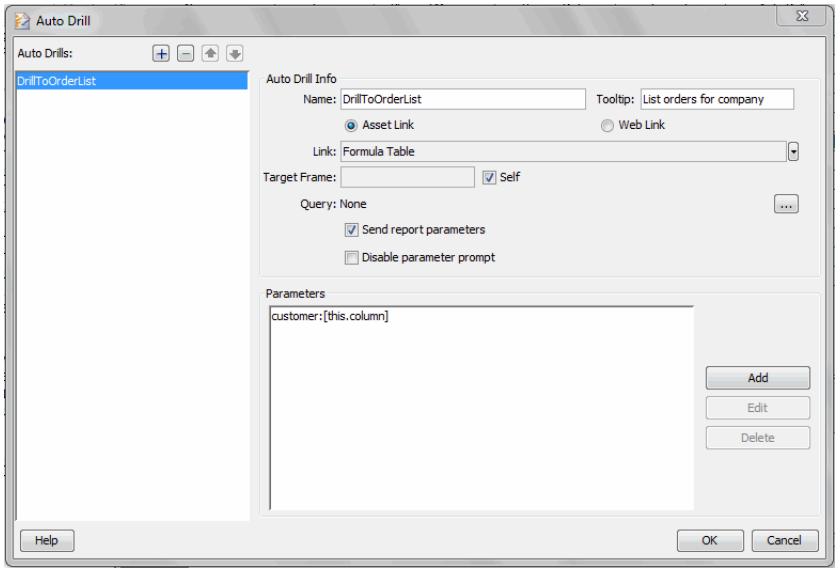
12. Press ‘OK’ to close the ‘Parameter’ dialog box.

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'.

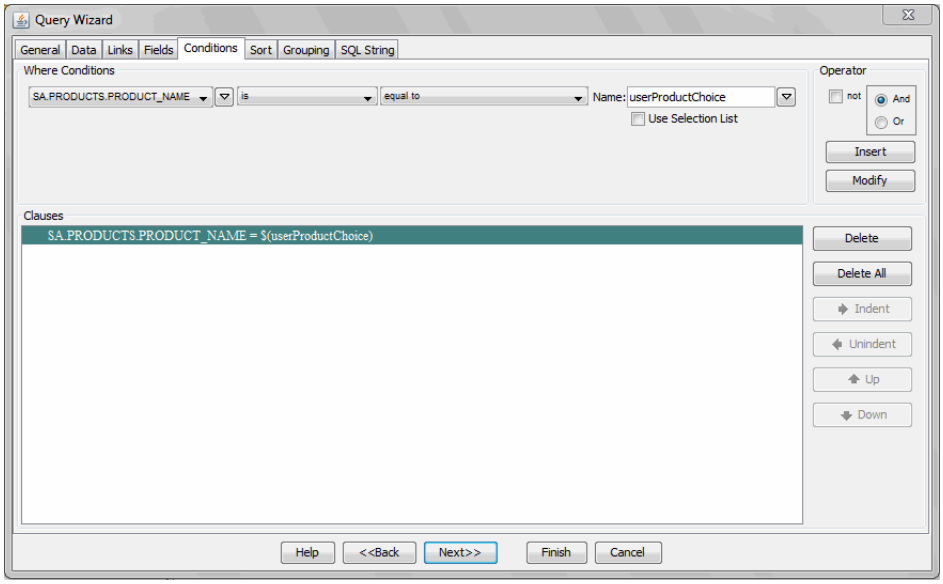
4. Under the **Data** tab of the Query Wizard, select the 'PRODUCTS' table from the 'TABLE' > 'SA' node.

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.

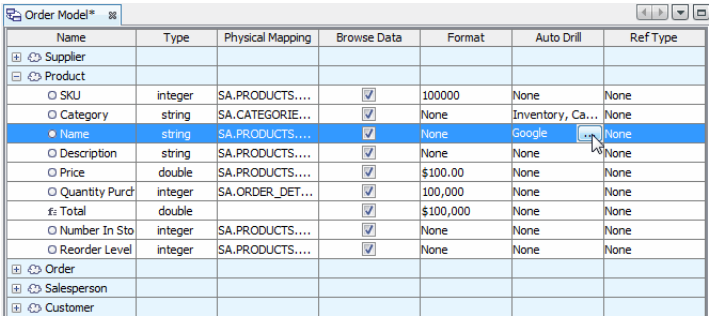
10. Click the 'Finish' button in the Query Wizard, and then click the 'Save' button in the toolbar to save this query.

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.

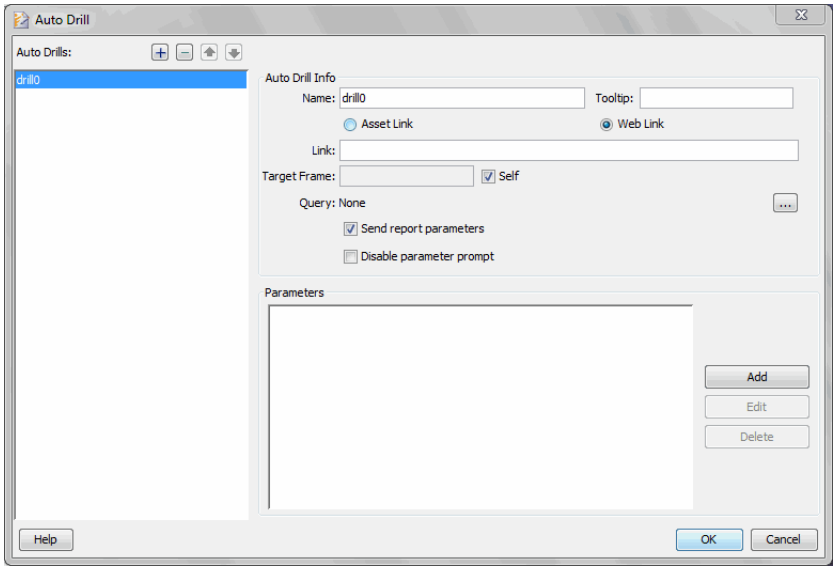
2. Double-click on the ‘Order Model’ to open the model for editing.

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.

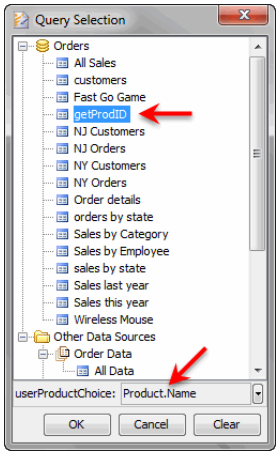
7. Select the 'Web Link' button. In the 'Link' field, enter the URL of the hyperlink destination with parameters omitted. For example, if the hyperlink is to pass a value as parameter to the Google search engine, the 'Link' field should specify "`http://www.google.com/search`".



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'.



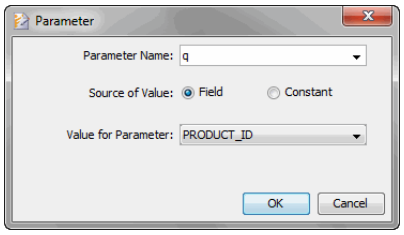
`Note: If the query takes multiple parameters, you can connect them to various attributes/columns from the current model/query. If those columns are present in the table that is displaying the auto-drill, the values of those columns will also be passed to the subquery.`

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.

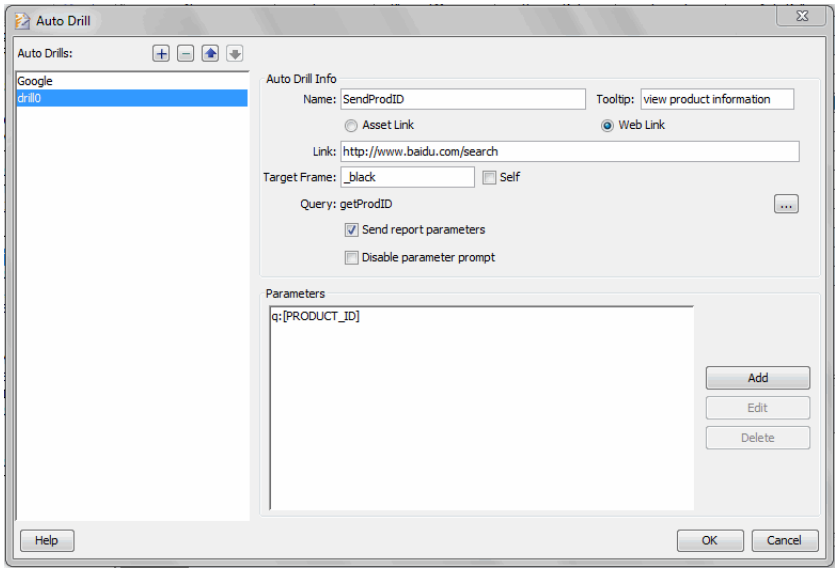
11. Enter the variable name “q” in the ‘Parameter Name’ field. This name will be the parameter name used in the URL that is automatically constructed. (The parameter name “q” is the parameter name expected by the Google search engine.)



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.

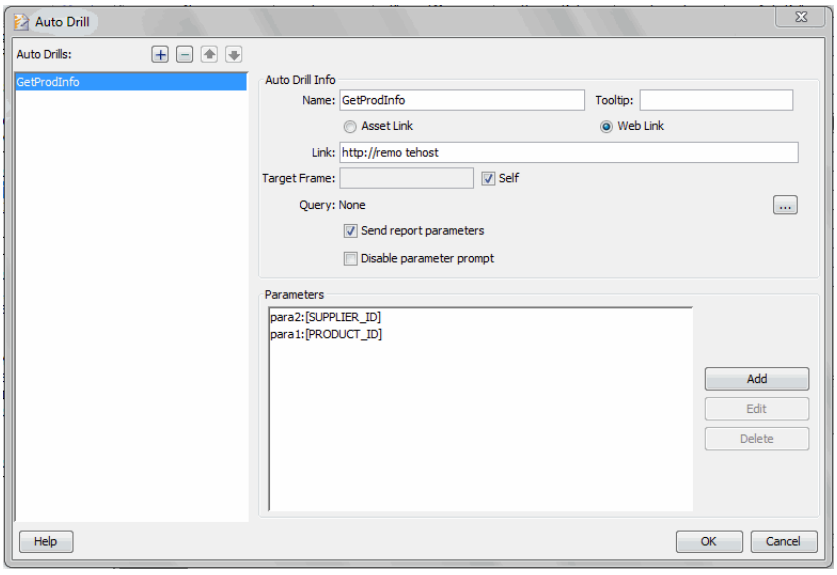
For an illustration of how these hyperlinks will appear in a deployed report, see Steps 16–20 of *Example 1: Passing column values in a drill*.

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

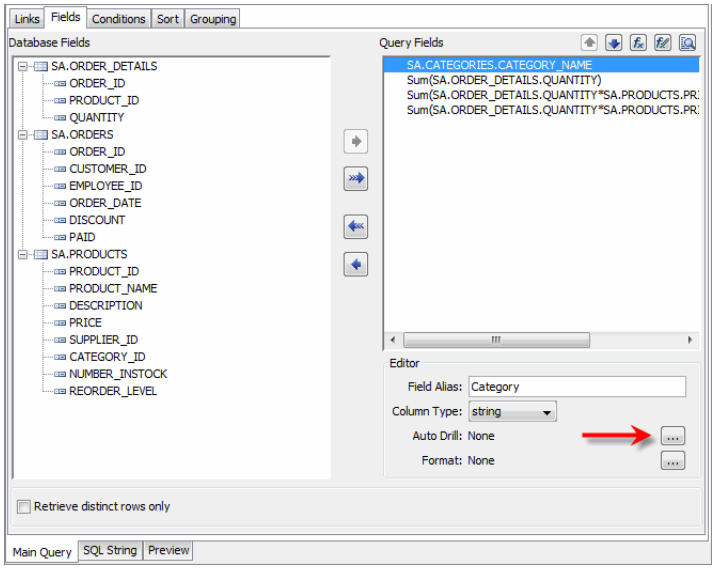
In the following example, you will add an auto-drill to the 'Sales by Category' query of the 'Orders' data source:

1. Open the **Asset** panel. Expand the 'Data Source' node and the 'Orders' data source.

2. Double-click the 'Sales by Category' query in the 'Sample Queries' folder of 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.

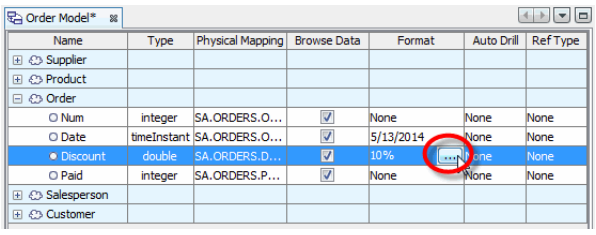
   The ‘Auto-Drill’ window that now opens is the same as the ‘Auto-Drill’ window for data models. For further instruction on using this ‘Auto-Drill’ window, see *Example 1: Passing column values in a drill* and *Example 2: Passing query-based values in drill*. After you add the auto-drill to this query 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.

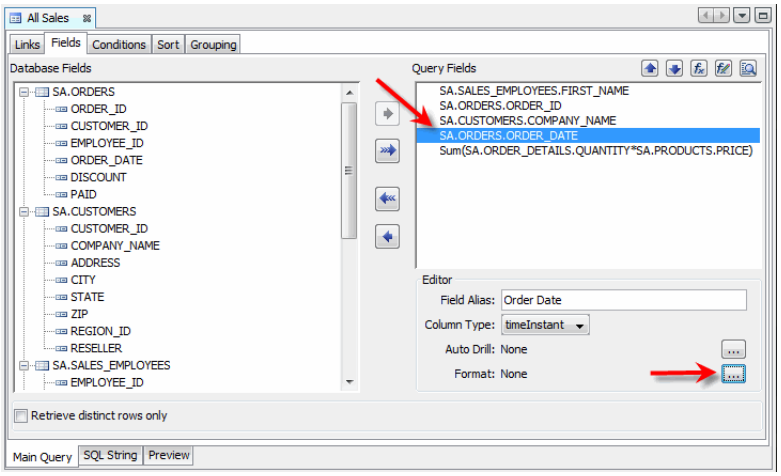
3.4.3 Adding a Format to a Data Field

To associate a data format with an attribute (column) at the model level or query level, follow the steps below:

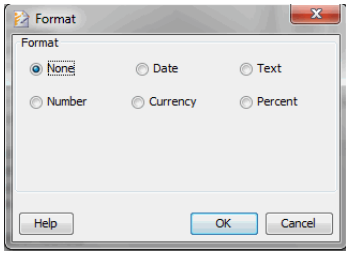
1. For a *data model*, open the model for editing, and click the button in the ‘Format’ column corresponding to the desired attribute.



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.

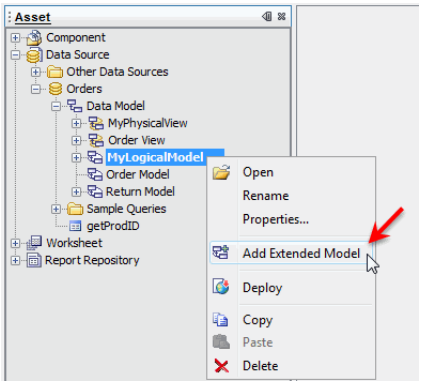
3.4.4 Extending a Logical Model

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:

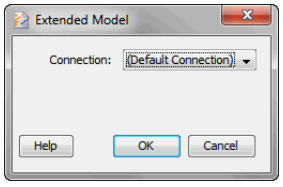
1. Open the **Asset** panel, and expand the 'Data Source' node.

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.

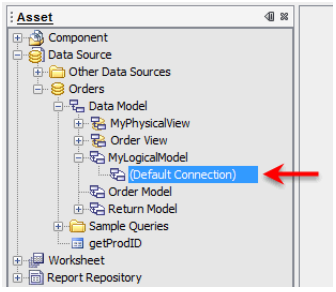


Note: An extended model is associated with one (and only one) connection and physical view. Likewise, a connection can be associated with only one extended model.

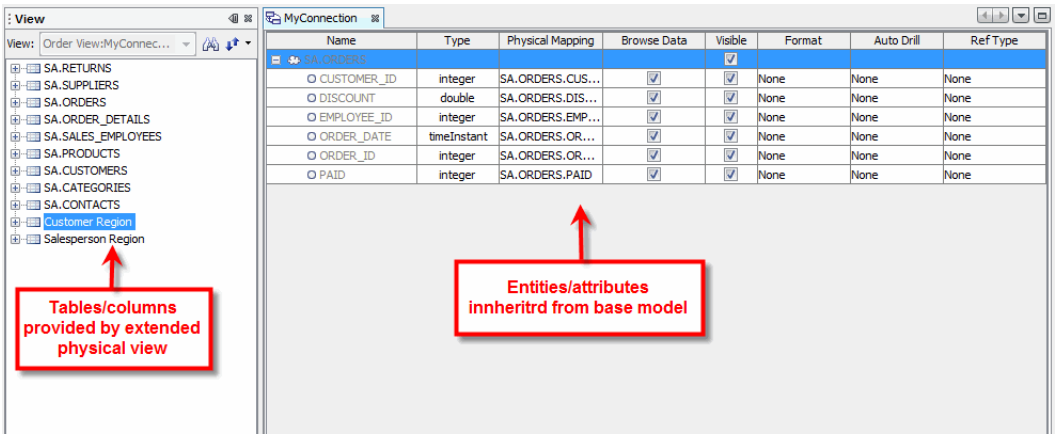
3. In the 'Extended Model' dialog box, choose a connection option for the new extended model, and press 'OK'. (You can select the default connection for the data source, or any of the additional connections.)



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.

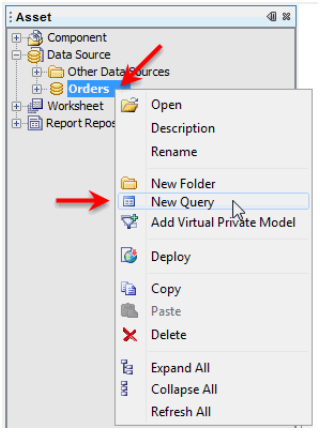
Queries and data models are independent of one another. However, you can transform and combine queries with data models for use in more sophisticated applications by using the Data Block technology provided by Data Worksheets. (See *Advanced Data Mashup* for more details.)

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.

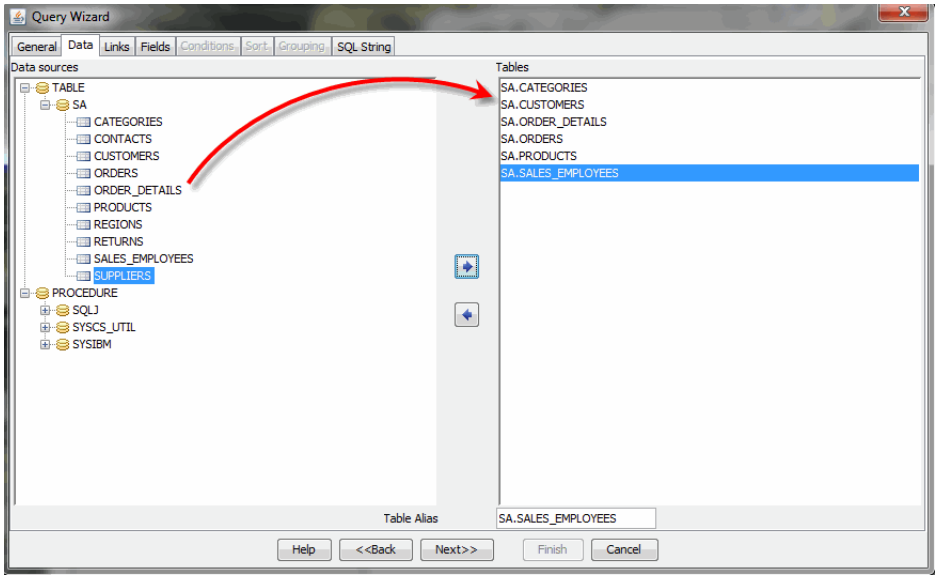
2. Right-click on the data source that you wish to query, and select ‘New Query’ from the context menu. This opens the ‘Query Wizard’ dialog box.



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.

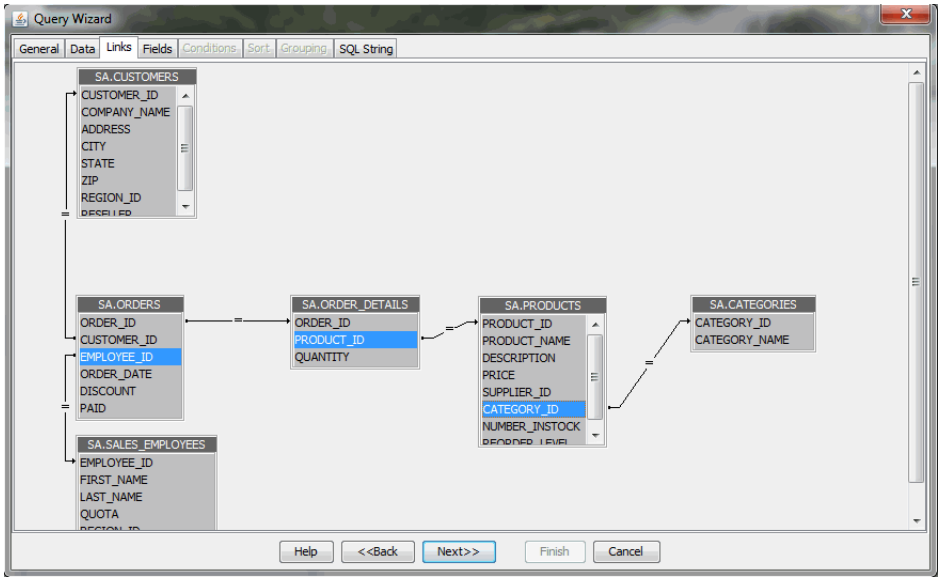
You can also double-click fields to add them, or use the arrow buttons.

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.



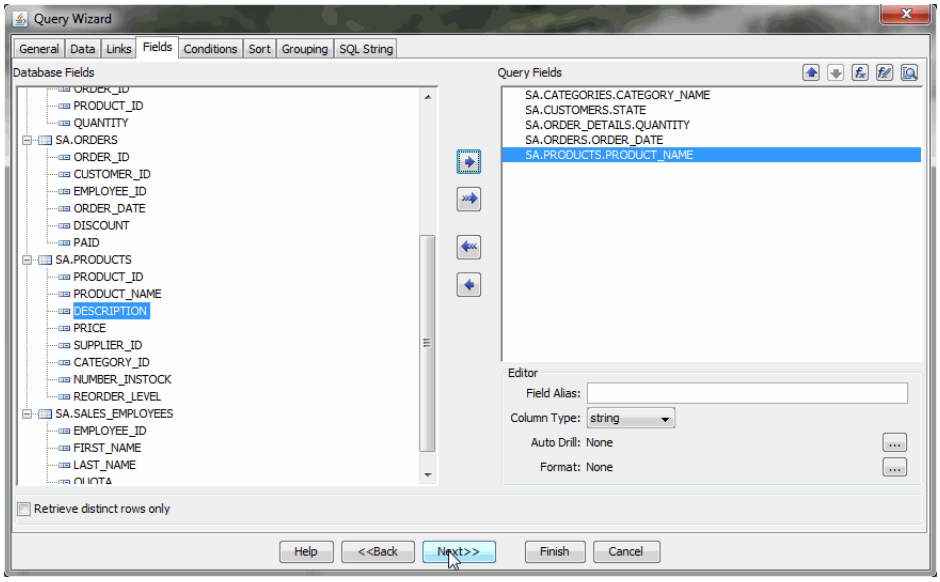
To manually enter SQL rather than using the Wizard, click the **SQL String** tab. See *Editing a Query SQL String*.

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.



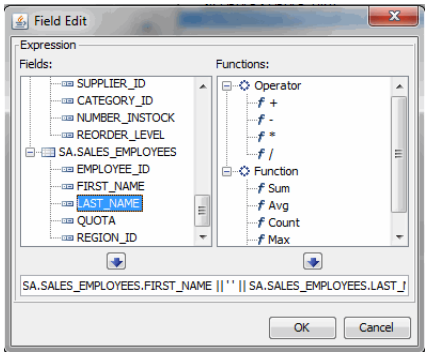
7. (Optional) To create a derived (expression) column, follow the steps below:



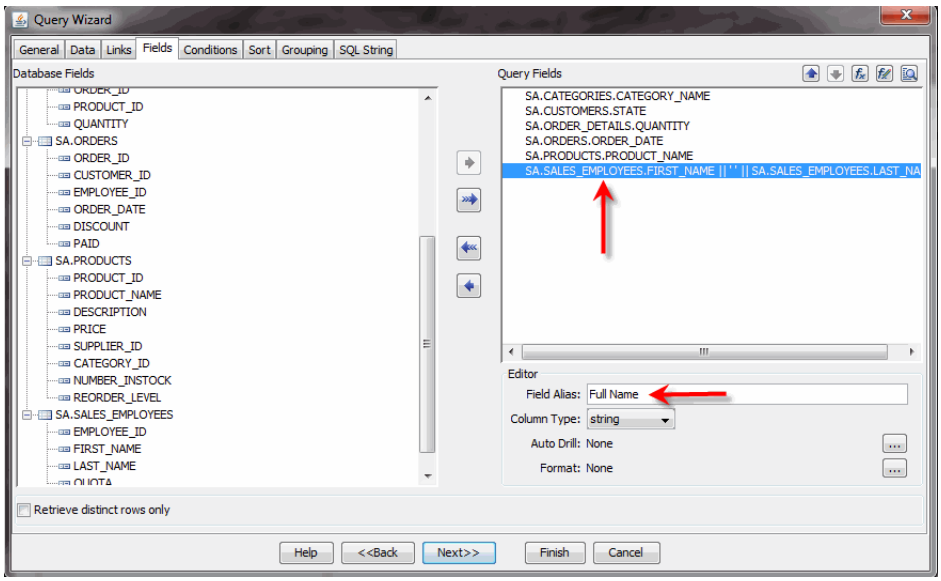
a. Click the 'Add Expression' button above the 'Query Fields' panel. This opens the 'Field Edit' dialog box.

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.)

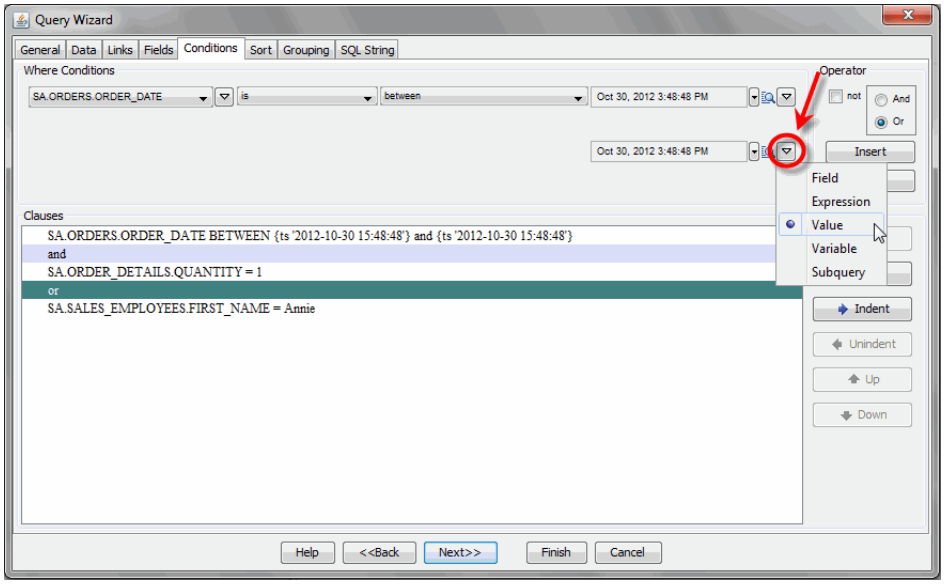


c. To create an alias for the expression column, select the expression column in the 'Query Fields' panel, and type a name into the 'Field Alias' box in the bottom panel.



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.



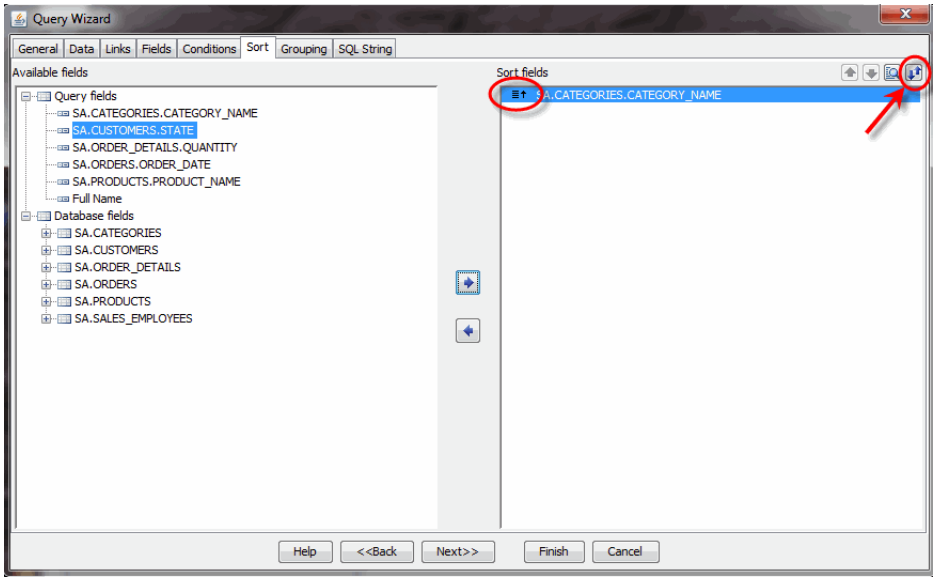
When you enter a fixed value as the right-side of the condition, use the 'Browse Data' button to assist the selection.

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

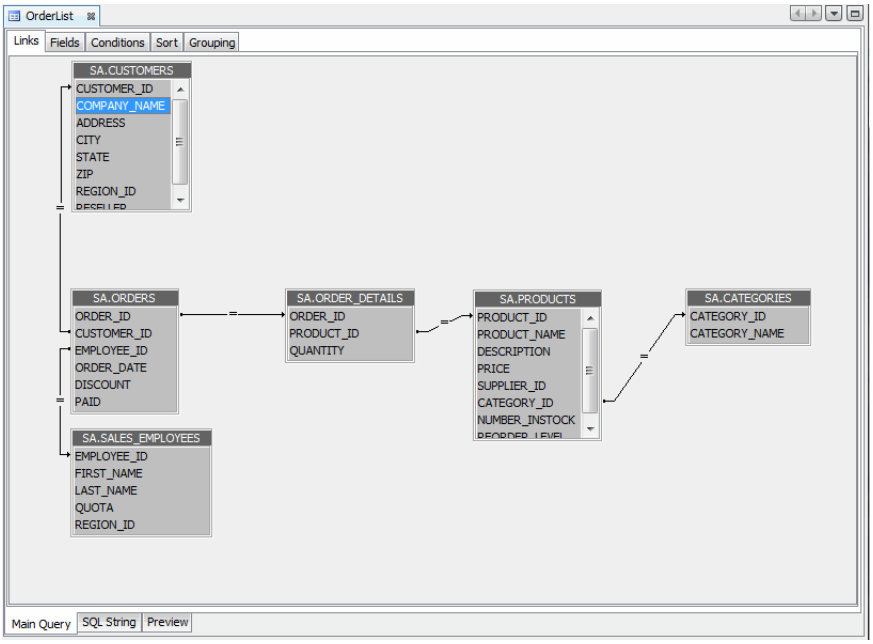
To edit an existing query, follow the steps below:

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.

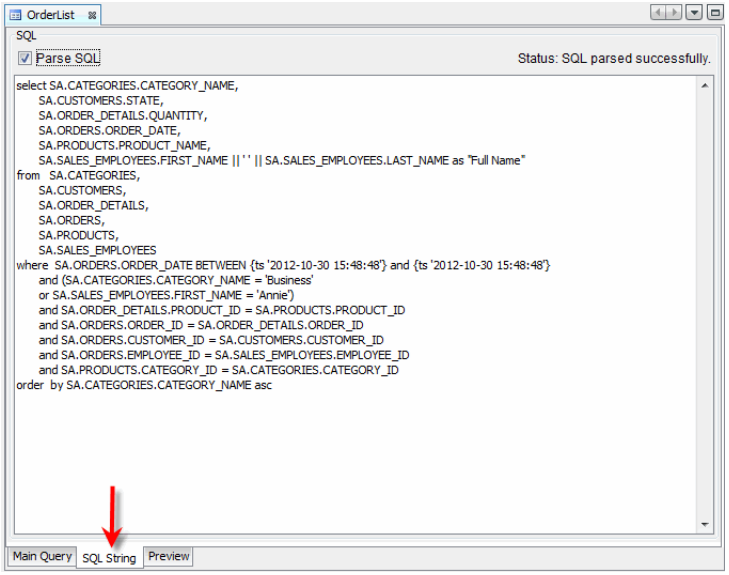
To view the results returned by the query, click the **Preview** tab at the bottom of the editor.

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 repository. 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.

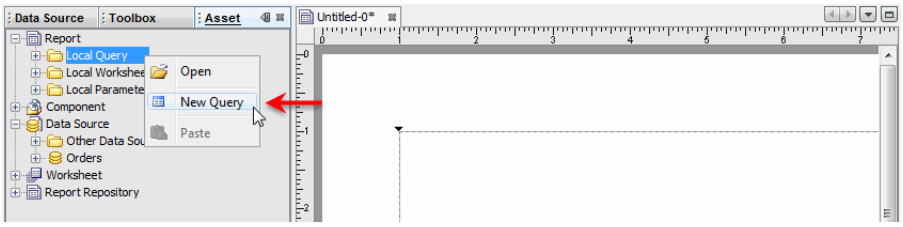
Creating a Local Query

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 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:

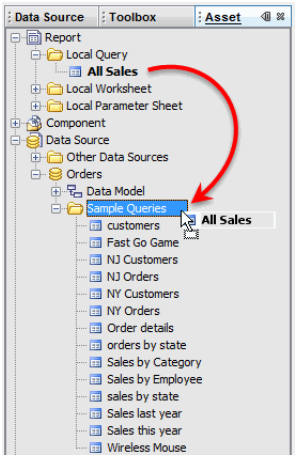
   a. Select the desired queries in the ‘Local Query’ folder, right-click, and choose ‘Copy’ from the context menu.

   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 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.
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.

Alternatively, drag-and-drop the query to the desired location while holding down the **Ctrl** key.

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

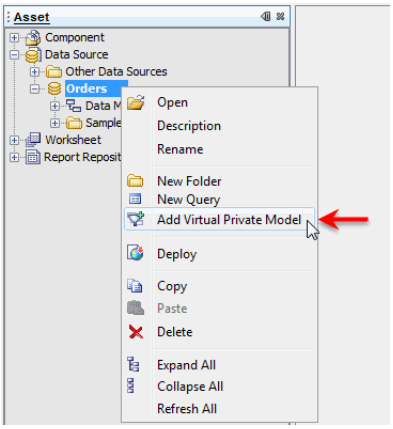
Design for Multi-Tenant Environment, in *Advanced Server Integration*, for connection-based access control.

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.

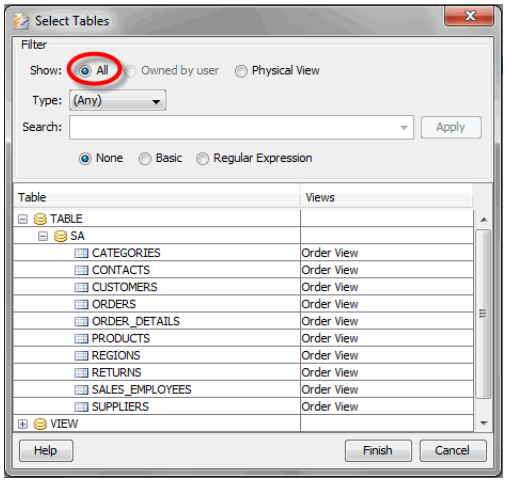
Previously defined conditions are listed in alphabetical order. (case insensitive).

5. Enter a name for the new condition in the ‘New Condition’ dialog box, and click ‘OK’.

6. In the ‘Define Conditions’ pane, click the ‘Choose Table’ button. This opens the ‘Select Table’ dialog box, showing a list of database tables.

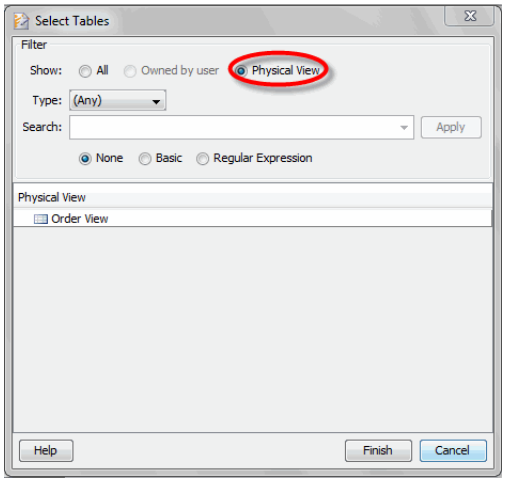
The dialog box provides controls for searching and sorting tables. See *Adding Database Tables to the Physical View* for details.

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.

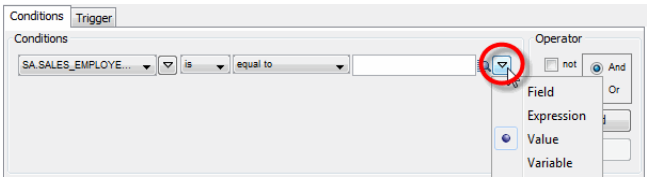


Note: If you have created one or more aliases of a table in the physical view, do not apply VPM conditions to the original table. (VPM conditions applied to the original table may affect its aliases in an unpredictable manner.) Rather, apply VPM conditions only to the aliased copies.

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.

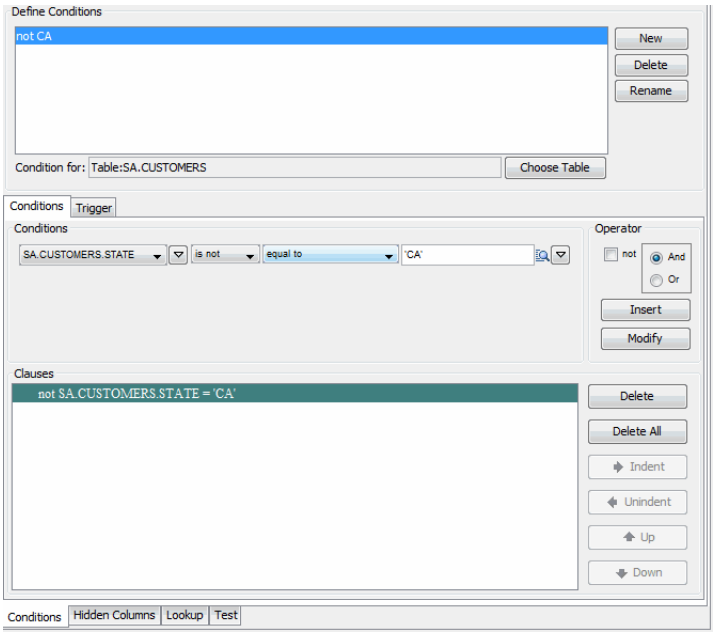
   'Field' allows you to select a different column from the same table or physical view. 'Expression' allows you to enter a regular expression using Perl5 regex syntax (see A.5 *Regular Expressions*.) 'Value' allows you to enter a fixed value. 'Variable' allows you to enter the name of a parameter to supply 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.

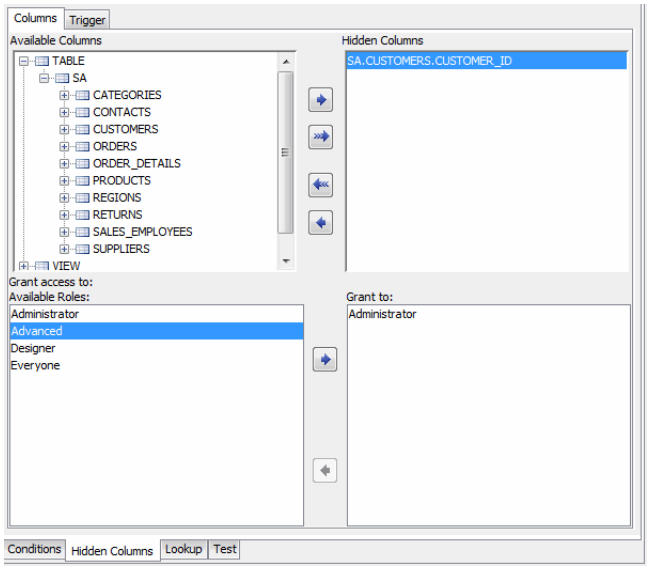
2. Expand the data source that contains the VPM you want to edit. Double-click the VPM to open it for editing.

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.

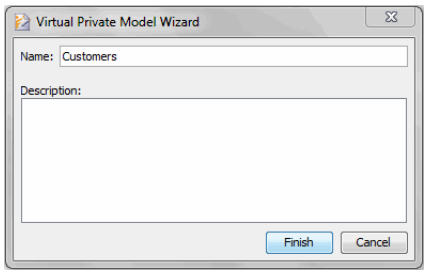
2. Create a new user “Eric,” and assign him the role of “CASalesDept”.

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.

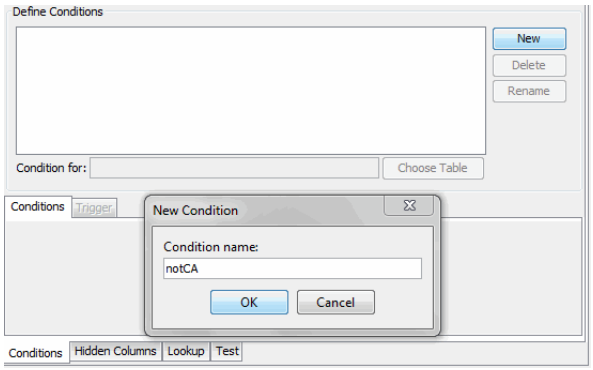


6. Click on the ‘Finish’ button. This opens the new VPM for editing.

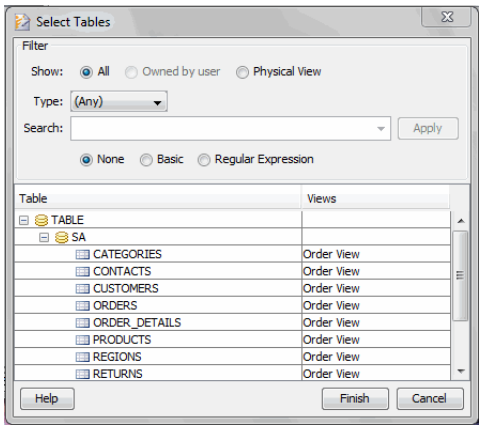
7. Click the **Conditions** tab at the bottom of the VPM editing panel.

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.



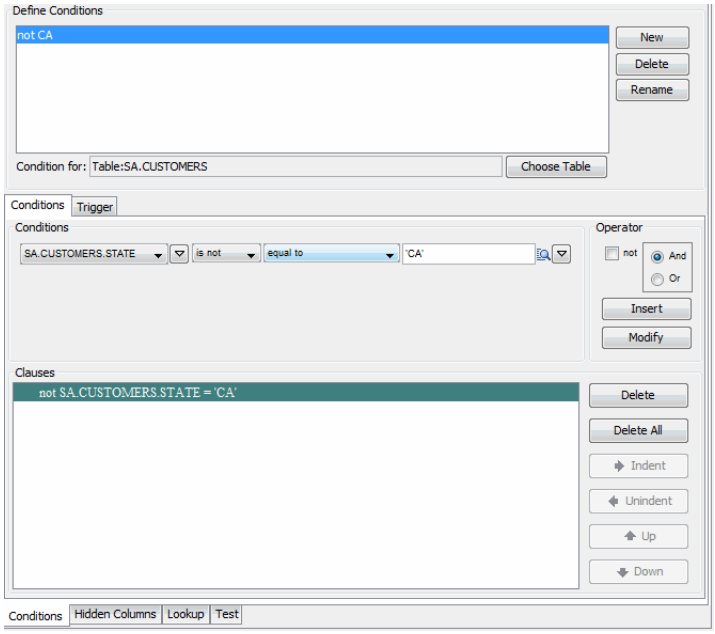
10. Click the ‘Choose Table’ button. This opens the ‘Select Table’ dialog box.



11. Select the ‘SA.CUSTOMERS’ table and click on the ‘Finish’ button.

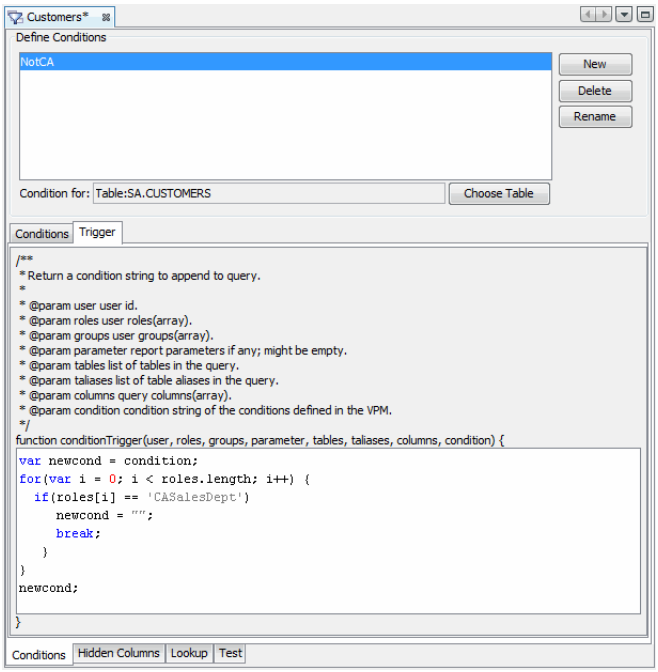
12. Specify the following condition and click the ‘Append’ button:

```
[SA.CUSTOMERS.STATE] [is not] [equal to] ['CA']
```



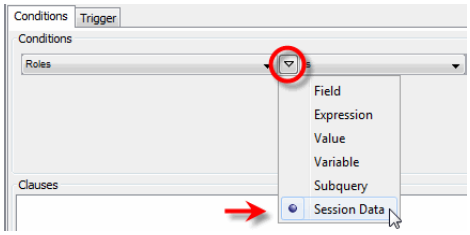
13. Click the **Trigger** tab and add the following script:

```
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'.

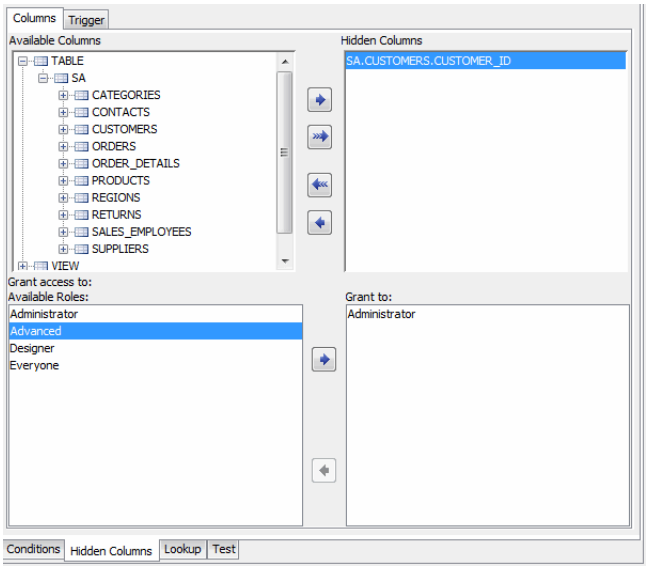
To achieve the same result *without* using a trigger script, join ("and") the following clause to the original condition: `[Roles] [is not] [equal to] ['CASalesDept']`. (To select the 'Roles' option in the left-menu, first choose the 'Session Data' option from the adjacent pop-up menu.)



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'.

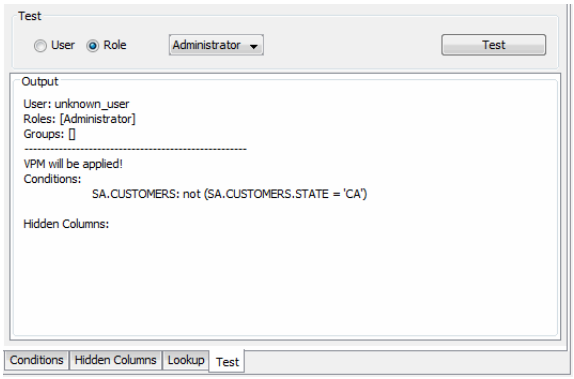
16. Select 'Administrator' from the list of 'Available Roles' under 'Grant access to'. Click the right-arrow button to grant permission to the 'SA.CUSTOMERS.CUSTOMER_ID' field to all users having role 'Administrator'.



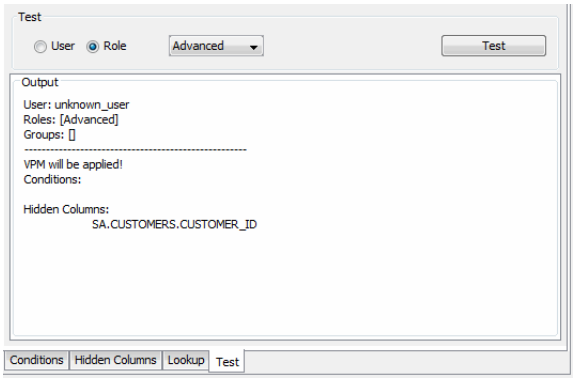
17. Click the **Lookup** tab (at bottom) and notice that the 'SA.CUSTOMERS' table is included in the list of lookup tables. All tables having conditions or hidden columns will appear in this list.

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.

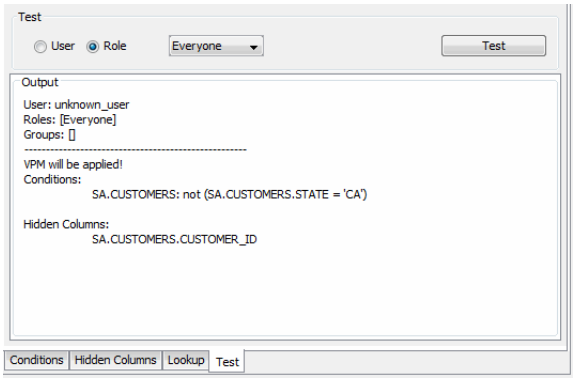


20. Now select 'CASalesDept' from the list of roles and click on the 'Test' button. Users belonging to this role do not have the 'CUSTOMERS.STATE' condition applied to their query. This is because of the condition trigger script. However, the 'CUSTOMERS.CUSTOMER_ID' column is hidden from all of these 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,

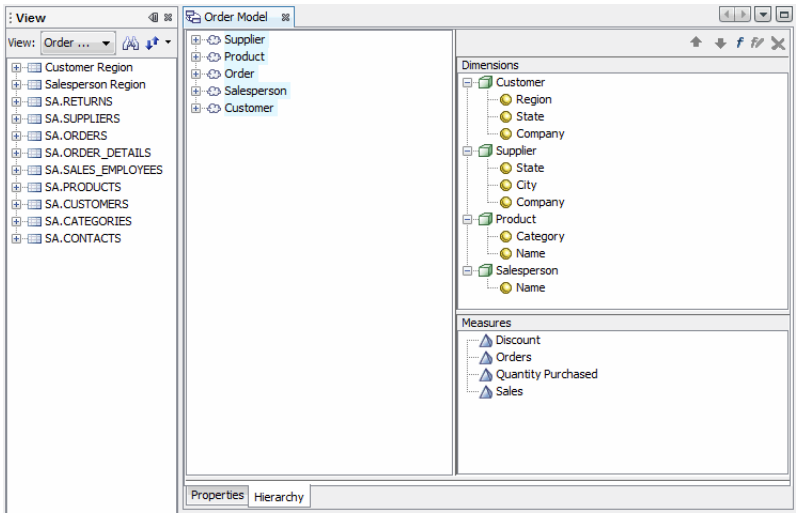
‘order sale amount’ fits this definition because adding all ‘order sale amounts’ will give the ‘sales total’. Bank account ‘daily balance’, on the other hand, is not additive because adding two days of a balance does not provide meaningful information. Therefore, account balance is considered semi-additive, 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.

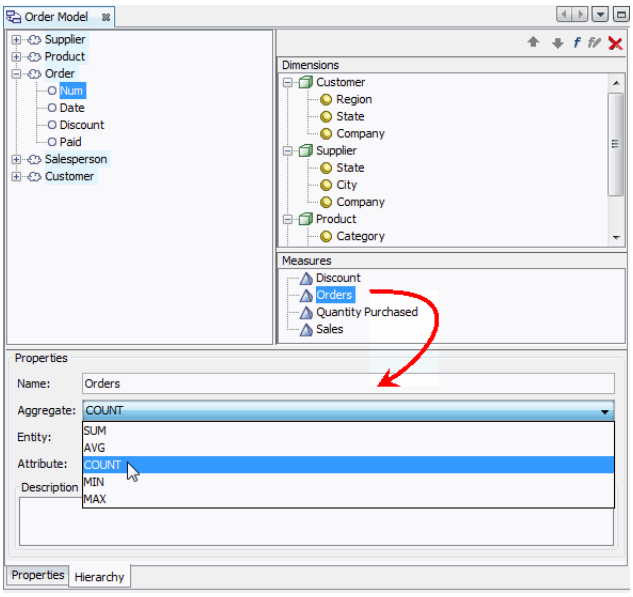
To create new date components of this kind, you can define expression attributes in the **Model** tab. These expression attributes will be available for use in the hierarchical overlay. For example, to create a new 'Month' attribute in which June is "calendar month 1," you could define a new expression 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,

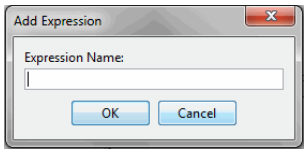
consider the case where you have the hierarchically-related dimensions ‘Month’ and ‘Day’, and a measure ‘Cost’. If you aggregate the measure by ‘AVG’, then the value of the measure when viewed along the ‘Month’ dimension will be “average cost per month,” while the value of the measure when viewed along 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

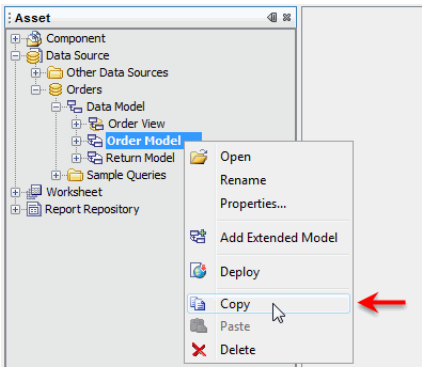
Walkthrough

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’.

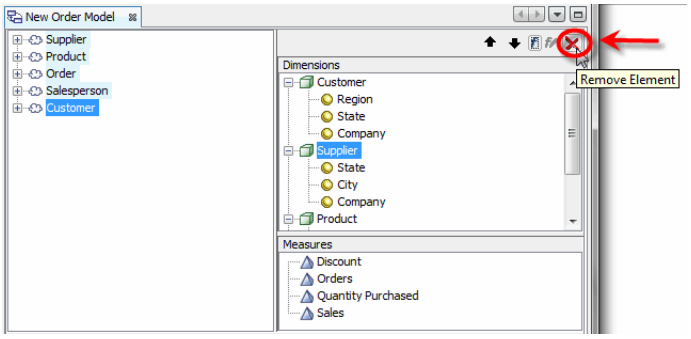
5. Right-click this copy and select ‘Rename’. Rename the copy ‘New Order Model’ and click ‘OK’.

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.

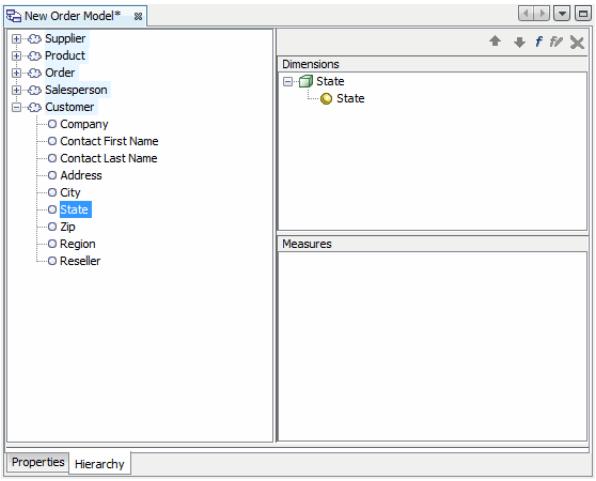
You will now specify dimensions and measures for the new hierarchical overlay. The four dimensions are as follows:

- ‘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’.

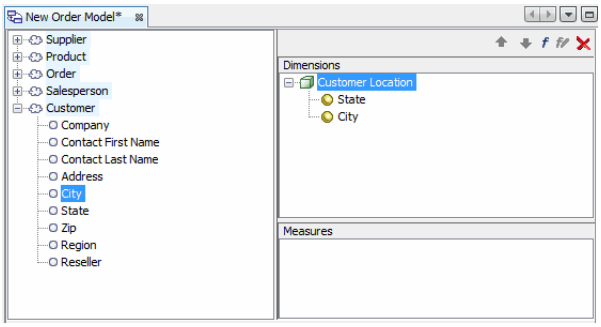
First you will create the dimensional hierarchy representing customer location. This hierarchy will be composed of ‘Customer State’ and ‘Customer City’:

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.



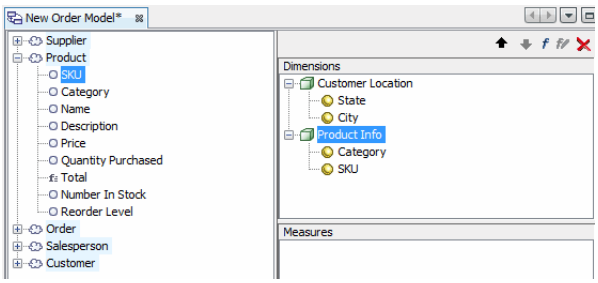
The ‘City’ attribute should now appear as a dimension in the ‘Customer Location’ hierarchy. Make sure the dimension order is correct: Finer-granularity dimensions should appear below the corresponding courser-granularity dimensions; thus, ‘City’ should appear below ‘State’. If the ordering is wrong, 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’.

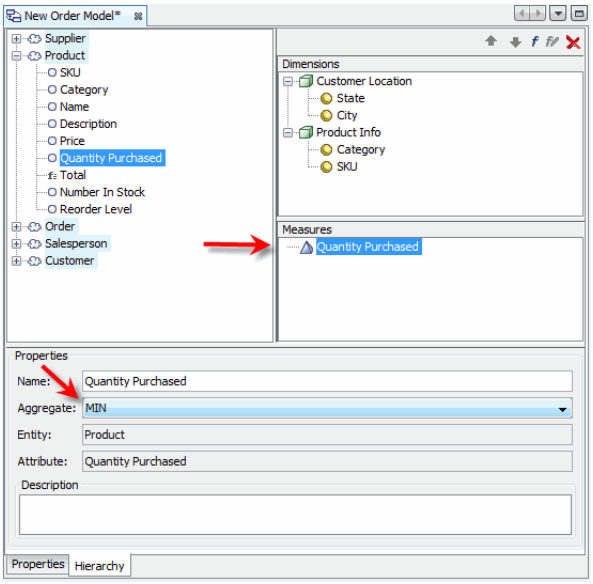
Now you will add two measures. The first measure will be the quantity of product purchased, and the second measure will be the dollar amount of discounts applied. Because this second measure is not available as an

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.

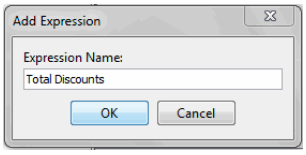
17. From the ‘Aggregate’ menu, select the ‘MIN’ option. This will compute the minimum quantity purchased for each level of the dimension.





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.

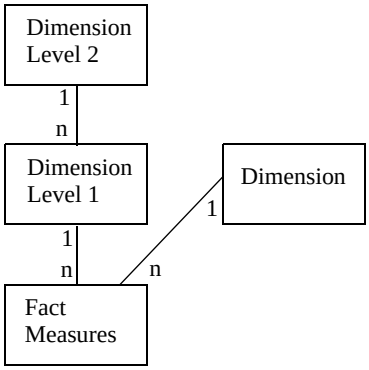
23. From the ‘Aggregate’ menu in the ‘Properties’ panel, select the ‘SUM’ option. This will compute the summed discount in dollars for each level of dimension granularity.

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.

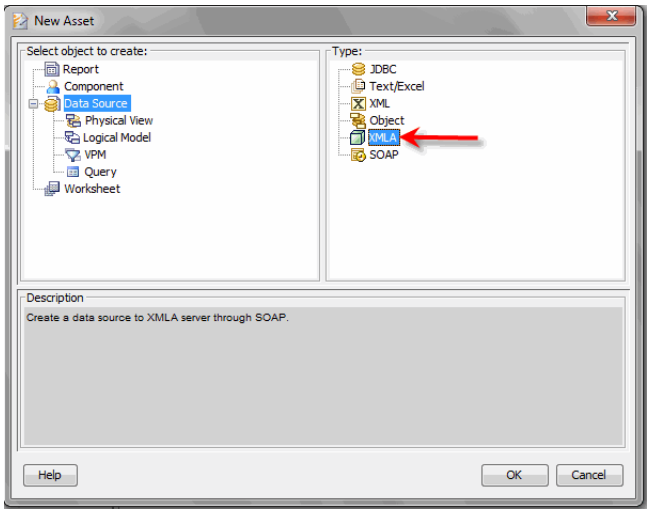
```
Recommendation: Within a given level of a dimension, do not create
multiple entries with the same name. This creates an ambiguity for
Viewsheet selections.
```

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.

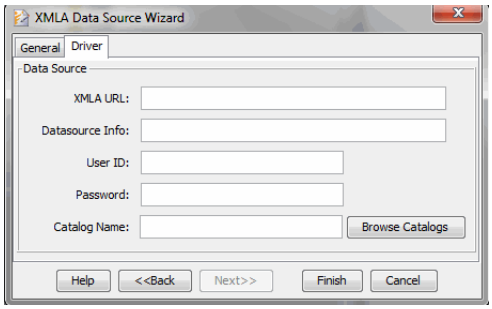
2. In the left panel, select the ‘Data Source’ node. In the right ‘Type’ panel, select ‘XMLA’ and press ‘OK’.



   This opens the ‘XMLA Data Source Wizard’ dialog box.

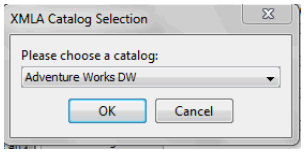
3. In the ‘XMLA Data Source Wizard’ dialog box, select the **General** tab and enter the data source ‘Name’ and ‘Description’.

4. Select the **Driver** tab and enter the 'XMLA URL'.

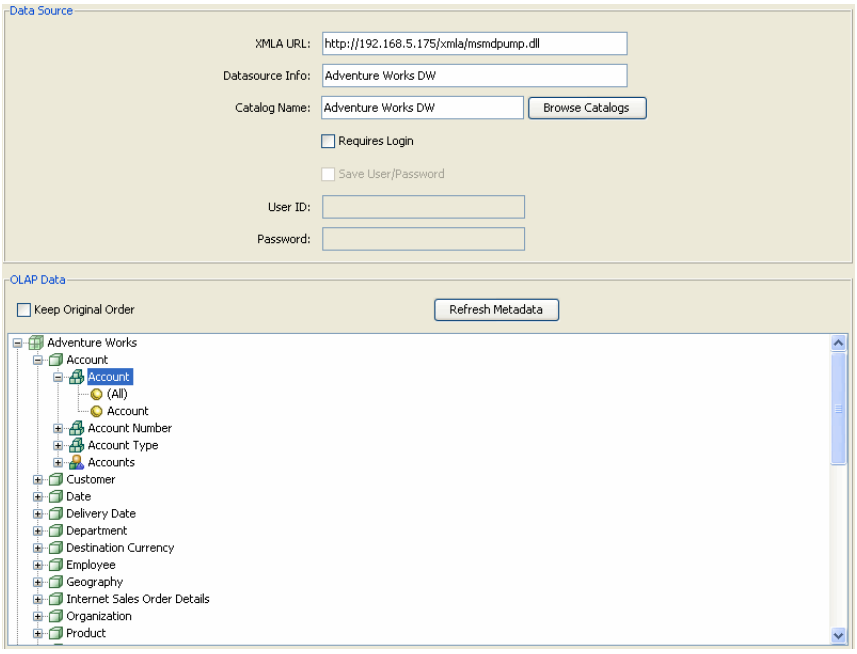


XMLA is used to communicate with the OLAP server. The URL is the path to the XMLA service; for example, the URL for SQL Server is the file *msxisapi.dll*, which is in the *Isapi* directory of the XMLA installation.

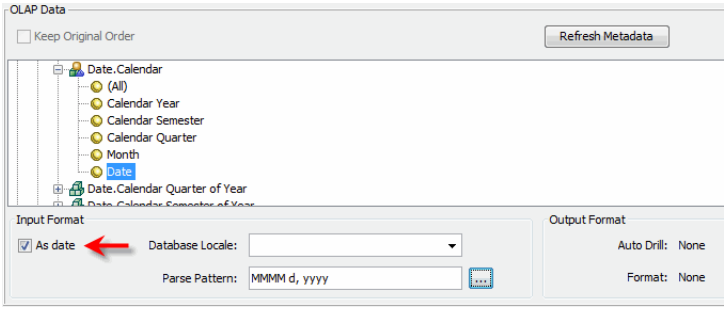
5. Press the 'Browse Catalogs' button and select the database catalog name from the menu. Press 'OK'.



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.



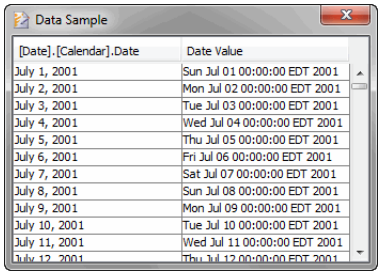
7. (Optional) For a date dimension, select a desired level of the date dimension in the tree, and enable the 'As Date' option. In the 'Parse Pattern' field, enter the date format corresponding to the date values stored in the data source.



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 Oracle Hyperion and Hyperion Essbase

To create a datasource for Hyperion Essbase cubes, follow the procedure described in *Generic Multidimensional Data Source*.

- The 'XMLA URL' should have the following form:

```
http://{host}:{port}/aps/XMLA
```

- The 'Datasource Info' field should specify the following:

```
Provider=Essbase; Data Source={host}
```

4.4 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.

4.4.1 Defining an XML Data Source

The following example illustrates how to configure an XML data source.

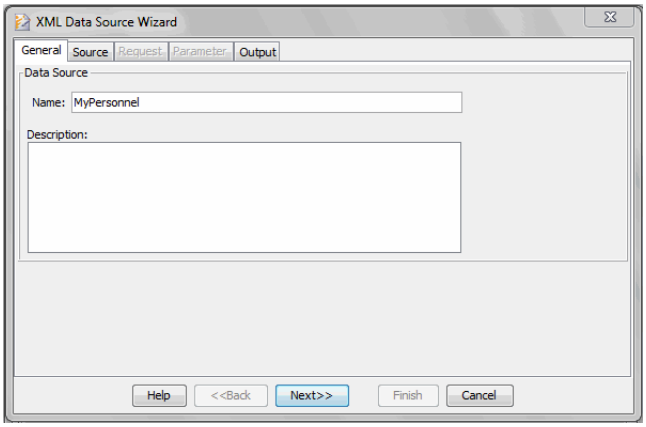
Walkthrough

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’.

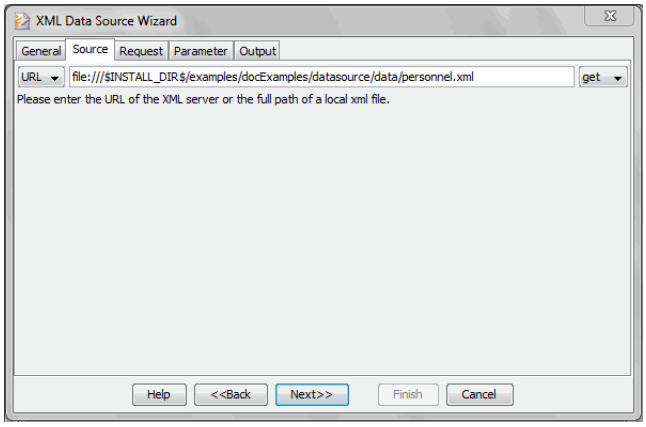


4. Select the **Source** tab. Enter a data source URL or file path. (For this example enter “`{Installation}/examples/docExamples/datasource/data/personnel.xml`”, where `{Installation}` is the Style Intelligence installation directory, for example “`C:\StyleIntelligence`”.)

   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.

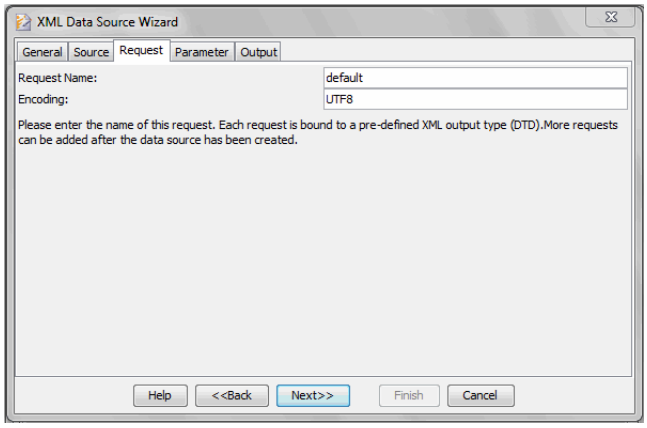
5. Select the HTTP method. If the URL does not use HTTP, select the GET method.



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.

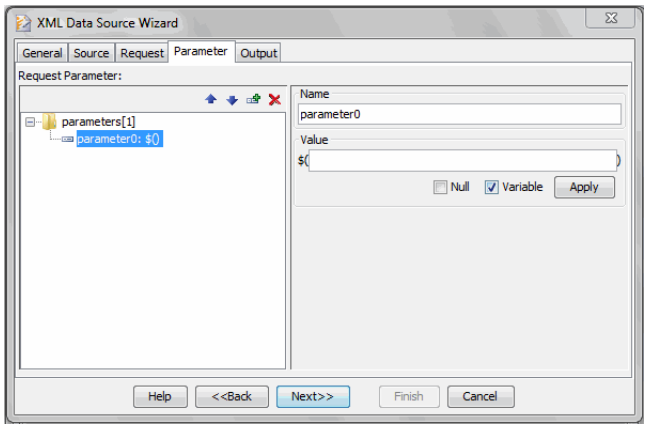


8. Select the **Parameter** tab.

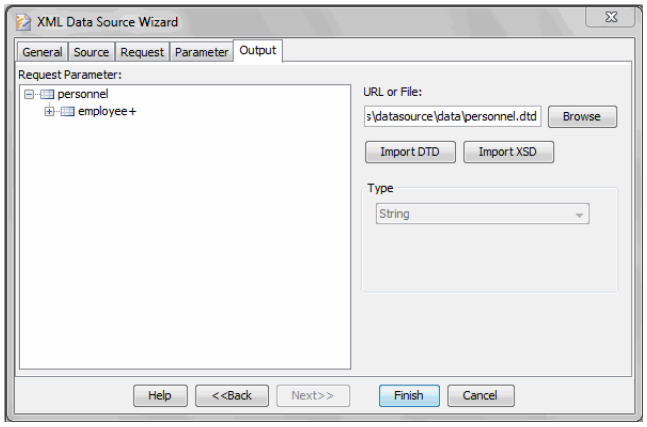
   The **Parameter** tab shows the current parameters for the request. For HTTP requests, the parameters are ordered in a parameter tree. The root of the tree is the 'Parameters' node. The child of the root is a sequence node, with one or more pairs of parameters. Each pair consists of a name and a value 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

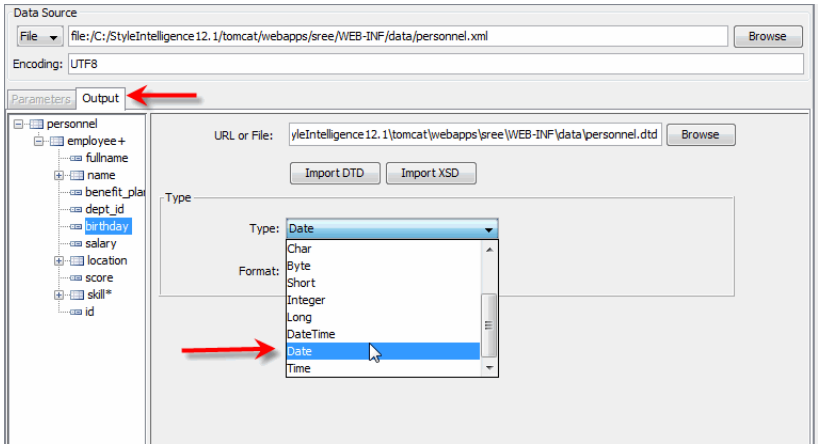
See *XML Element Types* for more information about the types.

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.

4.4.2 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 '`#, ###, ###. ##`'.

4.5 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.

4.5.1 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*.)

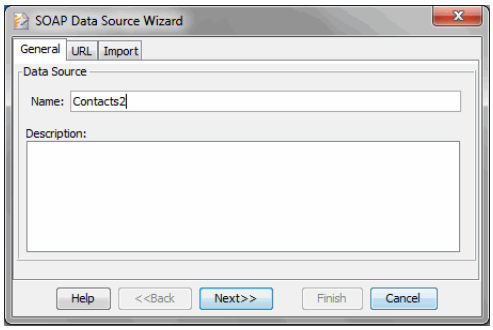
Then follow the steps below:



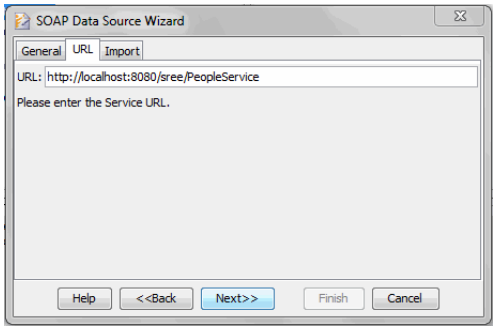
1. Press the 'New' button in the Style Studio toolbar. This opens the 'New Asset' dialog box.

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'.

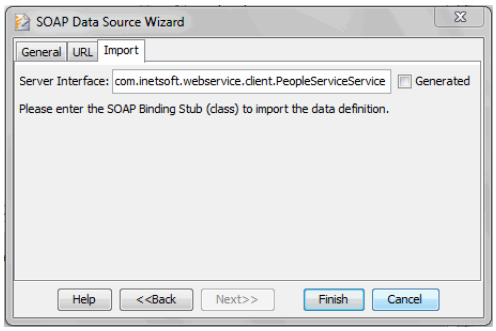
3. Select the **General** tab. In the 'Name' field, enter "Contacts2".



4. Select the **URL** tab. In the 'URL' field, enter the service URL: '`http://localhost:8080/sree/PeopleService`'

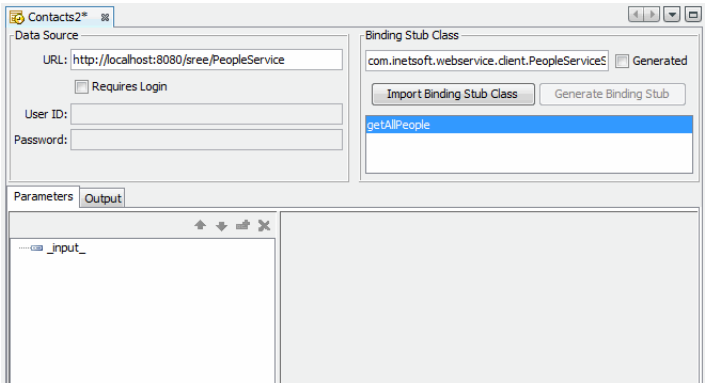


5. Select the **Import** tab. In the 'Server Interface' field, specify the client stub class (Soap Binding Stub): '`com.inetsoft.webservice.client.PeopleServiceService`'



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.





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.

4.5.2 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 -d 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.

4.6 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.

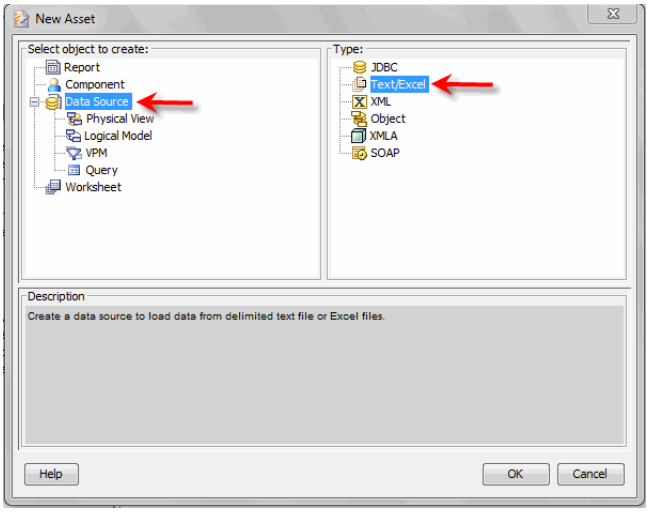
4.6.1 Defining a Text Data Source

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.

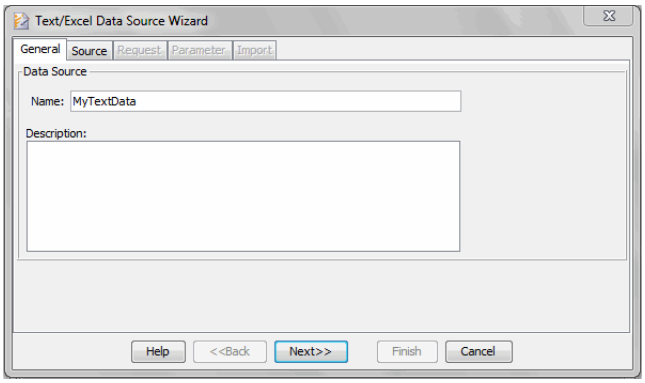
To create a Text data source, follow the steps below:

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.
```

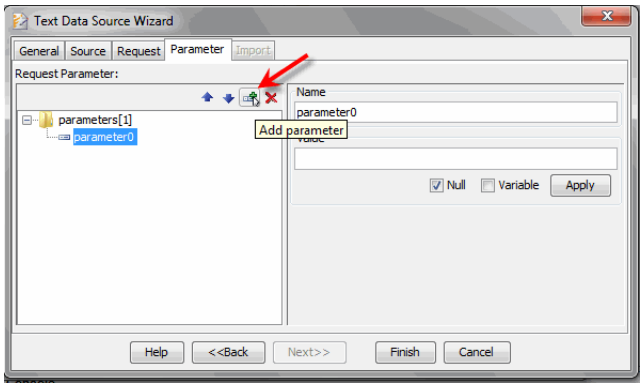
5. (Optional) Select the **Request** tab, and enter the name of the request. In a servlet- or CGI-based URL, each request corresponds to a parameter set for the HTTP request.

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.

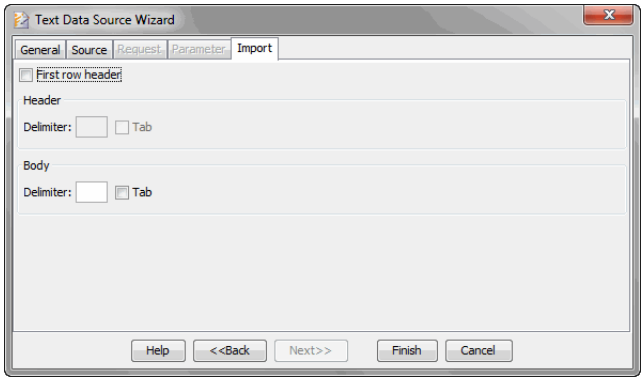


   c. To specify a null value for the parameter, select the ‘Null’ option.

   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’.

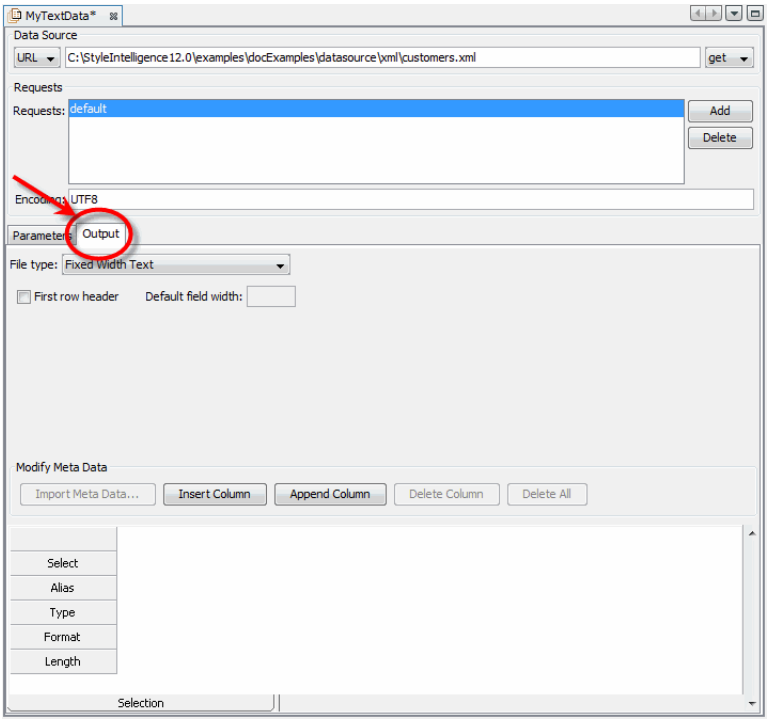
b. In the ‘Header’ panel, enter the character used as the header delimiter, or select ‘Tab’ for a tab-delimited header.

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.

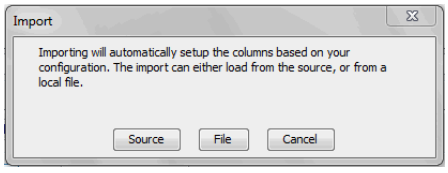
13. If you selected one of the ‘Microsoft’ formats, follow the steps below:

    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.

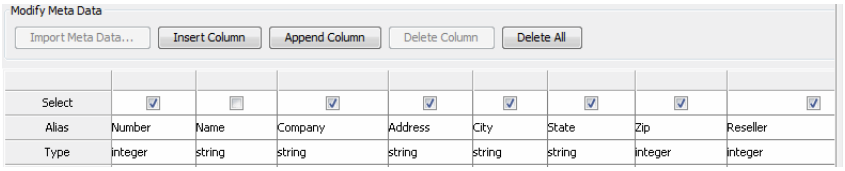
14. If you selected the ‘First header row’ option for any of the available formats, press the ‘Import’ button to import the column header meta-data. This opens the ‘Import’ dialog box.



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*.

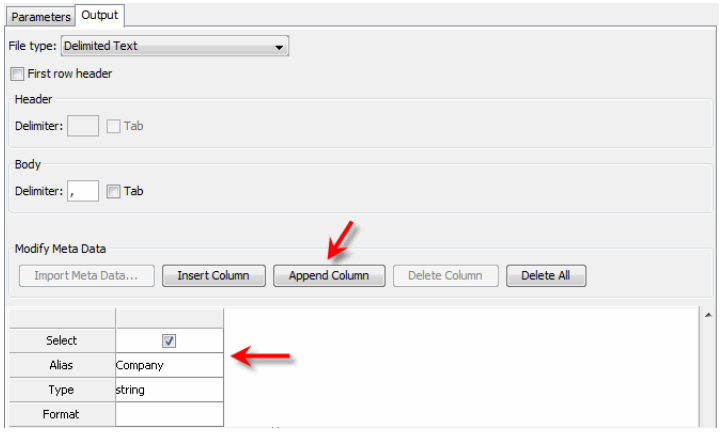
16. Press the 'Save' button in the Style Studio toolbar to save the new Text data source.

4.6.2 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\customers.txt* 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”.

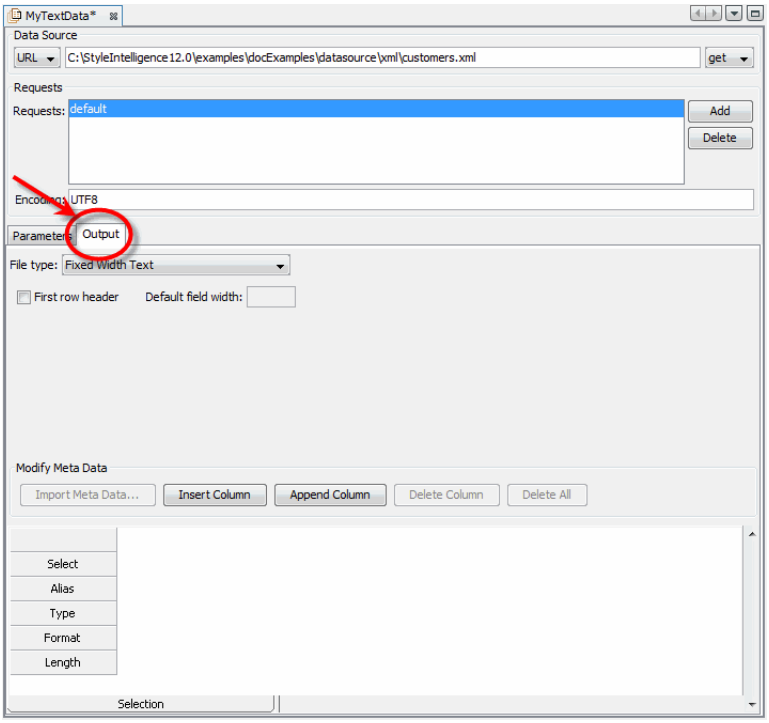
4. Select the **Source** tab. Press the ‘Browse’ button next to the ‘URL or File’ field, and select the following text file:

   `{StyleIntelligence}\examples\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.

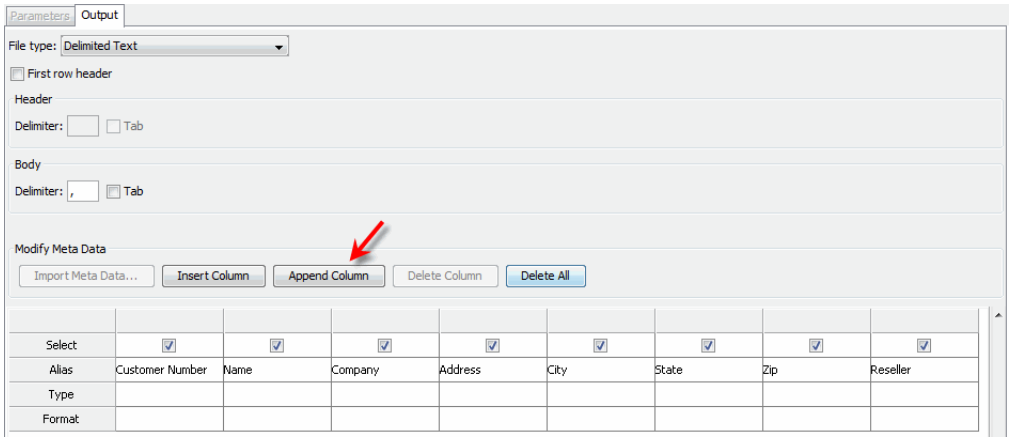
9. For the Body ‘Delimiter’, use the default comma delimiter.

For files with a header row, use the ‘Import’ feature to help define columns. See *Defining a Text Data Source*.

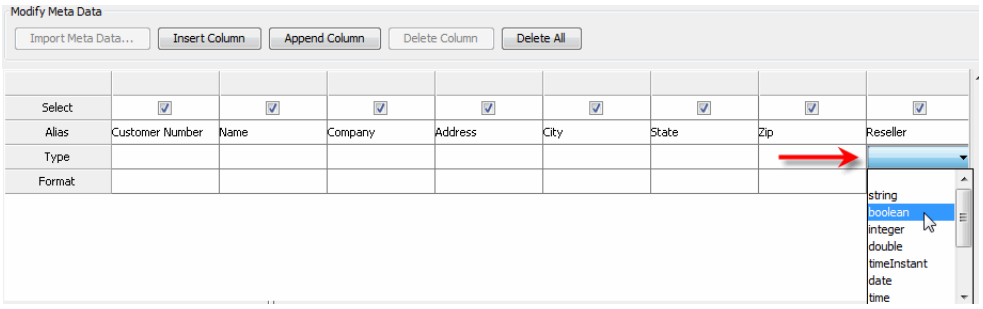
10. Press ‘Append Column’ to add a column to the table specification.

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.

¢

4.7 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.

4.7.1 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.

4.7.2 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.

4.7.3 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

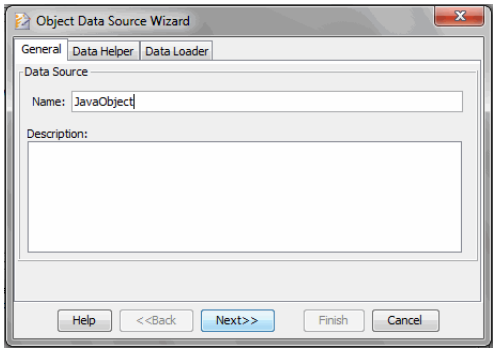
The parent directory of this '`object`' directory must be included in the classpath.

Follow the steps below to configure the data source.

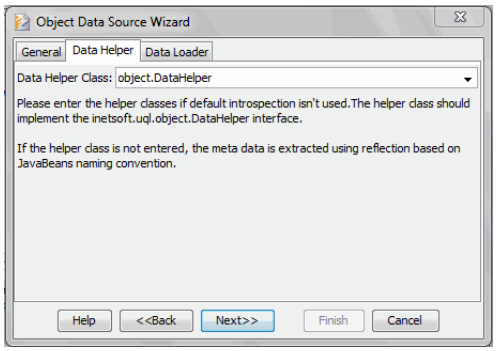
1. Press the 'New' button in the Style Studio toolbar. This opens the 'New Asset' dialog box.

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.

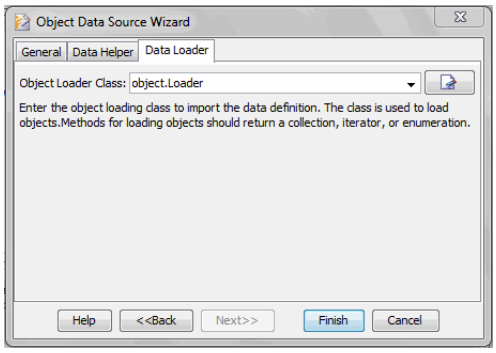
3. Select the **General** tab. In the 'Name' field, enter a name for the data source name, for example: "JavaObject".



4. Select the **Data Helper** tab. In the 'Data Helper Class' field, enter the helper class name (e.g., '`object.DataHelper`').

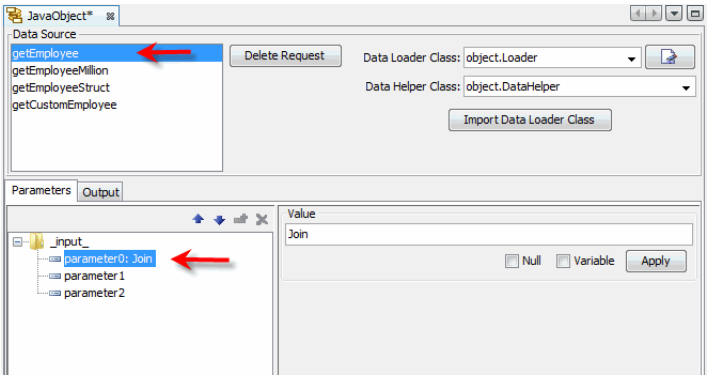


5. Select the **Data Loader** tab. In the 'Object Loader Class' field, enter the loader class name (e.g., '`object.Loader`').



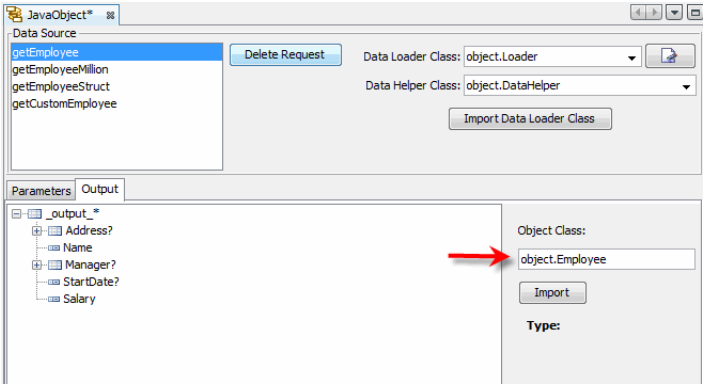
6. Press 'Finish' to create the data source.

7. In the 'Data Source' panel, select the desired request (e.g., 'getEmployee'). Select the desired parameter, and enter the desired parameter value in the 'Value' field. Press 'Apply'.



8. Repeat the above step to provide values for any other parameters.

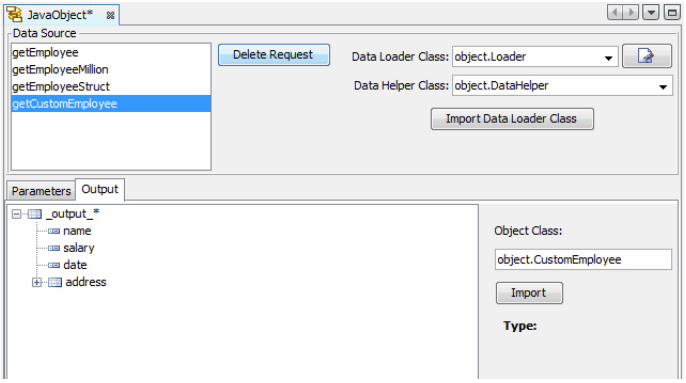
9. Select the **Output** tab. Type a class name (e.g., '`object.Employee`') in the right-side text field and press 'Import'.



10. In the 'Data Source' panel, select the request (e.g., 'getEmployeeStruct').

11. Select the **Output** tab, enter the class name (e.g., '`object.Employee`') in the right-side text field, and press 'Import'.

12. Repeat the above steps for any additional requests.





13. 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.

4.8 SAP Data Source

To connect to an SAP data source (including SAP ERP, SAP NetWeaver, SAP NetWeaver Business Warehouse), follow the steps below:

1. Verify that you have the .NET 1.1 runtime installed.

2. Install the SAP client tools on your machine. This will also install any required DLLs.

```
Contact your Account Manager to obtain the required inetsoft-
sap.jar file.
```

3. Download and copy the InetSoft JCo package (`inetsoft-sap.jar`) to a new directory, e.g., *C:\SAPJCo*.

4. Contact your SAP administrator to obtain the SAP JAR file (which may be named `sapjco.jar`).

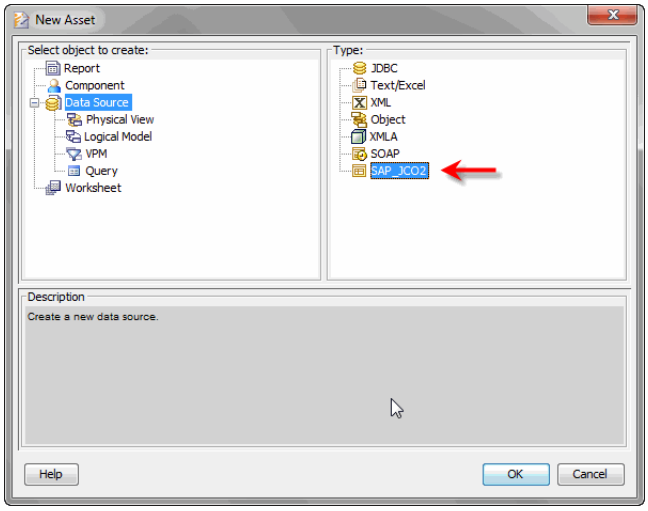
5. Add the `sapjco.jar` file (or equivalent) and the `inetsoft-sap.jar` file to your Style Studio classpath. (See *Configuring the Style Studio Classpath*.)

6. Place a copy of the `sapjco.jar` file and the `inetsoft-sap.jar` file into the *webapps\sree\WEB-INF\lib* directory.

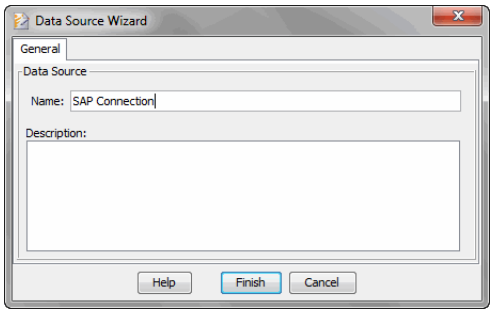
Once you have completed the above configuration, create the SAP data source by following the steps below:

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 ‘SAP_JCO2’, and press ‘OK’.

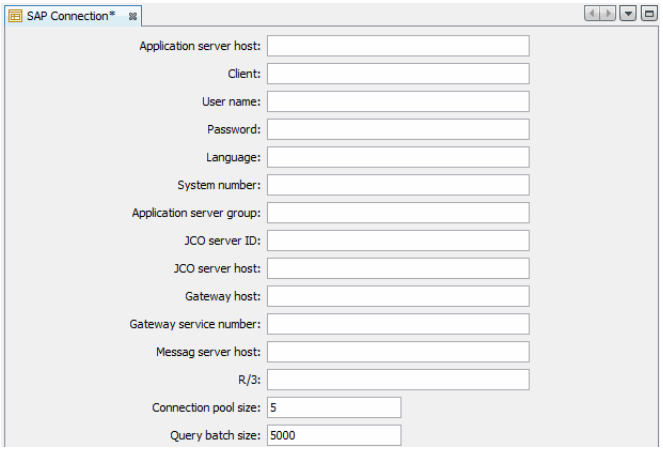


3. In the Data Source Wizard, enter a ‘Name’ and optional ‘Description’ for the data source. Press ‘Finish’.



This opens the SAP data source screen.

4. Enter the appropriate connection information appropriate to your SAP installation. See the SAP documentation for further information about the connection parameters.



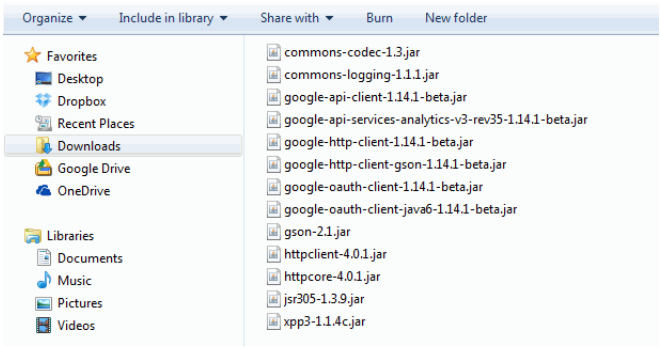
5. Press the ‘Save’ button in the toolbar to save the data source.

See Also
SAP Hana, for information on connecting to SAP Hana.

4.9 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.

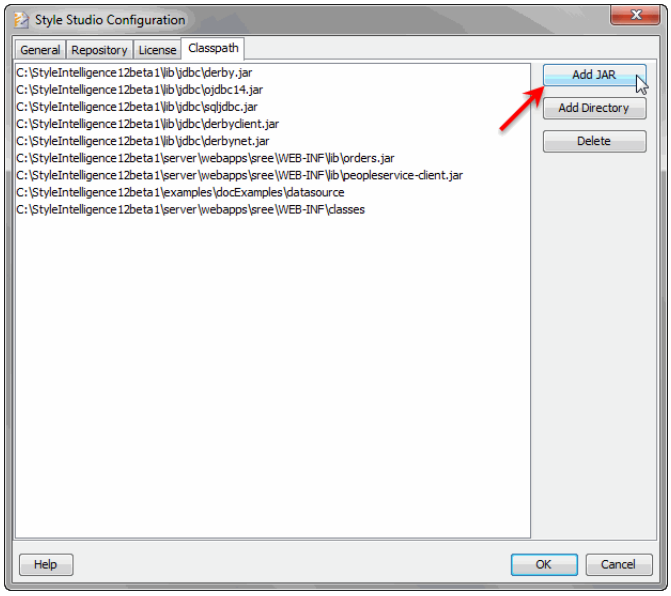


2. Extract all JAR files into the *sree/WEB-INF/lib* directory of the Style Intelligence server installation.

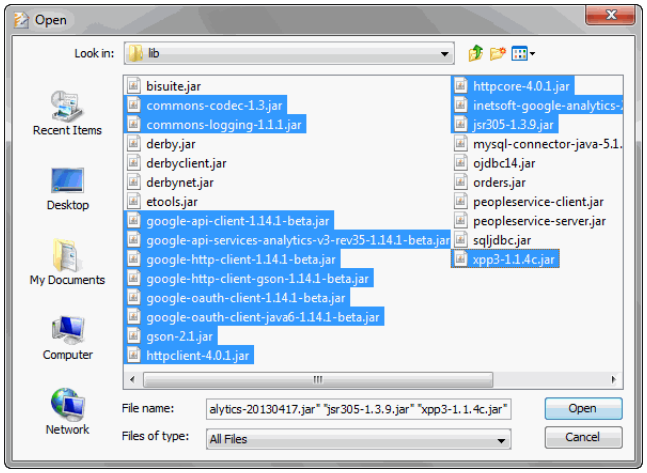
3. Launch Style Studio, and select ‘Configure’ from the ‘File’ menu. This opens the ‘Style Studio Configuration’ dialog box.

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.)



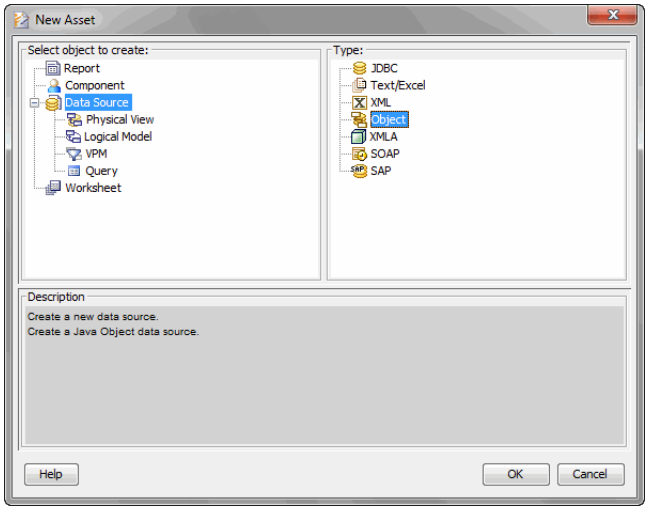
6. Navigate to the location where you copied the JAR files (*sree/WEB-INF/lib*). Select all of the extracted JAR files and press 'Open'. This adds the files to the classpath.



7. Press 'OK' to close the 'Style Studio Configuration' dialog box.

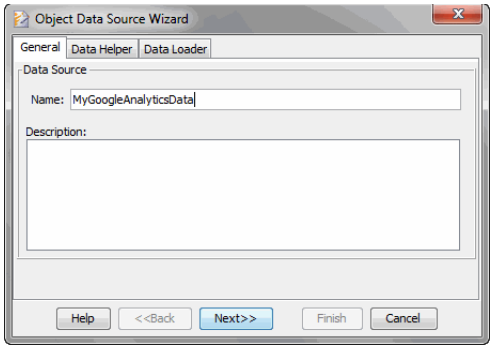
8. Select 'New' from the Style Studio 'File' menu. This opens the 'New Asset' dialog box.

9. In the left panel, select 'Data Source'. In the right panel, select 'Object'.



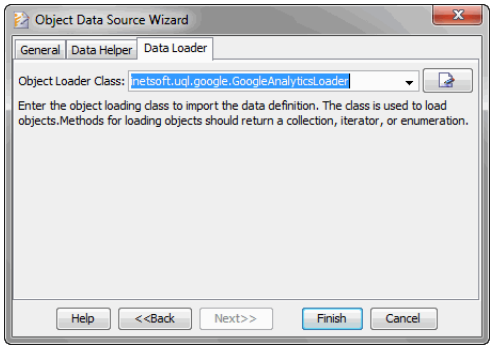
10. Press ‘OK’. This opens the Object Data Source Wizard.

11. Enter a ‘Name’ for the new data source.

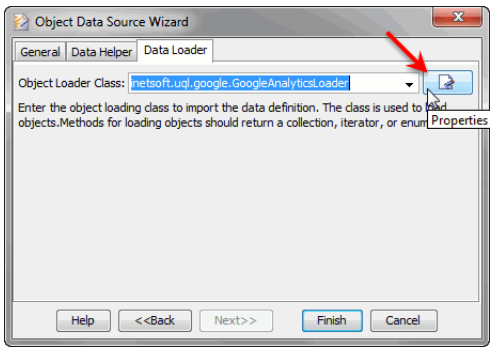


`Note: You do not need to provide any information in the` **`Data Helper`** `tab.`

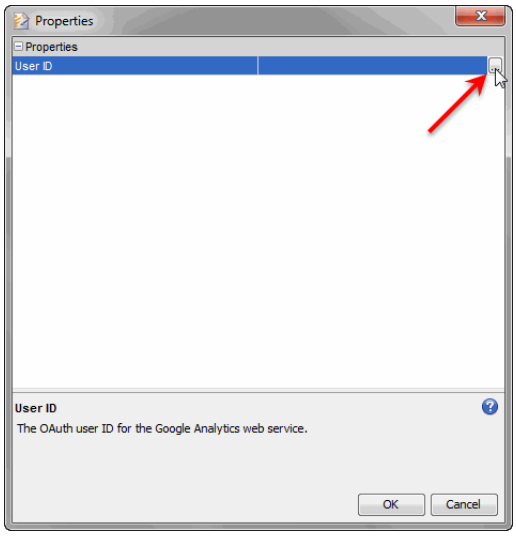
12. Click on the **Data Loader** tab. In the ‘Object Loader Class’ menu, select the `inetsoft.uql.google.GoogleAnalyticsLoader` class.



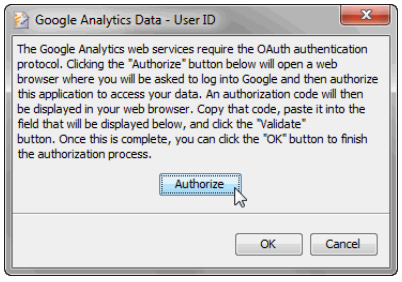
13. Press the ‘Properties’ button (next to the ‘Object Loader Class’ menu). This opens the ‘Properties’ dialog box.



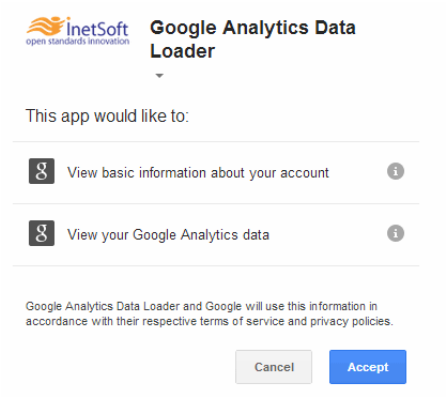
14. Press the ‘…’ button in the ‘Properties’ dialog box. This opens the ‘Google Analytics Data–User ID’ dialog box.



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).

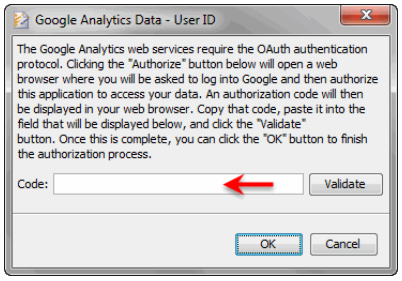


16. Log into the desired account. This prompts you to accept an agreement.



17. Press the ‘Accept’ button. This displays a code in the browser window.

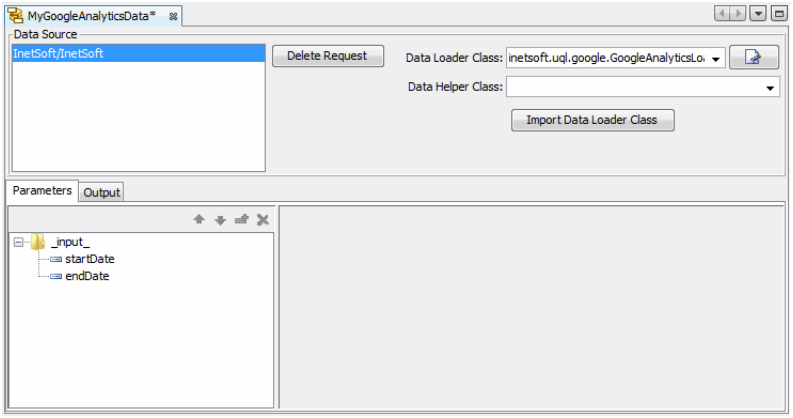
18. Copy the code from the browser window, and paste the code into the ‘Google Analytics Data–User ID’ dialog box. Press ‘Validate’. (Validation may take a few seconds.)



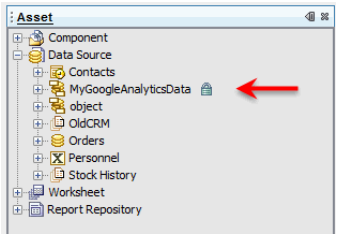
19. When the code has been validated (indicated by an “Authorization successful” message), press ‘OK’ to close the dialog box.

20. Press ‘OK’ to close the ‘Properties’ dialog box.

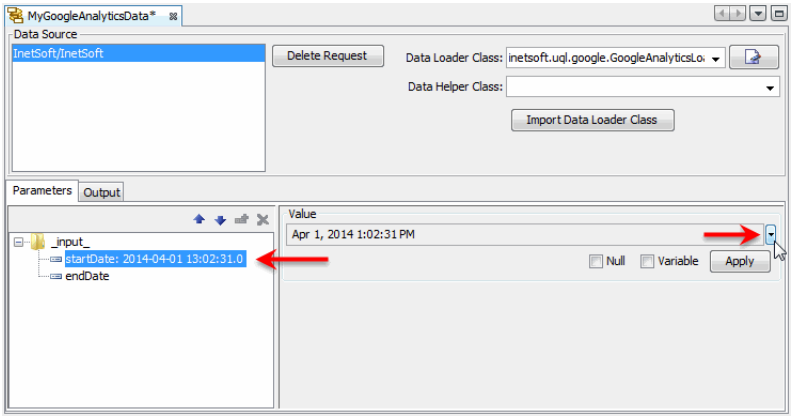
21. Press ‘Finish’ to close the ‘Object Data Source Wizard’. This opens the data source in Style Studio for editing.



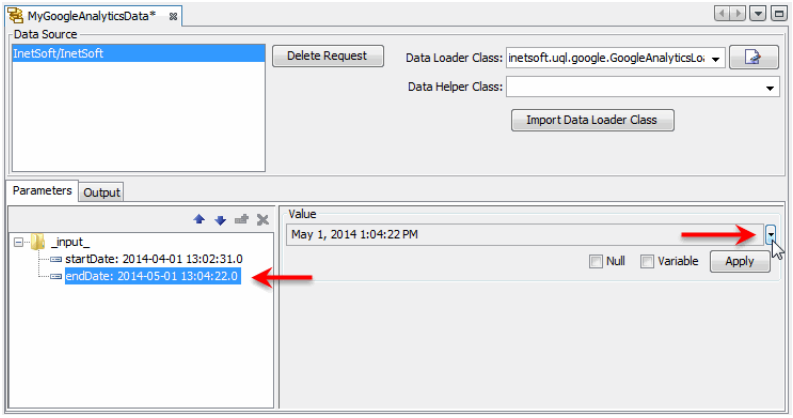
22. Press the 'Save' button in the Style Studio toolbar to save the data source. The saved data source now appears in the Style Studio Asset panel under the 'Data Source' node.



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'.

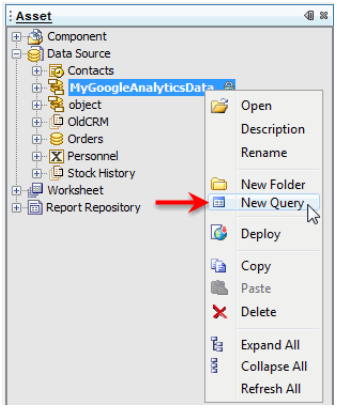


24. In the **Parameters** panel, click on the 'endDate' parameter. Click the arrow button on the right side of the date field, and enter an end date for the period of interest. Press 'Apply'.



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*.

4.10 SharePoint Data Source

To connect to a Microsoft SharePoint system, follow the steps below:

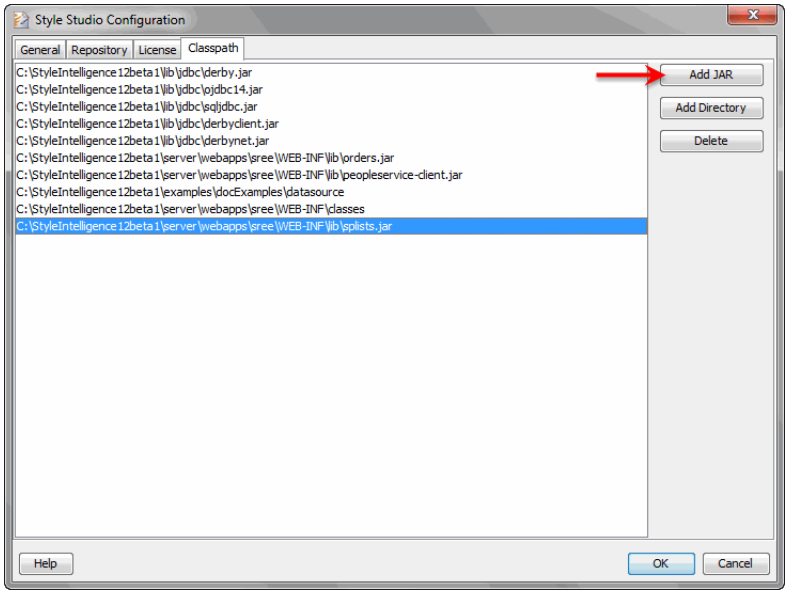
1. Download the *splists.jar* file from the URL that was provided to you.

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.)

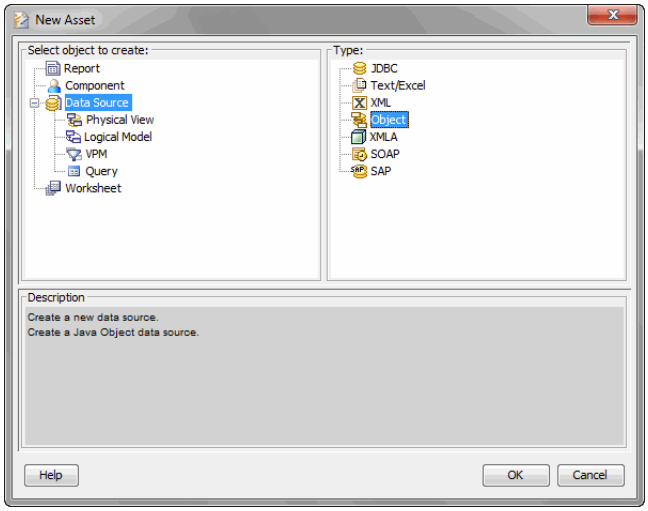
5. Navigate to the location where you copied the *splists.jar* file (*sree/WEB-INF/lib*). Select *splists.jar* and press 'Open'. This adds the file to the classpath.



6. Press 'OK' to close the 'Style Studio Configuration' dialog box.

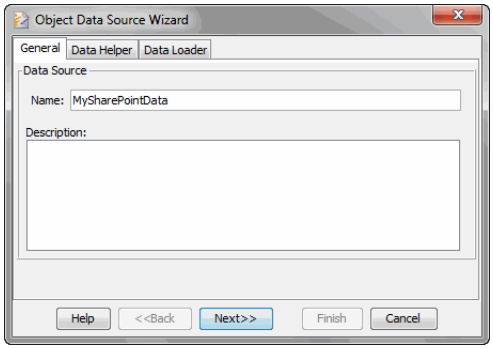
7. Select 'New' from the Style Studio 'File' menu. This opens the 'New Asset' dialog box.

8. In the left panel, select 'Data Source'. In the right panel, select 'Object'.



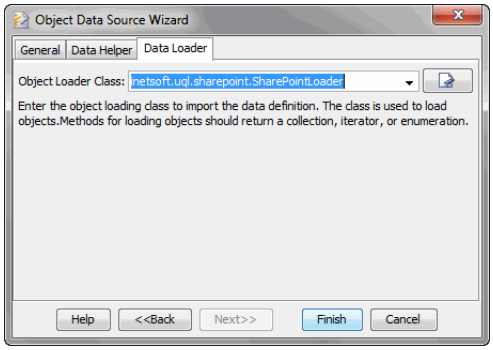
9. Press ‘OK’. This opens the Object Data Source Wizard.

10. Enter a ‘Name’ for the new data source.

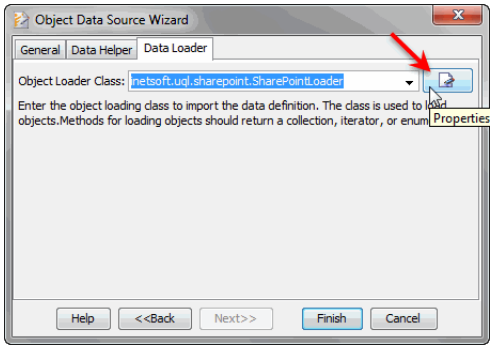


`Note: You do not need to provide any information in the` **`Data Helper`** `tab.`

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:

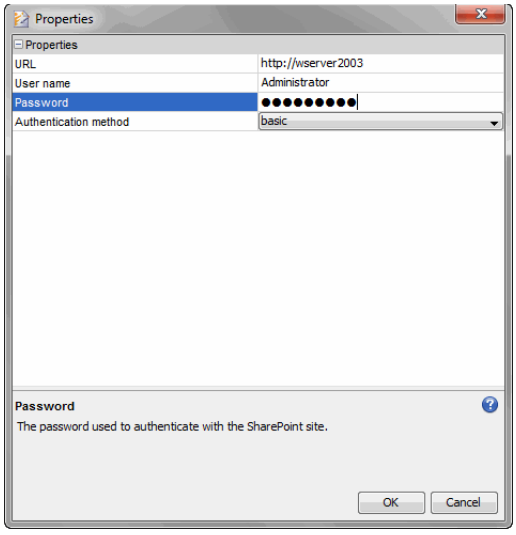
a. Select ‘basic’ for most cases.

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.

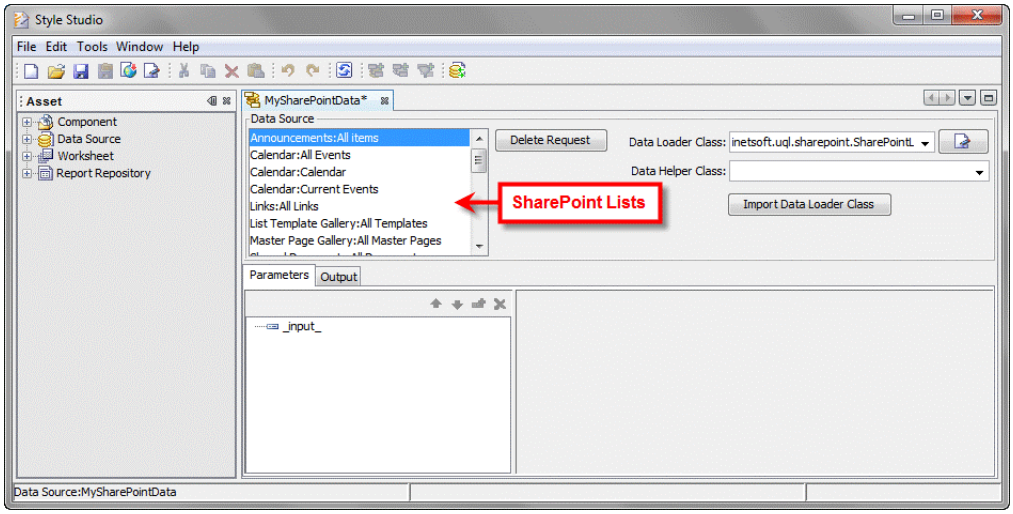
14. Enter your SharePoint server URL in the ‘URL’ field, and enter an appropriate user name and password in the ‘User Name’ and ‘Password’ fields.



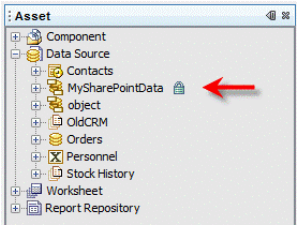
15. Press ‘OK’ to close the ‘Properties’ dialog box.

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.



4.11 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.

4.11.1 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.

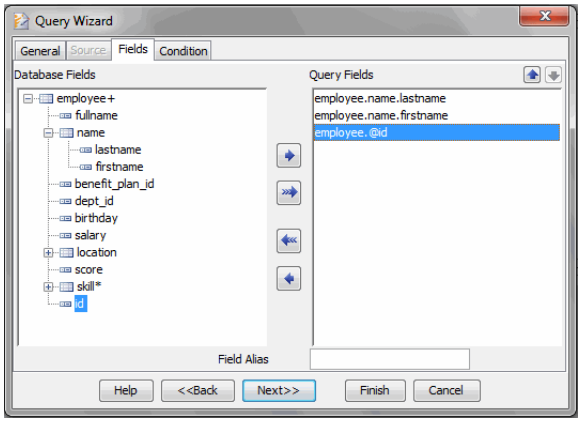
Follow these steps:

1. Press 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”.

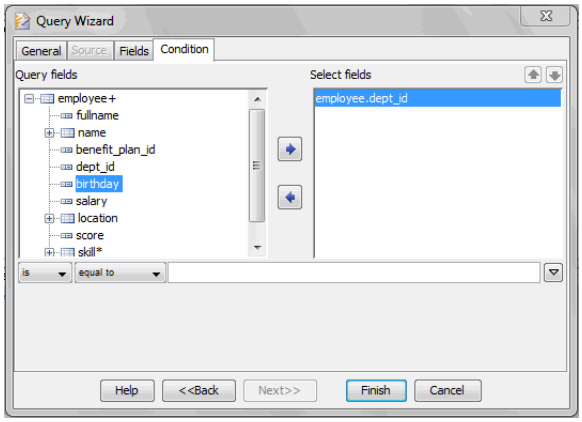
4. Select the **Fields** tab. Expand the ‘`name`’ node in the ‘Database Fields’ panel. Add ‘`lastname`’ and ‘`firstname`’ to the ‘Query Fields’ panel. Also add the ‘`id`’ node, then click ‘Finish’.



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.

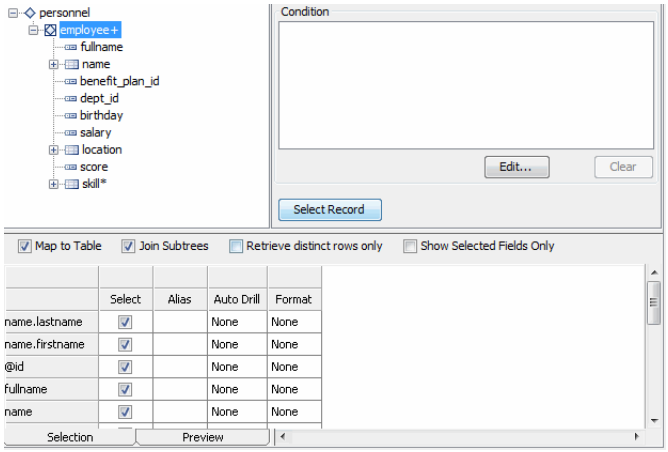
   In the menu, choose one of ‘User’, ‘Roles’, or ‘Groups’. These parameters return information about the user who is currently accessing the report; respectively, the user name, the array of roles to which the user belongs, and the array of groups to which the user belongs.

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

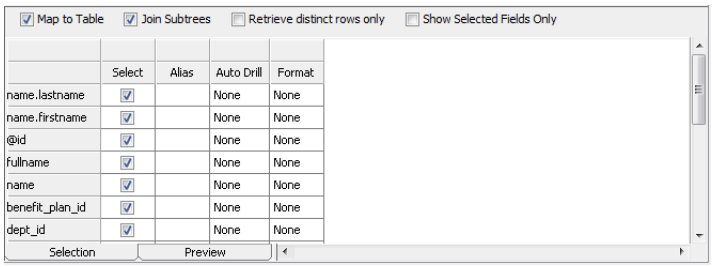
To add table construction rules to a query, follow the steps below:

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.

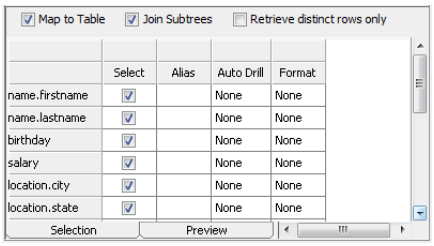
3. Deselect the ‘Join Subtrees’ option.

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

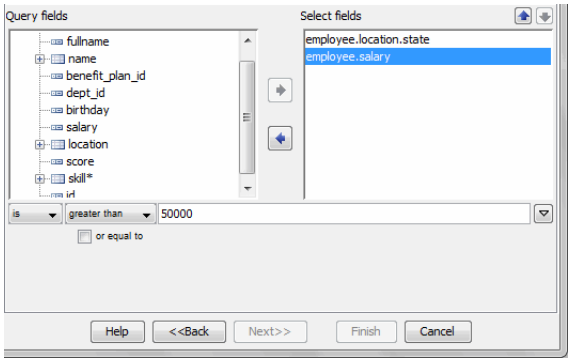
Tree Node Filtering

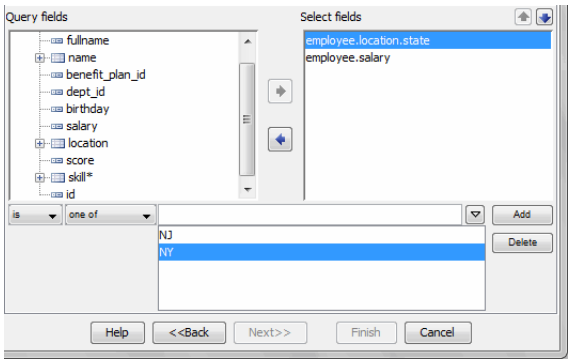
So far you have used the query engine to select all nodes on the specified tree path. As described before, you can attach conditions to selection nodes to filter the child sub-trees and extract a subset of the nodes.

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

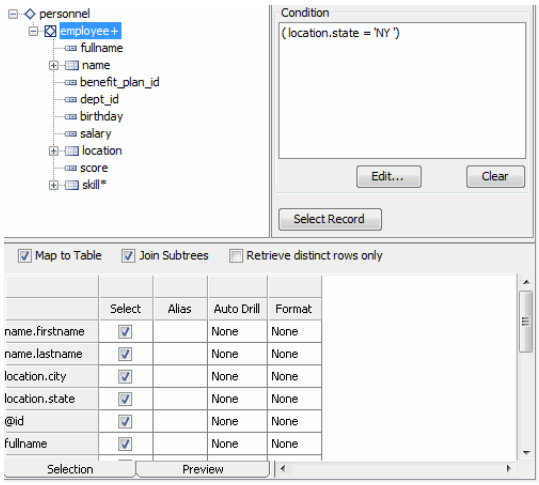
branches are processed. The collection of all selected sub-trees becomes the child nodes of a new root node.

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

4.11.2 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:

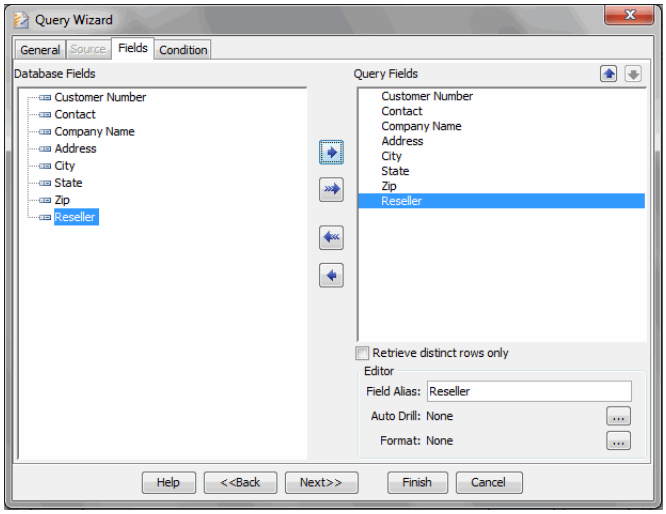
1. Click the 'New' button in the Style Studio toolbar. This opens the 'New Asset' dialog box.

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.)

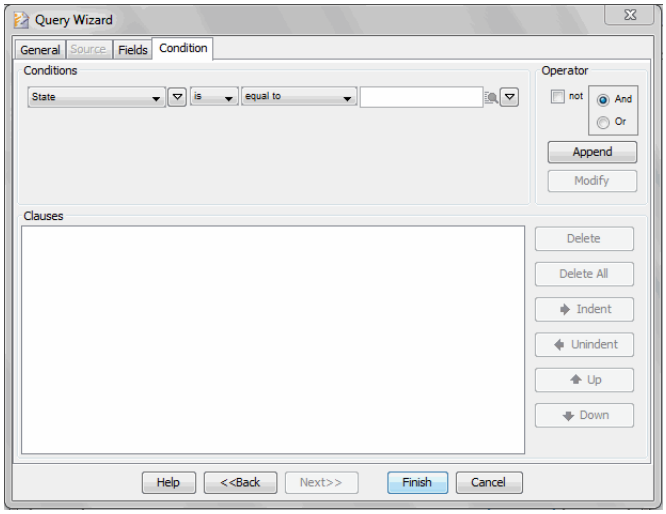
5. Select the **Fields** tab. Choose the database fields you want to include in the query.



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]
   ```

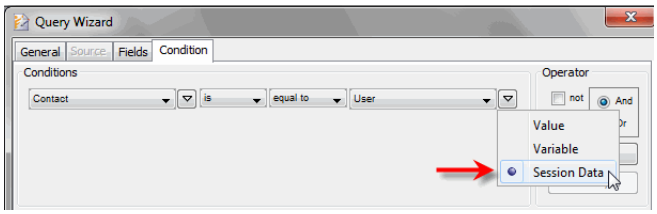


8. To use a value in the right-hand side of the condition, click the “arrow” button at the right side of the condition, and select the ‘Value’ option. Then enter the desired value in the text field.

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.

4.11.3 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.

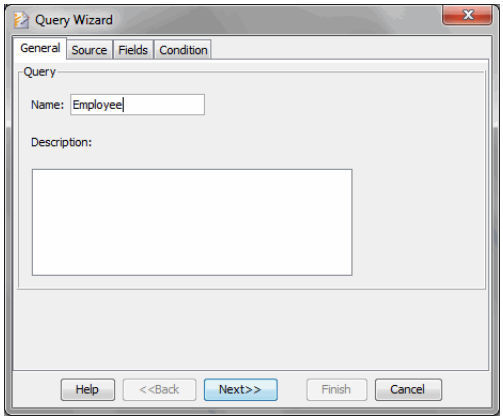
A query based on the Object data source is executed in the following sequence:

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.

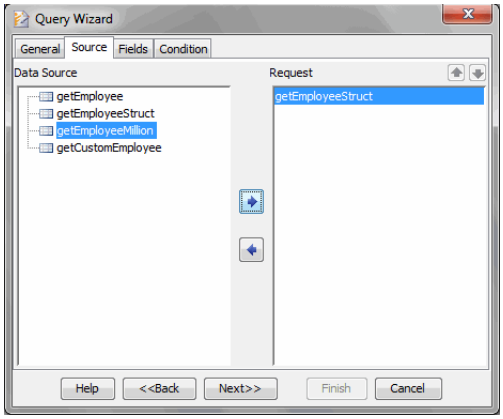
To define a query on the Object data source that you created in *Defining a Java Object Data Source*, follow the steps below:



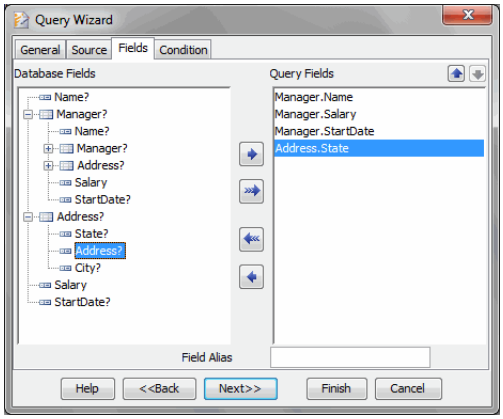
1. Click on the ‘New Query’ button to create a new query.
2. Type in “Employee” as the name of the query. Select ‘Object’ as the data source. Click ‘OK’.



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.

4.11.4 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.

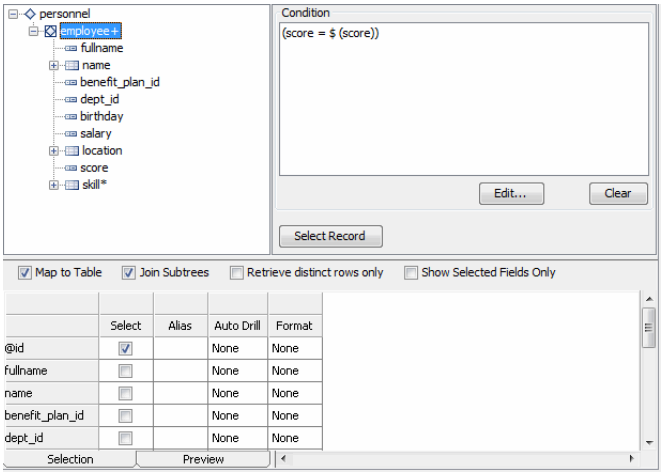
1. Create a query named "Scores" based on the sample 'Personnel' data source.
2. Select the 'default' request type.
3. On the **Fields** tab, select the 'id' field, and click the 'add' button.
4. On the **Condition** tab, select the 'score' field, click the 'add' button, and set the condition to:

   ```
   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 named “Above avg employees” based on 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))
```

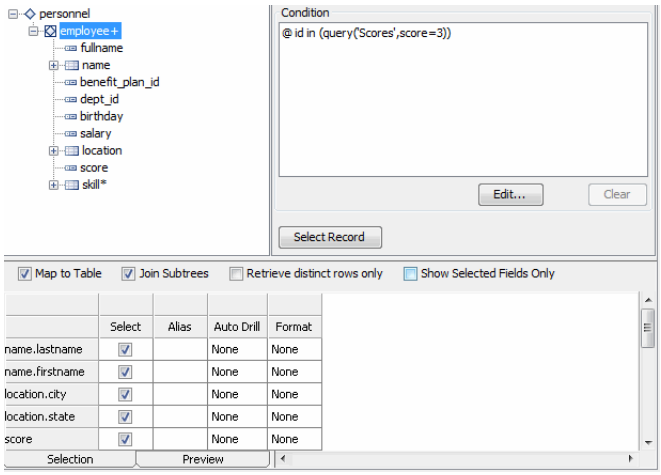


Figure 6. Query Output

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 Metadata

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 Metadata

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.

Walkthrough

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'
```

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 *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

A Data Worksheet is a powerful and easy-to-use tool that enables you to rapidly build complex data sets for tasks such as “what-if” and cross-domain analysis. Using the simple mouse-driven interface, you can mashup diverse data sources and local data. The data blocks you create in the Data Worksheet can 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

```
Note: The screen images shown in this guide are generally from the
Visual Composer.
```

In Style Scope and Style Intelligence, you can create and edit Worksheets using either the desktop *Style Studio* or the web-based *Visual Composer*. These tools provide identical functionality, although there are some minor differences in appearance.

If you are using Style Scope Agile Edition, see *Creating a Data Set by Mash-Up* for information on how to create a new Worksheet.

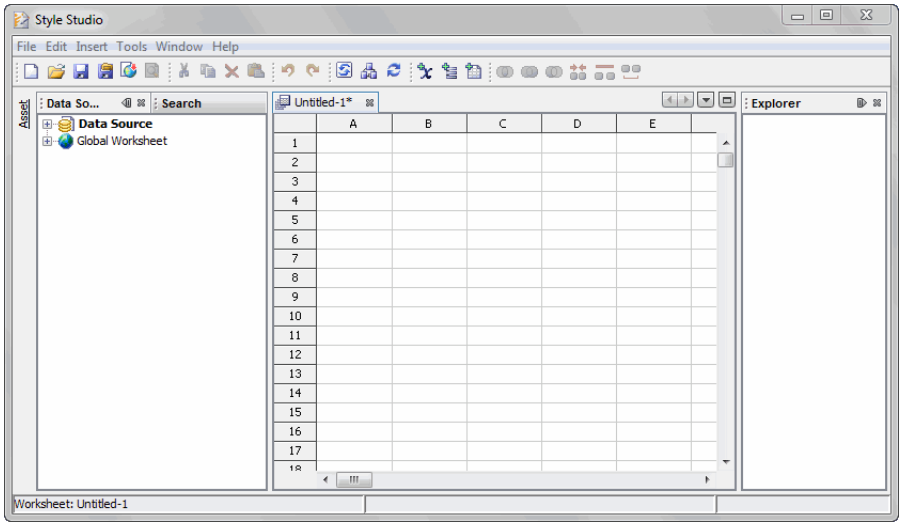
See Also
Saving an Asset, for information about asset storage.
Editing Data Worksheets, for information on opening and creating Worksheets.

1.1.1 Editing a Data Worksheet in Style Studio

```
Style Studio is not available in Style Scope Agile Edition.
```

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

`If you are using Style Scope Agile Edition, see` *`Creating a Data Set by Mash-Up`* `to create a new Worksheet.`

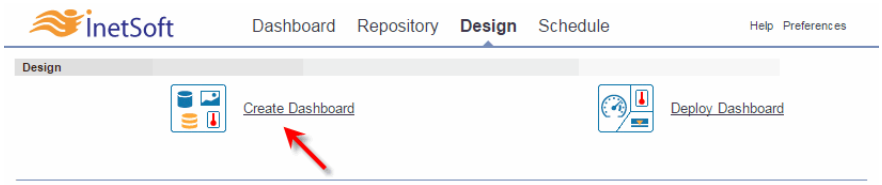
To launch the web-based Visual Composer, follow these steps:

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. Select the **Design** tab at the top of the User Portal.

3. Press the 'Visual Composer' link under the **Design** tab. This launches 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.

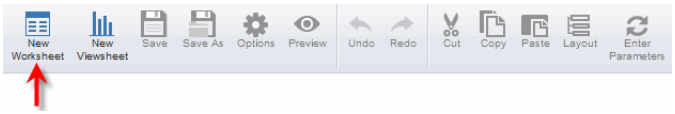
1.2 Editing Data Worksheets

This section provides basic information about working with Worksheets.

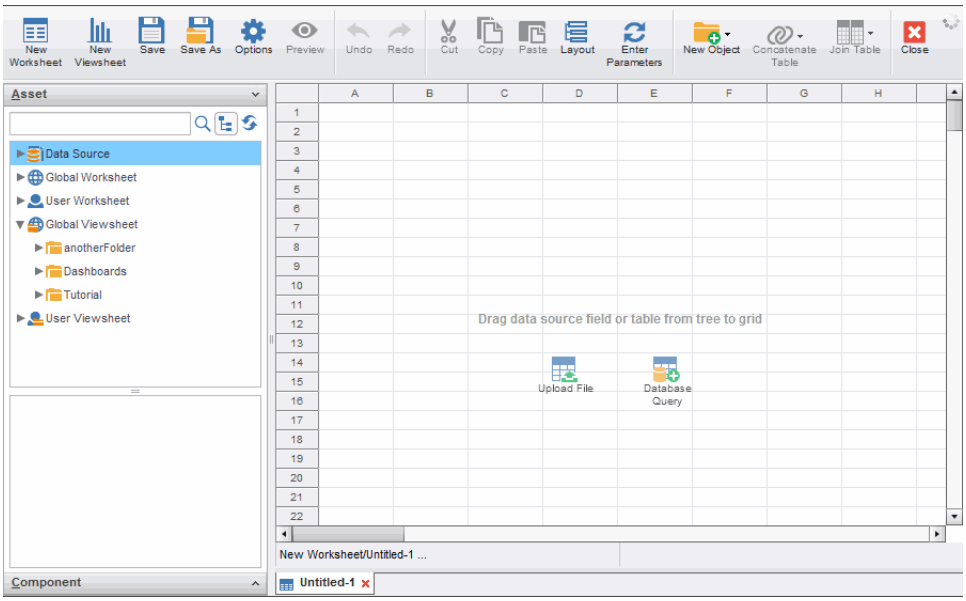
1.2.1 Creating a New Data Worksheet

`If you are using Style Scope Agile Edition, see` *`Creating a Data Set by Mash-Up`* `to create a new Worksheet.`

To create a new Worksheet (in Style Studio or Visual Composer), press 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

`Local Worksheets are not available for Style Scope Agile Edition.`

A *local Worksheet* is a Worksheet that is embedded within a report, rather than being stored in the common *asset.dat* file or *asset.dat.d* directory. 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.

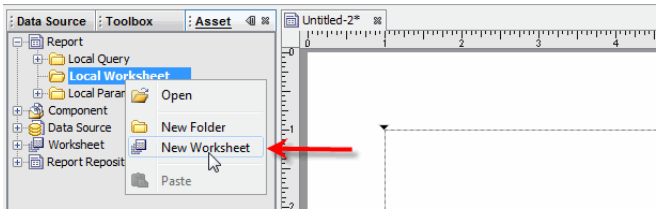
Creating a Local Worksheet

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* file or *asset.dat.d* directory 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.

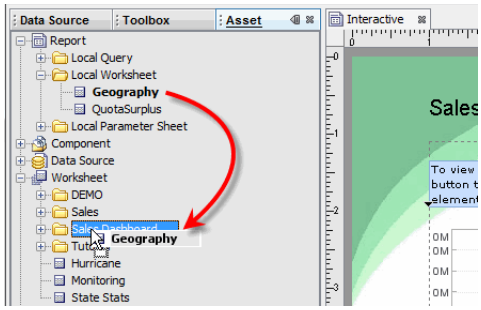
   b. Right-click the desired folder under the ‘Worksheet’ tree node, and select ‘Paste’ from the context menu.

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* file or *asset.dat.d* directory 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.

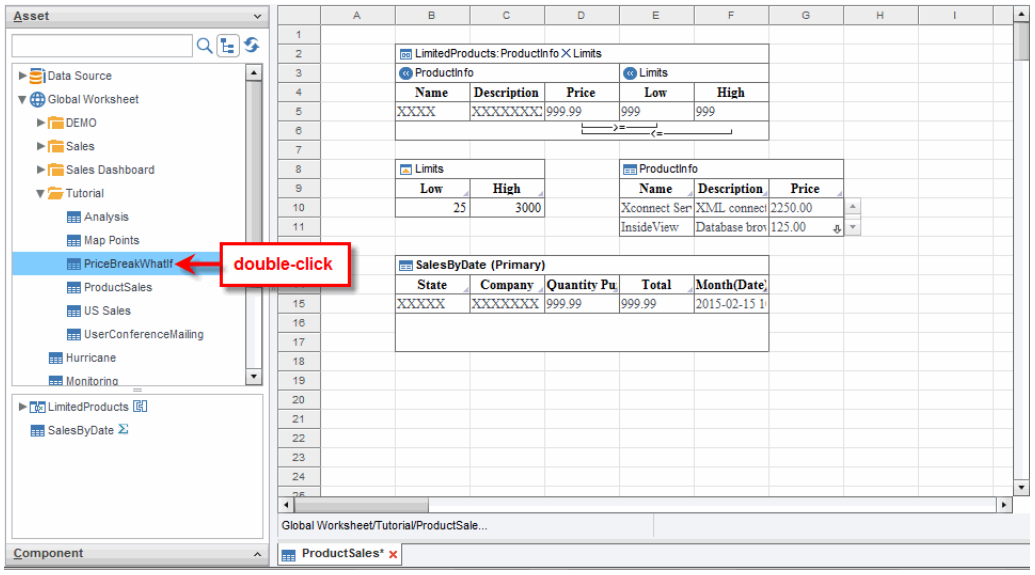
1.2.3 Opening a Data Worksheet

`If you are using Style Scope Agile Edition, see Editing a Data Set to open a Worksheet.`

To open a saved Worksheet (in Style Studio or Visual Composer), follow these steps:

1. Expand the ‘Global Worksheet’ or ‘User Worksheet’ node in the **Asset** panel, and locate the desired Worksheet.

2. Double-click the Worksheet in the **Asset** tree, or right-click the Worksheet and select ‘Open Sheet’.



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

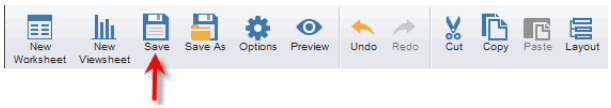
Assets: Reusable Worksheet Components, for information on editing a Worksheet.

1.2.4 Saving a Data Worksheet

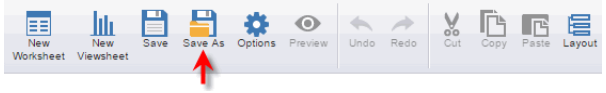
This section discusses Worksheet storage and scope.

Saving a New Version of an Existing Worksheet

To save a new version of an existing Worksheet, press 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.



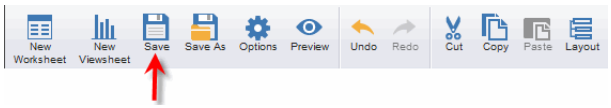
To save the new version of the Worksheet under a different name, press the ‘Save As’ button. This opens the ‘Save Worksheet’ dialog box. Then follow the instructions for saving a new Worksheet below.



Saving a New Worksheet

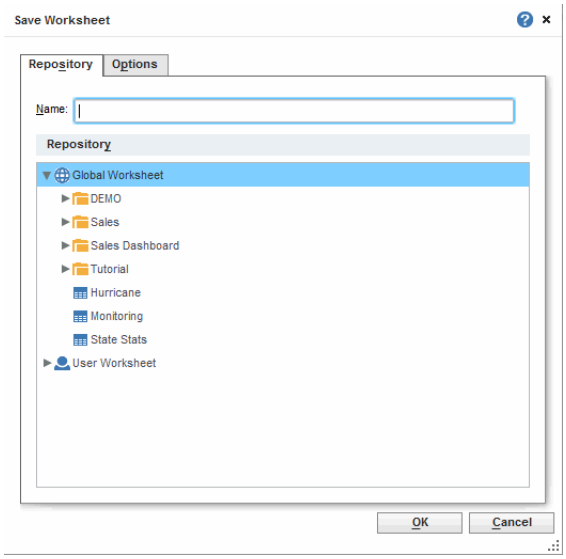
To save a new Worksheet, follow these steps:

1. Press the ‘Save’ button in the toolbar, or press Ctrl-S on the keyboard.

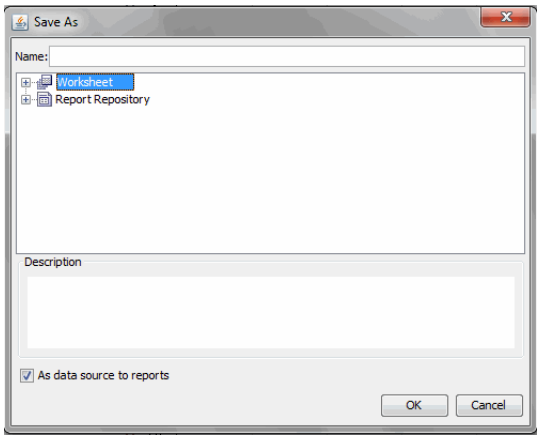


In Style Scope Agile Edition, this is the ‘Save Data Set’ dialog box.

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.

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.

```
'As data source to reports' is not available in Style Scope Agile Edition.
```

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 data blocks in the Worksheet to retrieve.

6. Press 'OK' to save the Worksheet.

The Worksheet is saved to the asset repository, and appears in the **Asset** tree under the specified scope. Saved Worksheets are stored in the *asset.dat* file or *asset.dat.d* directory in the SREE Home directory.

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 blocks, 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** tree. See *Using an Asset* for a list of icons.

1.2.5 Closing a Data Worksheet

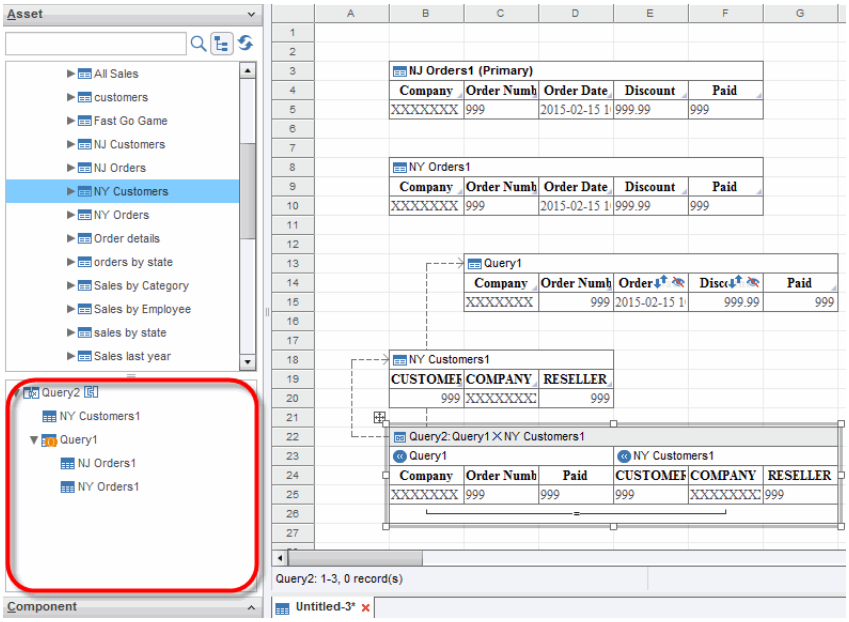
To close a Worksheet, do one of the following:

- 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.



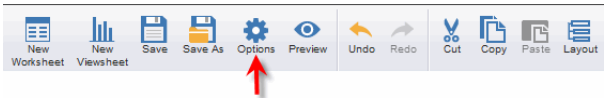
The hierarchy reflects the composition structure of data blocks in the Worksheet, with the participating (base) data blocks of each Composition Table listed as sub-nodes under the result.

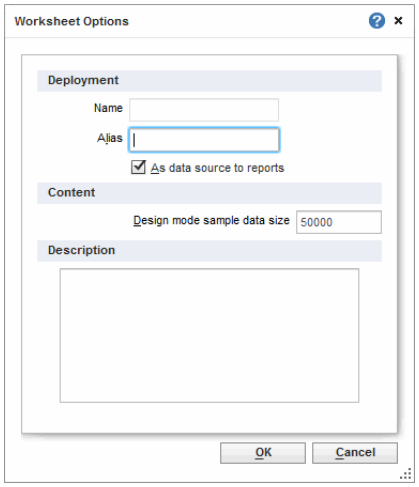
The icons in the tree indicate the type of operation performed to obtain the resulting data block. Click on an icon to open the corresponding dialog box (‘Conditions’, ‘Join Types’, etc.) and make modifications to the operation.

ICON	OBJECT/OPERATION
	Cross Join.
	Embedded table data block. (See *Creating an Embedded Table*.)
	Inner Join.
	Intersect.
	Merge Join.
	Minus.
	Mirror data block. (See *Mirroring a Table*.)
	Rotated data block. (See *Rotating a Table*.)
	Data block used as subquery within condition. (See *Using a Subquery in a Condition*.)
	Union.

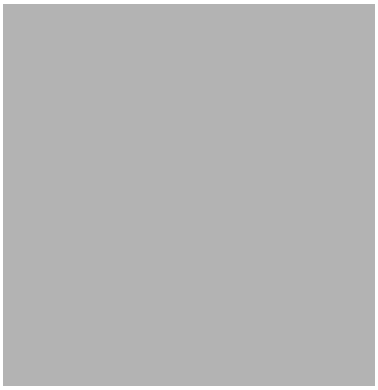
1.2.7 Setting Global Worksheet Options

To set global Worksheet options using Visual Composer, press the ‘Options’ button in the toolbar.





To set global Worksheet options using Style Studio, select the ‘Worksheet Properties’ option from the *File* menu.



Some options are not available in Style Scope Agile Edition.

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 data block 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 data block is itself row-limited. For example, if the final data block 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 Block 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 data block query.

2 Assets: Reusable Worksheet Components

An *asset* is a reusable Worksheet component. The three different types of Worksheet assets are listed below.

ASSET	DESCRIPTION
Data Block	An data table (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** 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 Block
	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 Block

You can use a data block from the **Asset** tree in a Viewsheet, report, or within another Worksheet.

Using a Data Block in a Viewsheet

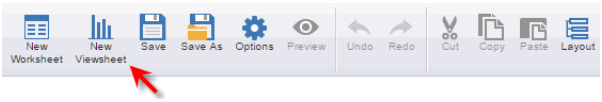
A Viewsheet can be based on a Worksheet. The Viewsheet will have access to *all* of the data blocks its underlying Worksheet, regardless of whether the data blocks are marked as Primary.

In Style Scope Agile Edition, see *Creating a New Dashboard*.

To create a new Viewsheet based on an existing Worksheet, follow these steps:

In Style Scope Agile Edition, press the 'New Dashboard' button.

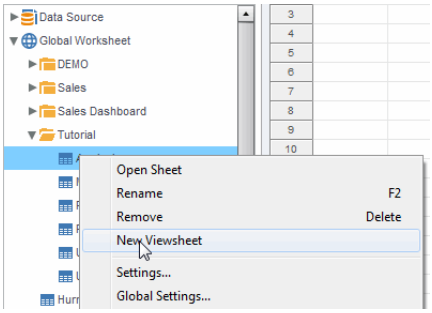
1. Press the 'New Viewsheet' button in the Visual Composer toolbar. This opens the 'New Viewsheet' dialog box.



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.

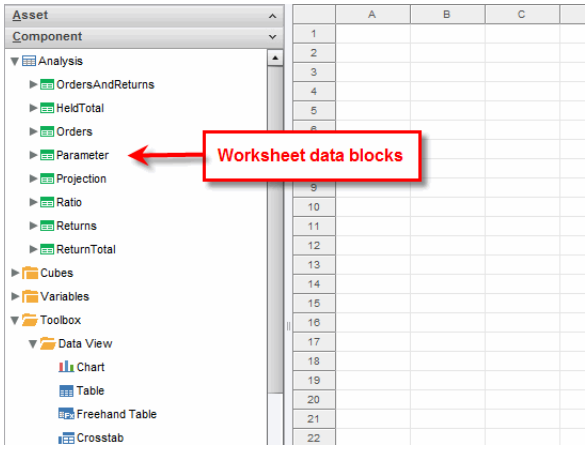
Alternatively, right-click the desired Worksheet in the **Asset** tree, and select 'New Viewsheet' from the context menu.



To see which data blocks 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 blocks in the underlying Worksheet. Although you can insert these data blocks directly into the Viewsheet, it is more typical to bind the data blocks to Viewsheet input and output controls that provide visualization and exploration capabilities.

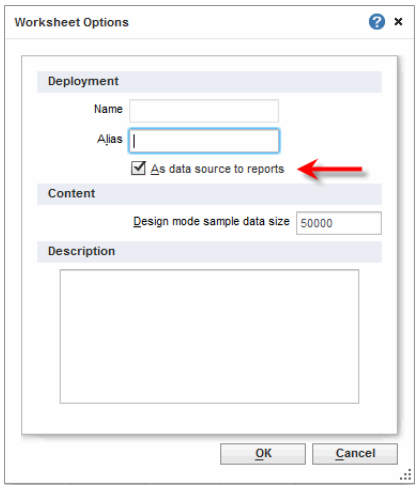
For more information on using Worksheet data within a Viewsheet, see *Advanced Dashboard Design*.

Using a Data Block in a Report

`This feature is not available in Style Scope Agile Edition.`

To use a Worksheet data block in a report, follow the steps below.

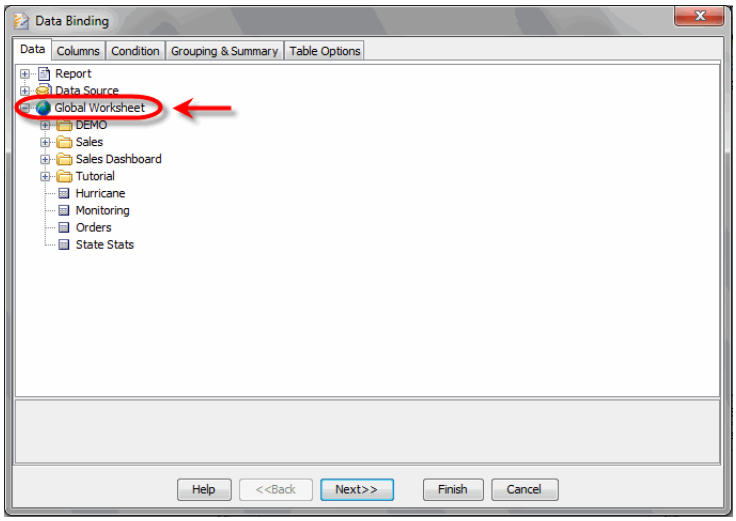
1. Press the 'Options' button in the Worksheet toolbar, and verify that 'As data source to reports' is enabled. This makes the Worksheet visible for data binding in reports.



2. In Style Studio, open the report template in which you want to access the Worksheet data block.

3. Open the 'Data Binding' dialog box for the report element (Table, Chart, Section, etc.) that you want to bind to the Worksheet data block. (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 blocks belonging to the 'Global' and 'Report' scopes of the asset repository. Data blocks belonging 'User' scope are not available for binding.

5. Select the desired data block, and proceed with the data binding.

Note that only the data block 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 block the Primary asset.
Setting Global Worksheet Options, to make Worksheets accessible to reports.

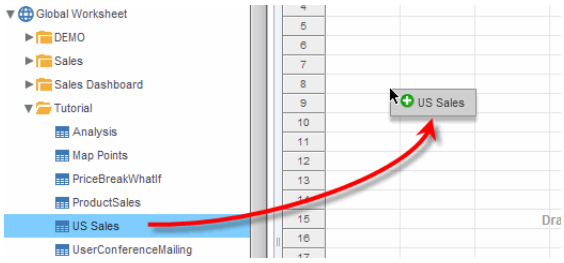
Using a Data Block within a Worksheet

To use a data block from the **Asset** tree in a Worksheet, follow these steps:

1. Open the Worksheet in which you want to use the data block.

2. Drag the data block from the **Asset** tree into the Worksheet.

This adds the data block to the Worksheet as a new data block. Note that only the Primary data block of the Worksheet in the Repository is added to the new Worksheet.



3. Rename the new table as desired.

The new data block 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 Block

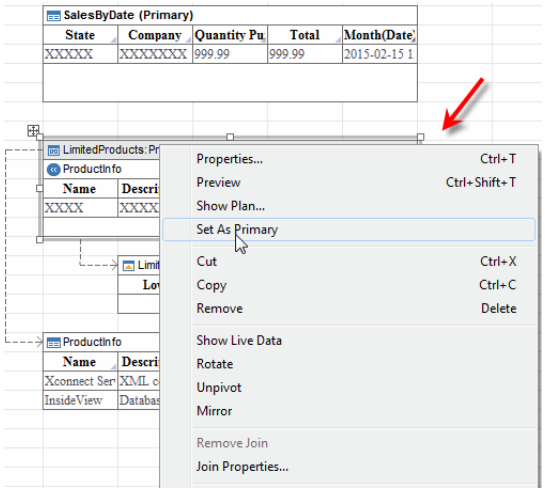
The sample 'ProductSales' Worksheet contains a joined table, 'LimitedProducts', that returns products in a particular price range. To use this data block within another Worksheet, follow the steps below:

This Worksheet can be found in the 'Tutorial' folder, or in the 'Orders Examples' data set folder in Agile Edition.

1. Open the 'ProductSales' Worksheet in Visual Composer.

   For information on how to open a Data Worksheet, see *Editing a Data Worksheet in Visual Composer* (Style Scope or Style Intelligence) or *Editing a Data Set* (Style Scope Agile Edition).

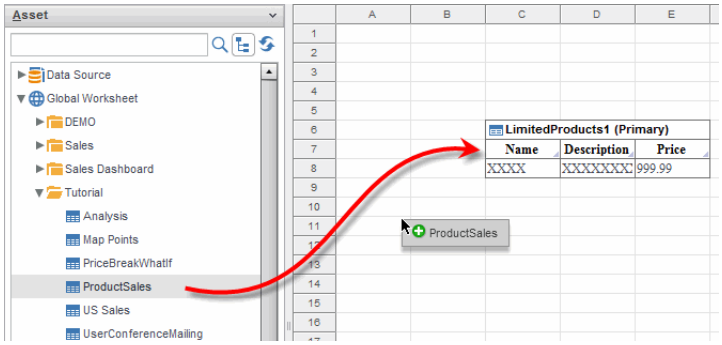
2. Right-click the 'LimitedProducts' data block, and select 'Set as Primary' from the context menu. This makes the 'LimitedProducts' data block visible to other Worksheets.



3. Save the ‘ProductSales’ Worksheet.

4. Create a new Data Worksheet.

5. 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** tree 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.

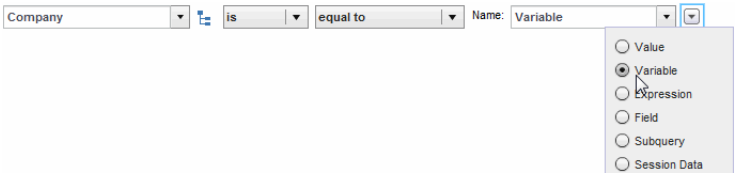
2.1.3 Using a Variable

You can use a Variable from the asset repository in the filtering condition of a Worksheet data block or Named Grouping. Follow these steps:

1. Drag the Variable from the **Asset** tree 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 Dashboard Options, in *Advanced Dashboard Design*, to enable Viewsheet prompting.
Creating a Named Grouping, for details on Named Groups.

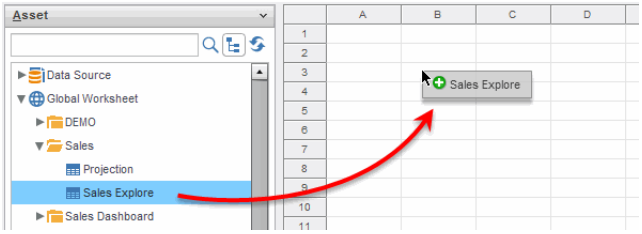
Example: Using a Variable

This example continues the example in *Defining a Variable*. Assume you have a table listing order data for all states, but you are interested in working with the data for only one particular state at a time. To do this, you will add a filtering condition on the 'State' field of the table based on the 'State' Variable. You will use the existing 'State' Variable from the asset repository. (See *Defining a Variable* for instructions on how to create this Variable).

1. Create a new Worksheet.

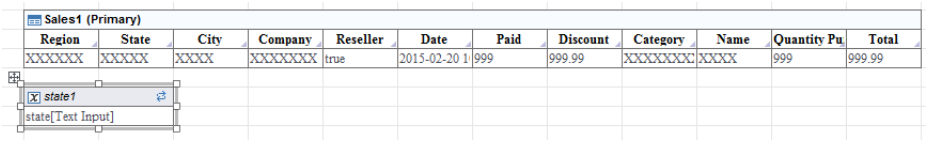
This Worksheet can be found in the 'Sales' folder, or in the 'Orders Examples' data set folder in Agile Edition.

2. From the **Asset** panel, drag the 'Sales Explore' Worksheet onto an empty cell on the Worksheet. This creates the data block 'Sales1'.



The Variable name is shown in italics to indicate that the asset is *referenced* from another Worksheet.

3. From the **Asset** panel, drag the 'State' Variable onto an empty cell in the Worksheet. This imports the variable into the Worksheet as '*state1*.'



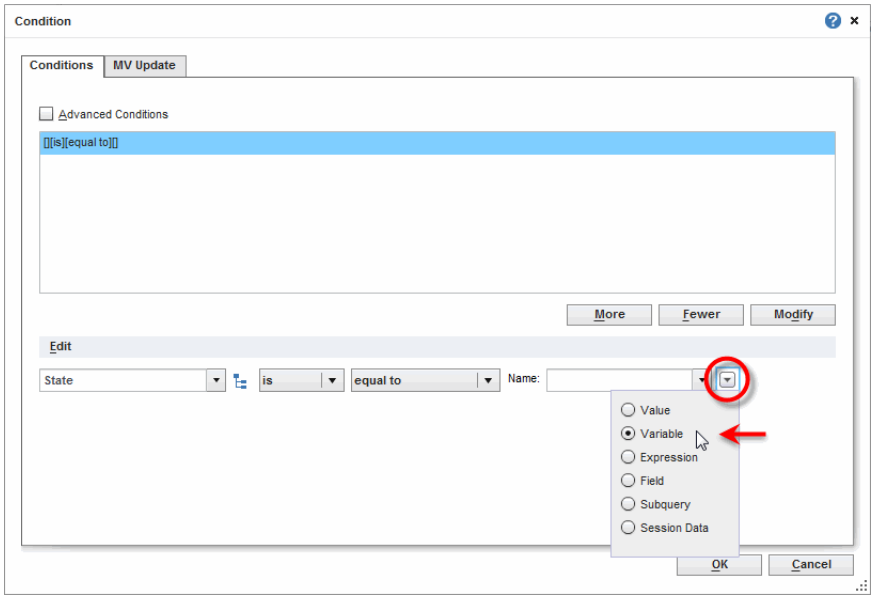
4. Press the 'Define Condition' button in the title bar of the 'Sales1' data block to open the 'Condition' dialog box.

5. Press the 'More' button to add a new condition.

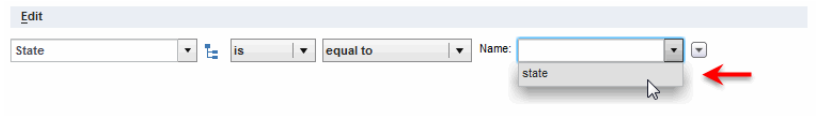
6. Select 'State' from the fields drop-down list.

7. Select "is equal to" as the condition operator.

8. In the right-side pop-up menu select 'Variable.

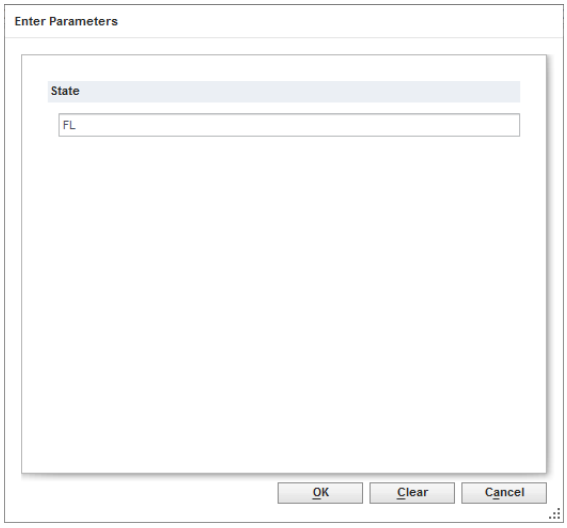


9. From the ‘Name’ menu, select ‘state’.

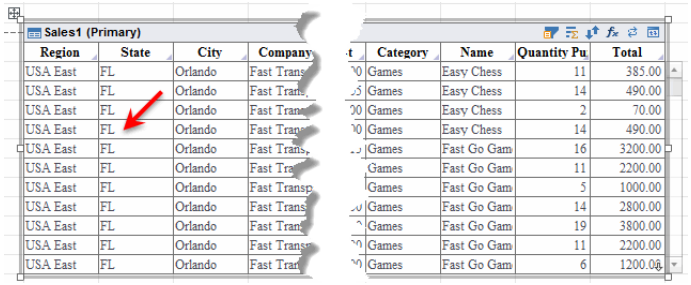


10. Press ‘OK’.

11. You will be prompted for the value of the ‘state’ Variable. Enter “FL” and press ‘OK’.



12. Preview the table (see *Previewing a Data Block*), 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.

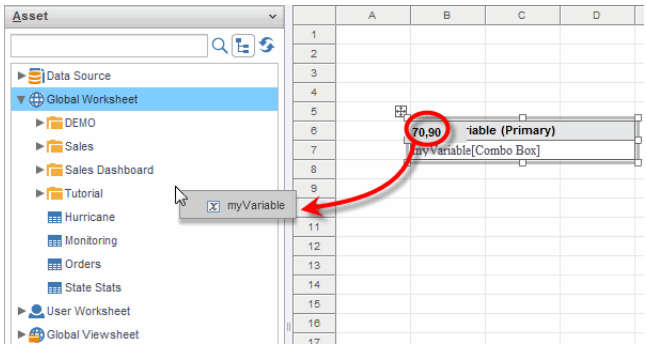
See Also

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

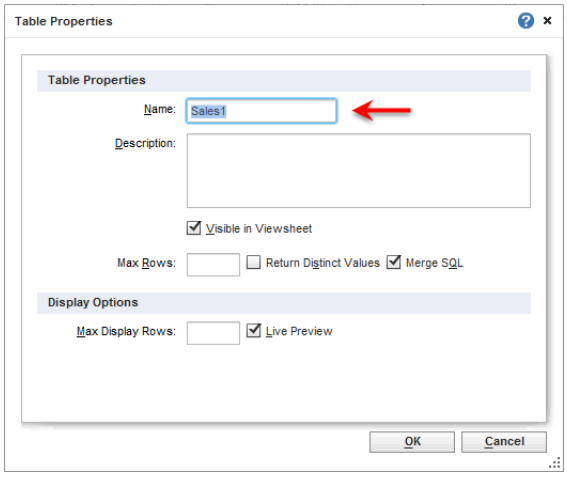
You can rename an asset inside the Worksheet that contains it. You can also rename an entire Worksheet in the **Asset** panel.

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’.

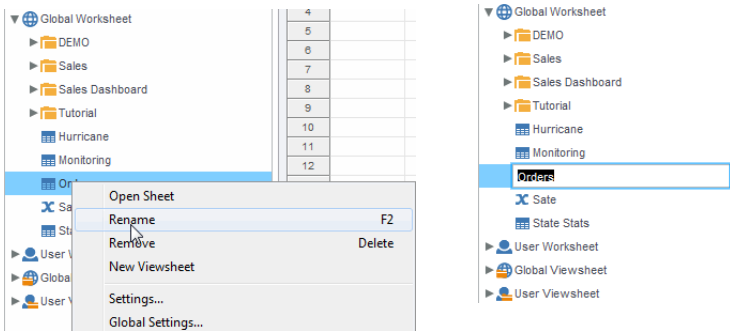


All of the assets in a given Worksheet must have unique names.

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.



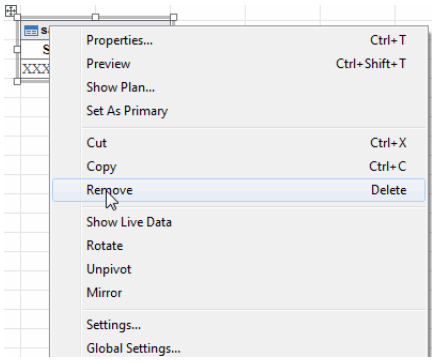
2.4 Deleting an Asset

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 bar to select the asset.
2. Right-click the asset title bar to open the context menu.
3. Select ‘Remove’ from the context menu.



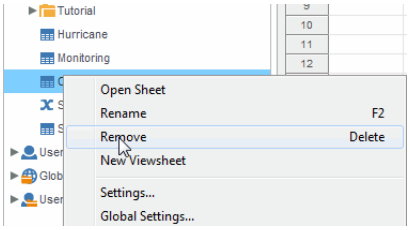
Alternatively, press the ‘Delete’ key on the keyboard to remove the assembly.

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.



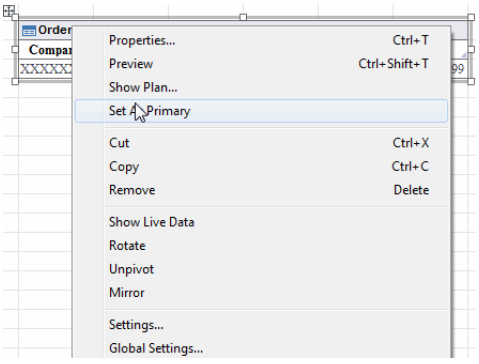
For Style Scope Agile Edition, see also *Deleting an Asset*.

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.



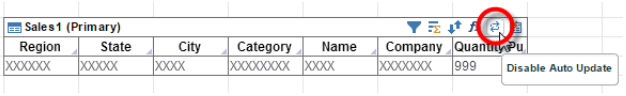
The Primary asset is the “main result” of the Worksheet. It is this Primary asset that you will access from within reports or other Worksheets.

When you save a Worksheet, the Primary asset determines the appearance of the Worksheet in the **Asset** panel, and how the Worksheet can be used. See *Using an Asset* for more information about accessing Worksheet assets.

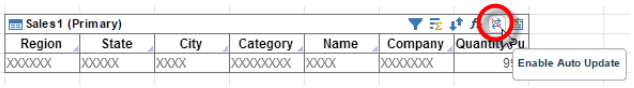
2.7 Automatically Updating an Asset

When you drag an asset from the **Asset** panel 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

`Style Scope Agile Edition only supports global 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 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

`Style Scope Agile Edition only supports global scope.`

To change the scope of an asset in the **Asset** panel tree, drag it to the branch of the 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 Block

You can combine data from multiple queries and different data sources into a data block, producing a single dataset that contains meaningful and useful information. Tables have many different roles; see *Using a Data Block* for more information.

Many data block operations create and maintain links between the component data blocks, 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.

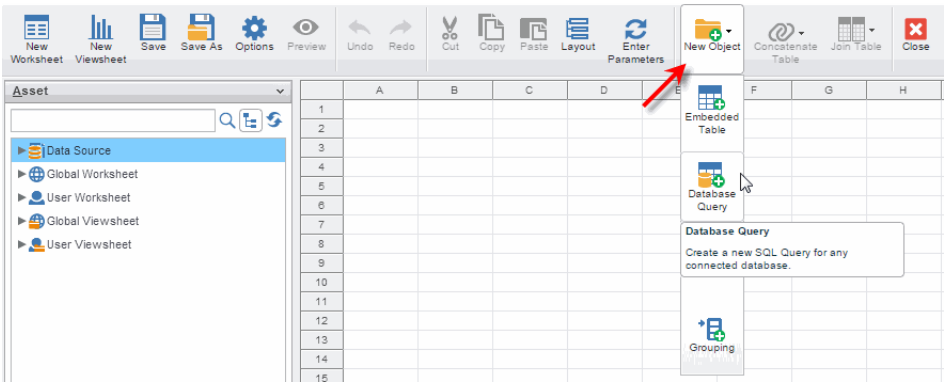
3.1 Creating a Regular Data Block

This section discusses the various ways to create a regular data block.

3.1.1 Creating a Data Block from a New Query

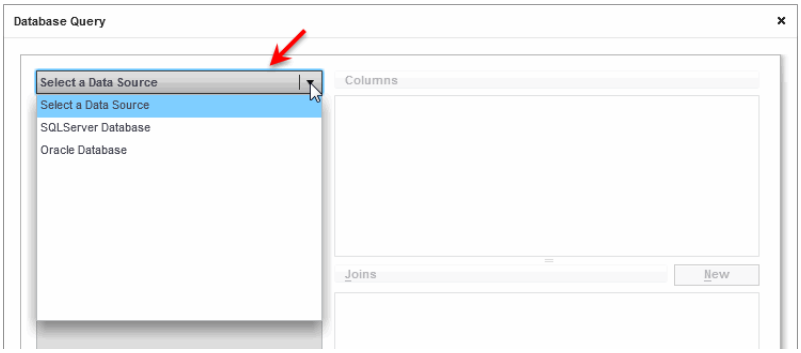
To create a data block by defining a new query, follow the steps below:

1. Press the ‘New Object’ button in the toolbar and select ‘Database Query’.

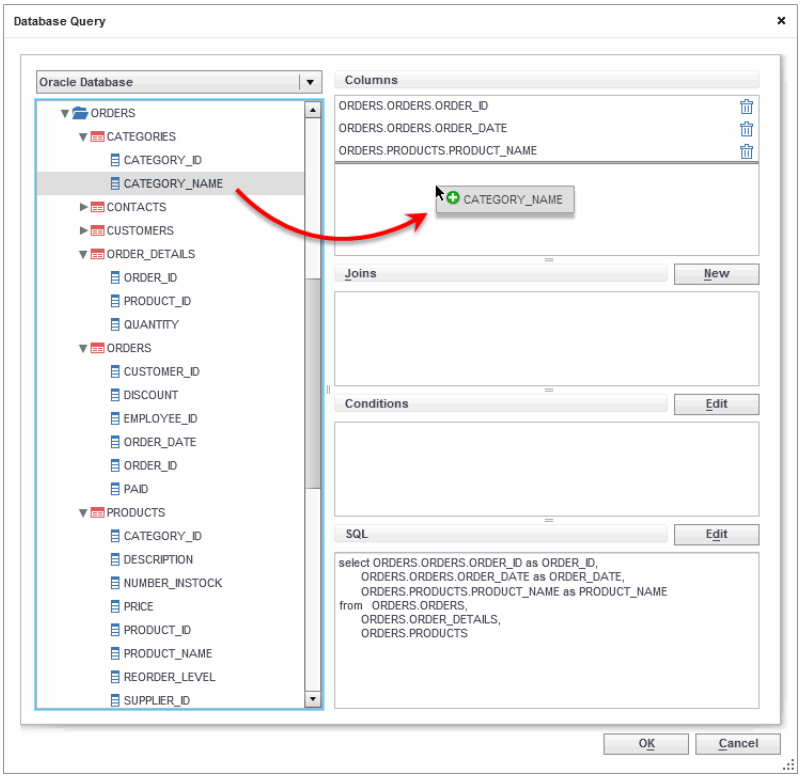


   This opens the ‘Database Query’ dialog box.

2. In the ‘Select a Data Source’ menu, choose the database that you want to supply data to the data set.



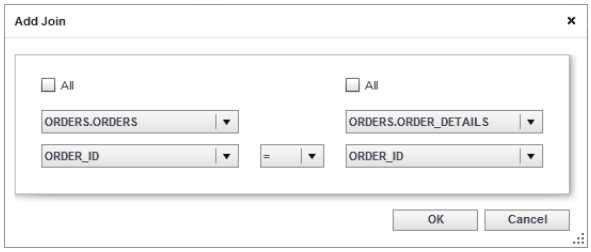
3. Select the database fields that you want to retrieve by dragging them from the left panel into the ‘Columns’ area.



4. Define any required joins. To do this, follow the steps below:

   a. Press the ‘New’ button next to the ‘Joins’ area to open the ‘Add Join’ dialog box.

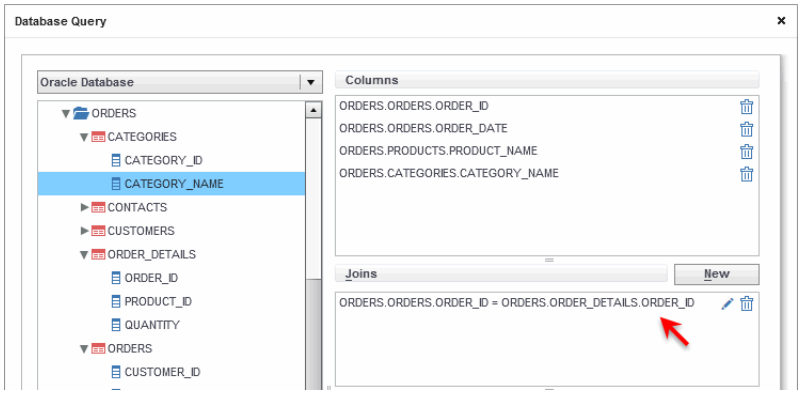
   b. Select a table from the top-left menu, and select the join field for that table from the bottom-left menu. Select a table from the top-right menu, and select the join field for that table from the bottom-right menu. From the center menu, select the join operator (equi-join or inequality join).



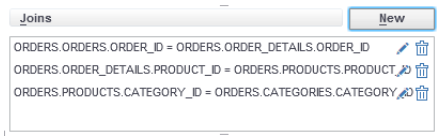
For full information about specifying joins, see *Changing the Join Type*.

c. (Optional) Select the ‘All’ option on the left side to retain all records from the left table, even those for which there is no match in the right table (left outer join). Select the ‘All’ option on the right side to retain all records from the right table, even those for which there is no match in the left table (right outer join). Select both ‘All’ options to retain all rows from both tables (full outer join).

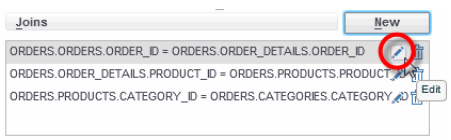
d. Press ‘OK’ to add the join. The join is now displayed in the ‘Joins’ area.



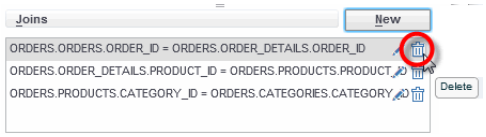
e. To add an additional join, press the ‘New’ button again to open the ‘Add Join’ dialog box. Repeat as necessary to add all desired joins.



f. To change an existing join, press the ‘Edit’ button next to the join.

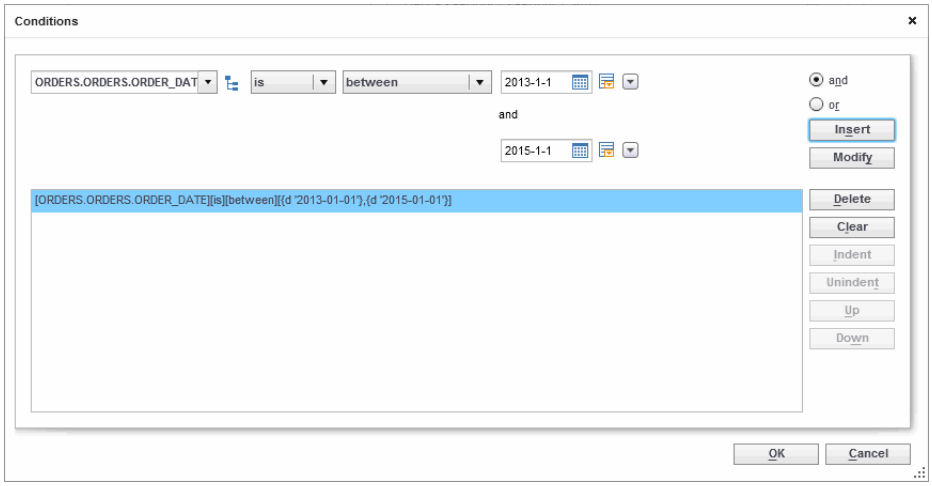


g. To delete an existing join, press the ‘Delete’ button next to the join.



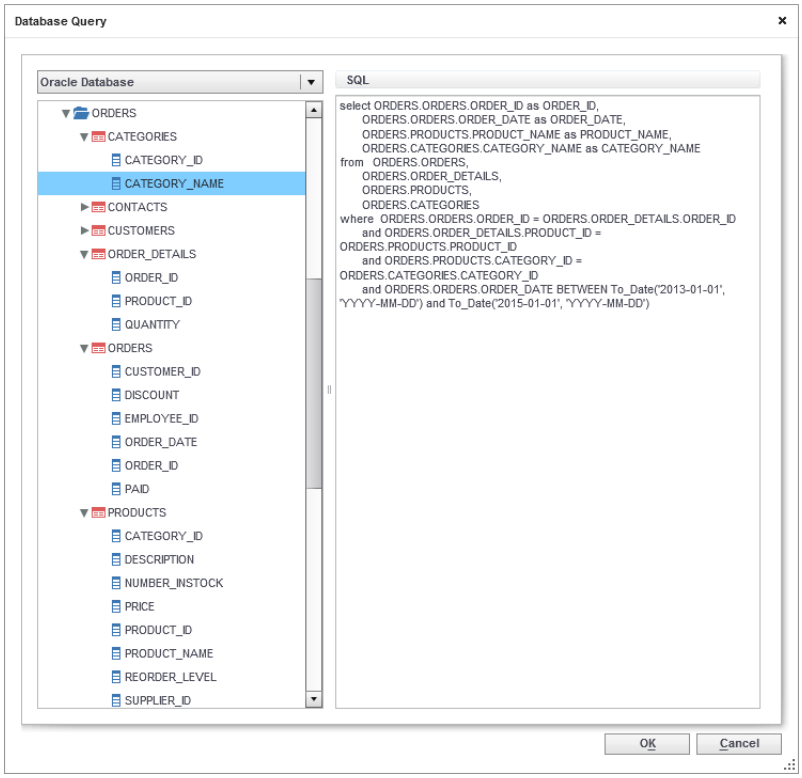
For full information about specifying conditions, see *Filtering*.

5. (Optional) Press the ‘Edit’ button next to the ‘Conditions’ area to open the ‘Conditions’ dialog box. Use the menus to enter a desired filter condition and press ‘Insert’. Then press ‘OK’.




```
Note: If you make manual edits to the SQL string, you will no
longer be able to modify the query using the 'Columns', 'Joins',
and 'Conditions' panels.
```

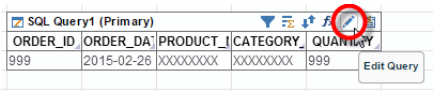
6. (Optional) To make manual edits to the SQL string that will be sent to the database, press the ‘Edit’ button next to the ‘SQL’ area to open the ‘SQL’ panel for editing. Make the desired edits.



7. Press ‘OK’ to close the ‘Database Query’ dialog box. This creates a new data block based on the specified query.



8. To edit the query further, press the ‘Edit Query’ button in the title bar. This reopens the ‘Database Query’ dialog box.



3.1.2 Creating a Data Block from an Existing Query or Data Model

`Style Scope Agile Edition does not support data models. See` *`Creating a Data Set from Database Query`* `for information on how to create a data block from a query.`

To create a regular data block from a query or data model, drag one of the following assets from the **Asset** tree into the Worksheet:

- Query
- Query column

- Data model entity
- Data model attribute

This creates a new data block 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 block 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 Data Block*.

Example: Creating a Data Block from a Query

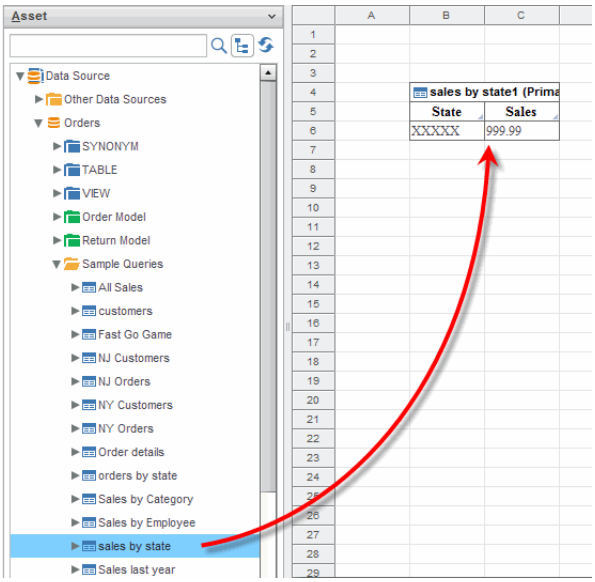
Assume that you want to create a table for the 'sales by state' query from the 'Orders' data source. Follow the steps below.

1. Create a new Worksheet by pressing the 'New Worksheet' button.

`For Style Scope Agile Edition, see Creating a Data Set from Database Query for information on how to create a data block from a query.`

2. In the **Asset** panel, expand the 'Data Source' node, the 'Orders' node, and the 'Sample Queries' 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

¢

3.1.3 Creating a Data Block from Database Tables

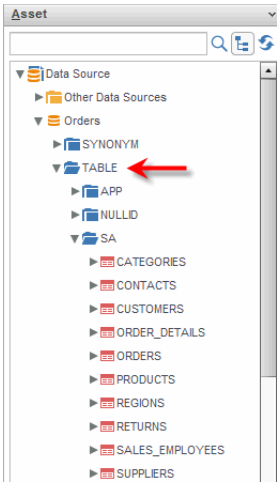
`To create a data block in Style Scope Agile Edition, see` *`Creating a Data Set by Mash-Up`*`.`

In most cases, you should access data from a relational database only via *queries* and *data models*. (See *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 under the desired data source in the **Asset** panel. The database tables are 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 Block* and *Adding a Table Column* for details.)

3.1.4 Creating a Data Block from Existing Table Columns

You can create a new data block by copying the columns of an existing regular data block. 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*.

3.2 Creating an Embedded Table

Instead of creating a regular table from a query or data model, you can create a data block by manually typing values into the Worksheet. Such a table is called an *Embedded Table*.

There are three ways to generate an embedded table:

- Create a new embedded table using the ‘Embedded Table’ option in the ‘New Object’ menu.
- Upload data from a file into a new embedded table.
- Convert a regular data block into an embedded table.

After you generate an embedded table, you can use it in the same way as a regular data block. 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 Block, to create a table from a query or data model.

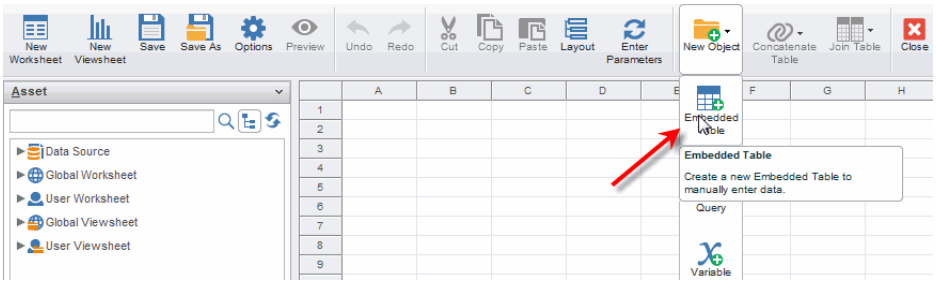
What-If Analysis, to use embedded tables in exploratory analysis.

Using Form 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 the steps below:

1. Press the 'New Object' button, and select 'Embedded Table'.



   The cursor changes to a crosshairs.

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.

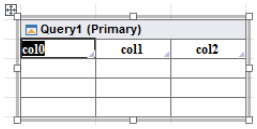
If you are using Style Scope Agile Edition, see *Creating a Data Set by Mash-Up* to create a new Worksheet.

1. Create a new Worksheet by pressing the ‘New Worksheet’ button.

2. Press the ‘New Object’ button, and select ‘Embedded Table’. The cursor changes to a crosshairs.

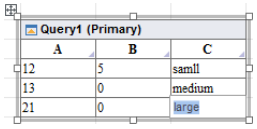
3. Drag across the Worksheet to highlight five rows and three columns. When you release the mouse, the selected range of the empty grid cells becomes part of the new table called ‘Query1’; the first row of the selected range becomes the header row, and the second row becomes the column header row.

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 block 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

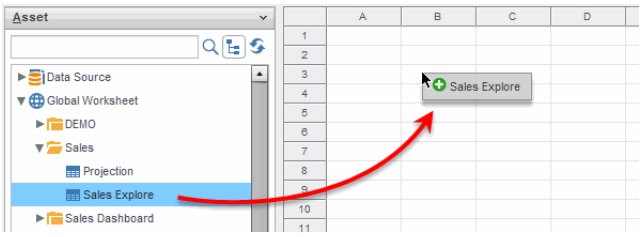
In the following example, the ‘Sales’ data block of the ‘Sales Explore’ Worksheet is converted to an editable embedded table.

If you are using Style Scope Agile Edition, see *Creating a Data Set by Mash-Up* to create a new Worksheet.

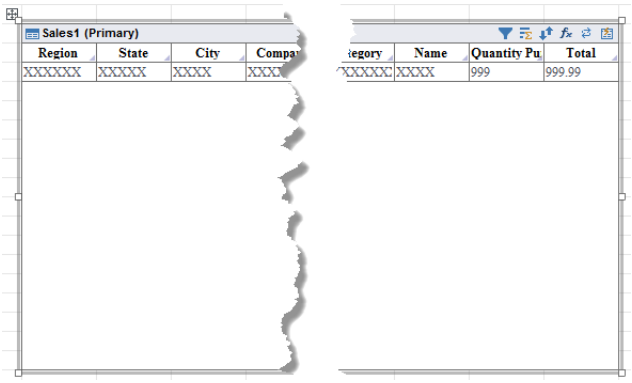
1. Create a new Worksheet by pressing the ‘New Worksheet’ button.

This Worksheet can be found in the ‘Sales’ folder, or in the ‘Orders Examples’ data set folder in Agile Edition.

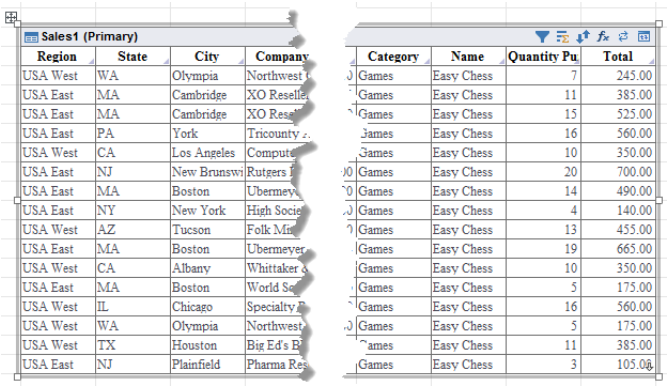
2. From the **Asset** panel, drag the ‘Sales Explore’ Worksheet onto an empty cell on the Worksheet. This creates the data block ‘Sales1’.



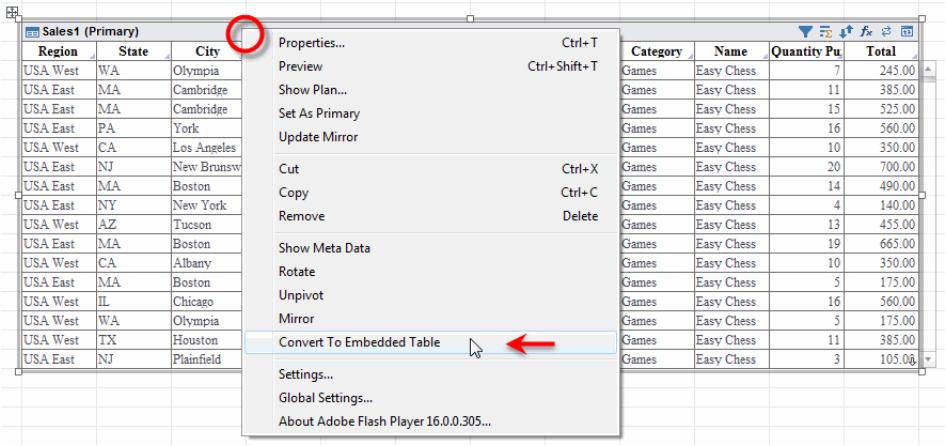
3. Expand the table vertically so that more rows are visible.



4. Press the ‘Change View’ button in the table title bar, and select ‘Live Preview’ from the menu.

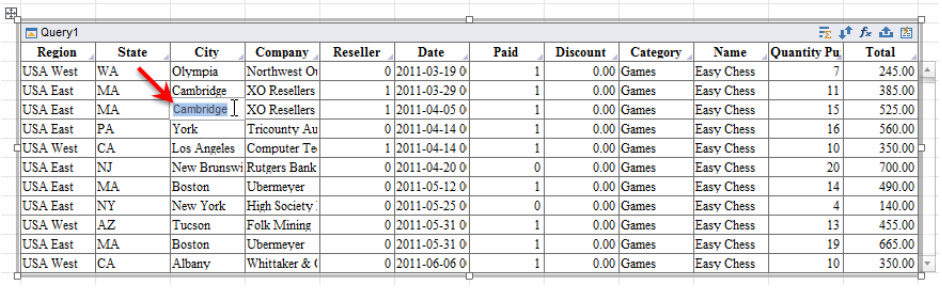


5. Right-click on the title row and select ‘Convert to Embedded Table’.



A new table will be created with name ‘Query1’. Expand the number of rows so that all rows are visible.

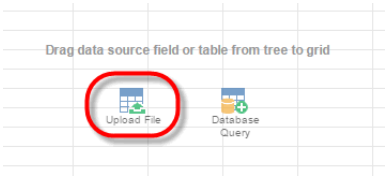
6. Click on any of the cells of the ‘Query1’ table. Notice that all of the cells in this table are editable.



¢

3.2.3 Importing Data Into an Embedded Table

If you have created a new blank Worksheet, press the ‘Upload File’ button in the center of the Worksheet to upload text or Excel data into an embedded table.



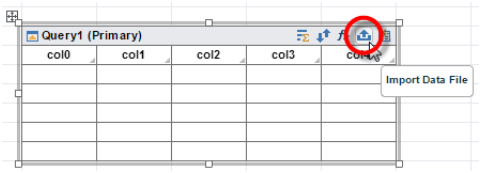
This opens the ‘Import Data File’ dialog box. For information about using this dialog box, see below.

The ‘Upload File’ button is only available when the Worksheet is first created.

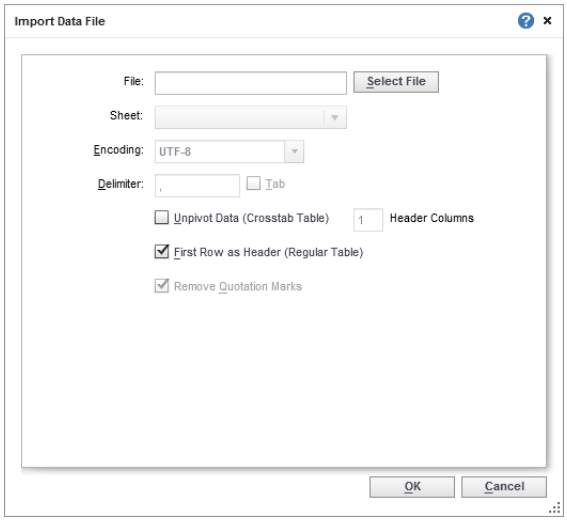
If you do not see the ‘Upload File’ button in the center of the Worksheet, follow the steps below to upload data from a text or Excel file into an embedded table:

1. Create a new embedded table (see *Creating a New Embedded Table*), or select an existing embedded table. Note that any existing data and column names in the embedded table will be overwritten by the imported data.

2. Press the ‘Import Data File’ button in the title bar of the embedded table.



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.

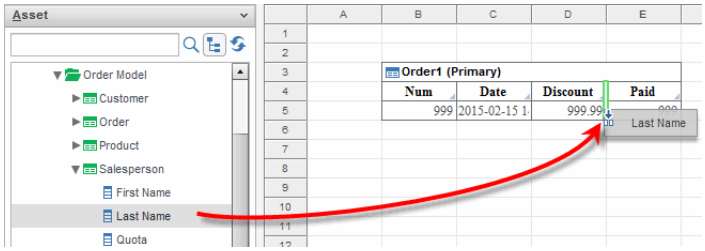
Example: Adding a Column to a Regular Table

In the following example, you will add a single attribute (salesperson last name) as a new column in an existing table.

```
Style Scope Agile Edition does not support data models.
```

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.)

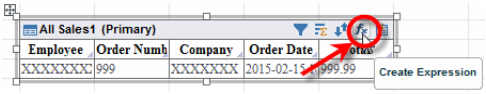
2. Right-click on the table row you want to delete, and choose 'Remove Row' from the context menu. This removes the row you clicked, and shifts subsequent rows upward in the table.

3.3.4 Creating an Expression/Formula Column

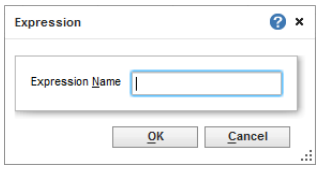
An expression column, or formula column, is a column whose data is generated from a script (expression) by processing existing data in the table. The expression can use either SQL or JavaScript syntax, and can reference any other columns within the data block.

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 block. This opens the ‘Expression’ dialog box.

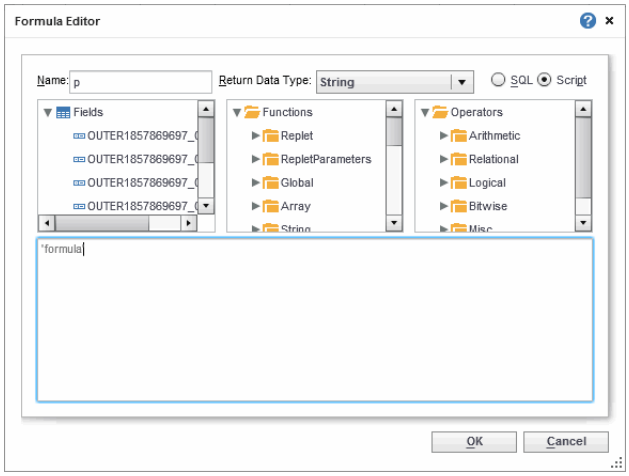


2. Enter a name for the new column in the dialog box, and click ‘OK’.



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 *Form 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 an Expression Column

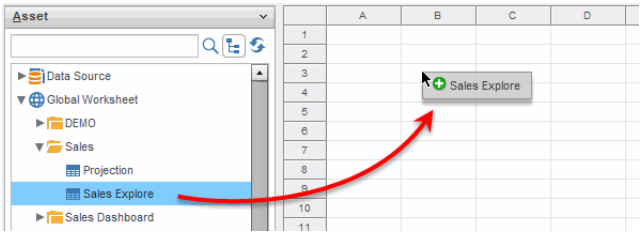
Use the data in the sample 'Sales Explore' Worksheet to compute the revenue for each sale:

`If you are using Style Scope Agile Edition, see Creating a Data Set by Mash-Up to create a new Worksheet.`

1. Create a new Worksheet by pressing the 'New Worksheet' button.

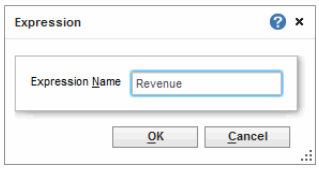
This Worksheet can be found in the 'Sales' folder, or in the 'Orders Examples' data set folder in Agile Edition.

2. From the **Asset** panel, drag the 'Sales Explore' Worksheet onto an empty cell on the Worksheet. This creates the data block 'Sales1'.



3. Press the 'Expression' button in the title bar of the 'Sales1' data block. This opens the 'Expression' dialog box.

4. Enter "Revenue" as the name of the new expression column.



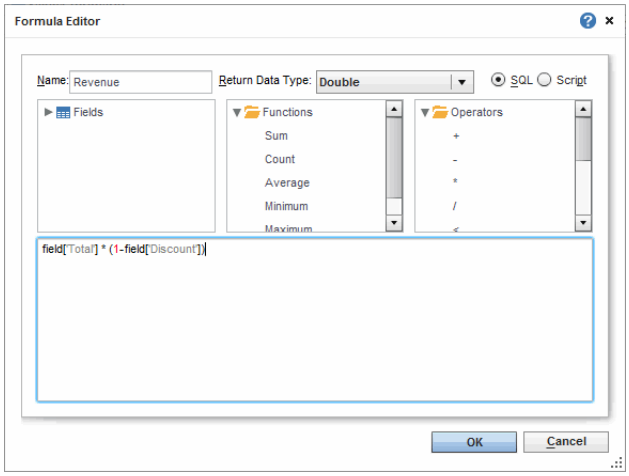
5. Press 'OK'. This appends a new 'Revenue' column to the right side of the data block, and opens the Formula Editor.

6. In the 'Return Data Type' menu, select the 'Float' option.

7. Enter the following formula in the text area and press 'OK':

```
field['Total'] * (1-field['Discounts']);
```

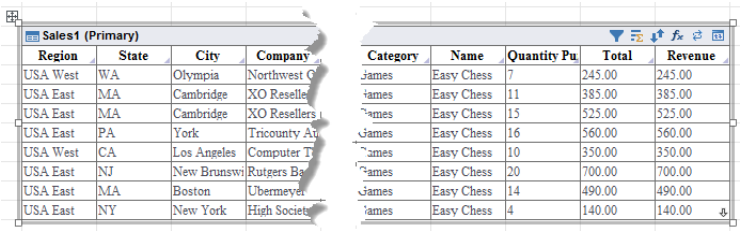
You can click on a field name in the left panel to add it to the expression area.



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.

8. (Optional) To edit the expression again at a later time, click the '*fx*' button in the expression column header (not the table title bar). This reopens the Formula Editor.

9. Preview the 'Sales1' data block and note the new 'Revenue' column.



¢

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 (Binning)

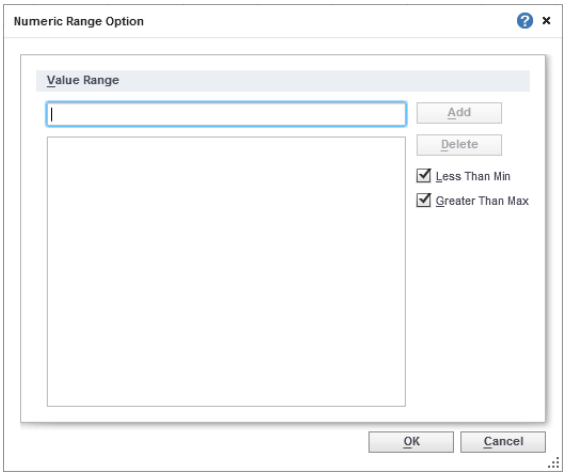
A numeric range column groups numeric data into a predefined set of bins or ranges, for example:

– 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 block. 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 ‘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. Press 'OK' to close the dialog box.

Example: Creating a Numeric Range Column

Consider the sample 'ProductSales' Worksheet, which contains a data block 'SalesByDate' that returns month-by-month sales. You can define a range column for the 'Total' field that places each amount into a predefined range or bin.

This Worksheet can be found in the 'Tutorial' folder, or in the 'Orders Examples' data set folder in Agile Edition.

1. Open the 'ProductSales' Worksheet in Visual Composer.

   For information on how to open a Data Worksheet, see *Editing a Data Worksheet in Visual Composer* (Style Scope or Style Intelligence) or *Editing a Data Set* (Style Scope Agile Edition).

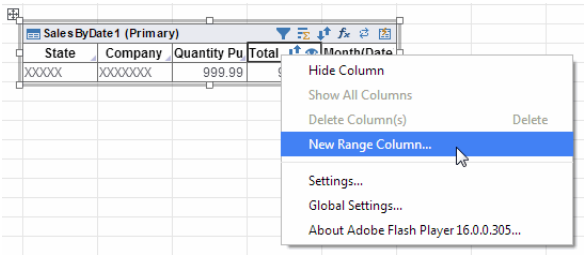
2. Right-click the 'SalesByDate' data block and select 'Set as Primary' from the context menu. This makes the 'SalesByDate' data block visible to other Worksheets.

3. Save the 'ProductSales' Worksheet.

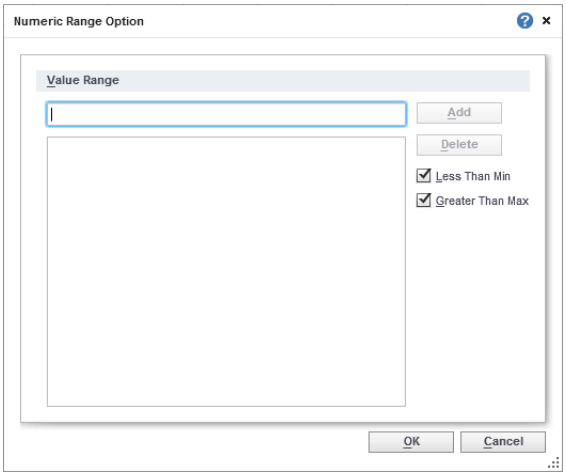
4. Create a new Worksheet.

5. Select the ‘ProductSales’ Worksheet in the **Asset** tree, and drag it onto one of the empty cells in the new Worksheet. This adds the ‘SalesByDate’ data block to the new Worksheet, and renames it to ‘SalesByDate1’.

6. Right-click the ‘Total’ column, and select the ‘New Range Column’ option. This opens the ‘Range Name’ dialog box.



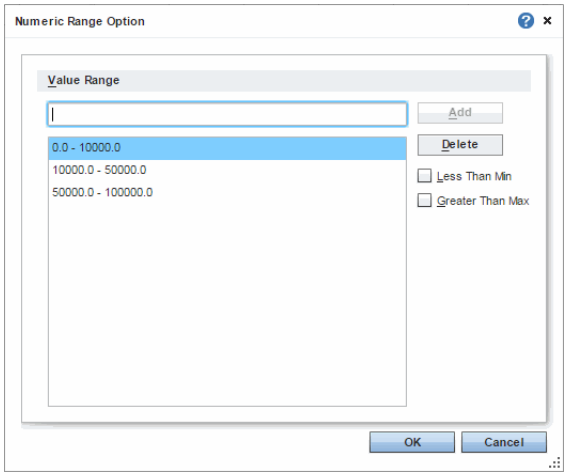
7. Enter the name “Revenue Range” and press ‘OK’. This creates the new column and opens the ‘Numeric Range Option’ dialog box, which allows you to specify the desired thresholds.



8. Deselect the ‘Less Than Min’ and ‘Greater Than Max’ options.

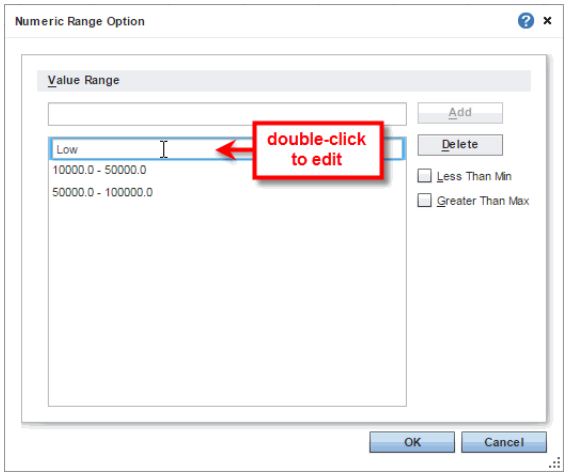
9. Enter the first threshold, 0, into the ‘Value Range’ field and press ‘Add’.

10. Enter the next three thresholds, pressing ‘Add’ after each one: 10000, 50000, 100000.



This creates the three ranges: (1) 0-10,000, (2) 10,000-50,000, and (3) 50,000-100,000.

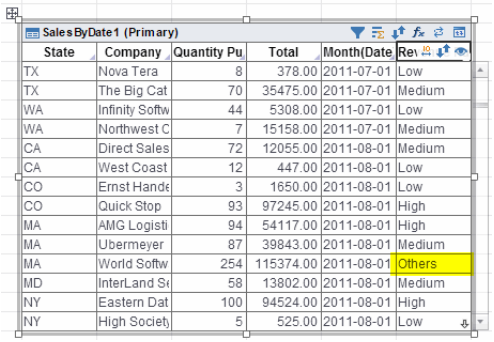
11. Double-click the '0-10,000' label and enter the new label "Low".



12. Repeat the above step to relabel the 10,000-50,000 range as "Medium" and the 50,000-100,000 range as "High".

13. Press 'OK' to close the dialog box.

14. Preview the data block. Note that each cell in the 'Total' column has a corresponding range in the 'Revenue Range' column. The values that lie outside the specified ranges are labeled 'Others'.

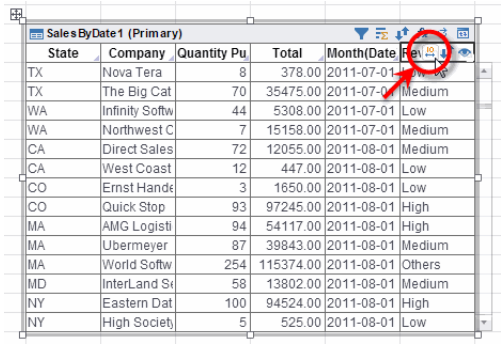


SalesByDate1 (Primary)

State	Company	Quantity Pu	Total	Month(Date	Rev
TX	Nova Tera	8	378.00	2011-07-01	Low
TX	The Big Cat	70	35475.00	2011-07-01	Medium
WA	Infinity Softw	44	5308.00	2011-07-01	Low
WA	Northwest C	7	15158.00	2011-07-01	Medium
CA	Direct Sales	72	12055.00	2011-08-01	Medium
CA	West Coast	12	447.00	2011-08-01	Low
CO	Ernst Hande	3	1650.00	2011-08-01	Low
CO	Quick Stop	93	97245.00	2011-08-01	High
MA	AMG Logisti	94	54117.00	2011-08-01	High
MA	Ubermeyer	87	39843.00	2011-08-01	Medium
MA	World Softw	254	115374.00	2011-08-01	Others
MD	InterLand Se	58	13802.00	2011-08-01	Medium
NY	Eastern Dat	100	94524.00	2011-08-01	High
NY	High Society	5	525.00	2011-08-01	Low

15. (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:

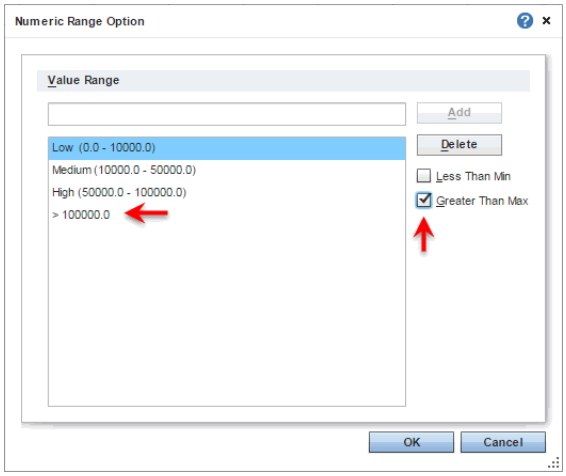
    a. Open the ‘Numeric Range Option’ dialog box by pressing the range button in the column header (see *Editing a Range Column*).



SalesByDate1 (Primary)

State	Company	Quantity Pu	Total	Month(Date	Rev
TX	Nova Tera	8	378.00	2011-07-01	
TX	The Big Cat	70	35475.00	2011-07-01	Medium
WA	Infinity Softw	44	5308.00	2011-07-01	Low
WA	Northwest C	7	15158.00	2011-07-01	Medium
CA	Direct Sales	72	12055.00	2011-08-01	Medium
CA	West Coast	12	447.00	2011-08-01	Low
CO	Ernst Hande	3	1650.00	2011-08-01	Low
CO	Quick Stop	93	97245.00	2011-08-01	High
MA	AMG Logisti	94	54117.00	2011-08-01	High
MA	Ubermeyer	87	39843.00	2011-08-01	Medium
MA	World Softw	254	115374.00	2011-08-01	Others
MD	InterLand Se	58	13802.00	2011-08-01	Medium
NY	Eastern Dat	100	94524.00	2011-08-01	High
NY	High Society	5	525.00	2011-08-01	Low

    b. Select the ‘Greater than Max’ option, and press ‘OK’.



c. Preview the data block and note that the values outside the specified range (formerly labelled ‘Others’) are now labelled ‘>100000’.

¢

See Also

Creating a Named Grouping, for another way of partitioning column data.
Creating a Date Range Column for partitioning date information.

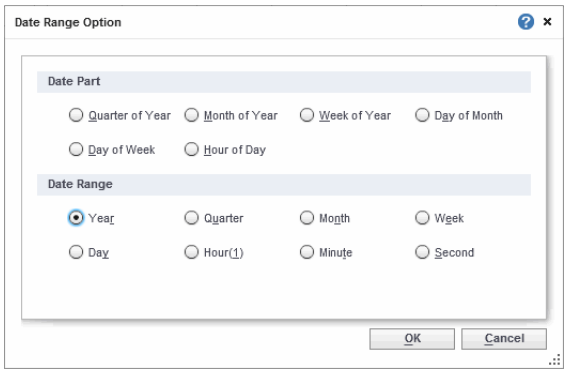
3.3.7 Creating a Date Range Column

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 block.

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 press ‘OK’. This opens the ‘Date Range Option’ dialog box.



3. Select the desired partitioning (year, month, quarter, etc.), and press ‘OK’.

See *Creating a Numeric Range Column (Binning)* 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 Data Block 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

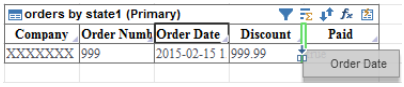
```
Warning: If there is a Viewsheet that uses on this Worksheet
column, renaming the column will cause the Viewsheet to lose its
reference.
```

To change a column name, double-click the column header and enter the desired text.

3.4.2 Reordering a Column

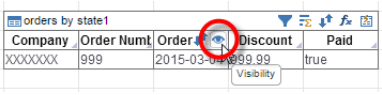
To change the position of a column, click on the column header cell, and drag the column to the desired location in the table. A green bar shows the location where the column will be inserted.

In the figure below, the ‘Order Date’ column is positioned between the ‘Discount’ and ‘Paid’ columns.

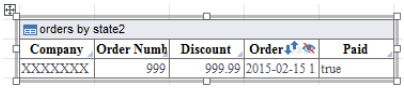


3.4.3 Showing and Hiding a Column

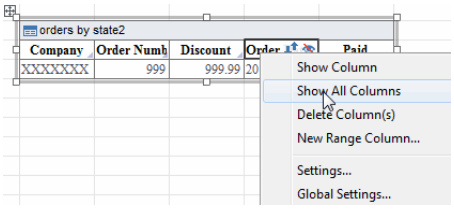
To change the visibility of a column (show or hide), click the ‘visibility’ button next to the column name in the column header cell.



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.

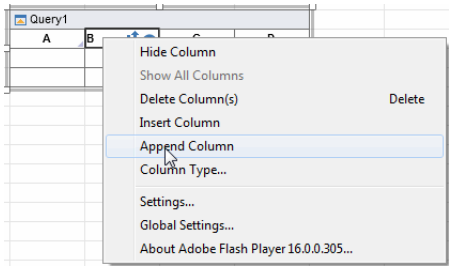
Example: Changing the Column Type

In the following example, you will change the data type of an embedded table column from ‘String’ to ‘Integer’:

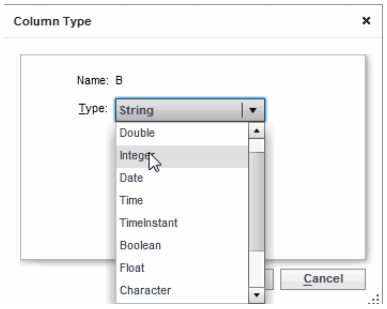
1. Create a new Worksheet.

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.

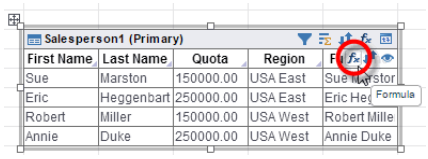
Format Tab, in *Advanced Dashboard Design*, for more information on Viewsheet formatting.

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. Press the ‘Formula’ button in the column header. This opens the Formula Editor.



3. Make the desired changes to the formula, and press ‘OK’.

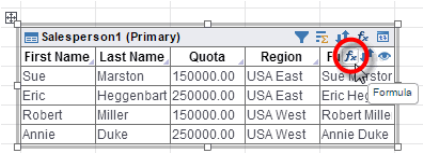
See *Creating an Expression/Formula Column* for more information about using the Formula Editor.

3.4.6 Editing a Range Column

To edit a date range column or numeric range column, follow these steps:

1. Click the column header to select it.

2. Press the Date Range or Numeric Range button in the column header.



This opens the 'Numeric Range Option' or 'Date Range Option' dialog box. See *Creating a Numeric Range Column (Binning)* and *Creating a Date Range Column* for more details about using these dialog boxes.

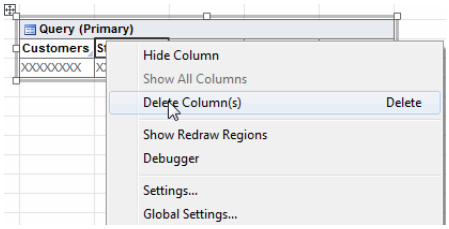
3.5 Deleting a Table Column

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.

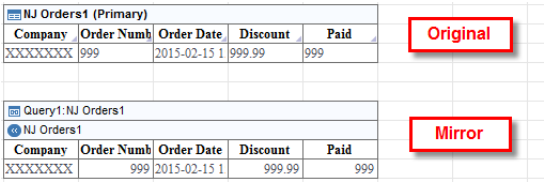
A new table is created in the Worksheet at that location. Although the newly created table is a copy of the existing table, the two tables are completely independent of each other. Therefore, changes made to one table do not affect the other table.

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.



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.

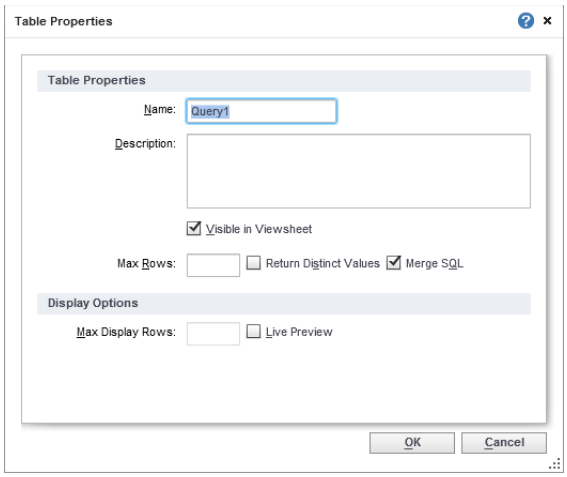
3.8.1 Setting Table Query and Display Properties

You can change various properties of a data block, 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.



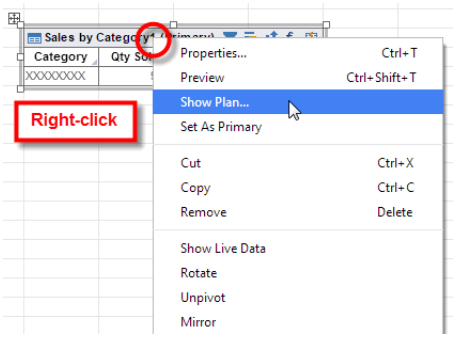
The table properties are described below.

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 Block* 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 Data Block* and *Showing Live Data in a Table*.)
Return Distinct Values	Display only distinct rows in the table. (Remove duplicate rows.)

PROPERTY	DESCRIPTION
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.

```
This data source is not available in Style Scope Agile Edition.
```

1. Expand the 'TABLE' > 'SA' node of the 'Orders' data source.

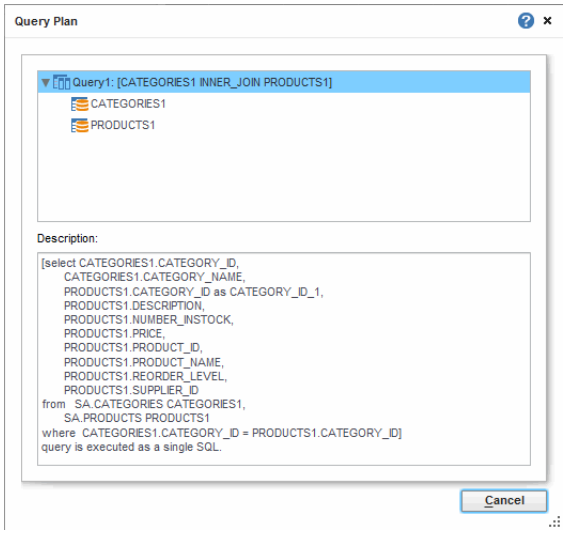
2. Drag the 'PRODUCTS' table to an empty location on the Worksheet.

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

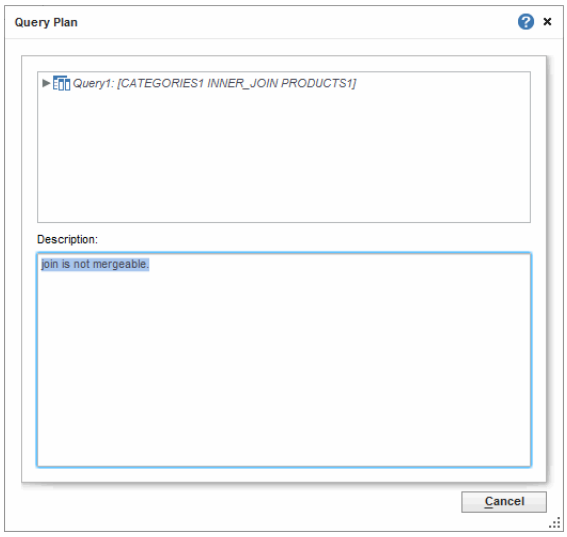
database as a single SQL statement, which is also indicated by the single icon in the top panel.

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.

¢

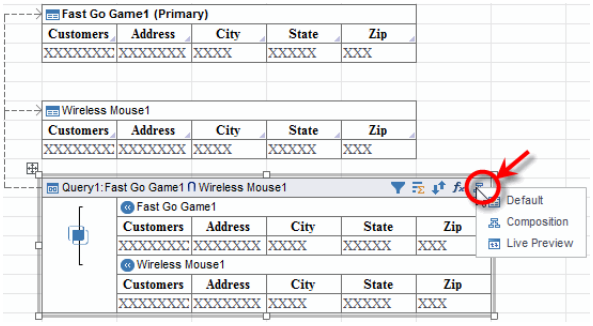
3.8.3 Editing a Composition Table

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 block. 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 returning customers who purchased the Fast Go Game product and one returning customers who purchased the Wireless Mouse product. Both of these tables have a column named 'Zip' that you would like to hide. They also both have a column named 'Customers' that you would like to rename to 'Company'.

To create this Composition Table, and then hide and rename columns, follow the steps below:

1. Create a new Worksheet.

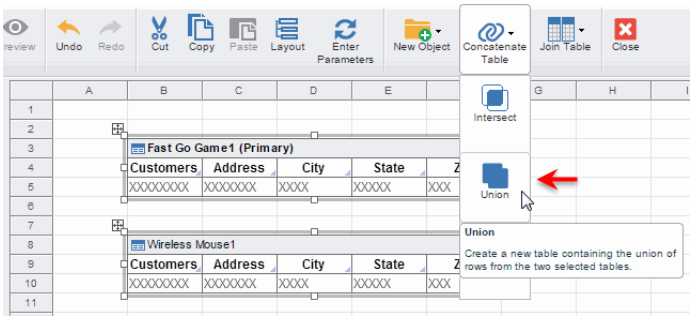
These data sets can be found in the 'Sample Queries' folder, or in the 'Orders Examples' data set folder in Agile Edition.

2. Drag the 'Fast Go Game' query from the **Asset** panel onto one of the empty cells in the Worksheet. This creates a new data block, named 'Fast Go Game1'.

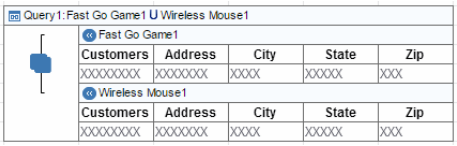
3. Drag the 'Wireless Mouse' query from the **Asset** panel onto one of the empty cells in the Worksheet. This creates a new data block, named 'Wireless Mouse1'.

You can also drag the 'Fast Go Game' block immediately above or below the 'Wireless Mouse' block. Release when you see the 'union' symbol appear.

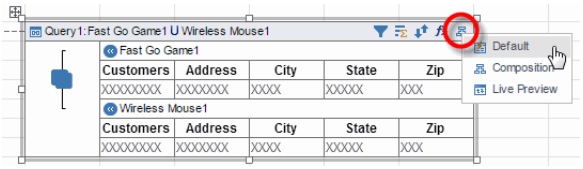
4. Drag across both data blocks to select them. Press the 'Concatenate Table' button in the toolbar and select 'Union'.



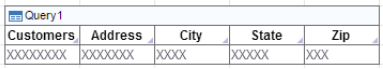
This creates a new Concatenated Table, ‘Query1’. Notice that the ‘Query1’ table is in the ‘Composition’ view since the ‘union’ symbol is visible on the left of the table, indicating its type, and the table’s title row lists all of the tables participating in the union operation.



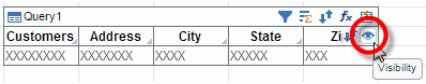
5. Press the ‘Change View’ button in the table title bar, and select ‘Default’ from the menu.



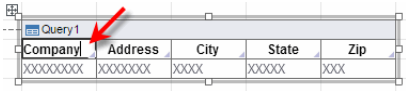
This changes the view to default view, which allows editing of columns.



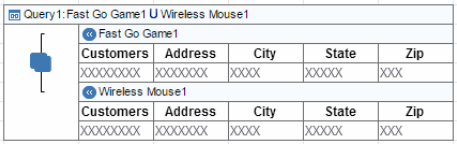
6. Press the ‘Visibility’ button in the ‘Zip’ cell header to hide the column.



7. Double-click the ‘Customers’ header, and change the name to ‘Company’.

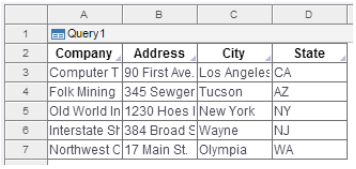


8. Switch back to Composition view by pressing the ‘Change View’ button in the title bar. Select ‘Composition’ from the menu.



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.

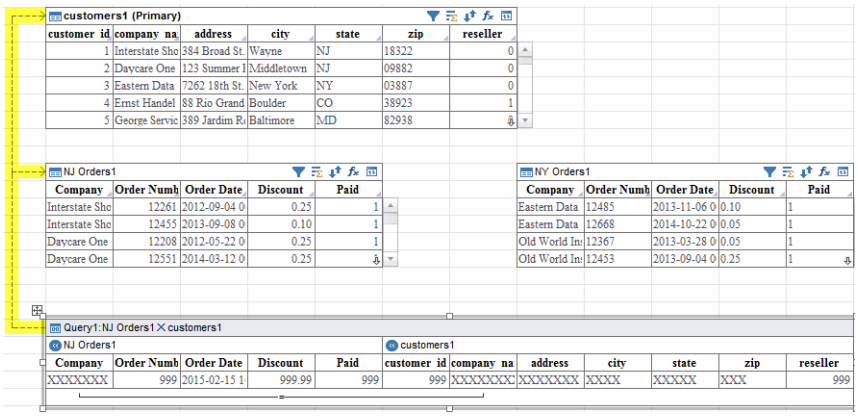
9. Preview the table. Notice that the ‘Zip’ field is hidden and the ‘Customer’ column header has been renamed to ‘Company’.



¢

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:

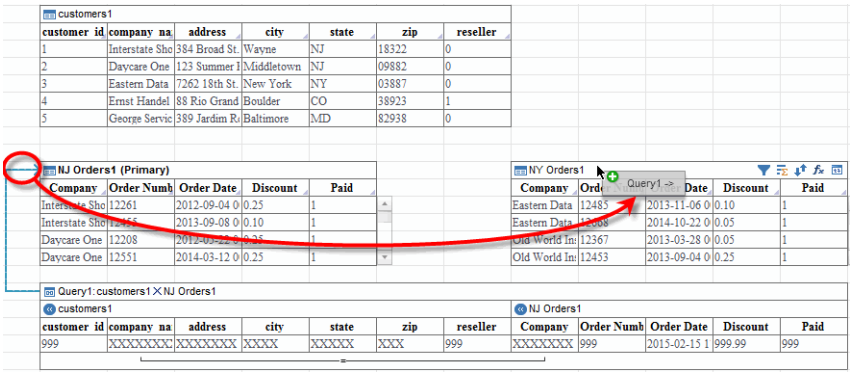
```
Note: The new data block must be compatible with the data block it
replaces. See below.
```

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.

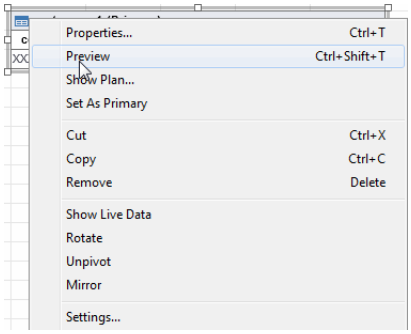
- For a Concatenated Table, the number of columns and the data types in the replacement data block must match the old data block.

See Also

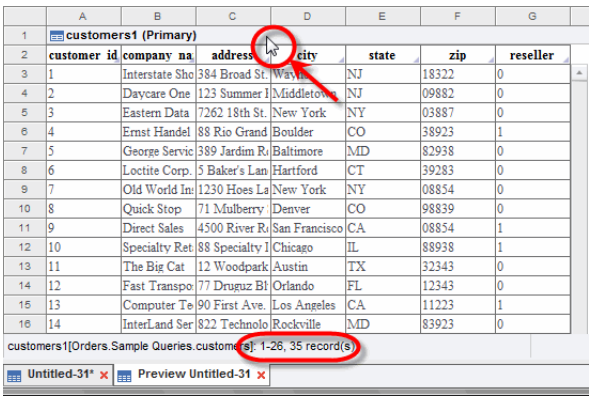
Changing the Join Type, to change the existing join operator.
Changing the Concatenation Type, to change the existing concatenation operator.

3.8.5 Previewing a Data Block

To preview a data block, right-click the 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 Block, for information about creating a new table.
Previewing a Data Block, 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

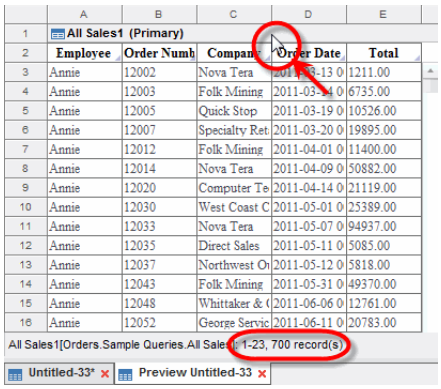
The length of a table is the number of records that the table retrieves from the database. You can check this length in two ways, by using Live Preview mode or by using Preview.

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.

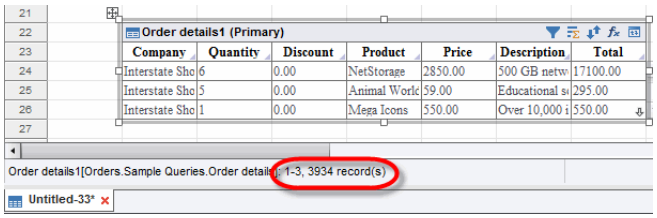
2. Open the ‘Table Properties’ dialog box.

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).



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

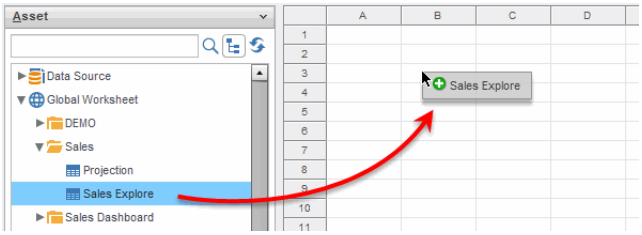
To understand how to unpivot a crosstab, first create a crosstab by following the steps below:

`If you are using Style Scope Agile Edition, see Creating a Data Set by Mash-Up to create a new Worksheet.`

1. Create a new Worksheet by pressing the 'New Worksheet' button.

This Worksheet can be found in the 'Sales' folder, or in the 'Orders Examples' data set folder in Agile Edition.

2. From the **Asset** panel, drag the 'Sales Explore' Worksheet onto an empty cell on the Worksheet. This creates the data block 'Sales1'.



You will use this table to create a crosstab.

3. Press the 'Group and Aggregate' button in the table's title bar. This opens the 'Group and Aggregate' dialog box.



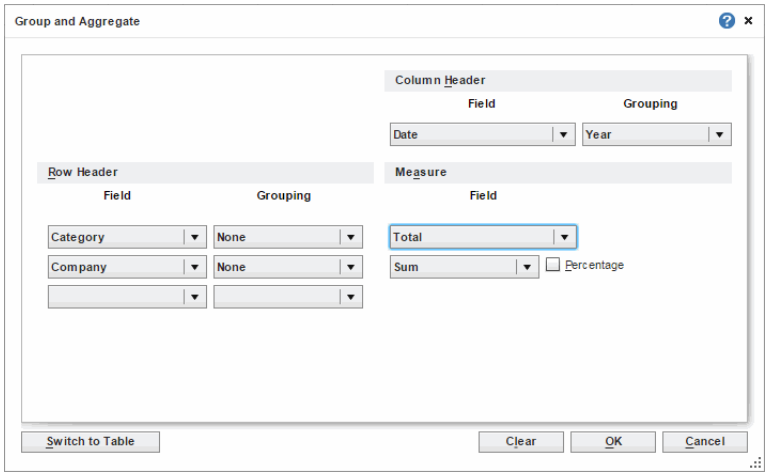
4. Press the 'Switch to Crosstab' button at the bottom of the dialog box. This updates the dialog box to display the crosstab binding interface.

5. In the 'Row Header' region, select the following two fields: 'Category' and 'Company'.

6. In the 'Column Header' region, select the 'Date' field.

7. In the menu next to the 'Date' field, select 'Year'.

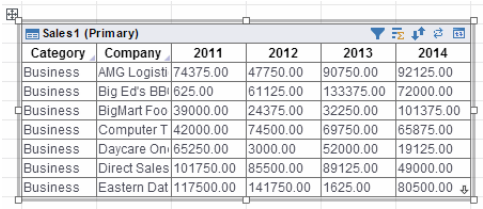
8. In the 'Measure' region, select the 'Total' field.



9. Press ‘OK’. This converts the data block to a crosstab.

10. In the crosstab title bar, press the ‘Change View’ button, and select ‘Live Preview’ from the menu.

11. Drag the bottom border of the table to expand the table.

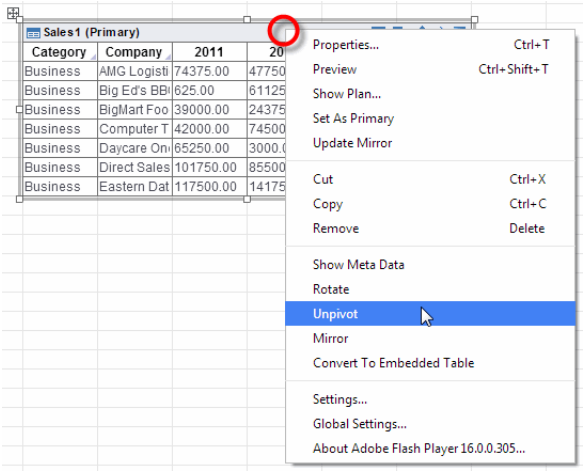


Category	Company	2011	2012	2013	2014
Business	AMG Logisti	74375.00	47750.00	90750.00	92125.00
Business	Big Ed's BBI	625.00	61125.00	133375.00	72000.00
Business	BigMart Foo	39000.00	24375.00	32250.00	101375.00
Business	Computer T	42000.00	74500.00	69750.00	65875.00
Business	Daycare On	65250.00	3000.00	52000.00	19125.00
Business	Direct Sales	101750.00	85500.00	89125.00	49000.00
Business	Eastern Dat	117500.00	141750.00	1625.00	80500.00

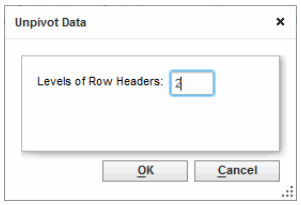
This crosstab has two levels of row headers (‘Category’ and ‘Company’), and one level of column headers (‘Order Date’, broken out by year).

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.



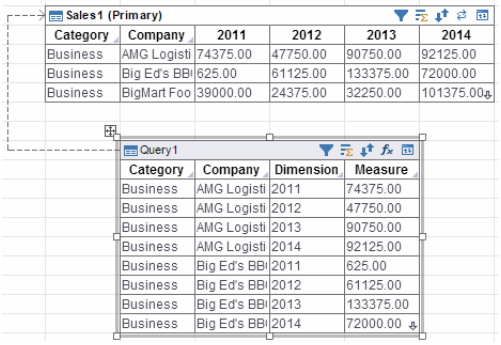
2. In the ‘Unpivot Data’ dialog box, enter a value of “2” for the ‘Levels of row headers’ value. (The two levels are ‘Category’ and ‘Company’.)



3. Press ‘OK’ to close the dialog box. This creates a new table containing the unpivoted (flattened) data.

4. In the table title bar, press the ‘Change View’ button and select ‘Live Preview’.

5. Drag the bottom border of the table to expand the table.



Observe that the unpivoting operation transforms the values of the column header in the crosstab (‘Date’) into a new column called ‘Dimension’. The new unpivoted table retains the row headers of the original crosstab in the first two columns (‘Category’ and ‘Company’), but now repeats each combination of ‘Category’ and ‘Company’ for each value of ‘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 Blocks

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 blocks 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.

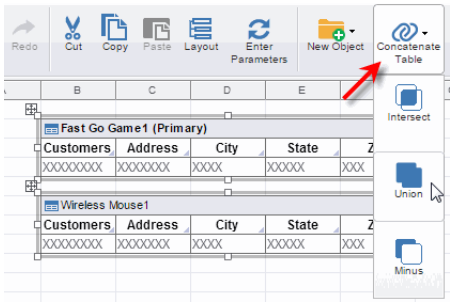
4.1.1 Creating a Concatenated Table

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

One query returns a list of customers who have purchased more than 100 pieces of 'Wireless Mouse' and a second query returns a list of customers who have purchased more than 100 pieces of 'Fast Go Game'. To see the contact information of all customers who have purchased more than 100 pieces of either 'Wireless Mouse' or 'Fast Go Game' in a single data block, perform a union. Follow the steps below:

`If you are using Style Scope Agile Edition, see Creating a Data Set by Mash-Up to create a new Worksheet.`

1. Create a new Worksheet by clicking on the 'New Worksheet' button.

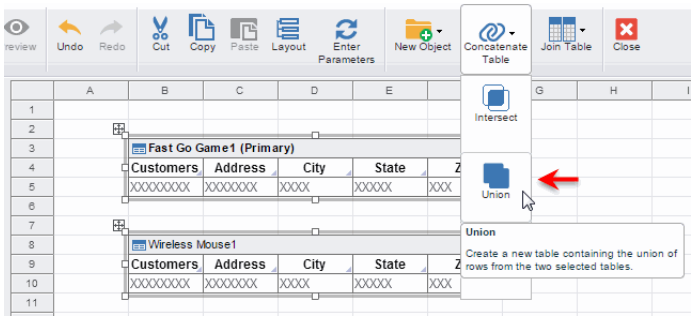
These data sets can be found in the 'Sample Queries' folder, or in the 'Orders Examples' data set folder in Agile Edition.

2. Drag the 'Fast Go Game' query from the **Asset** panel onto one of the empty cells in the Worksheet. This creates a new data block, named 'Fast Go Game1'.

3. Drag the 'Wireless Mouse' query from the **Asset** panel onto one of the empty cells in the Worksheet. This creates a new data block, named 'Wireless Mouse1'.

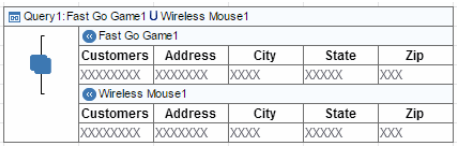
You can also drag the 'Fast Go Game' block immediately above or below the 'Wireless Mouse' block. Release when you see the 'union' symbol appear.

4. Drag across both data blocks to select them. Press the 'Concatenate Table' button in the toolbar and select 'Union'.



This creates a new Concatenated Table, 'Query1'. Notice that the 'Query1' table is in the 'Composition' 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.



5. To preview 'Query1', right-click on the title bar and select 'Preview'. 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	Query1				
2	**Customers**	**Address**	**City**	**State**	**Zip**
3	Computer Te	90 First Ave.	Los Angeles	CA	11223
4	Folk Mining	345 Sewgen L	Tucson	AZ	11234
5	Old World In	1230 Hoes La	New York	NY	08854
6	Interstate Sho	384 Broad St.	Wayne	NJ	18322
7	Northwest Or	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.

`If you are using Style Scope Agile Edition, see Creating a Data Set by Mash-Up to create a new Worksheet.`

1. Create a new Worksheet by clicking on the 'New Worksheet' button

These data sets can be found in the 'Sample Queries' folder, or in the 'Orders Examples' data set folder in Agile Edition.

2. Drag the 'Fast Go Game' query from the **Asset** panel onto one of the empty cells in the Worksheet. This creates a new data block, named 'Fast Go Game1'.

3. Drag the 'Wireless Mouse' query from the **Asset** panel onto one of the empty cells in the Worksheet. This creates a new data block, named 'Wireless Mouse1'.

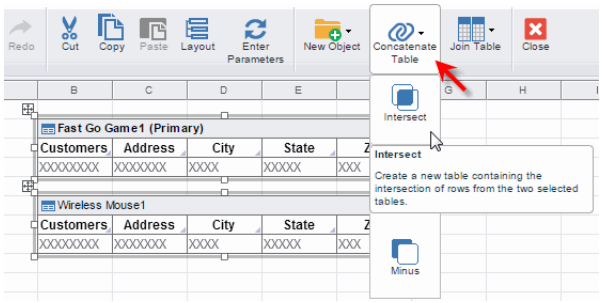
Fast Go Game1 (Primary)

Customers	Address	City	State	Zip
XXXXXXX	XXXXXXX	XXXX	XXXXX	XXX

Wireless Mouse1

Customers	Address	City	State	Zip
XXXXXXX	XXXXXXX	XXXX	XXXXX	XXX

4. Drag across both data blocks to select them. Press the ‘Concatenate Table’ button in the toolbar and select ‘Intersect’.



5. This creates a new table, ‘Query1’. 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).

Query1: Fast Go Game1 ∩ Wireless Mouse1

Fast Go Game1

Customers	Address	City	State	Zip
XXXXXXX	XXXXXXX	XXXX	XXXXX	XXX

Wireless Mouse1

Customers	Address	City	State	Zip
XXXXXXX	XXXXXXX	XXXX	XXXXX	XXX

6. To preview ‘Query1’, right-click on the title bar and select ‘Preview’. Only those customers who purchased more than 100 pieces of both ‘Wireless Mouse’ and ‘Fast Go Game’ are included in the table.

Query1

Customers	Address	City	State	Zip
Old World In	1230 Hoes I	New York	NY	08854
Folk Mining	345 Sewger	Tucson	AZ	11234

¢

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 obtain that result.

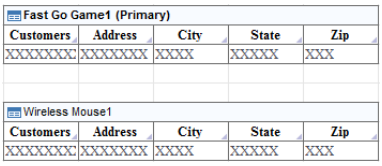
If you are using Style Scope Agile Edition, see *Creating a Data Set by Mash-Up* to create a new Worksheet.

1. Create a new Worksheet by pressing the ‘New Worksheet’ button.

These data sets can be found in the ‘Sample Queries’ folder, or in the ‘Orders Examples’ data set folder in Agile Edition.

2. Drag the ‘Fast Go Game’ query from the **Asset** panel onto one of the empty cells in the Worksheet. This creates a new data block, named ‘Fast Go Game1’.

3. Drag the ‘Wireless Mouse’ query from the **Asset** panel onto one of the empty cells in the Worksheet. This creates a new data block, named ‘Wireless Mouse1’.

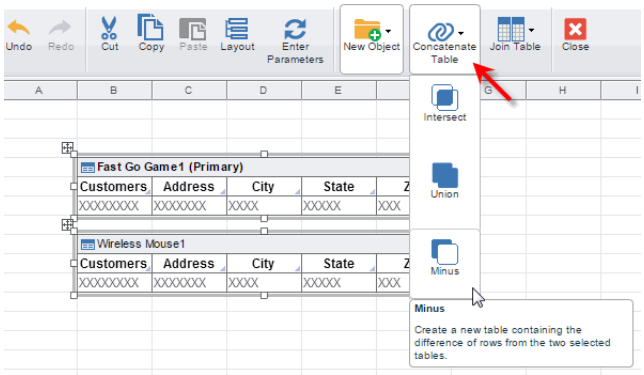


For a Minus operation, the order in which you select the data blocks is important.

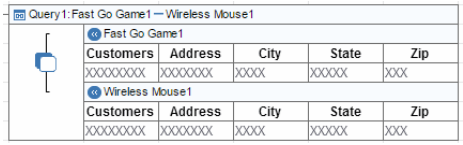
4. First select ‘Fast Go Game1’ by clicking the table’s title bar.

5. Now select ‘Wireless Mouse1’ by Ctrl-clicking the table’s title bar (It is important to select the ‘Wireless Mouse1’ table after selecting the ‘Fast Go Game1’ table.)

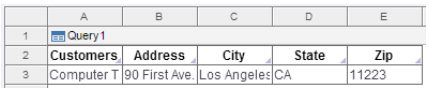
6. Press the ‘Concatenate Table’ button in the toolbar and select ‘Minus’. This creates a new table, ‘Query1’.



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).



7. To preview ‘Query1’, right-click on the title bar and select ‘Preview’. 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.



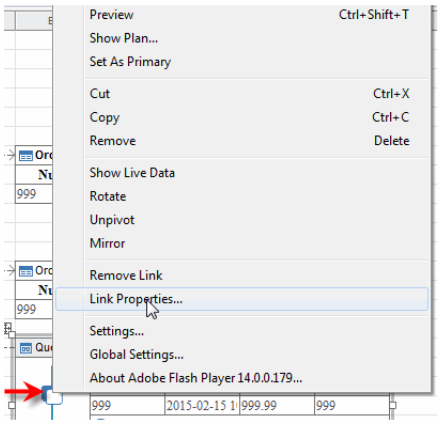
¢

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.

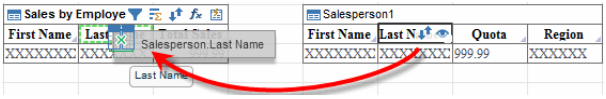
You can create an inner join either by dragging columns or by using the 'Join Table' button.

Creating a Inner Join by Dragging Columns

To create an inner join between two base tables by dragging columns, follow these steps:

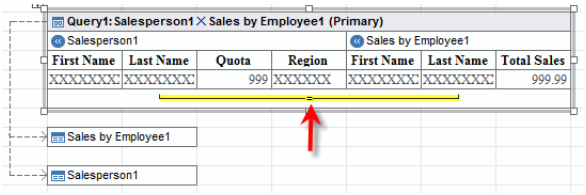
1. Identify the join columns, one from each table. These are the columns to which the join condition is applied.

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*.



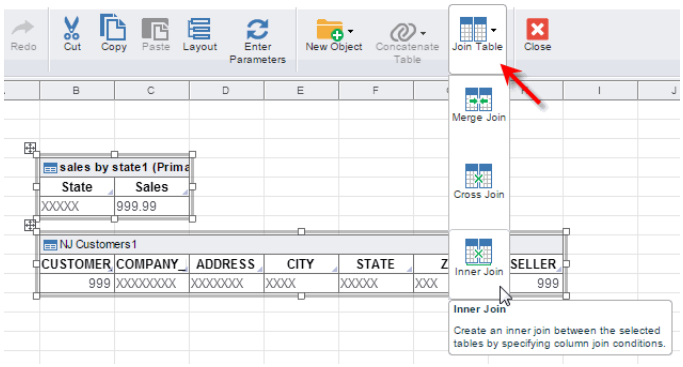
4. If desired, repeat the above steps to implement multiple join conditions.

Creating an Inner Join with the 'Join Table' Button

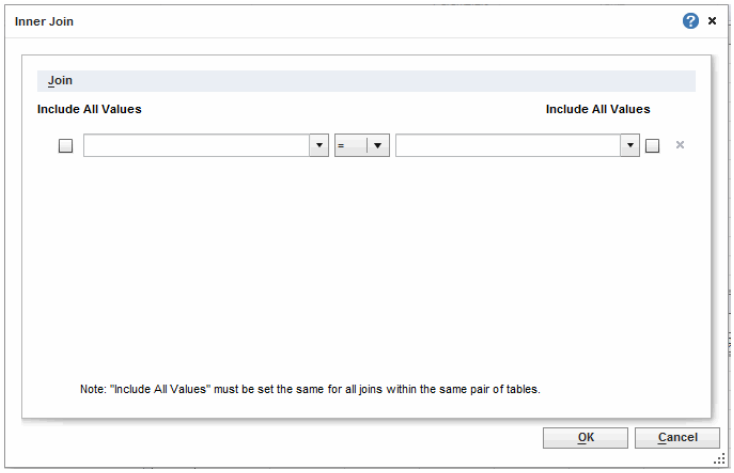
To create an inner join between two base tables, follow these steps:

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

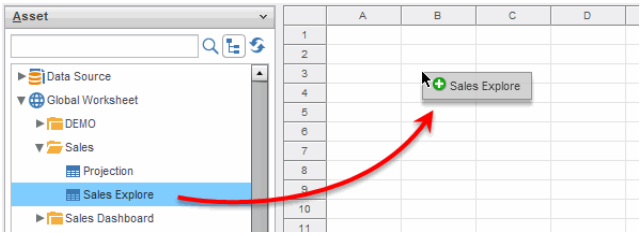
The sample 'Sales Explore' Worksheet contains individual order information. If you want to add customer information such as addresses, you need to join this data block to the data set returned by the sample 'customers' query. Follow the steps below:

`If you are using Style Scope Agile Edition, see Creating a Data Set by Mash-Up to create a new Worksheet.`

1. Create a new Worksheet by pressing the 'New Worksheet' button.

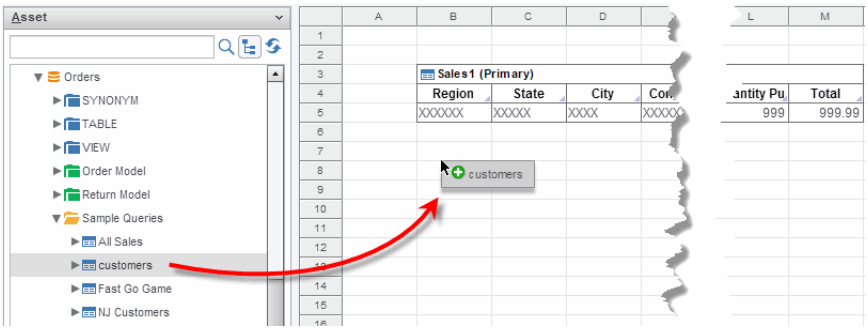
This Worksheet can be found in the 'Sales' folder, or in the 'Orders Examples' data set folder in Agile Edition.

2. From the **Asset** panel, drag the 'Sales Explore' Worksheet onto an empty cell on the Worksheet. This creates the data block 'Sales1'.

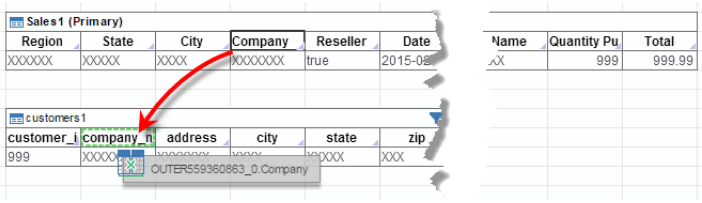


This query can be found in the 'Sample Queries' folder, or in the 'Orders Examples' data set folder in Agile Edition.

3. From the **Asset** panel, drag the 'customers' query onto an empty cell on the Worksheet. This creates the data block 'customers1'.

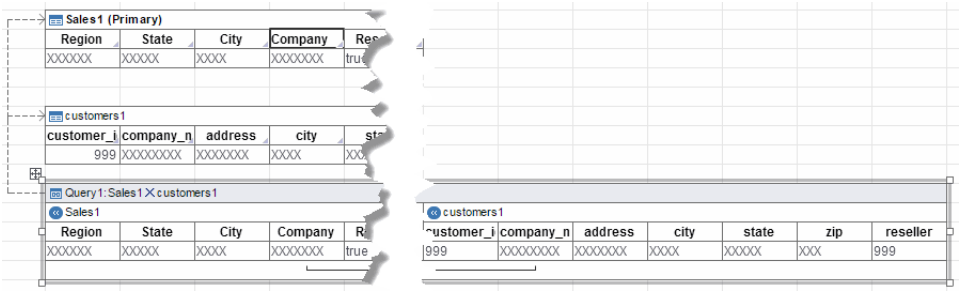


4. Drag the 'Company' column header from the 'Sales1' data block onto the 'company_name' column header of the 'customers1' data block. Notice that the cursor changes to a join icon.

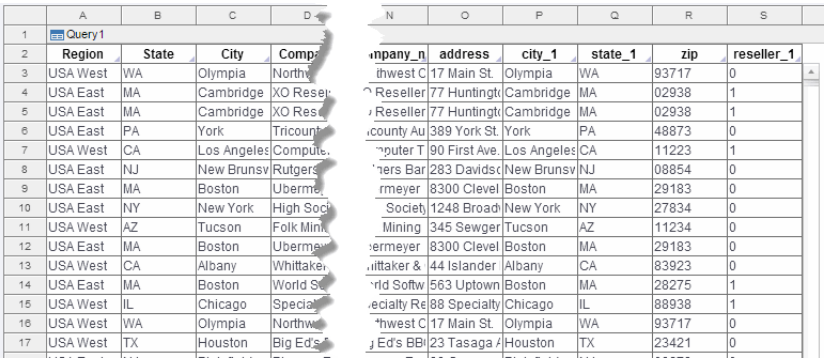


Notice that the connector line on the bottom row of the table shows the join relationship by connecting the join columns together.

5. Drop the ‘Company’ column header on the ‘company_name’ column header to create the join. This creates a new joined table, ‘Query1’.



6. To preview the ‘Query1’ data block, right-click on the title row and select ‘Preview’.



¢

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.

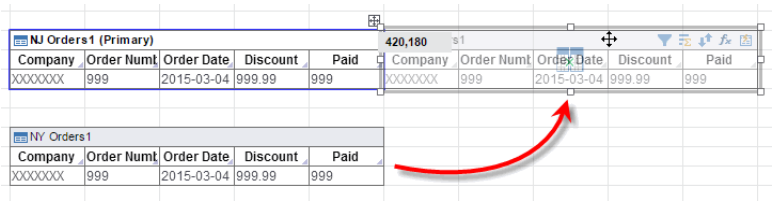
4.2.3 Cross Join

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.

Example: Cross Join

The 'customers' query returns information about different companies. Consider a case where you want to mash up this company information with salesperson information such that each row of company data is repeated for each of several different salespeople. This requires a cross-join, and will yield a result whose length is equal to the product of the rows of the two base tables.

`If you are using Style Scope Agile Edition, see` *`Creating a Data Set by Mash-Up`* `to create a new Worksheet.`

1. Create a new Worksheet by pressing the 'New Worksheet' button.

2. From the **Asset** panel, drag the 'customers' query onto an empty cell on the Worksheet. This creates the data block 'customers1'.

This query can be found in the 'Sample Queries' folder, or in the 'Orders Examples' data set folder in Agile Edition.

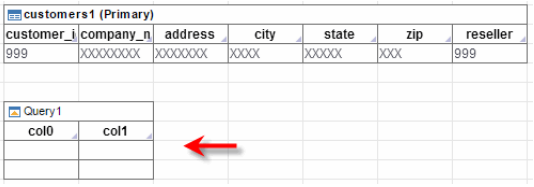
3. Create an embedded table containing the salesperson information. Follow the steps below:

   a. Press the 'New Object' button in the toolbar, and select 'Embedded Table'. The mouse changes to a crosshairs.

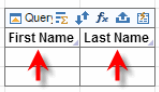
   b. Drag with the crosshairs to highlight four rows and two columns on the grid, and release.



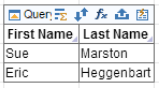
   This creates a new embedded table called 'Query1'.



4. Double-click on the 'col0' column header, and edit the column name to be 'First Name'. Double-click on the 'col1' column header and edit the column name to be 'Last Name'.



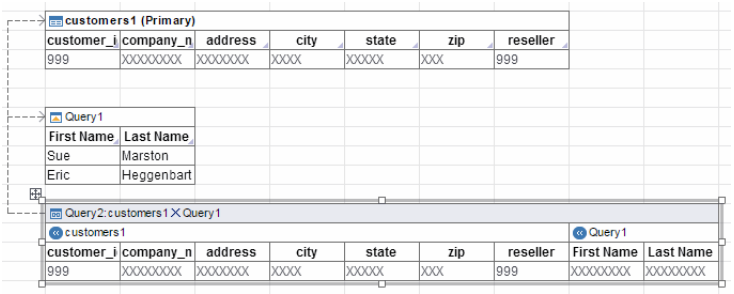
5. In the first row of the 'First Name' column, enter "Sue". In the first row of the 'Last Name' column, enter "Marston". In the second row of the 'First Name' column, enter 'Eric'. In the second row of the 'Last Name' column, enter "Heggenbart".



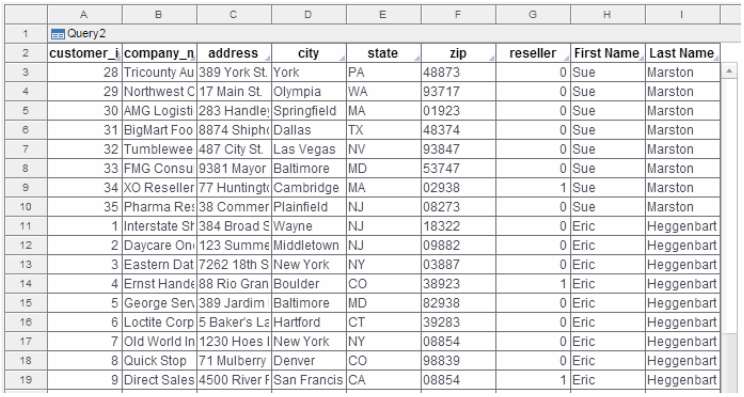
6. Join the two data blocks using a cross-join. To do this, click on the title bar of the 'Query1' data block, and drag the entire data block right next to the 'customers1' data block.



7. When the cross-join icon appears, release the mouse to create the join. This creates a new joined table, ‘Query2’.



8. To preview the ‘Query2’ table, right-click on the title bar and select ‘Preview’. Notice that the ‘Query1’ table contains the cross-product of the data from the ‘customers1’ and ‘Query1’ tables. That is, every row in the ‘Query1’ table is paired with every row in the ‘customers1’ table.



¢

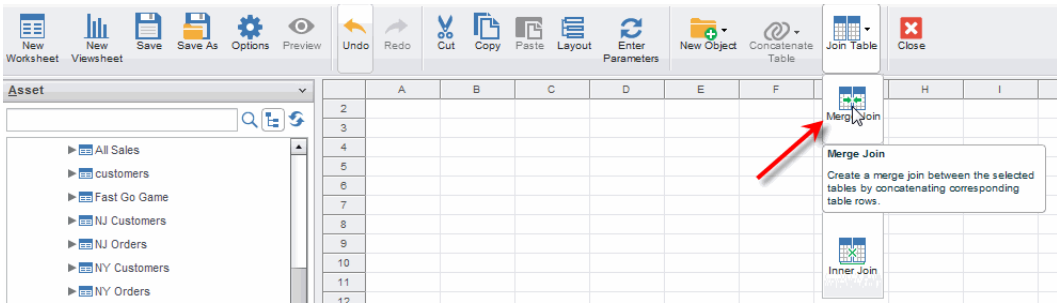
4.2.4 Merge Join

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.

To create a merge join, follow these steps:

1. Select both tables by Ctrl-clicking on each table's title row, or by dragging across both tables.

2. Press 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

Place orders of Wireless Mouse and Fast Go Game side-by-side within a dingle data block by using a merge join.

1. Create a new Worksheet.

These data sets can be found in the 'Sample Queries' folder, or in the 'Orders Examples' data set folder in Agile Edition.

2. Drag the 'Fast Go Game' query from the **Asset** panel onto one of the empty cells in the Worksheet. This creates a new data block, named 'Fast Go Game1'.

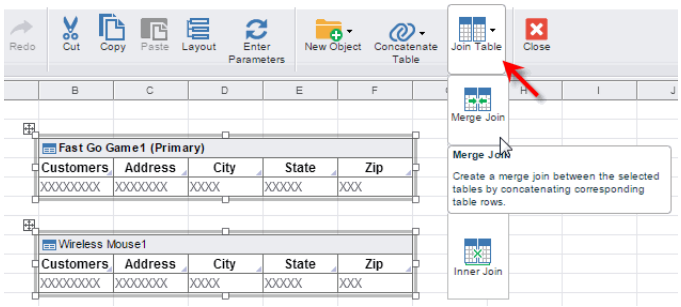
3. Drag the 'Wireless Mouse' query from the **Asset** panel onto one of the empty cells in the Worksheet. This creates a new data block, named 'Wireless Mouse1'.

Fast Go Game1 (Primary)				
Customers	Address	City	State	Zip
XXXXXXX	XXXXXXX	XXXX	XXXXX	XXX

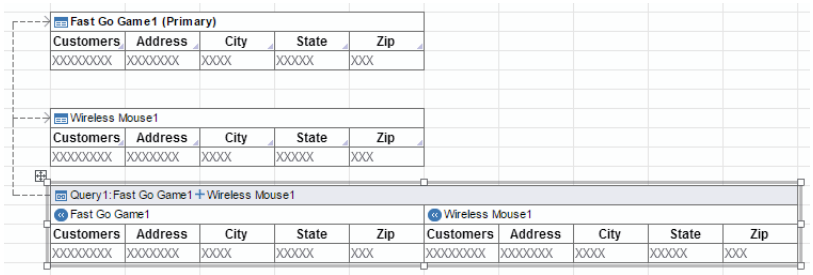
Wireless Mouse1				
Customers	Address	City	State	Zip
XXXXXXX	XXXXXXX	XXXX	XXXXX	XXX

4. Select both data blocks by holding down the Ctrl key and clicking each one.

5. Press the 'Join Table' button on the toolbar and select 'Merge Join'.

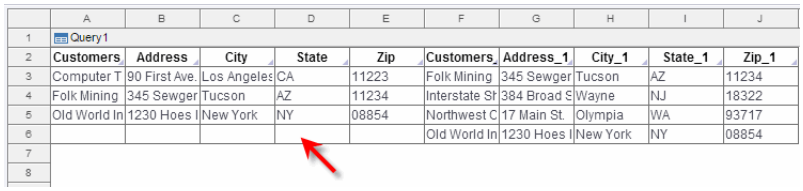


This creates a new data block, ‘Query1’.



Since the merge join does not create any relationship between the tables, there is no connector shown. This join type, therefore, cannot be changed.

6. To preview ‘Query1’, right-click on the title bar and select ‘Preview’. Notice that since the ‘Wireless Mouse1’ table has more rows than the ‘Fast Go Game1’ table, the number of rows of the ‘Query1’ table is equal to the number of rows of the ‘Wireless Mouse1’ table. The ‘Fast Go Game1’ data is padded with an empty row.



¢

4.2.5 Changing the Join Type

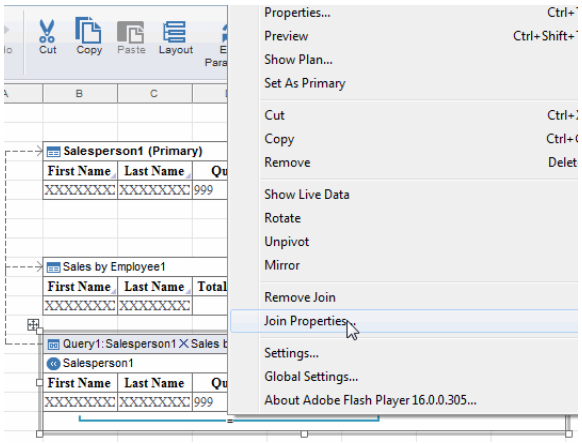
The join type can be changed for all joined tables except for Merge-joins and Cross-joins. (Merge joins and cross joins do not define a relationship between tables, so there is no relationship that can be changed.)

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:

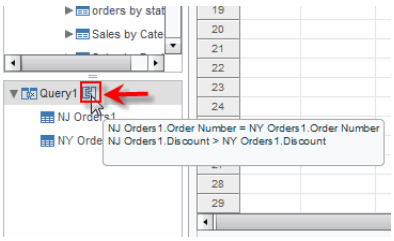
   a. Click the join connector link to select it. The selected link is highlighted in blue. Right-click on the highlighted join connector link and select ‘Join Properties’ from the context menu. This opens up the ‘Join Types’ dialog box for the selected join.



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.

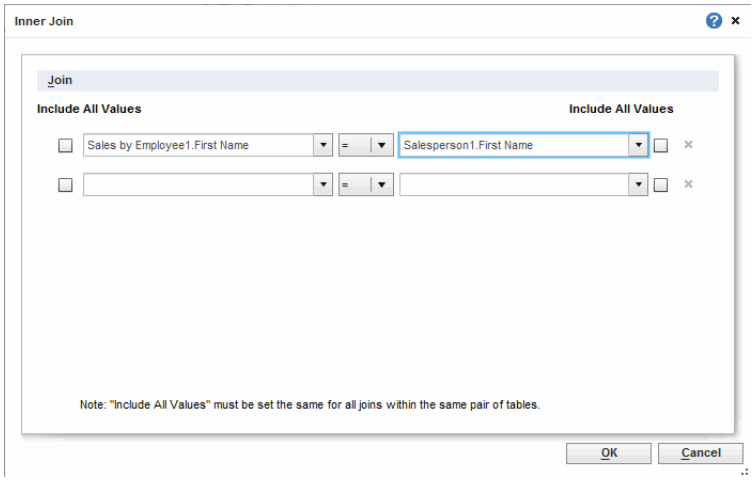


3. In the ‘Join Types’ dialog box, select the join columns from the drop-down menus for each table.

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.

`If you are using Style Scope Agile Edition, see` *`Creating a Data Set by Mash-Up`* `to create a new Worksheet.`

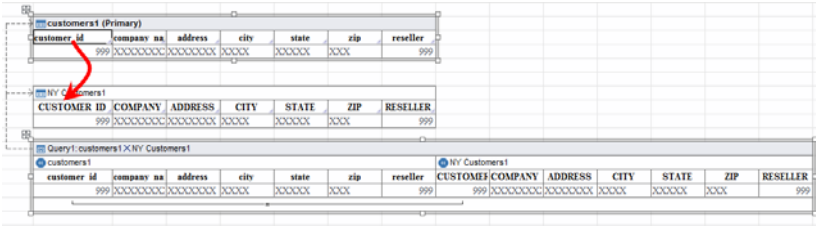
1. Create a new Worksheet by pressing the ‘New Worksheet’ button in the toolbar.

2. Drag the ‘customers’ query from the **Asset** tree to an empty cell in the Worksheet. This creates a new data block named ‘customers1’.

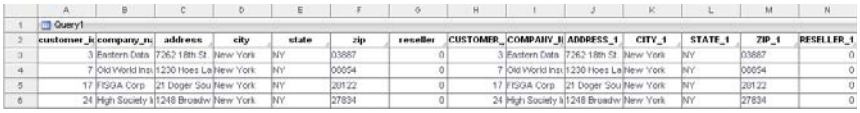
   This query can be found in the ‘Sample Queries’ folder, or in the ‘Orders Examples’ data set folder in Agile Edition.

3. Drag the ‘NY Customers’ query from the **Asset** tree to an empty cell in the Worksheet. This creates a new data block named ‘NY Customers1’.

4. Drag the ‘customer_id’ column header from ‘customers1’ onto the ‘CUSTOMER_ID’ column header of ‘NY Customers1’. This creates a new joined data block called ‘Query1’.



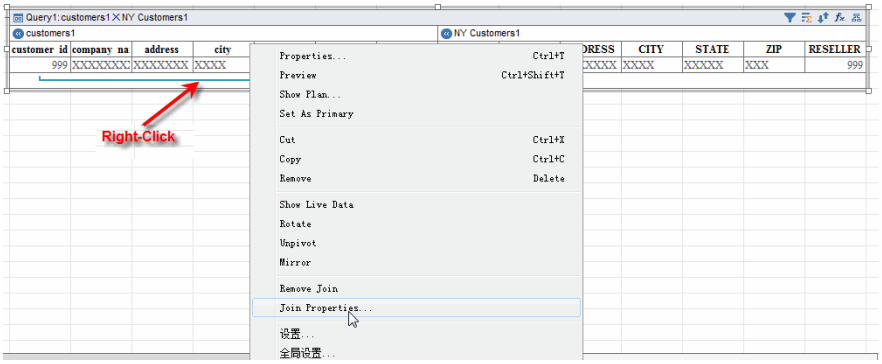
5. Right-click on the title row of the ‘Query1’ table and select ‘Preview’ from the context menu.



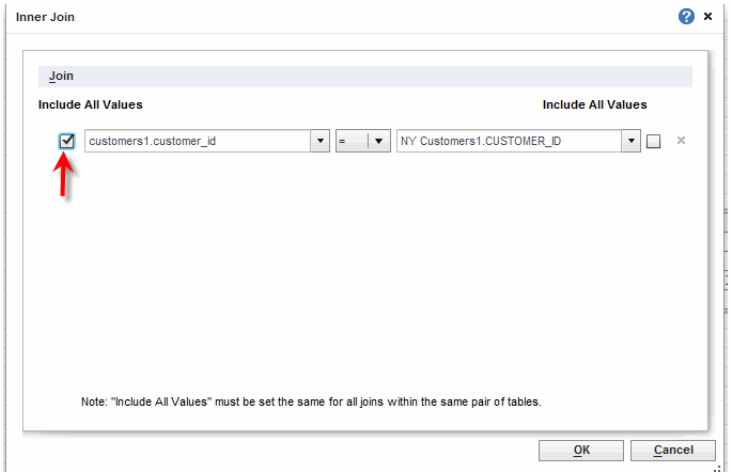
Notice that only the customers from the state of NY are listed, because the join is currently an inner join.

6. In the Data Worksheet, click the join connector in table ‘Query1’ to select it.

7. Right-click the connector and select ‘Join Properties’ from the context menu. This opens the ‘Join Types’ dialog box.



8. Select ‘include all values’ for the ‘customers1’ table and press ‘OK’.



9. Preview the table again. Notice that the two tables are now joined using a left outer join.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Query1													
2	customer_id	company_na	address	city	state	zip	reseller	CUSTOMER_	COMPANY_N	ADDRESS_1	CITY_1	STATE_1	ZIP_1	RESELLER_1
3	1	Interstate Sho	384 Broad St	Wayne	NJ	18322	0							
4	2	Daycare One	123 Summer F	Middletown	NJ	09882	0							
5	3	Eastern Data	7262 10th St	New York	NY	03887	0	3	Eastern Data	7262 10th St	New York	NY	03887	0
6	4	Ernst Handel	88 Rio Grand	Boulder	CO	38923	1							
7	5	George Servic	389 Jordan Ro	Baltimore	MD	82938	0							
8	6	Loctite Corp.	5 Baker's Lan	Hartford	CT	29283	0							
9	7	Old World Insi	1230 Hoes La	New York	NY	00854	0	7	Old World Insi	1230 Hoes La	New York	NY	00854	0
10	8	Quick Stop	71 Mulberry S	Denver	CO	98839	0							
11	9	Direct Sales	4600 River Ro	San Francisco	CA	08854	1							
12	10	Specialty Reti	88 Specialty D	Chicago	IL	88938	1							
13	11	The Big Cat	12 Woodpark	Austin	TX	32343	0							
14	12	Fast Transpor	77 Druguz Blv	Orlando	FL	12343	0							
15	13	Computer Tec	90 First Ave.	Los Angeles	CA	11223	1							
16	14	InterLand Ser	822 Technolo	Rockville	MD	83923	0							
17	15	Scheggia Ser	839 Via Gaty	Camden	NJ	12334	0							
18	16	Whittaker & C	44 Islander La	Albany	CA	83923	0							
19	17	FISGA Corp.	21 Doger Sou	New York	NY	20122	0	17	FISGA Corp.	21 Doger Sou	New York	NY	20122	0
20	18	Ubermeyer	8300 Clevelar	Boston	MA	29183	0							
21	19	Software Spe	300 Great Sw	Newark	NJ	23445	1							
22	20	Folk Mining	345 Servgem I	Tucson	AZ	11234	0							
23	21	West Coast C	460 Newcom	Los Angeles	CA	73646	0							
24	22	Big Ed's BBQ	23 Tasaga Av	Houston	TX	23421	0							
25	23	World Softwa	563 Uptown D	Boston	MA	28275	1							
26	24	High Society I	1248 Broadw	New York	NY	27834	0	24	High Society I	1248 Broadw	New York	NY	27834	0
27	25	Rutgers Bank	203 Davidson	New Brunswi	NJ	00854	0							
28	26	Nova Tera	987 First Ave	Dallas	TX	23123	0							

¢

5 Manipulating Tabular Data

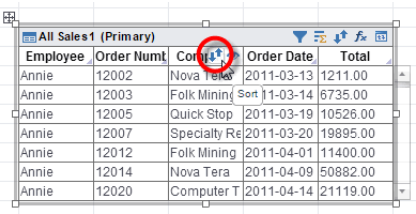
This section discusses various techniques for manipulating the data within a table.

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, press 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.



When a column is sorted, its sort order is indicated by the small arrow next to the column name in the column header cell.

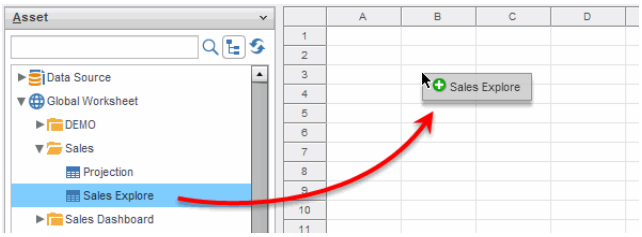
Example: Sorting a Single Column

Sort the data in the sample 'Sales Explore' Worksheet in ascending order by state name. Follow these steps:

1. Create a new Worksheet.

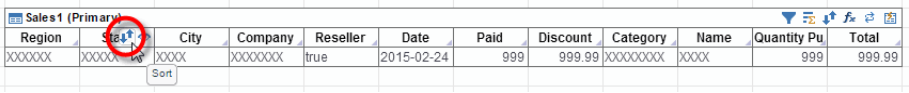
This Worksheet can be found in the 'Sales' folder, or in the 'Orders Examples' data set folder in Agile Edition.

2. From the **Asset** panel, drag the 'Sales Explore' Worksheet onto an empty cell on the Worksheet. This creates the data block 'Sales1'.

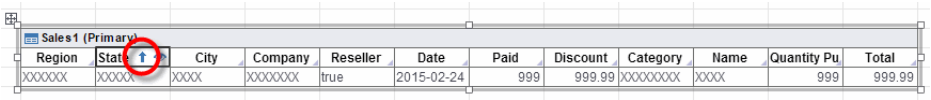


3. Select the 'State' header cell.

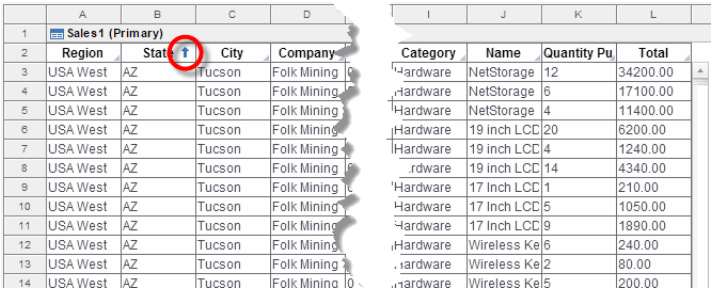
4. Press the 'Sort' button in the column header cell.



Notice the up-arrow in the ‘State’ column header. This indicates the current ascending sort order.

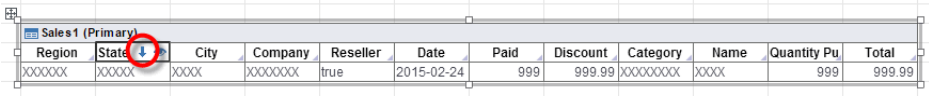


5. Preview the table.



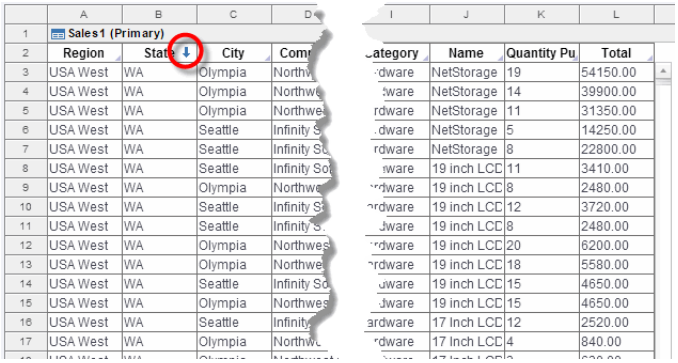
6. Close the ‘Preview’ window, and select the ‘State’ column header again.

7. Press the ‘Sort’ button again. Notice that the ‘State’ column is now sorted in descending order, indicated by the down-arrow in the column header.



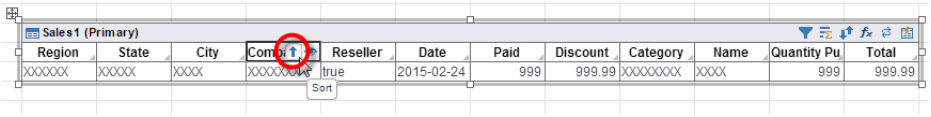
Press the ‘Sort’ button a third time to return the data to its original order.

8. Preview the table.

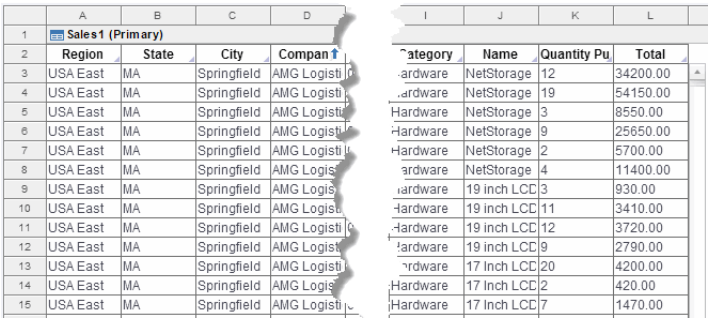


9. Now select the ‘Company’ column header cell.

10. Press the ‘Sort’ button in the ‘Company’ column header. Notice that the ‘Company’ column is now sorted in ascending order and the ‘State’ column is no longer sorted.



11. Preview the table.

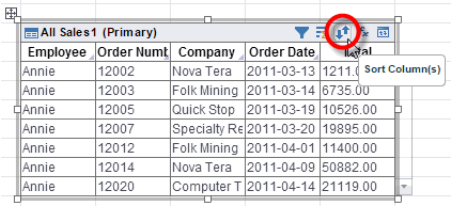


¢

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

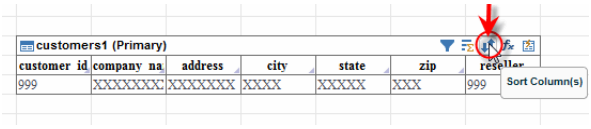
Say you have a table listing the information of all of our customers and you would like to sort the data based on the state first, and then the city. To sort both the 'state' and the 'city' columns of the table, follow these steps:

1. Create a new Worksheet.

2. Drag the 'customers' query from the **Asset** tree to an empty cell in the Worksheet. This creates a new data block named 'customers1'.

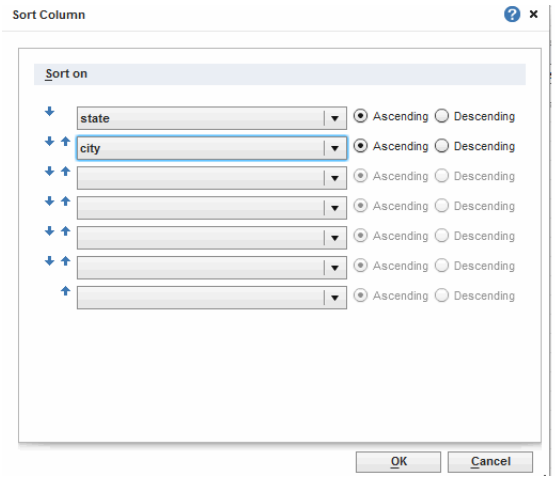
This query can be found in the 'Sample Queries' folder, or in the 'Orders Examples' data set folder in Agile Edition.

3. Press the 'Sort column(s)' button in the table title bar. This opens the 'Sort Column' dialog box.

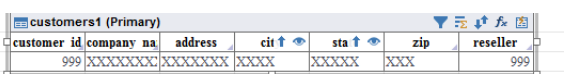


4. Select 'state' from the first drop-down list, and ensure that the 'Ascending' radio button is selected.

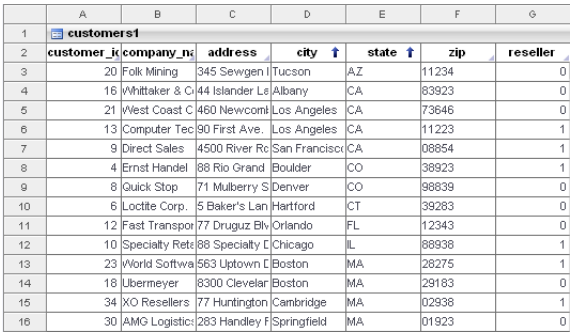
5. Select 'city' from the second drop-down list, and ensure that the 'Ascending' radio button is selected.



6. Press 'OK'. Notice that the up arrows next to the column names 'state' and 'city' indicate that both of those columns are sorted in ascending order.



7. 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 Newcomb	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.

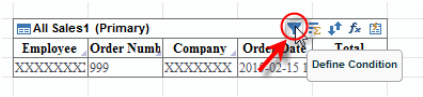
5.2.1 Defining Simple Conditions

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*.

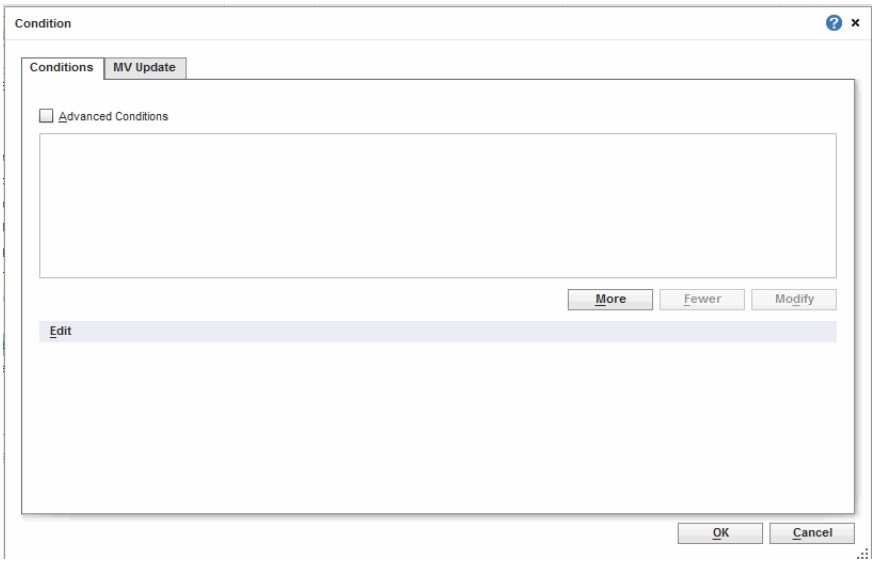
Note: Filtering conditions are only applied to data that is retrieved from the database. The amount of data (number of rows) retrieved is determined by the ‘Max Rows’ setting and the global ‘Design mode sample data size’ setting. See *Previewing a Data Block* and *Showing Live Data in a Table* for more information.

To define a simple condition, follow these steps:

1. Press the ‘Define Condition’ button on the title bar of a data block.



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.

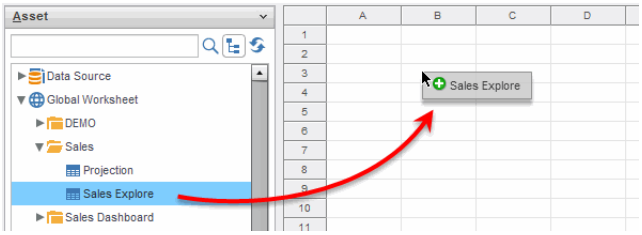
Example: Defining a Simple Condition

In the sample Sales Explore Worksheet, retrieve only orders with sales of at least $1,000.

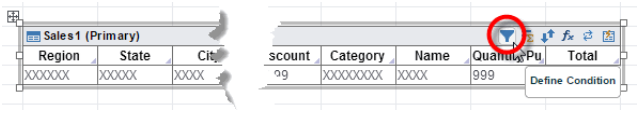
1. Create a new Worksheet.

2. From the **Asset** panel, drag the ‘Sales Explore’ Worksheet onto an empty cell on the Worksheet. This creates the data block ‘Sales1’.

This Worksheet can be found in the ‘Sales’ folder, or in the ‘Orders Examples’ data set folder in Agile Edition.



3. Press the ‘Define Condition’ button in the upper right corner of the table’s title bar.



This opens the ‘Conditions’ dialog.

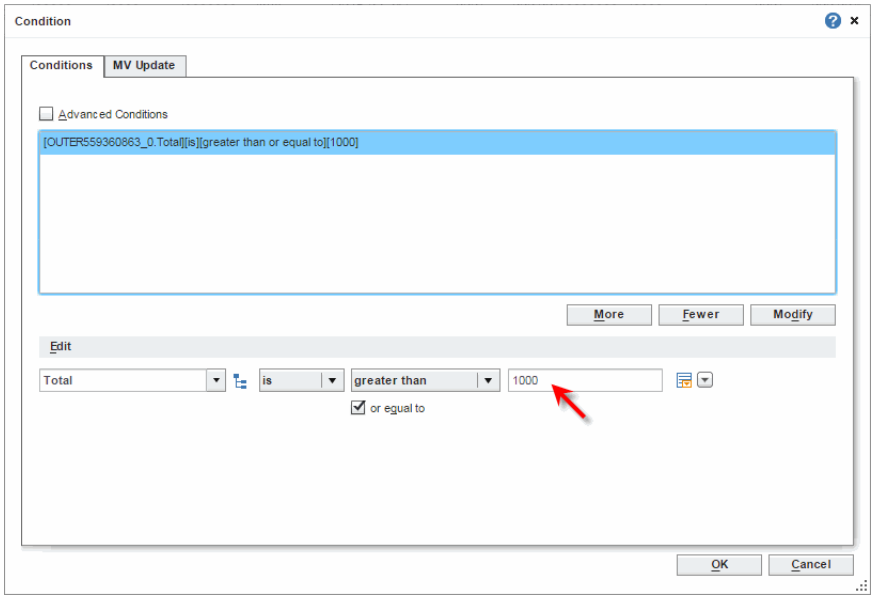
4. Press the ‘More’ button to create a new condition.

5. Select ‘Total’ from the left menu.

6. Select ‘greater than’ from the center menu and select the ‘or equal to’ option.

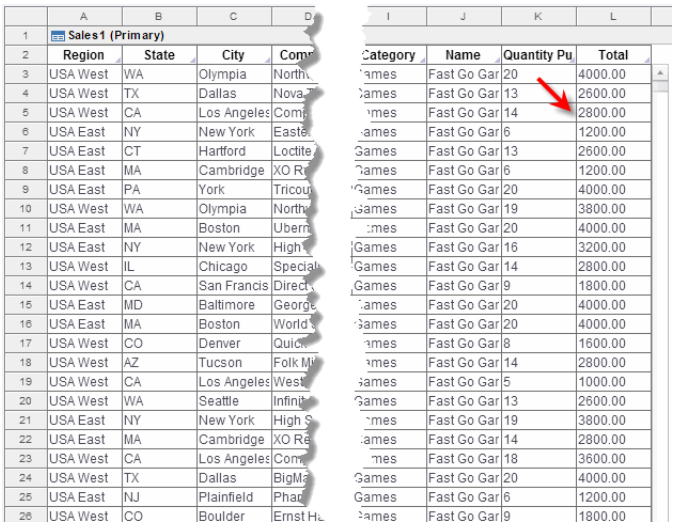
A Variable or another field value can be used in the condition by selecting ‘variable’ or ‘field’ and specifying either the Variable or the field name. See *Creating a Variable* for more information.

7. Specify “1000” for the value.



8. Press ‘OK’.

9. Right-click the table title bar, and select ‘Preview’ from the context menu.



	A	B	C	D	I	J	K	L
1	Sales1 (Primary)							
2	Region	State	City	Comp	Category	Name	Quantity Pu	Total
3	USA West	WA	Olympia	North	ames	Fast Go Gar	20	4000.00
4	USA West	TX	Dallas	Nova	ames	Fast Go Gar	13	2600.00
5	USA West	CA	Los Angeles	Comp	mes	Fast Go Gar	14	2800.00
6	USA East	NY	New York	Easte	ames	Fast Go Gar	6	1200.00
7	USA East	CT	Hartford	Loctite	Games	Fast Go Gar	13	2600.00
8	USA East	MA	Cambridge	XO Re	ames	Fast Go Gar	6	1200.00
9	USA East	PA	York	Tricou	Games	Fast Go Gar	20	4000.00
10	USA West	WA	Olympia	North	Games	Fast Go Gar	19	3800.00
11	USA East	MA	Boston	Uberr	mes	Fast Go Gar	20	4000.00
12	USA East	NY	New York	High	Games	Fast Go Gar	16	3200.00
13	USA West	IL	Chicago	Special	Games	Fast Go Gar	14	2800.00
14	USA West	CA	San Francis	Direct	Games	Fast Go Gar	9	1800.00
15	USA East	MD	Baltimore	George	ames	Fast Go Gar	20	4000.00
16	USA East	MA	Boston	World	ames	Fast Go Gar	20	4000.00
17	USA West	CO	Denver	Quick	mes	Fast Go Gar	8	1600.00
18	USA West	AZ	Tucson	Folk M	mes	Fast Go Gar	14	2800.00
19	USA West	CA	Los Angeles	West	ames	Fast Go Gar	5	1000.00
20	USA West	WA	Seattle	Infinit	Games	Fast Go Gar	13	2600.00
21	USA East	NY	New York	High S	mes	Fast Go Gar	19	3800.00
22	USA East	MA	Cambridge	XO Re	ames	Fast Go Gar	14	2800.00
23	USA West	CA	Los Angeles	Comp	mes	Fast Go Gar	18	3600.00
24	USA West	TX	Dallas	BigMa	Games	Fast Go Gar	20	4000.00
25	USA East	NJ	Plainfield	Phar	Games	Fast Go Gar	6	1200.00
26	USA West	CO	Boulder	Ernst H	ames	Fast Go Gar	9	1800.00

Note that only totals greater than $1000 are returned.

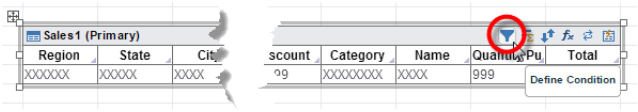
¢

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:

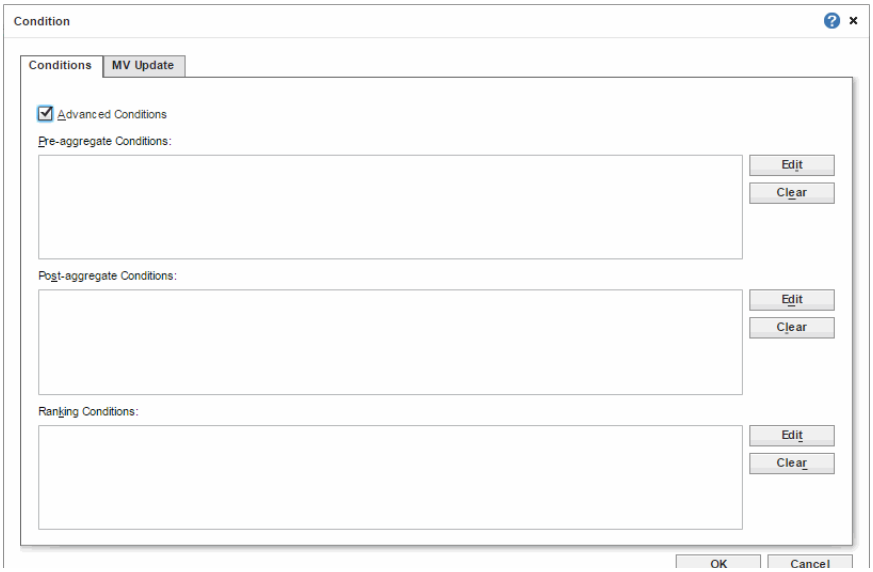
1. Press the ‘Define Condition’ button on the title bar of a data block.



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.

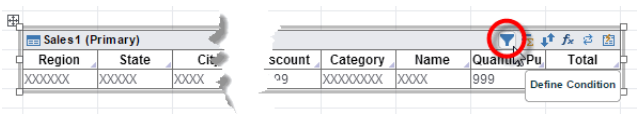
Pre-aggregate, Post-aggregate, and Ranking Conditions

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 an Advanced Condition

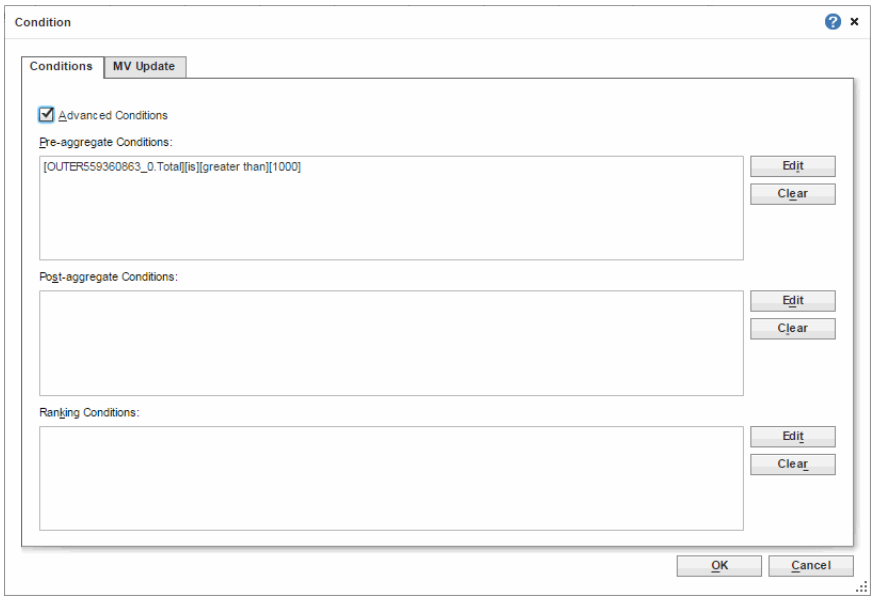
This example continues the example from the previous section, *Defining Simple Conditions*. Adjust the condition you specified previously to return only the top 10 totals. Follow the steps below:

1. Press the 'Define Condition' button located in the data block title bar.



This opens the 'Conditions' dialog box.

2. Select the ‘Advanced Conditions’ option. Notice that the condition added to the ‘Total’ field is included in the ‘Pre-aggregate’ conditions list.

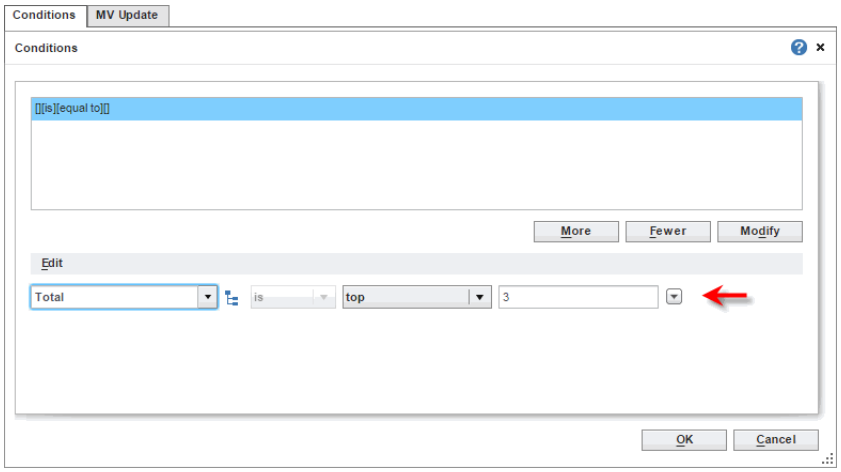


3. Press the ‘Edit’ button next to the ‘Ranking Conditions’, and then press the ‘More’ button.

4. From the left menu, select ‘Total’.

5. In the right menu select ‘top’ and enter “10” as the value.

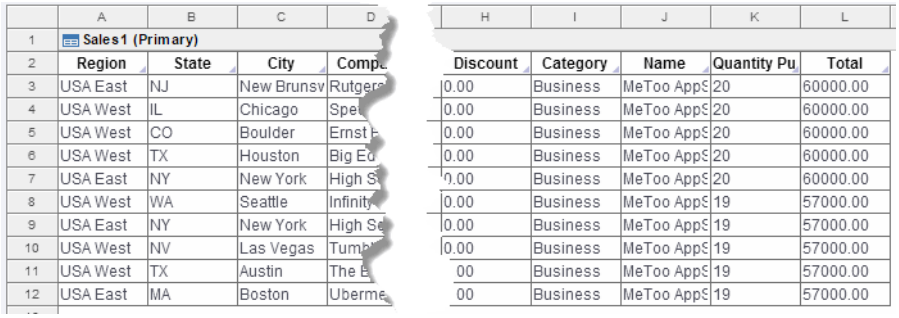
A Variable value can be used as the ranking condition value by selecting ‘variable’ from the drop-down list to the right of the value text box and specifying the parameter name in the text box. See *Creating a Variable* for more information.



6. Press ‘OK’ to return to the ‘Conditions’ dialog box.

7. Press ‘OK’ to apply all of the conditions to the table.

8. Preview the data block. Data for only the top 10 values of ‘Total Sales’ value are listed.



	A	B	C	D	H	I	J	K	L
1	Sales1 (Primary)								
2	Region	State	City	Comp	Discount	Category	Name	Quantity Pu	Total
3	USA East	NJ	New Brunsv	Rutger	0.00	Business	MeToo Apps	20	60000.00
4	USA West	IL	Chicago	Spe	0.00	Business	MeToo Apps	20	60000.00
5	USA West	CO	Boulder	Ernst	0.00	Business	MeToo Apps	20	60000.00
6	USA West	TX	Houston	Big Ed	0.00	Business	MeToo Apps	20	60000.00
7	USA East	NY	New York	High S	0.00	Business	MeToo Apps	20	60000.00
8	USA West	WA	Seattle	Infinity	0.00	Business	MeToo Apps	19	57000.00
9	USA East	NY	New York	High Se	0.00	Business	MeToo Apps	19	57000.00
10	USA West	NV	Las Vegas	Tumb	0.00	Business	MeToo Apps	19	57000.00
11	USA West	TX	Austin	The E	00	Business	MeToo Apps	19	57000.00
12	USA East	MA	Boston	Uberme	00	Business	MeToo Apps	19	57000.00

¢

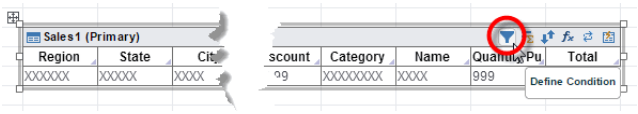
5.2.3 Defining a Materialized View Update Condition

You can create conditions for a data block which determine how and when the data block’s materialized view is updated. A materialized view is a high-performance caching mechanism for dashboards. (See *Incrementally Updating a Materialized View* in *Advanced Server Administration* for background on incremental updates.)

If you are using Style Scope Agile Edition, see *Refreshing Data in a Dashboard* for information about the high-performance cache.

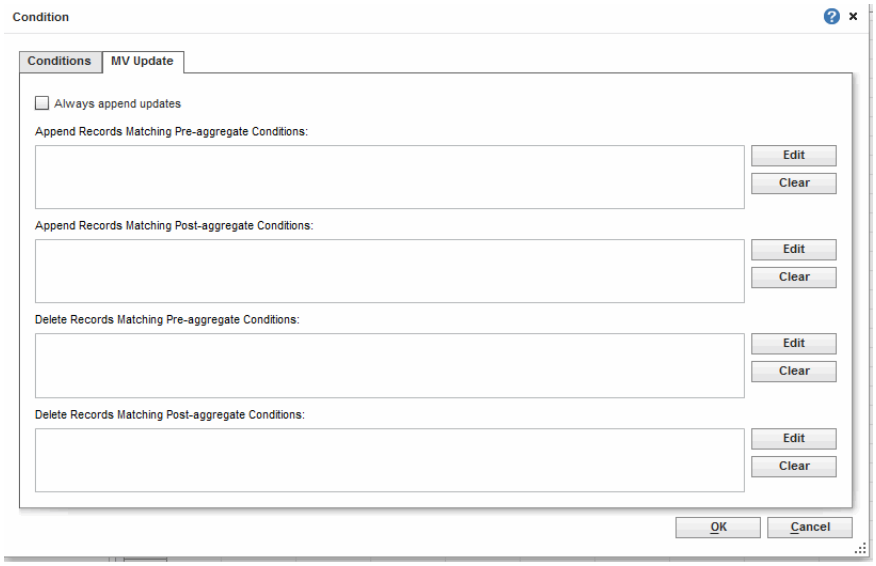
To specify the materialized view update conditions for a data block, follow the steps below:

1. Press the ‘Define 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. (Optional) In the special case where you are using an unmergeable query, enable the 'Always append updates' option. Otherwise, leave it disabled. See *Incrementally Updating a Materialized View* in *Advanced Server Administration* for information about using this property for unmergeable queries.

4. 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).

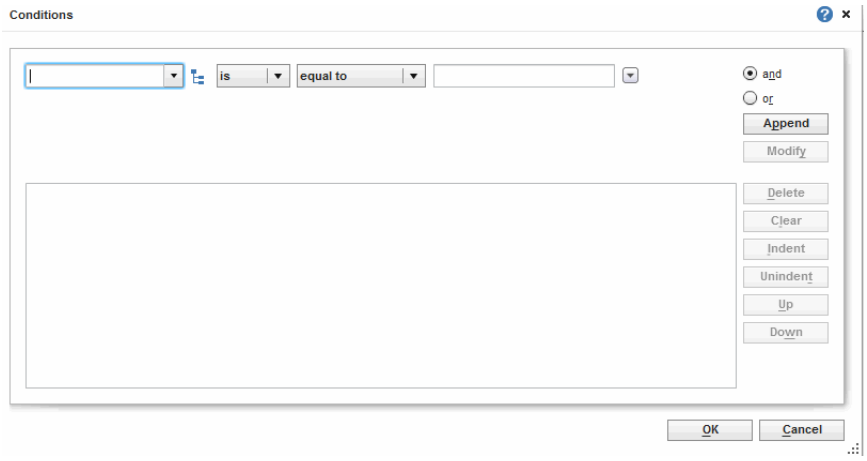
   b. **Append Records Matching Post-aggregate Condition**: Create a condition which selects records to add to the existing materialized view based on the post-aggregate values of data (subsequent to grouping and summarization).

   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).

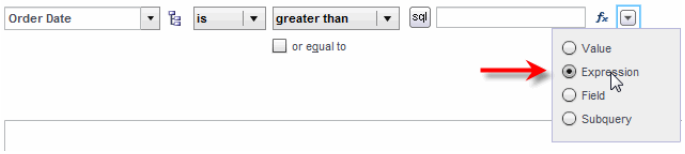
   d. **Delete Records Matching Post-aggregate Condition**: Create a condition which selects records to delete from the existing materialized view based on the post-aggregate values of data (subsequent to grouping and summarization).

When you press the desired ‘Edit’ button, this opens the ‘Conditions’ dialog box.

5. 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.

6. 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.

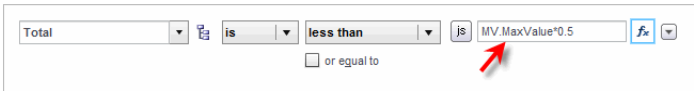
Special Parameters for Materialized View Update Conditions

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.)

OPERATOR	DESCRIPTION
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.)
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:

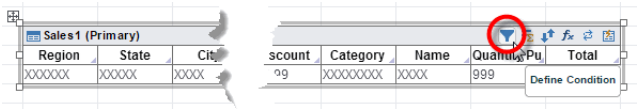
- Viewsheet input component. (See *Passing Inputs to an Embedded Table or Variable* in *Advanced Dashboard Design*.)
- User session data. (See *SRPrincipal Properties* in *Advanced Server Integration*.)

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. Press the ‘Define 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.

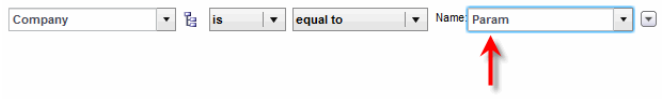


In the Combo Box, do one of the following:

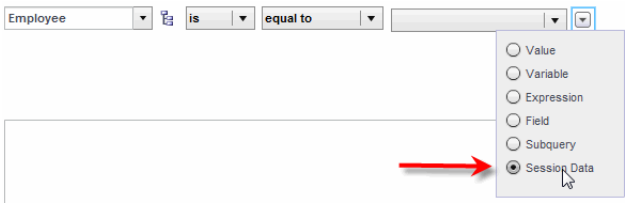
a. Select an existing Worksheet Variable from the menu.

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.

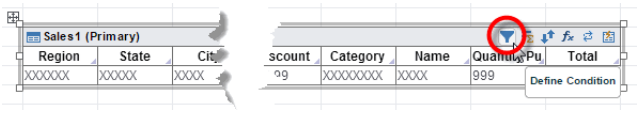
5.2.6 Using a Field in a Condition

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. Press the ‘Define Condition’ button on the title bar of the table you wish to filter.

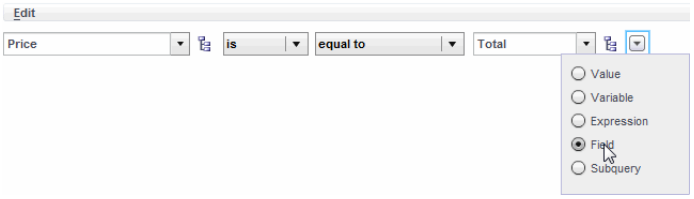


This opens the ‘Condition’ dialog box.

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.

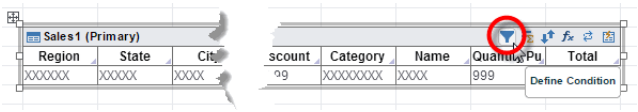
See Also
Filtering, for details on specifying filtering conditions.

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.

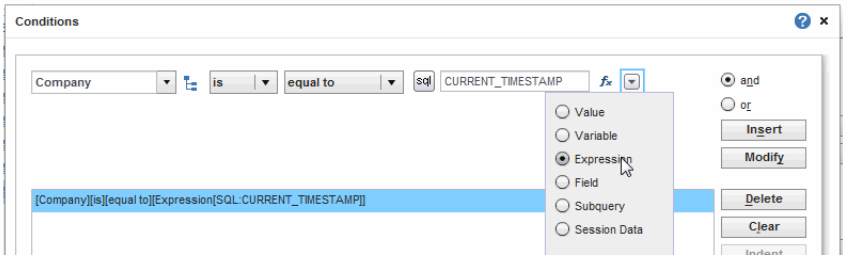
2. Press the ‘Define Condition’ button on the title bar of the data block you wish to filter.



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. Press 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]
```

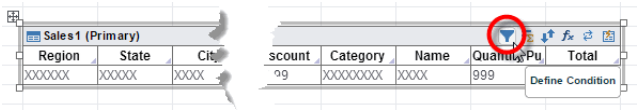
The list of “companies in NY” generated by the subquery will always be up-to-date, because the subquery itself executes at runtime.

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. Press the ‘Define Condition’ button on the title bar of the table you wish to filter.



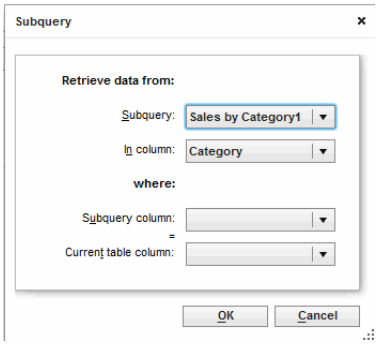
This opens the ‘Condition’ dialog box.

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

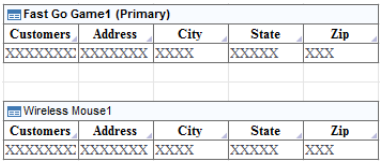
Assume that you have a table of companies that purchased a ‘Wireless Mouse’ product, and a second table of companies that purchased a ‘Fast Go Game’ product. You would like to filter the ‘Wireless Mouse’ table to show only the companies that *did not* purchased the ‘Fast Go Game.’ To do this, define 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.

These data sets can be found in the ‘Sample Queries’ folder, or in the ‘Orders Examples’ data set folder in Agile Edition.

2. Drag the ‘Fast Go Game’ query from the **Asset** panel onto one of the empty cells in the Worksheet. This creates a new data block, named ‘Fast Go Game1’.

3. Drag the ‘Wireless Mouse’ query from the **Asset** panel onto one of the empty cells in the Worksheet. This creates a new data block, named ‘Wireless Mouse1’.



4. In the ‘Fast Go Game1’ table, rename the ‘Customers’ column to ‘Customers_FGG’. This will make the next steps more clear.

5. Press the ‘Define Condition’ button in the title bar of the ‘Wireless Mouse1’ data block.



6. This opens the ‘Conditions’ dialog box.

7. Uncheck the ‘Advanced Conditions’ option.

8. Press the ‘More’ button.

9. Use the menus to specify the following expression:

```
[Customers][is not][one of]
```

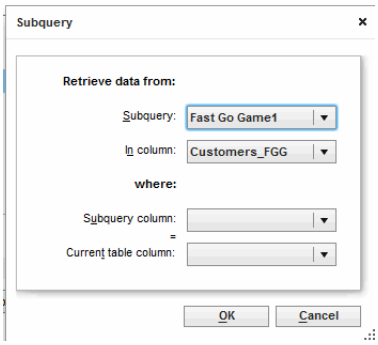
10. In the right-side pop-up menu button select ‘Subquery’.



11. Press the ‘Subquery’ button. This opens the ‘Subquery’ dialog box.

12. Select ‘Fast Go Game1’ from the ‘Subquery’ menu. This is the name of the subquery table to use.

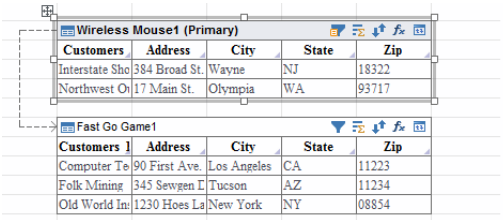
13. Select ‘Customers_FGG’ from the ‘In column’ menu. This is the column of the subquery which contains the list of values to use in the condition expression.



14. Press ‘OK’ to close the ‘Subquery’ dialog box. (You do not need to make selections from the bottom two menus.)

15. Press ‘OK’ to exit the ‘Condition’ dialog box.

16. 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.

¢

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.

These data sets can be found in the 'Sample Queries' folder, or in the 'Orders Examples' data set folder in Agile Edition.

2. Drag the 'Sales this year' query from the **Asset** panel onto one of the empty cells in the Worksheet. This creates a new data block, named 'Sales this year1'.

3. Drag the 'Sales last year' query from the **Asset** panel onto one of the empty cells in the Worksheet. This creates a new data block, named 'Sales last year1'.

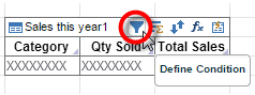
4. Rename the following columns in the 'Sales last year1' table:

   a. Rename 'Category' to 'Category_LY'.

   b. Rename 'Total Sales' to 'Total_Sales_LY'.

   This will make the following steps more clear.

5. Press the 'Define Condition' button in the title bar of the 'Sales this year1' table.



   This opens the 'Conditions' dialog box.

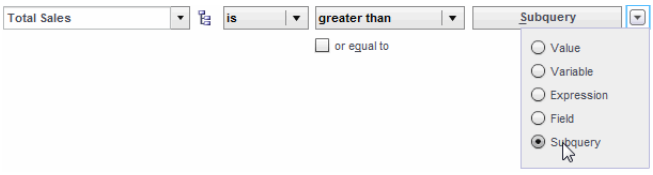
6. Uncheck the 'Advanced Conditions' option.

7. Press the 'More' button.

8. Use the menus to specify the following expression:

   ```
   [Total Sales][is][greater than]
   ```

9. From the right-side menu select 'Subquery'.



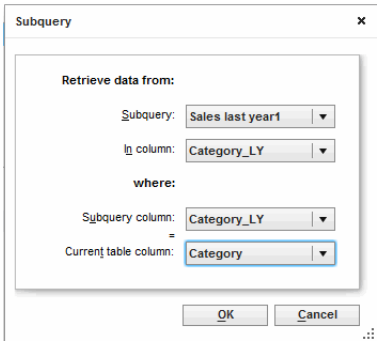
10. Press the 'Subquery' button. This opens the 'Subquery' dialog box.

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.

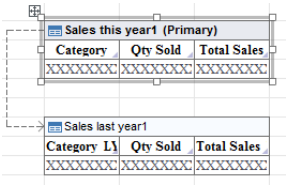
13. Select ‘Category_LY’ from the ‘Subquery column’ column. This is the subquery column that will be matched up against the ‘Current table column’ in the main table. (It is the “join column” of the subquery table.)

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

16. Press ‘OK’ 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.



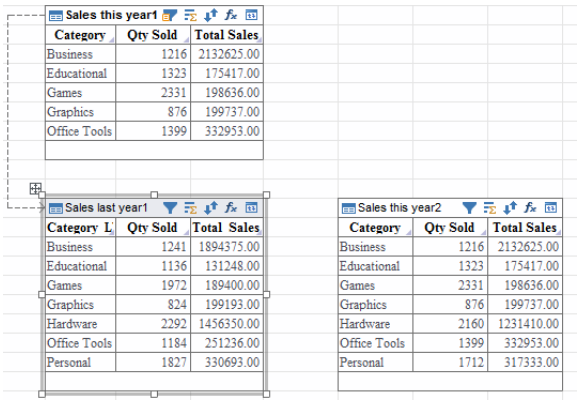
17. For each table, press the ‘Change View’ button in the title bar of the table and select ‘Live Preview’ from the menu.

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	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

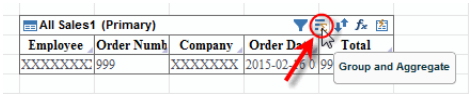
Sales this year2

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 block, follow these steps:

1. Press the ‘Group and Aggregate’ button in the table’s title bar.



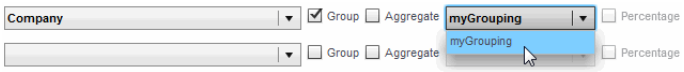
   This opens the ‘Group and Aggregate’ dialog box.

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

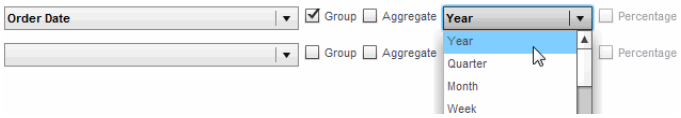
names, each distinct state name will define its own group, and aggregate values will be computed for each of those groups.

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.).

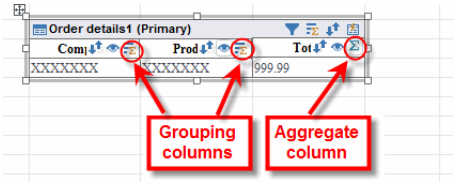
7. (Optional) For the univariate aggregation methods (Sum, Max, etc.), select the 'Percentage' check box to display the aggregated measure as a percentage of the grand total.

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 (Binning), to create a new grouping of existing column.

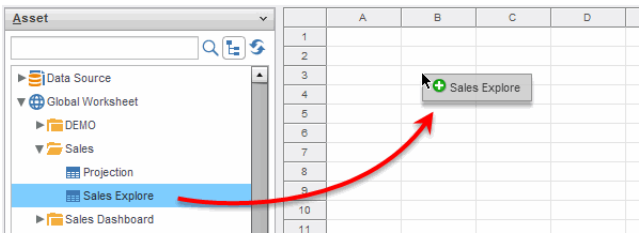
Example: Grouping and Aggregating Data

The sample ‘Sales Explore’ Worksheet contains order information for several products. Find the total sales for each individual product. To do this, group the table data by product name and summarize by ‘Total’. Follow the steps below:

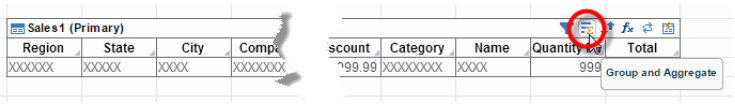
1. Create a new Worksheet.

This Worksheet can be found in the ‘Sales’ folder, or in the ‘Orders Examples’ data set folder in Agile Edition.

2. From the **Asset** panel, drag the ‘Sales Explore’ Worksheet onto an empty cell on the Worksheet. This creates the data block ‘Sales1’.



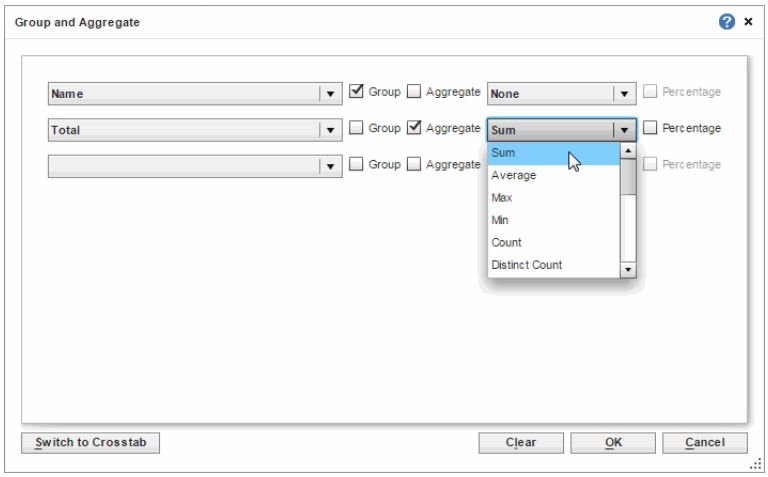
3. Press the ‘Group and Aggregate’ button in the title bar.



This opens the ‘Group and Aggregate’ dialog box.

4. Select ‘Group’ for the ‘Name’ column and ‘Aggregate’ for the ‘Total’ column.

5. Set the aggregate function for the ‘Total’ column to ‘Sum’.



6. Press ‘OK’ to apply the grouping and summarization.

7. Preview the table. Notice that only the ‘Product’ and ‘Total’ fields are included in the table, and the table now provides the summed total for each product. Columns that are not designated as ‘Group’ or ‘Aggregate’ columns have been removed.

	A	B
1	Sales1 (Primary)	
2	Name	Total
3	17 Inch LCD	378420.00
4	19 inch LCD	504060.00
5	Animal World	105020.00
6	Barbie's Fashion	158553.00
7	Combat Hero	44950.00
8	Easy Chess	53935.00
9	Fancy Menus	98245.00
10	Fast Go Game	309600.00
11	Fast Mail	68133.00
12	Green Planet	116925.00
13	House Hunter	232716.00
14	Info Folder	628425.00
15	InsideView	184750.00
16	Learn ABC	105000.00
17	Math for Me	415480.00
18	MeToo AppServer	4446000.00
19	Mega Icons	813450.00

¢

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.

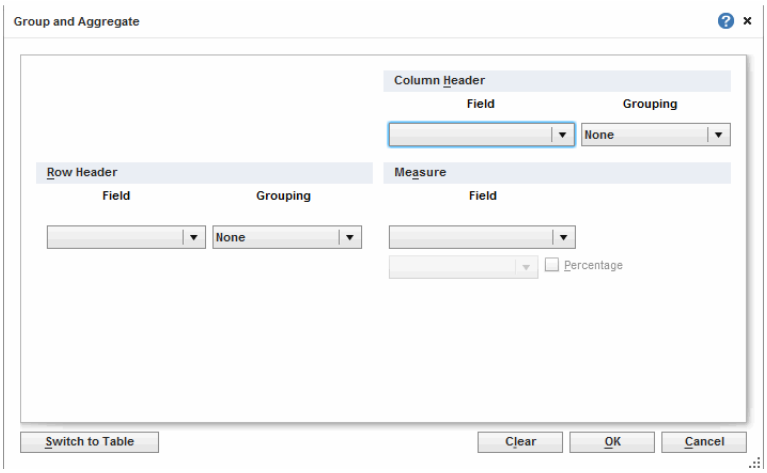
The values at the row-column intersections of the crosstab table represent summary (*aggregate*) information of the measure. For example, a crosstab with row headers representing ‘Salesperson’ and column headers

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 block, 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’.

AGGREGATE MEASURE	DESCRIPTION
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.
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’.

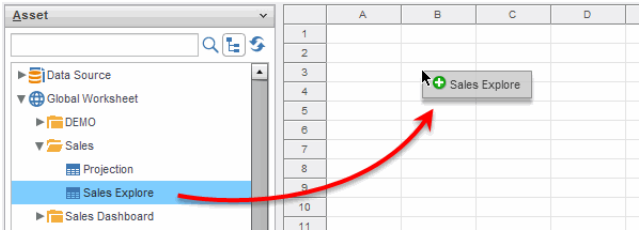
Example: Crosstab Aggregation

The sample ‘Sales Explore’ Worksheet contains order information for a number of different customers. Display the total amount of money spent by each customer on each of the products by using a crosstab table. Follow the steps below:

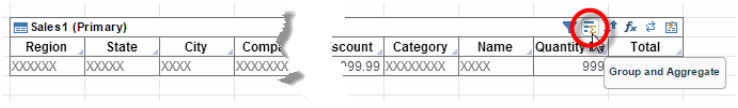
1. Create a new Worksheet.

2. From the **Asset** panel, drag the ‘Sales Explore’ Worksheet onto an empty cell on the Worksheet. This creates the data block ‘Sales1’.

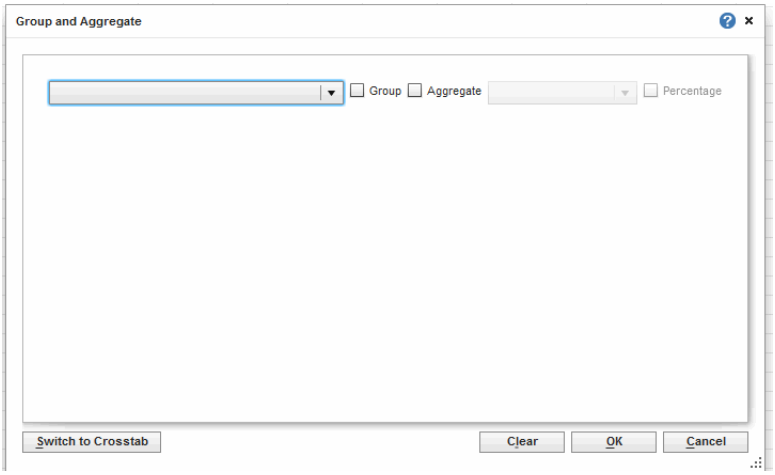
This Worksheet can be found in the ‘Sales’ folder, or in the ‘Orders Examples’ data set folder in Agile Edition.



3. Press the ‘Group and Aggregate’ button in the title bar.



This opens the ‘Group and Aggregate’ dialog box.

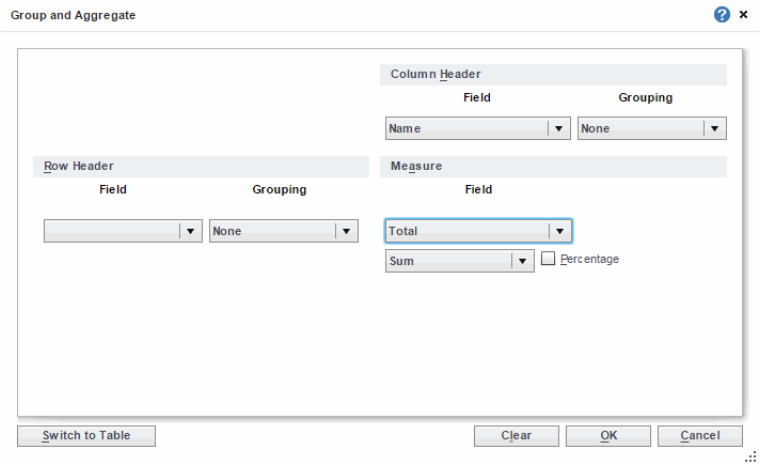


4. Press the ‘Switch to Crosstab’ button. The view changes to enable you to specify the column headers and the summary column.

5. Select ‘Company’ from the first drop-down list under the ‘Row Header’.

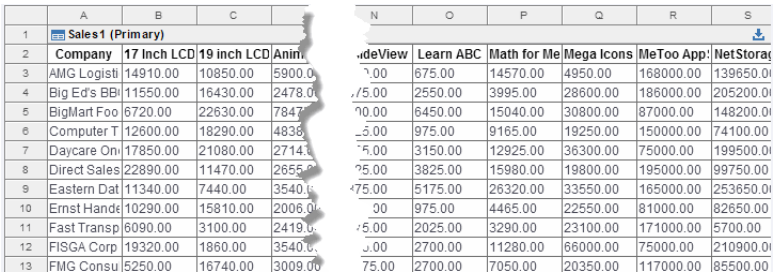
6. Select ‘Name’ from the ‘Column Header’ drop-down list.

7. Select ‘Total’ from the ‘Measure’ drop-down list and ensure that the aggregate function is set to ‘Sum’.



8. Press ‘OK’.

9. Preview the table.



	A	B	C	N	O	P	Q	R	S
1	Sales1 (Primary)								
2	Company	17 Inch LCD	19 inch LCD	...deView	Learn ABC	Math for Me	Mega Icons	MeToo App:	NetStorag
3	AMG Logisti	14910.00	10850.00	...00	675.00	14570.00	4950.00	168000.00	139650.0
4	Big Ed's BBI	11550.00	16430.00	...75.00	2550.00	3995.00	28600.00	186000.00	205200.0
5	BigMart Foo	6720.00	22630.00	...00.00	6450.00	15040.00	30800.00	87000.00	148200.0
6	Computer T	12600.00	18290.00	...5.00	975.00	9165.00	19250.00	150000.00	74100.00
7	Daycare On	17850.00	21080.00	...5.00	3150.00	12925.00	36300.00	75000.00	199500.0
8	Direct Sales	22890.00	11470.00	...25.00	3825.00	15980.00	19800.00	195000.00	99750.00
9	Eastern Dat	11340.00	7440.00	...75.00	5175.00	26320.00	33550.00	165000.00	253650.0
10	Ernst Hande	10290.00	15810.00	...00	975.00	4465.00	22550.00	81000.00	82650.00
11	Fast Transp	6090.00	3100.00	...5.00	2025.00	3290.00	23100.00	171000.00	5700.00
12	FISGA Corp	19320.00	1860.00	...00	2700.00	11280.00	66000.00	75000.00	210900.0
13	FMG Consu	5250.00	16740.00	75.00	2700.00	7050.00	20350.00	117000.00	85500.00

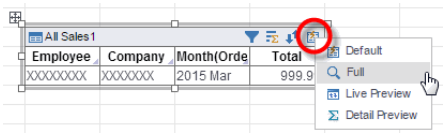
Observe that totals are shown for every combination of product and company.

¢

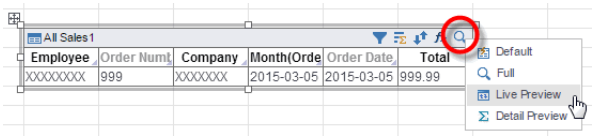
5.3.4 Switching Between Detail and Aggregate View

When you define grouping or aggregation on a data block, 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), press the ‘Change View’ button in the title bar and select ‘Full’ (meta-data) or ‘Detail Preview’ (live data).



- To switch a table from detail view to aggregate view (hiding all non-aggregate and non-grouping columns), press the ‘Change View’ button in the title bar and select ‘Default’ (meta-data) or ‘Live Preview’ (live data).



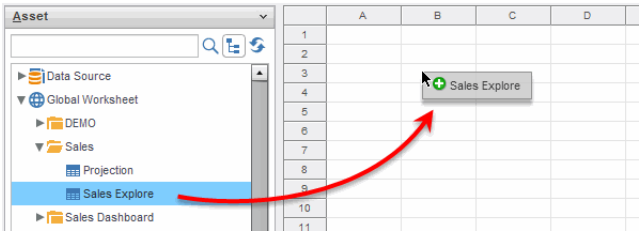
Example: Switching Between Detail and Aggregate View

The following example illustrates how to toggle from detail to aggregate view.

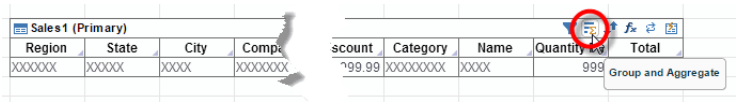
1. Create a new Worksheet.

This Worksheet can be found in the 'Sales' folder, or in the 'Orders Examples' data set folder in Agile Edition.

2. From the **Asset** panel, drag the 'Sales Explore' Worksheet onto an empty cell on the Worksheet. This creates the data block 'Sales1'.



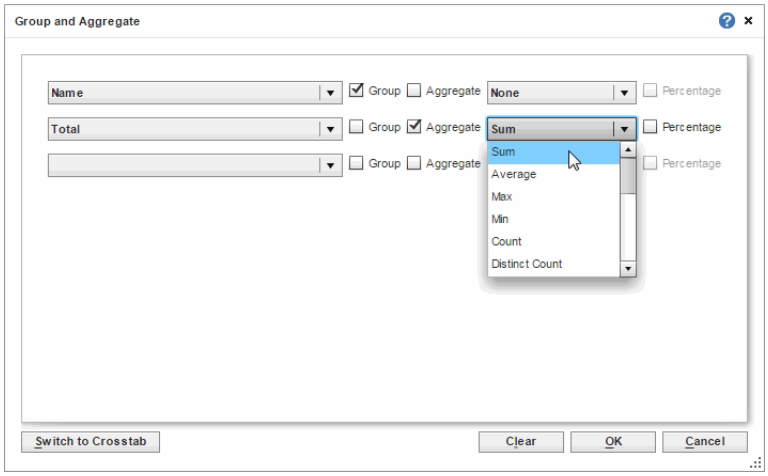
3. Press the 'Group and Aggregate' button in the title bar.



This opens the 'Group and Aggregate' dialog box.

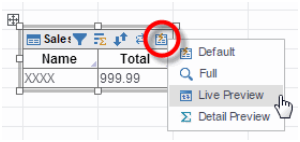
4. Select the 'Group' option for the 'Name' column and 'Aggregate' for the 'Total' column.

5. Ensure that the aggregate function for the 'Total' column is set to 'Sum'.

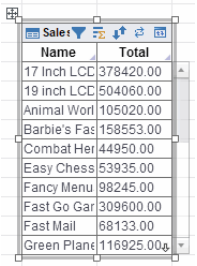


6. Press 'OK' button to apply the grouping and summarization.

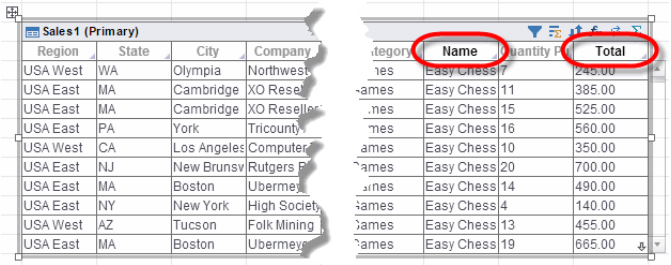
7. Press the 'Change View' button in the table title bar, and select 'Live Preview' from the menu.



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.



8. 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.



9. 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*.

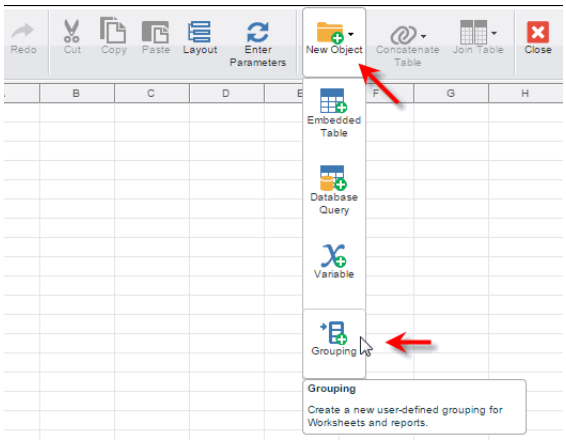
6.1 Creating a Named Grouping

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. Press 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. Press 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.

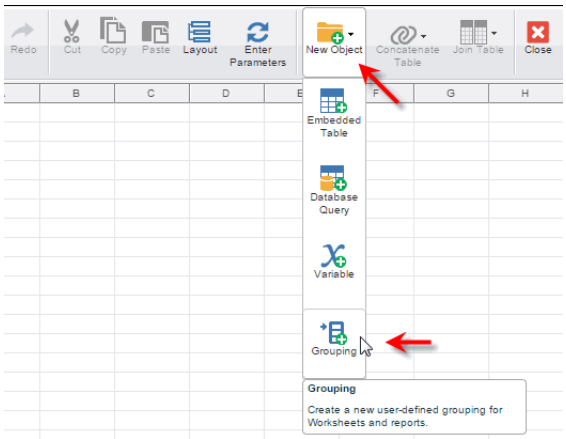
Saving an Asset, to save the defined grouping to the Repository.
Using a Named Grouping, to use the grouping in other Worksheets and reports.

Example: Creating a Named Grouping

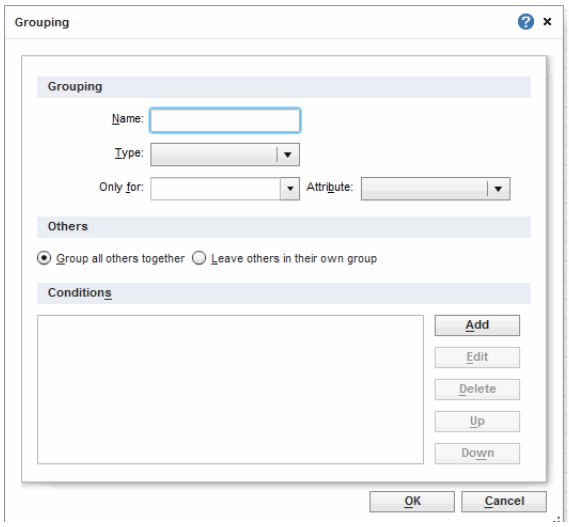
Suppose you want to create a group for all the states on the west coast, so that every time the customer data is displayed in either a table, chart, or section element in the report, all of the customers from the west coast states are grouped together.

Follow these steps to create the ‘WestCoast’ group.

1. Create a new Worksheet.
2. Press the ‘New Object’ button, and select the ‘Grouping’ option.



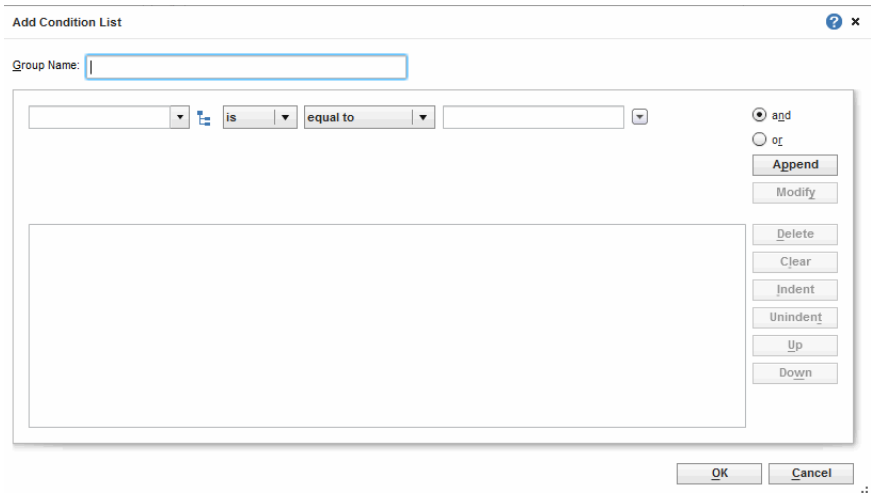
This opens the ‘Grouping’ dialog box.



3. Specify “WestCoast” as the name.

4. For the ‘Type’ select ‘String’.

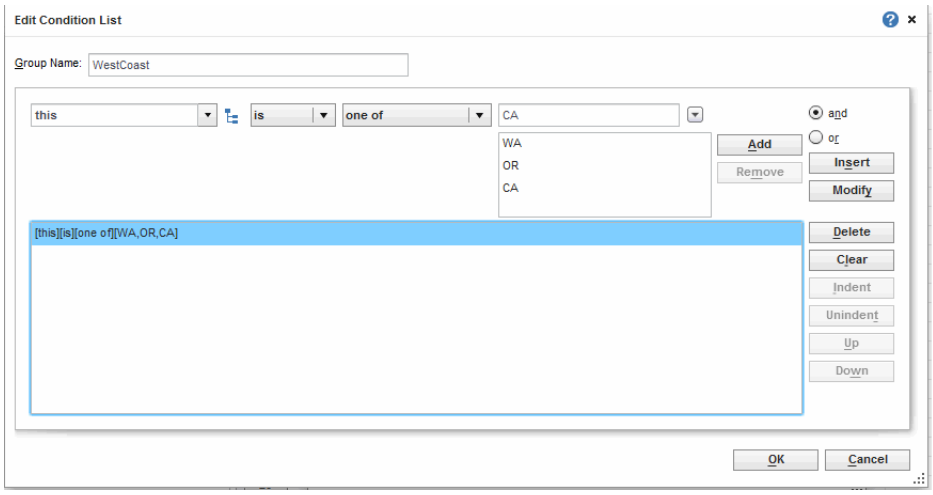
5. Press the ‘Add’ button to specify the conditions which the data has to satisfy in order to qualify to be a part of this new group. This opens the ‘Add Condition List’ dialog.



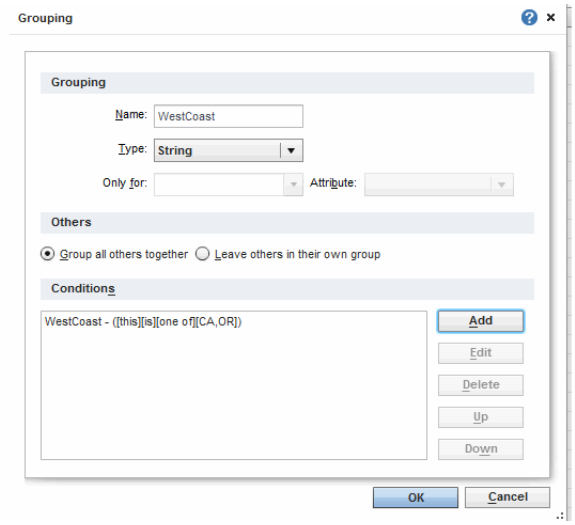
6. Specify ‘WestCoast’ as the Group Name.

7. Specify the following condition, and click the ‘Append’ button.

```
[this][is][one of][WA,OR,CA]
```

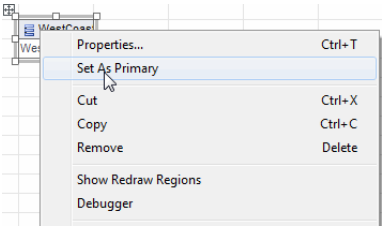


8. Press the ‘OK’ button to exit the ‘Add Condition List’ dialog and go back to the ‘Group’ dialog.



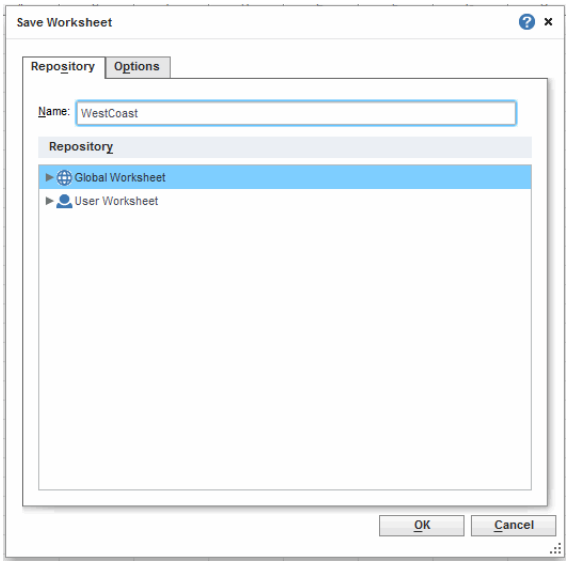
9. Press the ‘OK’ button.

10. Right-click on the title for the ‘WestCoast’ group and select ‘Set as Primary’.

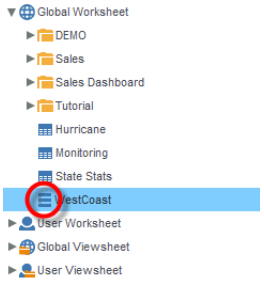


11. Press the ‘Save’ button in the top toolbar. The ‘Save as Worksheet’ dialog appears.

12. Specify “WestCoast” as the name of the asset and select ‘Global Worksheet’ as its scope.



13. Press ‘OK’ 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

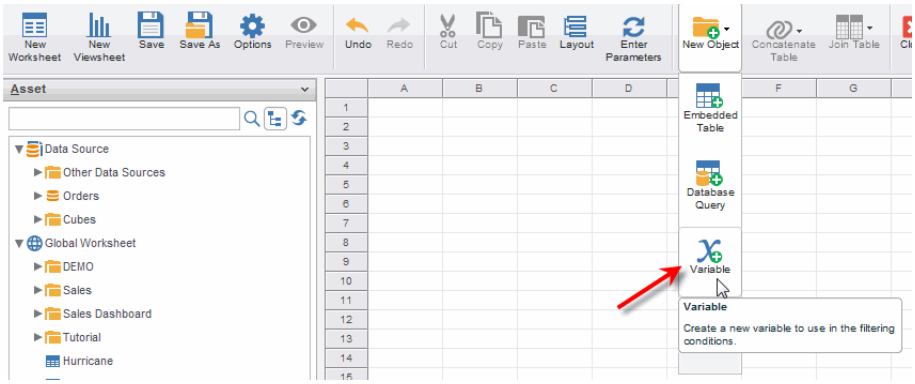
Variables provide a way to prompt the user to enter specific filtering conditions for tables in the Worksheet. You can use Variables in conditions on tables and Named Groups.

See *Saving an Asset* and *Using a Variable* for more information about working with Variables.

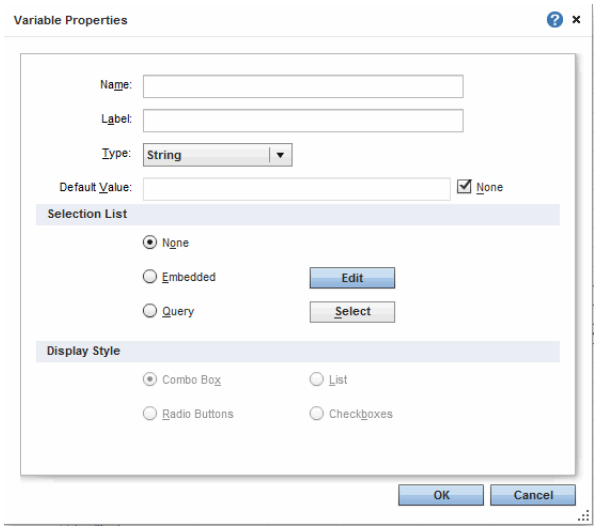
6.2.1 Defining a Variable

To create a new Variable, follow these steps:

1. Press 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.

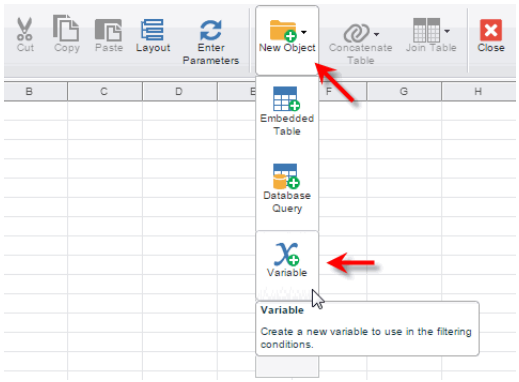
Example: Defining a Variable

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.

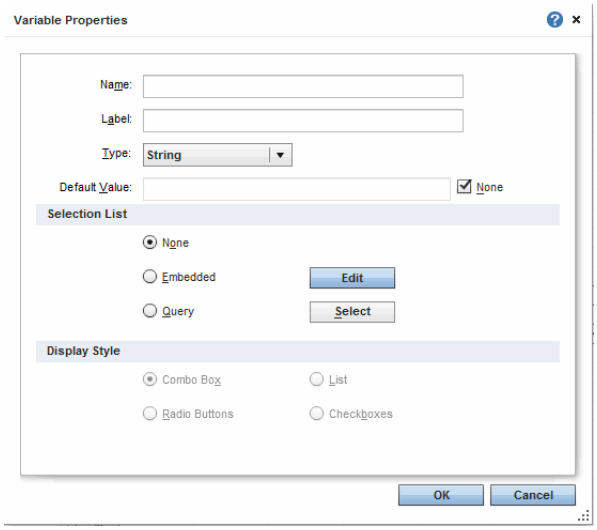
Instead of creating the ‘state’ Variable for all of the Data Worksheets, you can create it once and save it as a ‘Variable’ asset in the asset repository. Then, you can reuse the Variable in all of the relevant Data Worksheets. Follow these steps to create a Variable asset:

1. Create a new Worksheet.

2. Press the ‘New Object’ button located in the top toolbar, and select the ‘Variable’ option.



This opens the ‘Variable Properties’ dialog box.



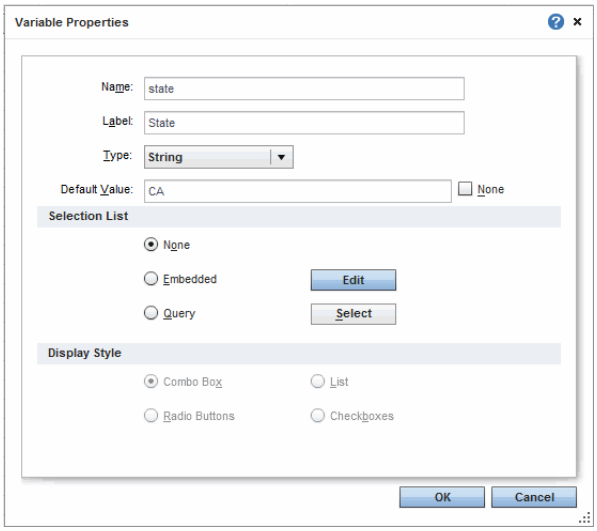
3. Specify “state” as the name of the Variable.

4. Specify “State” as the Label.

5. Select ‘String’ from the drop-down list for Type.

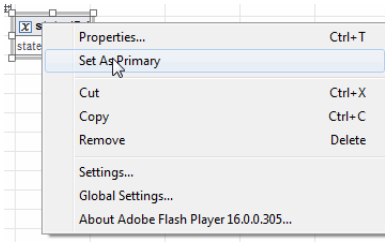
6. Uncheck the ‘None’ check box next to ‘Default Value’, and enter “CA” as the default value.

7. Ensure that ‘None’ is selected for the Selection List.



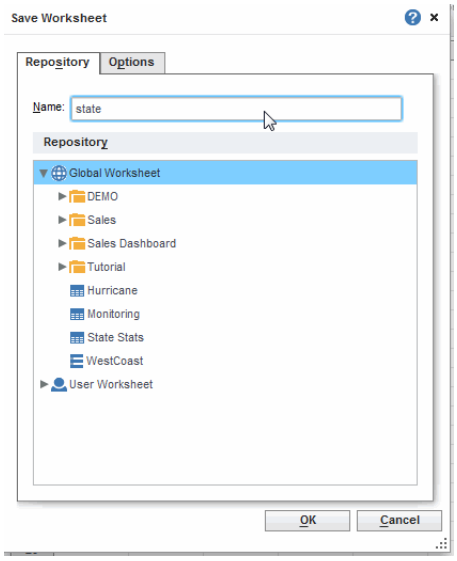
8. Press ‘OK’.

9. Right-click on the Variable’s title and select ‘Set as Primary’.

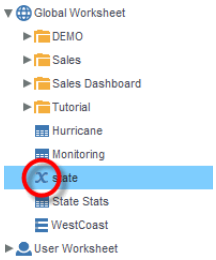


10. Press the ‘Save’ button in the top toolbar. This opens the ‘Save as Worksheet’ dialog.

11. Specify “State” as the asset name and select ‘Global Worksheet’ as its scope.



12. Press ‘OK’ to add the ‘State’ Variable to the **Asset** tree. Notice that the name of the asset on the **Asset** tree is shown 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.

¢

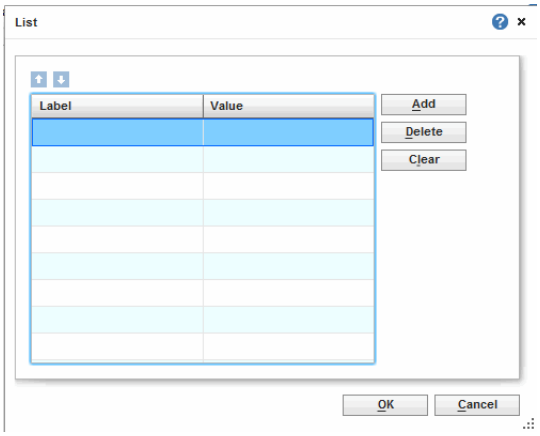
6.2.2 Creating an Embedded Variable List

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. Press the ‘Edit’ button to open the ‘List’ dialog box.



4. Press ‘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 press the ‘Up’ or ‘Down’ arrow button.

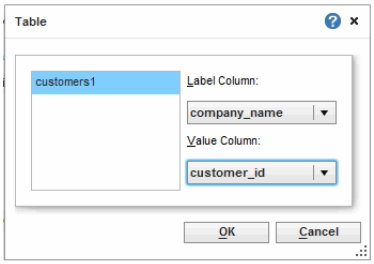
9. Click ‘OK’ to exit the ‘List’ dialog box.

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 are not available in Style Scope Agile Edition.
```

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

```
This example is not available in Style Scope Agile Edition.
```

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.

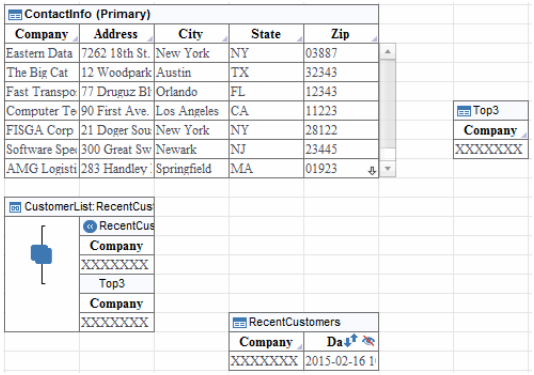
   To do this, apply an ‘in range’ filter condition on the ‘OrderInfo’ table that restricts the data to ‘Last 4 Weeks’. Name this new table ‘RecentCustomers’. The ‘RecentCustomers’ table will to have two columns, ‘Customer Company’ and ‘Order Date’. You require the ‘Order Date’ field to specify the date 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.

```
This example is not available in Style Scope Agile Edition.
```

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
   ```

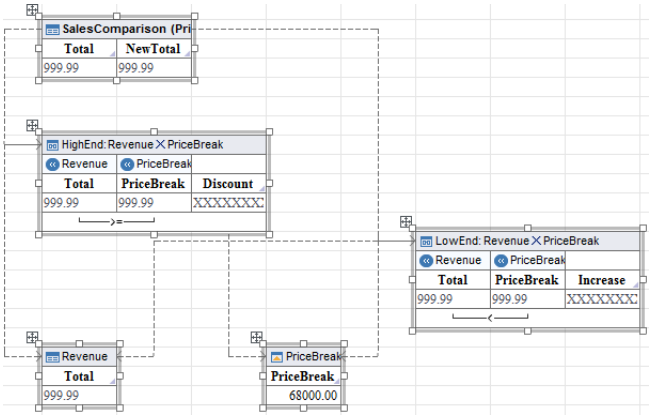
Now that you have the new sales total values for the large and small orders, you can analyze the benefits of the current ‘price break’ value by comparing the difference between the original sales total and the new sales total.

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 Form 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 block 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 blocks 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.

PRODUCTS1 (Primary)

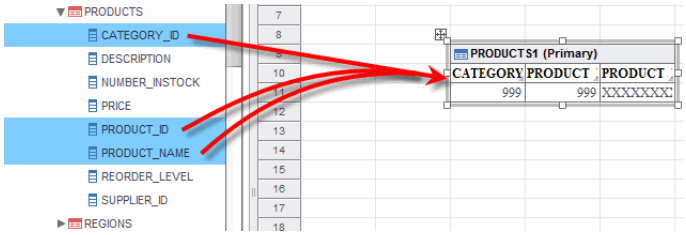
PRODUCT	PRODUCT	CATEGORY	DESCRIPTI
1	Xconnect Ser	3	XML connect
2	InsideView	3	Database brov
3	Info Folder	6	Document ma
4	Green Planet	4	Personal healt

```
This data source is not available in Style Scope Agile Edition.
```

1. In the **Asset** panel, expand the ‘Data Source’ node, the ‘Orders’ node, and the ‘TABLE’ node.

2. 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’.

3. Rename the ‘CATEGORY_ID’ column as ‘CAT_ID’. This will help to distinguish it from the table you will add next.

4. Drag the ‘TABLE’ > ‘SA’ > ‘CATEGORIES’ table to the Worksheet. This creates a new table named ‘CATEGORIES1’.

CATEGORIES1

CATEGORY	CATEGORY
999	XXXXXXX

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.

5. Press the '*fx*' button in the 'PRODUCTS1' table title row. This opens the 'Expression' dialog box. (See *Creating an Expression/ Formula Column* for more information.)

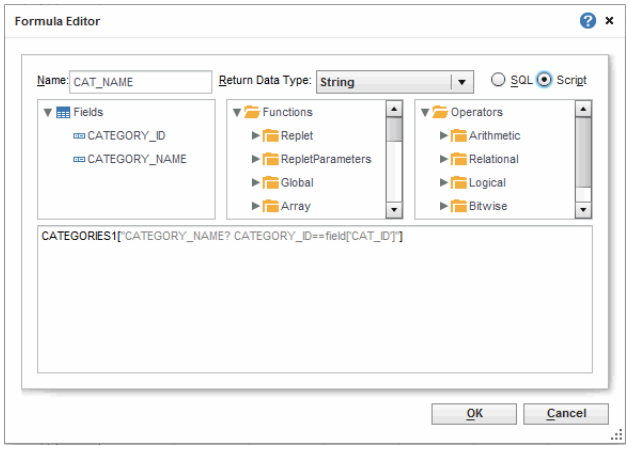
6. Enter 'CAT_NAME' as the name of the new expression column, and click 'OK'. This opens the Formula Editor.

7. 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."

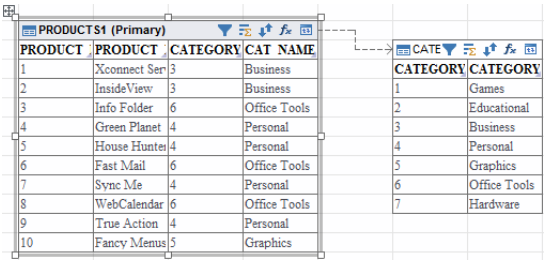
8. Uncheck the 'SQL' option at the top-right of the Editor.



9. 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.

10. 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 (Primary)

PRODUCT	PRODUCT	CATEGORY	CAT_NAME
1	Xconnect Ser	3	Business
2	InsideView	3	Business
3	Info Folder	6	Office Tools
4	Green Planet	4	Personal
5	House Hunte	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

CATE

CATEGORY	CATEGORY
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 blocks. 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 Report Design

Style Studio allows you to rapidly define sophisticated enterprise-grade reports, bind report elements to data sources, and deploy reports to a server.

1 Contents

This section explains the report design features of Style Studio. It covers the following major topics.

- *Introduction*

  Introduction to report design.

- *Report Design Tools*

  Introduction to Style Studio interface components for report design.

- *Report Layout*

  Description of how to set up report templates.

- *Report Elements*

  Description of available report elements.

- *Data Binding*

  Description of the Data Binding Wizard.

- *Advanced Topics*

  Advanced features such as report parameters and form elements.

- *Viewing a Report*

  Explains how to view and export reports.

- *Best Practices*

  Presents best practices for designing a large group of reports.

2 Introduction

A report consists of report elements such as text, charts, and tables, arranged on the page according to your particular design. Style Studio's report design features allow you to fully specify the structure and aesthetics of a report. This complete report specification is called a *report template*, and is stored with the file extension "*.srt*," which designates "Style Report Template." The report template in general does *not* contain any data.

In order for report elements (tables, charts, etc.) to display data, you must *bind* these elements to a data source such as a query, data model, or Data Worksheet. When the user executes the report, report elements that possess data bindings are populated by the data returned by the associated data sources. You can also specify the contents of report elements dynamically at runtime by using script. For more information, see *Advanced Report Scripting*. The following sections explain both the report layout process, and the data binding procedure.

Style Studio provides a number of tools that allow you to design report data sources. The *Advanced Data Modeling* component allows you to specify data sources, and to design data models and queries. The *Advanced Data Mashup* tools allow you to create assets based on data models, queries, or database tables, that represent the results of more complex data operations.

Once you have finished laying out the report and set the component data bindings, you can deploy (publish) your template to the server.

3 Report Design Tools

This section discusses some basic report design tools.

3.1 The Report Design Toolbox

The Style Studio **Toolbox** panel allows you to add various components to a report. The table below lists the element types. For information on adding report components, see *Inserting Elements*.

Table 1. Report Design Toolbox

	*Table Element*		*Report Bean*
	*Crosstab Element*		*Editable Region*
	*Chart Element*		*Separator*
	*Section Element*		*Bullet*
	*Text Element*		*Table of Contents*
	*TextBox*		*Heading*
	*Image*		*Running Total*
	*Painter*		*Newline*
	*Text Field*		*Space*
	*Text Area*		*Tab*
	*Button*		*Page Break*
	*Image Button*		Conditional Page Break
	*Check Box*		*Area Break*
	*Radio Button*		
	*Choice*		
	*List*		
	*Date Combo*		

See Also

Using Style Studio, for information on how to access Style Studio panels.

3.2 Using Grids and Rulers

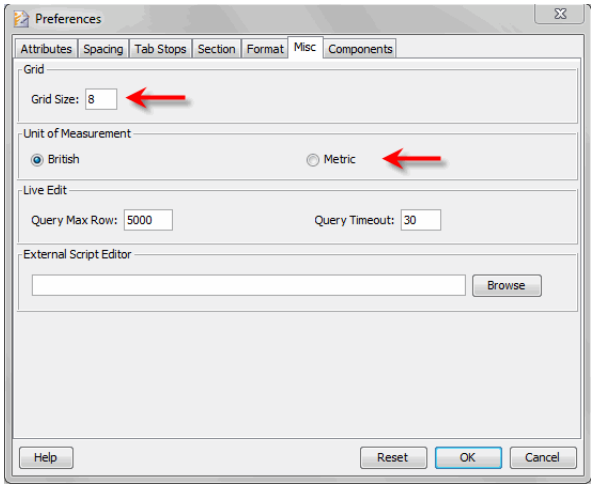
To display grids in the report design view, select the ‘Grid’ option from the Style Studio ‘View’ menu. These grids will help you to position elements in the report template.

To display rulers along the top and left side of the report, select the ‘Ruler’ option from the Style Studio ‘View’ menu

To specify the grid size and ruler units, follow the steps below:

1. Select ‘Preferences’ from the Style Studio ‘Report’ menu. This opens the ‘Preferences’ dialog box.

2. Select the **Misc** tab.



3. In the ‘Grid’ panel, enter the desired grid size.

4. In the ‘Unit of Measurement’ panel, specify the desired units: ‘British’ or ‘Metric’.

5. Click ‘OK’ to close the ‘Preferences’ dialog box.

4 Report Layout

The term *report* and *report template* are used interchangeably. here. "Report template" stresses the fact that the designed report contains no actual data, only layout and formatting information.

When you create a new report template, you can base the report template on a pre-designed meta-template (see *Meta-Templates*) or create a "blank" report template (see *Creating a New Blank Report*).

A blank report contains no elements, but it does represent a particular model. The report engine uses two layout models, *Flow Layout* and *Tabular Layout*. The layout method affects the manner in which report elements appear when the report is executed.

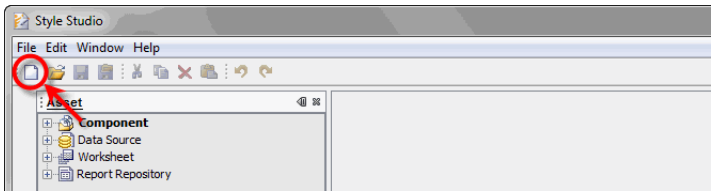
In flow layout, the report page consists of one or more page areas that all represent a single *flow*. Elements that you add to the report page will *flow* from one page area into the next based on the ordering that you assign. See *Flow Report Layout* for more information about flow layout.

In tabular layout, the report page is divided into independent grid cells, each of which hosts an *independent* sequential flow of elements. Elements that you place in one grid cell cannot flow into other grid cells. See *Tabular Report Layout* for more information about tabular layout.

4.1 Creating a New Blank Report

To create a new blank report, follow the steps below:

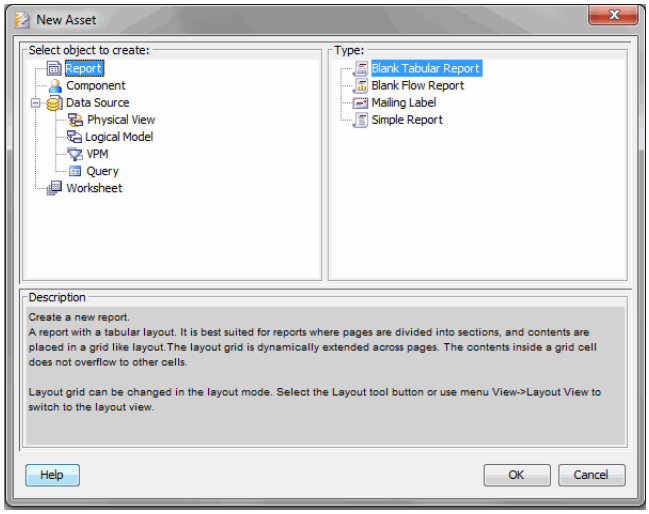
1. Press the 'New' button on the Style Studio toolbar.



   This opens the 'New Asset' dialog box.

2. In the left panel, select the 'Report' node.

3. In the right 'Type' panel, select either 'Blank Tabular Report' or 'Blank Flow Report'.



4. Click ‘OK’ to open the new report for editing.

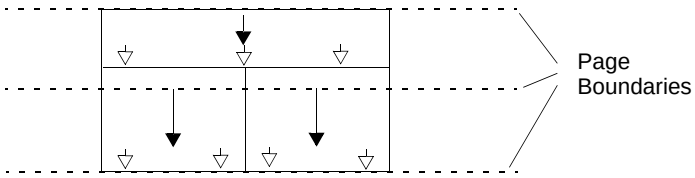
See Also

Creating a Report Based on a Meta-Template, to design a new report by using a boilerplate report layout.

4.2 Tabular Report Layout

The tabular layout model is ideal for reports whose contents are partitioned into rectangular regions. Instead of a single document flow, the report is separated into multiple grid cell regions, where each cell in the grid contains a separate flow.

Figure 1. Tabular Layout Grid



Report elements within a single tabular region are laid out inside the region, one by one in a flow. If the elements take up more space than the space allocated for the grid cell, the cell is extended vertically, so a given cell may span multiple pages. This in turn extends other cells on the same row in the grid. Elements in a tabular cell never overflow into other cells.

Examples of tabular reports are a stock performance report or an order form with distinct regions for customer details and order items.

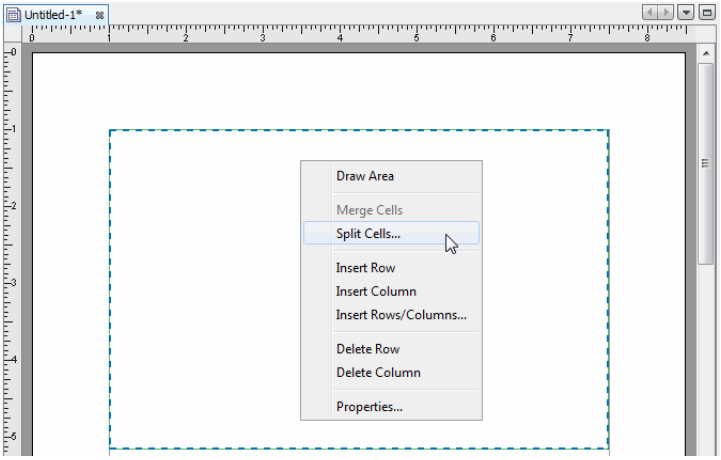
4.2.1 Setting up the Layout Grid

To create the grid layout for a tabular report, follow the steps below:



1. Click the ‘Layout View’ button in the toolbar. This toggles the report into Layout mode. (The report by default has just a single cell.)

2. Right-click in an existing cell of the tabular layout, and select the desired operation from the context menu.



To select multiple cells, drag the mouse across the desired cells.

The available operations include inserting new rows or columns into the grid, deleting rows or columns from the grid, splitting a cell into multiple rows or columns, and merging cells into a single cell.

3. (Optional) To quickly create a new grid cell at a particular location, follow the steps below:

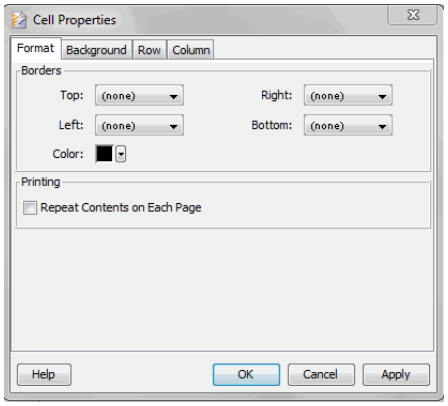


   a. Click the ‘Draw Area’ button on the toolbar.

   b. Draw a new cell at the desired location. This creates rows and columns on the grid to accommodate the new cell.

4. Repeat the above steps to create the desired layout grid.

5. To set properties for a grid cell, right-click the cell and select ‘Properties’ from the context menu. This opens the ‘Cell Properties’ dialog box.



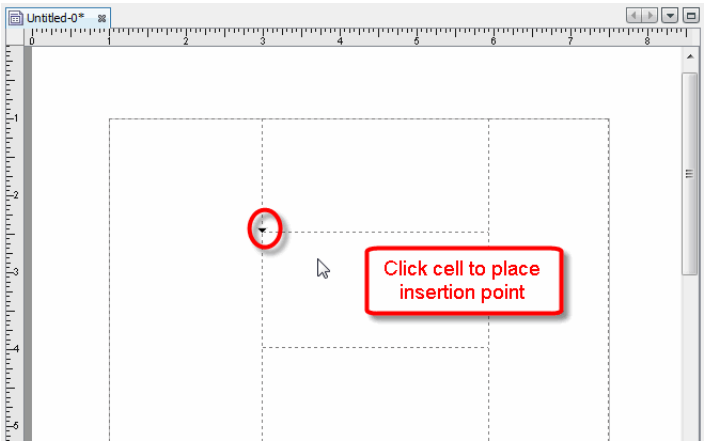
a. Under the **Format** tab, set the desired cell border style and color. Select 'Repeat Contents on Each Page' to print the contents of the region on each page of the report.

b. Under the **Background** tab, set the background color or image for the region.



6. Click the 'Layout View' button in the toolbar again to exit Layout mode and return to Normal mode.

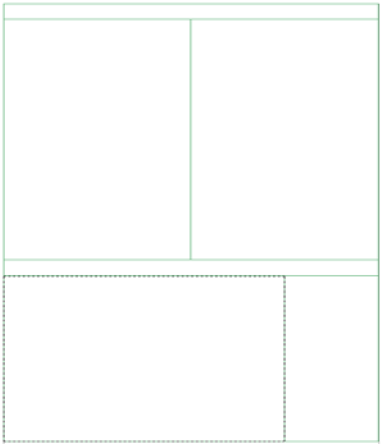
In Normal mode, you can add report elements to the grid cells that you created. Click in the desired grid cell to place the insertion point in that cell. Then click or drag an element from the **Toolbox** panel. (See *Inserting Elements*.)



Drawing the Layout Grid

Walkthrough

For this example, you will create the following layout:



To create the grid shown above, follow the steps below:

1. Create a new blank tabular report. (See *Creating a New Blank Report*.)

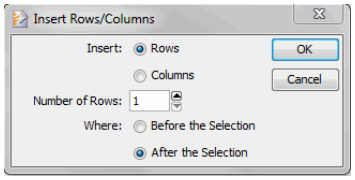


2. Click the 'Layout View' button in the toolbar to enter Layout mode. By default, the report has just one grid cell.

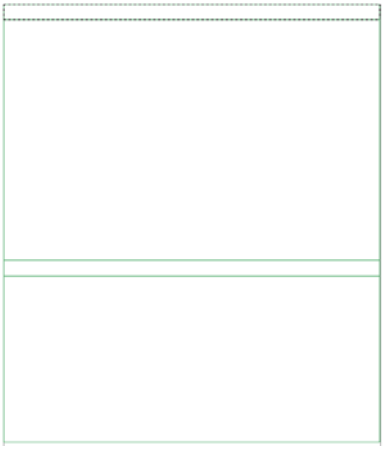
There are many approaches to arrive at the same layout. The steps shown here are just one possiblility.

3. Right-click the existing cell, and select 'Insert Row' from the context menu. This adds a narrow row at top.

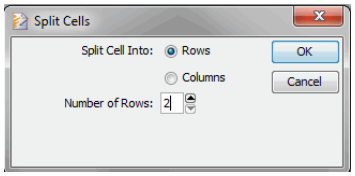
4. Right-click the larger cell, and select 'Insert Rows/Columns' from the context menu. Specify (as shown below) that you want to add two rows after the current selection, and click 'OK'.

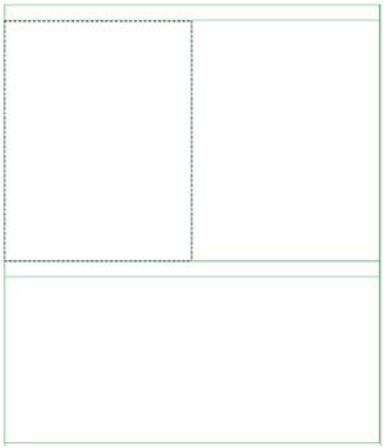


5. Resize the rows by dragging the cell borders to match the size of the rows that you want.

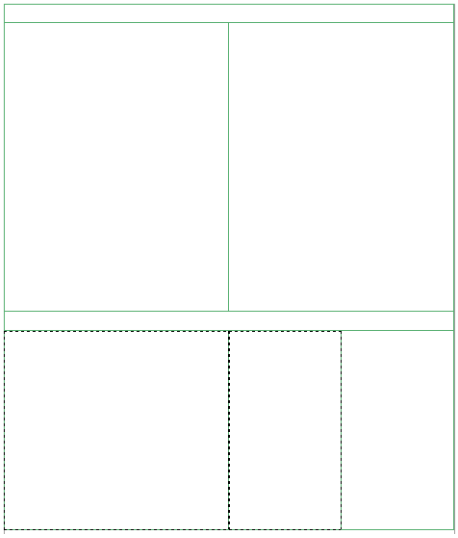


6. Right-click the second row, and select ‘Split Cell’ from the context menu. Specify (as shown) that you want to split into two columns.



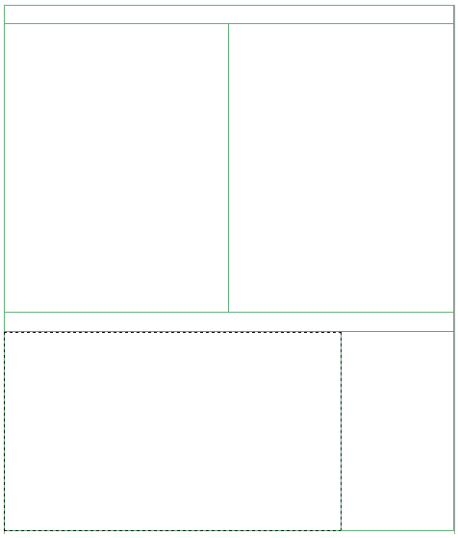


7. Right-click the bottom cell, and select ‘Split Cell’ from the context menu. Specify that you want to split the cell into three columns.



8. Drag across the two left cells (of the bottom row) to select them both.

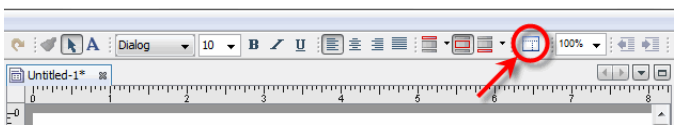
9. Right-click in either of the selected cells and choose ‘Merge Cells’ from the context menu.



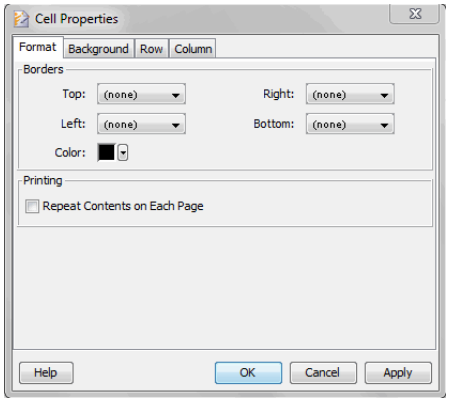
Setting Layout Grid Properties

To set layout properties for a tabular report, follow the steps below:

1. Press the ‘Layout View’ button in the toolbar to enter Layout mode.



2. Right-click in any grid cell, and select ‘Properties’ from the context menu. This opens the ‘Cell Properties’ dialog box.



3. Select the **Format** tab.

   a. Choose line styles and colors for the top, left, bottom, and right borders of the grid cell.

   b. (Optional) Select ‘Repeat Contents on Each Page’ if a particular cell must be repeated on every page (for example, if a cell on the right spans three pages, and the cell on the left must be repeated on every page).

4. Select the **Background** tab. Specify a background color or image, as desired.

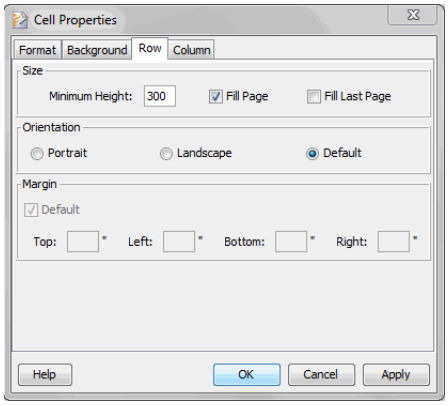
5. Select the **Row** tab.

   a. Specify a value for ‘Minimum Height’, if desired.

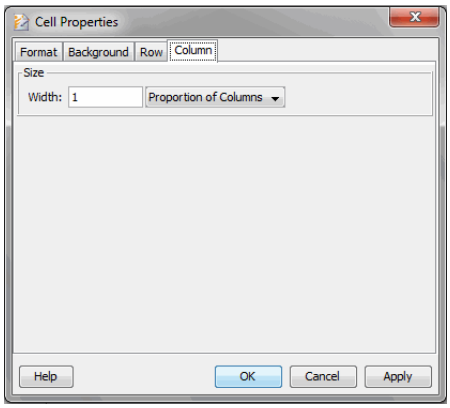
      To force the layout to span more than one page, set the minimum height to a large number, such as 700. This forces a new page to be added. Once the second page is available for layout, you can resize the grid cells appropriately.

   b. Select ‘Fill Page’ to force the bottom border of the grid cell to fall at the bottom of the page (rather than just satisfying the minimum height requirements).

   c. Add margins as desired in the ‘Margin’ panel. A margin introduces a page break for each cell whose margins are uniquely specified.



6. Select the **Column** tab. Select the desired method for specifying column width from the menu.



'Proportion of columns' specifies the width as a proportion of all the columns in the grid. In this case, one column is usually set to "1" and the other column widths are set as relative proportions (e.g., "2" would be twice as wide as the first). 'Percentage of page' specifies the width as a percentage of the page width, and 'Size in Points' specifies an absolute size in points.

See Also

Setting Page Orientation, for information on setting orientation for a region.

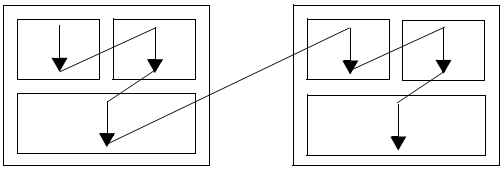
4.3 Flow Report Layout

The flow-based layout is similar to the document flow in a regular word processor. The simplest flow-layout report consists of one element flow, where each report element is placed in sequence on the page from top to bottom.

The page layout you specify for a flow report will modify the flow of the report elements. A page layout divides the page into multiple page areas,

and places these areas into a sequence that you define. Elements then "flow" from one page area to the next (on the same page), until all areas have been filled, at which point printing advances to the next page.

Figure 2. Page Areas



The page layout that you specify will be used for all successive pages until a new page layout is specified. For reports with a flow layout, you can specify the orientation (landscape/portrait) for each page individually.

Flow Layout is best suited for multi-column reports and reports that contain similar repetitive information. These include inventory listings or an employee contact information listings, where the contents are not divided into side-by-side view of report elements (as would be the case in a tabular report). Such multi-column reports cannot be created using a tabular report layout because the grid cells in a tabular report are independent of each other, and the contents of one cell cannot flow into another.

See Also

Creating a New Blank Report, for information on how to create a flow report.

Page Areas, for information on how to configure flow report areas.

4.4 Freehand Shapes

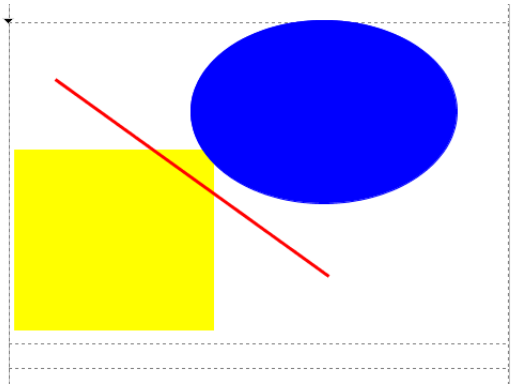
To embed a shape in a report as part of the document flow, you need to add a *Painter* element to the report via the reporting API by implementing the `Painter` class. See the `inetsoft.report.painter` package in the API documentation.

You can add shapes to the page in a more straightforward way by using Layout mode. To add a shape, follow these steps:

1. Click the 'Layout View' button in the Style Studio toolbar to enter Layout mode. The **Toolbox** panel changes to show the available shapes.

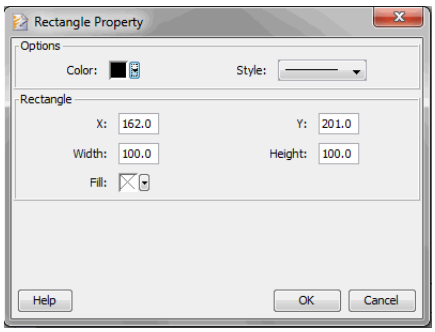
2. Double-click a shape on the toolbar, and then draw the shape on the page.

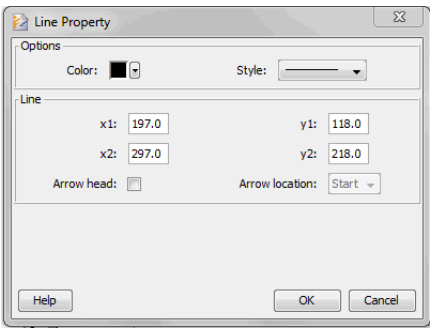
Hold down the **Shift** key to force equal width and height (circle or square).



3. To set the properties for a shape, right-click the shape, and select ‘Properties’ from the context menu. This opens the ‘Property’ dialog box for the shape.

   Properties of ovals include dimension, location, line color, and fill color. Rectangles additionally provide a line style property. Lines provide an arrowhead option, and choice of arrowhead placement (start or end).





You can add shapes only to the first page of a tabular report. However, in a flow report, shapes are attached to a specific page layout. Therefore, if a page layout is set on the report level, the shapes will be drawn on all pages until the page layout changes.

4.5 Meta-Templates

It is easy to design a single simple report from a blank template. (See *Creating a New Blank Report*.) However, it is more common to create a set of multiple related reports that share certain report properties, layout, and even report elements. In this situation, you should first design a *meta-template* or “boilerplate report” that encapsulates these common report features. You can then design the individual reports using the meta-template as a common prototype. This allows you to create a set of reports with a common look-and-feel.

4.5.1 Creating a Report Based on a Meta-Template

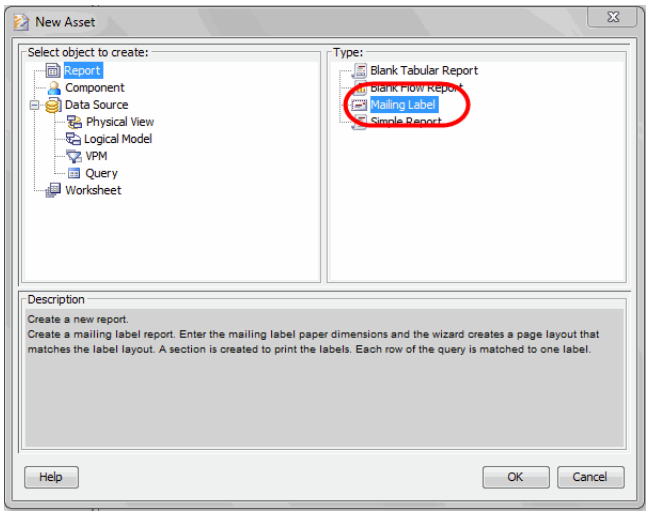
To create a new report based on a meta-template, follow these steps:

1. Click the ‘New’ button on the Style Studio toolbar. This opens the ‘New Asset’ dialog box.

2. In the left panel, click the ‘Report’ node.

3. In the right ‘Type’ panel, select a meta-template on which to base the report. The ‘Type’ panel lists both the predefined meta-templates and any custom meta-templates that you have designed.

4. Click ‘OK’.



   This opens a new report based on the selected meta-template. (The report may prompt you to first enter some customization information, such as data binding specification.)

A. report based on a meta-template generally contains a set of predefined report components (created by the meta-template designer to enforce a common look-and-feel) one or more *editable regions*. You can add new report components and modify predefined report components *only* within the editable regions. You cannot modify components or report properties (including page setup) that are outside of the editable regions.

When you save the report, changes that you made within an editable region (including modifications to pre-defined components within the editable region) are embedded together with the report template (*.srt* file). Subsequently, when you open or execute the report, layout and properties of components within the editable regions are read from the report file. However, layout and properties of components outside of the editable regions are always read from the external meta-template definition.

4.5.2 Designing a Meta-Template

Before you begin designing a meta-template, think about which properties and elements are common to the reports which will be created from the meta-template. For example, the layout will be identical for each report, as well as the page orientation. The reports will also generally share some elements as well, particularly in the report headers and footers.

When you design your meta-template, try to factor out as many of these common report components as possible, and design them into the meta-template. By incorporating these common components in the meta-template definition, you will be able to make future modifications in a single location, greatly improving maintainability. The changes that you make to the meta-template (outside of editable regions) will be automatically propagated to all reports that are based on the meta-template.

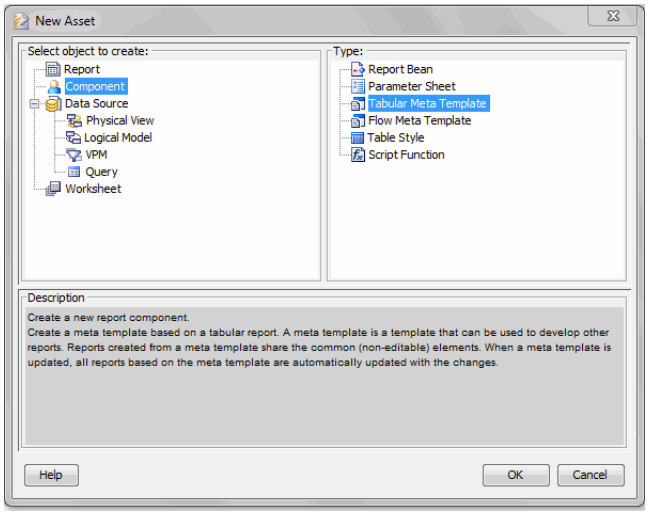
To define a new meta-template, follow the steps below:



1. Click the ‘New’ button in the Style Studio toolbar. This opens the ‘New Asset’ dialog box.

2. In the left panel, select ‘Component’ node.

3. In the right ‘Type’ panel, select either the ‘Flow Meta Template’ or ‘Tabular Meta Template’.



   The distinction between *flow* and *tabular* is the same as for blank reports, and determines the way that elements are positioned on the page. See *Tabular Report Layout* and *Flow Report Layout* for further details.

4. Click ‘OK’ to open the new meta-template.



5. Click the ‘Layout View’ toolbar button. Create a layout for the template as described earlier (i.e., *Setting up the Layout Grid* for tabular layout).

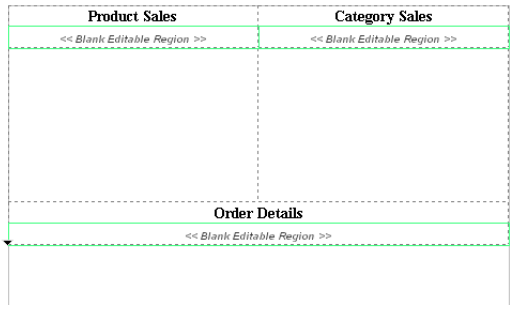
6. Click the ‘Layout View’ toolbar button again to exit Layout Mode.

7. Add *editable regions* to all the areas of the meta-template that you want the report designer to be able to modify. Follow the steps below:

   a. Click the report area where you want to add the editable region. This places the insertion point at that location.



   b. Click the ‘Editable Region’ button in the **Toolbox** panel. This adds a new editable region at that location



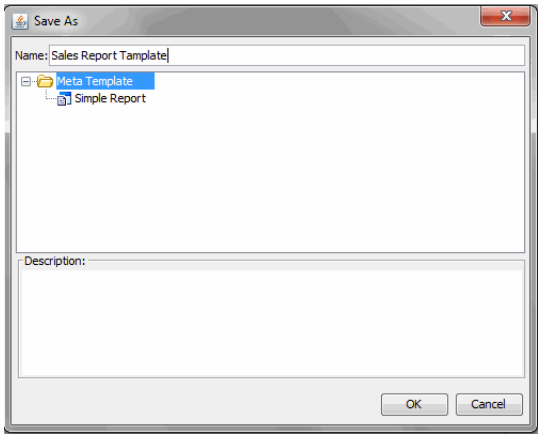
8. Insert report elements into the meta-template (inside or outside of the editable regions), as desired. See *Inserting Elements*.

   Report element that you insert within an editable region can be modified by the report designer when they create a new report based on this meta-template. Report elements that you insert outside of an editable region cannot be modified by the report designer (i.e., these elements are always referenced from the central meta-template definition). If you add a data element (chart or table) into an editable region, the report designer will be prompted to specify the ‘Data Binding’ for that element when they create a new report based on the meta-template.

Note: Meta-templates are saved together with other reusable components in *stylereport.srl.d*.

9. Click the ‘Save’ button. This opens the ‘Save As’ dialog box.

10. In the ‘Save As’ dialog box, enter a name for the meta-template and optional description, and click ‘OK’.



You should enter a meaningful name and useful description so that report designers can understand the proper utilization of the meta-template. The name and description will appear in the 'New Asset' dialog box.

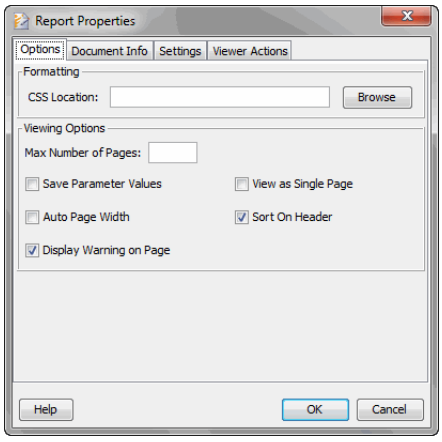
See Also

Creating a Report Based on a Meta-Template, for details on using a meta-template.

4.6 Report Properties

You can change global report settings from the 'Report Properties' dialog box. To open the 'Report Properties' dialog box, select 'Report Properties' from the Style Studio 'File' menu. The dialog box contains four tabs: **Options**, **Document Info**, **Settings**, and **Viewer Actions**.

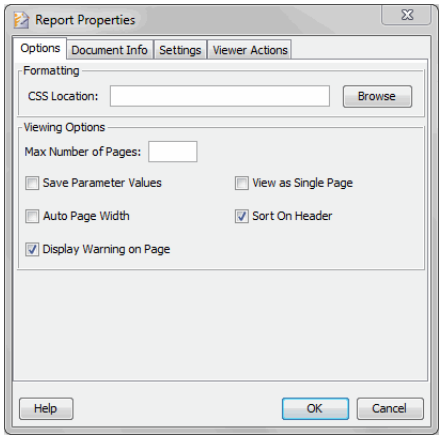
The **Options** tab provides properties related to report features.



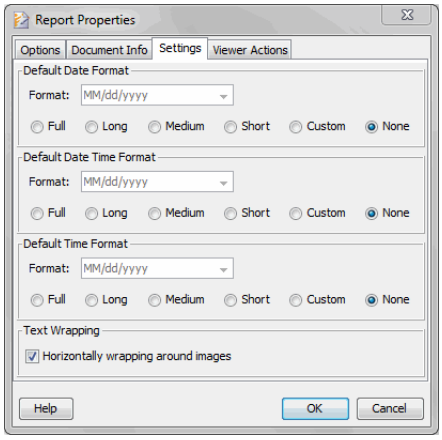
- **CSS Location**: Location of a CSS file which specifies formatting information for the report element styles. The location can be specified as a resource path (path relative to any folder on the classpath) or an absolute path. See *Styling a Report with CSS* for more information.

- **Max Number of Pages**: Sets the maximum number of report pages to be displayed in the Portal. This prevents a user from accidently generating an unexpectedly large report (for example, by entering an unreasonably large date range as a parameter). When the report exceeds the explicit limit, the displayed report is truncated as required, and an optional warning message is issued to the user (see below). To specify a global maximum for all reports, see the 'Report Maximum Page Count' property in the *Performance Options and Safeguards* section of the *Advanced Server Administration*.
- **Save Parameter Values**: Causes user-entered parameter values to be saved and reused every time the report is opened. (This applies to all types of parameters.) Users can change parameters by using the 'Customize' button in the Portal toolbar. See *Report Toolbar* in *Interacting with Reports* for more information.
- **View as Single Page**: Causes the report to be generated in the Portal as a single HTML page, without any pagination. This is useful for applications such as parameter sheets, where pagination is not desired. For most other cases, leave this option disabled.
- **Auto Page Width**: Specifies that the report width should expand to fit its contents. This only works if all table elements in the report are configured for 'Fit to Page'. See *Table Element* for more details.
- **Sort On Header**: Adds sorting functionality to table and crosstab headers. This allows the user to click the table column header to sort the corresponding data in ascending or descending order.
- **Display Warning Information**: If the 'Max Number of Pages' setting results in truncation of the displayed report, this options displays a warning message to the user on the final page of the report. The `warning.position.onheader` in the *sree.properties* file determines the position of the warning message. A value of `true` displays the message in the top right corner of the page header, while a value of `false` (default) displays the message in the bottom right corner of the page footer.

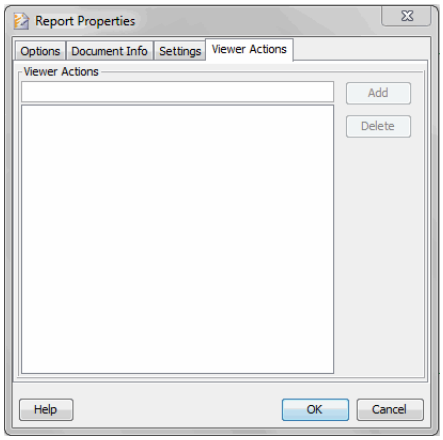
The **Document Info** tab lists special document properties. The Author property is set automatically when the report is initially saved.



The **Settings** tab contains options for default date format and text wrapping. See *Formatting* for more information about the available date format options.



The **Viewer Actions** tab lets you specify custom `ViewerAction` classes (implementing the `ViewerAction` interface; see the API documentation for more information). A `ViewerAction` object represents a custom button in viewer, specifying the icon and the label for the button, as well as its behavior.

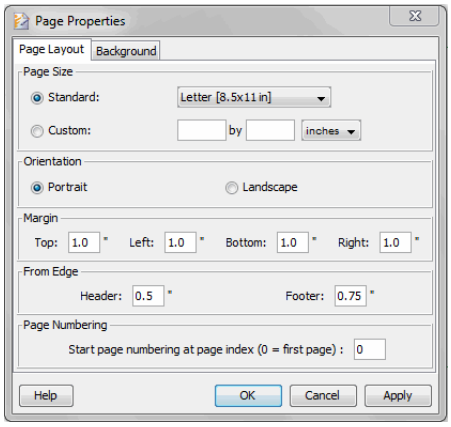


4.7 Page Setup

The default page setup places the header ½" from the top, and footer ¾" from the bottom. The printable area is the paper size minus the one-inch margins.

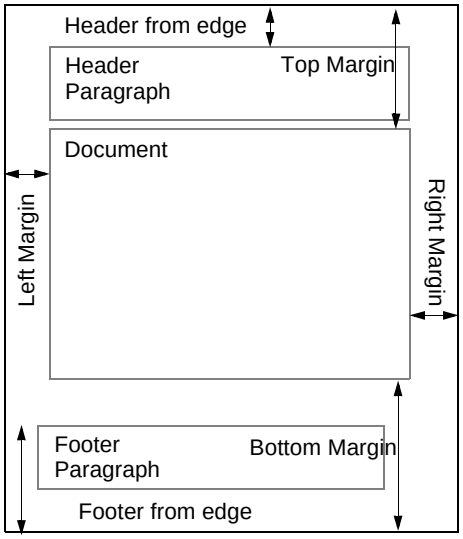
4.7.1 Configuring Page Properties

To configure page properties, select ‘Page Setup’ from the Style Studio ‘File’ menu. This opens the ‘Page Properties’ dialog box.



In addition to the paper size and orientation controls, the ‘Margin’ and header/footer ‘From Edge’ parameters control the size and position of the header and footer areas. ‘Start at Page Index (0)’ changes the page numbering starting page index, which also affects the total page count.

Figure 3. Page Margins, Headers and Footers



The ‘Top’ value in the ‘Margin’ panel and the ‘Header’ value in the ‘From Edge’ panel controll the position and size of the header. In the same manner, the ‘Bottom’ value in the ‘Margin’ panel and ‘Footer’ value in the ‘From Edge’ panel control the position and size of the footer.

4.7.2 Setting Page Orientation

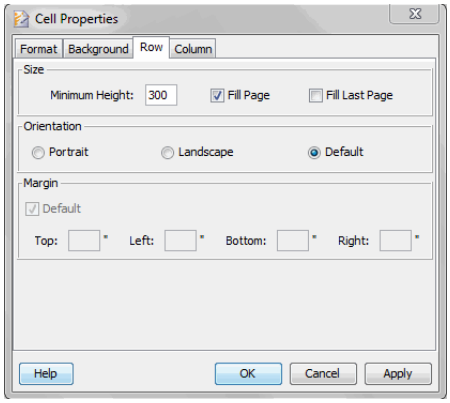
To change the *default* page orientation (landscape/portrait) for a tabular report or a flow report, follow these steps:

1. Select ‘Page Setup’ from the Style Studio ‘File’ menu. This opens the ‘Page Properties’ dialog box.
2. Choose the desired orientation from the ‘Orientation’ panel.
3. Click ‘OK’.

For a *tabular report* layouts (only), you can change the orientation for a *particular page* by associating a new layout with a particular *row* in the layout grid. Follow the steps below:



1. Click the ‘Layout View’ button in the toolbar to enter Layout mode.
2. Right-click the cell whose orientation you wish to change, and select ‘Properties’ from the context menu. This opens the ‘Cell Properties’ dialog box.
3. In the ‘Cell Properties’ dialog box, select the **Row** tab.



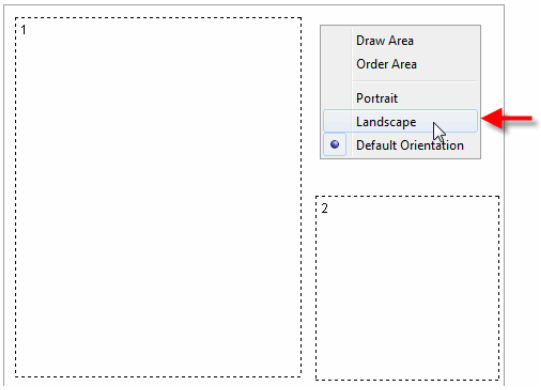
4. Select the desired orientation (‘Portrait’ or ‘Landscape’) and click ‘OK’.

For *flow report* layouts, you can also change the orientation for a particular page. To do this, follow the steps below:



1. Click the ‘Layout View’ button in the toolbar to enter Layout mode.

2. Right-click anywhere on the page (outside of the page areas) and select the appropriate orientation from the context menu.



See Also

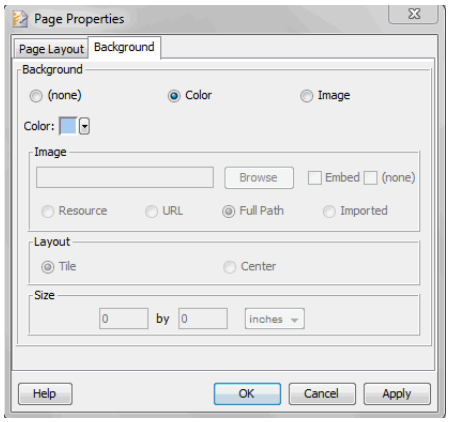
Setting up the Layout Grid, for more information about report layouts.
Setting Layout Grid Properties, for more information about grid cell properties.

4.7.3 Setting a Background Color/Image

To set a background color or image for a report, follow the steps below:

1. Select ‘Page Setup’ from the Style Studio ‘File’ menu. This opens the ‘Page Properties’ dialog box.

2. Select the **Background** tab.



3. To set a background color, click the 'Color' button, and select a color from the menu.

4. To set a background image, click the 'Image' button, and specify the location of the 'Image'.

   The options available for specifying a background image (i.e., 'Resource', 'URL', 'Full Path', 'Embed') are the same as those for the Image report element. See *Image* for more details.

5. (Optional) Specify the 'Size' and 'Layout' options for your image.

See Also

Freehand Shapes, for information on adding shapes to the background of a report.

4.8 Headers and Footers

Most reports use headers and footers to display report information, such as the report title, date/time, and page numbering.

4.8.1 Adding Headers and Footers to a Report

Walkthrough

The following example shows you how to add headers and footers that display title, page, and date information:



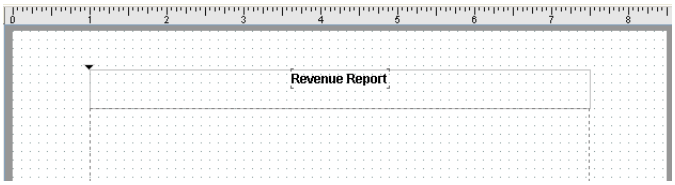
1. Click the header area or the 'Header' button to select the *Header* as the current editing target. The insertion point moves to the header area, indicating that new elements will be inserted into the header (rather than the report body).



2. Click the 'Text' button in the **Toolbox** panel. This adds a text element into the header.

3. Type the static text “Revenue Report” into the text element, and click away from the element.

4. Click the text element to select it. In the Style Studio toolbar, click the ‘Center’ alignment button and the ‘Bold’ button to change the formatting.





5. Click the ‘Footer’ button in the toolbar to move the insertion point to the footer.

6. Click the ‘Text’ button in the **Toolbox** panel. This adds a text element into the footer.

7. Type the text “`Page {P} of {N}`” in the text element. The symbols “`{P}`” and “`{N}`” are special page number tags that can *only* be used in headers and footers. When the report is generated, these tags are replaced with the current page number and the total number of pages, respectively. (See *Adding Page Numbers and Dates in Headers/Footers* for additional information.)

8. Select the footer again, and click the ‘Text’ button to add a second text element to the footer.

9. Enter the text “`{D} {T}`” in the text field. The symbols “`{D}`” and “`{T}`” are special date/time tags that can only be used in headers and footers. When the report is generated, the tags are replaced with the current date and time, respectively.



10. Click the new text element to select it. Click the ‘Right’ alignment button in the Style Studio toolbar to position the element as shown below.



The completed example is saved as *headerfooter1.srt* in the */examples/docExamples/design* directory.

4.8.2 Adding Page Numbers and Dates in Headers/Footers

Note: The final page count of a report depends on several factors, including current records in the database and any end-user parameter selections.

To insert page numbers or current date/time into a report, use the special page number and time/date tags (`{P}`,`{N}`,`{D}`,`{T}`) within a Text element. In general, this is much more reliable and robust than hard-coding these values into the report template itself.

The following table lists the special tags with their complete syntax:

Table 2. Header and Footer Tags (Page Numbers, Date/Time)

TAG	DESCRIPTION
{P}	Page number
{P,format}	Page number and number format
{N}	Number of pages
{N,format}	Number of pages and number format
{D}	Generation date
{D,format}	Date and format (*java.text.SimpleDateFormat*)
{T}	Generation time
{T,format}	Generation time and format (*java.text.SimpleDateFormat*)

To edit a Text element that contains these special tags, use one of the regular methods for editing text:

- Right-click the Text element, and select ‘Edit’ from the context menu.
- Right-click the Text element, and select ‘Properties’ from the context menu (or double-click the Text element) to open the ‘Properties’ dialog box. Select the **Contents** tab, and edit the contents as desired.

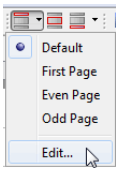
These special tags for page number and date/time are only effective when used within headers and footer Text (or Textbox) elements. If you use these tags within any other report element or in the report body, the runtime substitution is not performed. However, you can insert runtime page-numbering and date/time information into other elements or the report body by using script. For more information, see the *onPageBreak Handler* section the *Advanced Report Scripting*, as well as Appendix JS.12.1, *Date and Time Functions*.

4.8.3 Displaying a Header/Footer for Specific Page Types

To display a header or footer only on a a specific page type (first page, even pages, odd pages, or all pages default), follow the steps below:



1. Click the “arrow” icon next to the ‘Header’ or ‘Footer’ menu. This opens the ‘Header’ or ‘Footer’ menu.



2. Select the desired page type to which the header or footer should be applied. The available targets are first page, even pages, odd pages, or all pages (default).

3. Add components to the header or footer, as desired.

The components that you add to the header or footer will only display on pages of the selected type. To target headers or footers to specific *elements*, see *Associating Headers/Footers with Specific Elements*.

4.8.4 Associating Headers/Footers with Specific Elements

Sometimes you may need to specify headers and footers independent of the page numbering. For example, in a report that consists of multiple sections, you may need to display a different header and footer for each section.

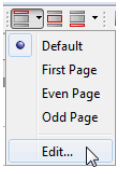
To do this, you can associate the header/footer specification with an *element* in the report (any element). Then, whenever the specified element is displayed on the page, the associated header/footer will be displayed as well. You can choose whether this element-associated header/footer should remain in effect only on pages where the specified element is displayed, or persist until a new element-associates header/footer is encountered in the report.

This element-based header/footer mechanism is more flexible than specifying the header/footer for specific pages using page numbers (see *Displaying a Header/Footer for Specific Page Types*). Because the page layout is based on the flow model, contents can shift and print on different pages depending on previous elements. By associating the header/footer with an element, you can ensure that the flow does not change the header/footer association.

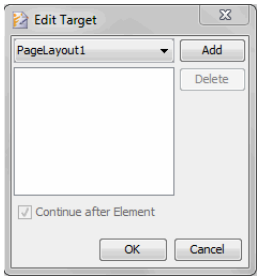
To associate a header/footer with an element, follow these steps:



1. Click the “arrow” icon next to the ‘Header’ or ‘Footer’ button. This opens the ‘Header’ or ‘Footer’ menu.



2. Select the ‘Edit’ option. This opens the ‘Edit Target’ dialog box. This lists all elements in the report.



3. Select an element to associate with the header (footer), and click ‘Add’. Repeat to add any additional elements that will have distinct headers/footers.

   To display distinct headers/footers for different sections of the report, add the heading elements (e.g., section titles) to the list. You can then create a distinct header for each section. If more than one element on a page has an associated header/footer, the header/footer corresponding to the last element is used.

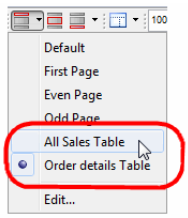
   If necessary, you can insert invisible elements into the report for the purpose of changing the header/footer on the page. For example, the last page of the report requires a different header, you can add a “space” element at the end of the report, and associate a new header with this element.

4. (Optional) Select ‘Continue after element’ to make the header/footer continue until the next element-associated header/footer. Otherwise, the header/footer will display only on pages that contain the element.

5. Click ‘OK’ to close the dialog box.



6. Again click the “arrow” icon next to the ‘Header’ or ‘Footer’ button to open the menu. Observe that the elements which you added are now listed in the menu.



7. Select the element from the list to which you want to associate the distinct header/footer.

8. Add components to the header or footer, as desired.



To create a header/footer for a different report element, again click the ‘Header’ (‘Footer’) button, and select the desired target from the menu.

5 Report Elements

A report consists of a sequence of report elements. The report engine processes elements sequentially during formatting and printing in the order that they appear in the report.

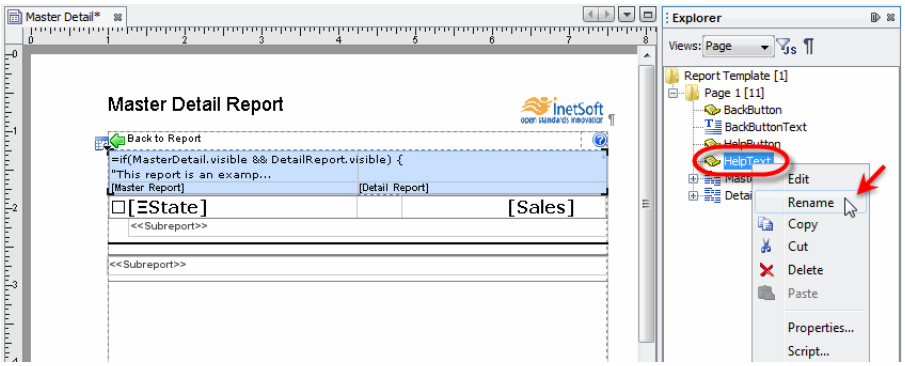
5.1 Element ID and Alias

Each element is identified by a unique element ID. The element ID is assigned dynamically when you add an element to the report. The element ID allows you to refer to an element from within report script, when the template is accessed programmatically, and for many other purposes.

Use meaningful names as element IDs to facilitate maintenance and debugging.

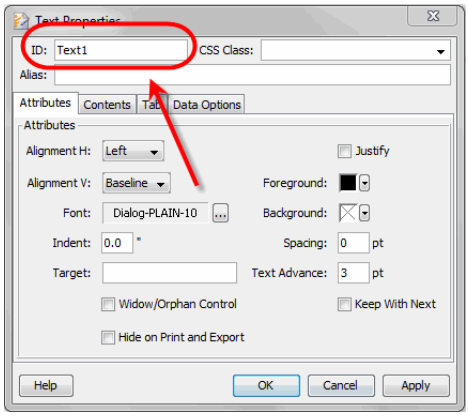
You can view the ID of the currently selected element in the status bar at the bottom of the Style Studio window. Likewise, in the Style Studio **Report Explorer** panel, report elements are identified by their element IDs.

To change an element's ID, right-click the element in the **Report Explorer** panel, and select 'Rename' from the context menu.



Alternatively, follow the steps below:

1. Right-click the element, and select 'Properties' from the context menu. This opens the element's 'Properties' dialog box.



2. Enter a meaningful name in the 'ID' field, and click 'OK'.

3. (Optional) Enter a meaningful name in the 'Alias' field. This is the name that identifies the element in the User Portal's Report Explorer tool. If left unspecified, the ID will be displayed for this purpose.

5.2 Inserting Elements

To insert an element into a report, follow these steps:

1. Click in the region of the report (body, header, etc.) where you wish to insert the element. This moves the insertion point (black triangle) to indicate the specified location.

   If an existing element is already selected, the new element will be inserted *before* the selected element.

2. In the **Toolbox** panel, click the element you wish to insert. This inserts the element at the specified location.

   Alternatively, you can drag the element from the **Toolbox** panel to the desired location in the report.

The alignment of consecutive elements determines which element is placed on a new line. When the alignment changes from one element to the next in a leftward direction (right → centered, centered → left), the next element is automatically placed on a new line. The element type also affects its placement on a new line.

To position elements with complete freedom, place the elements inside a Section. See *Section Element* for full details.

5.2.1 Text Mode vs. Insertion Mode

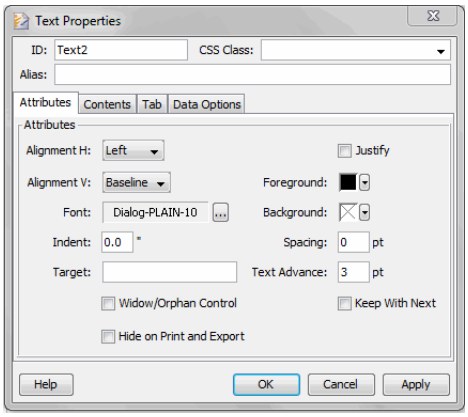
By default, when you click in a report, this places the insertion point at the specified location. However, you can also use Style Studio in a “text insertion” mode, similar to a word processor. To do this, click the ‘Text Tool’ button in the toolbar. Now, when you click in the report, this automatically opens a text element for editing.

To return to the default insertion behavior, click the ‘Pick’ tool.

5.3 Property Dialog Boxes

Every element has a ‘Property’ dialog box that allows you to modify attributes of the corresponding element. To open the ‘Property’ dialog box, do one of the following:

- Right-click on the element and select ‘Properties’ from the context menu.
- Double-click on the element.



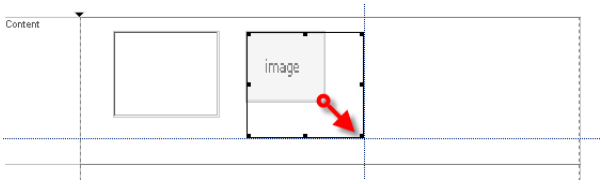
To simultaneously edit the attributes of multiple elements (of the same element type), Ctrl-click the elements to select them all. Then right-click on one of the elements, select ‘Properties’ from the context menu. This will open a dialog box exposing the common properties that you can set.

Most element ‘Properties’ dialog boxes consist of several tabbed panes. The common **Attributes** tab offers common features such as alignment, font, color and options to hide the element on print/export. Many elements also have a **Layout** tab, which has the anchor, size, and wrapping attributes.

You can also set properties for single or multiple components by using the **Property** panel of the Report Explorer. See *Report Explorer View* for full details.

5.4 Positioning and Sizing a Report Element

You can reposition and resize a report element from within its 'Properties' dialog box. (See *Property Dialog Boxes* for more information.) For an anchored element, or an element placed within a Section, you can also move and resize the element by dragging the element handles.



5.5 Brushing Element Styles

You can copy or *brush* visual properties from one element to another element. This provides a quick way to give the same style to multiple elements. The following properties can be brushed:

- Font
- Foreground color
- Background color

To brush styles from one element to another element, follow the steps below:

1. Select (click) the report element from which to copy the styles.
2.  Click the 'Brush' button in the toolbar. (The pointer icon changes to indicate that a brushing operation is in progress.)
3. Click the report element to which you want to apply the styles.

This applies the styles from the first element to the second. You can make further modifications to the copied styles in the usual manner (see *Property Dialog Boxes* for information on modifying visual properties).

Example: Brushing

In the following example, you will copy styles from one Text element to another Text element.

1. Create a new report. (See *Creating a New Blank Report*.)
2. Click the 'Text' button on the left-side toolbar to add a new Text element to the report.
3. Type the text "All Sales" into the element, and then click away from the element to end editing.

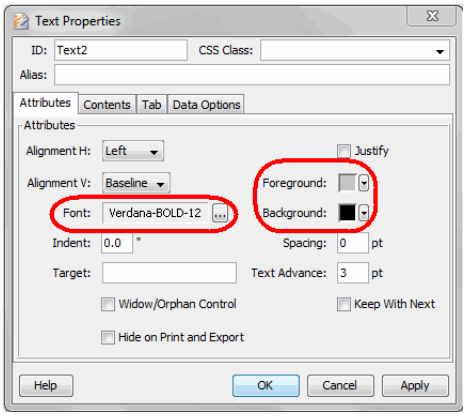
4. Right-click the Text element and select ‘Properties’ from the context menu. This opens the ‘Text Properties’ dialog box.

5. In the ‘Text Properties’ dialog box, make the following settings:

   a. Click the ‘Font’ button, and select “Verdana Bold 12pt.”

   b. Click the ‘Foreground’ menu, and select a light-gray color.

   c. Click the ‘Background menu, and select a dark-gray color.



6. Click ‘OK’ to close the ‘Text Properties’ dialog box.

7. Click to the right of the Text element to place the insertion point there, and press ‘Enter’ twice on the keyboard to insert two line feeds.

8. Click the ‘Text’ button on the left-side toolbar to add a second Text element. The new Text element is in edit mode by default.

9. Enter the text “Top Sales” into the element, and then click away from the element to end editing.

10. Brush the style of the first Text element onto the second Text element. Follow these steps:

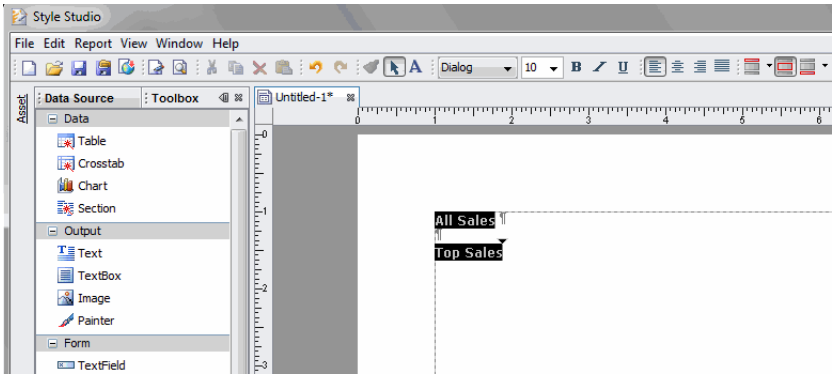
    a. Click the first Text element to select it.



    b. Click the ‘Brush’ button in the toolbar. (The pointer icon changes to indicate that a brushing operation is in progress.)

    c. Click second Text element.

This applies the styles from the first element to the second, so that they both share the same font, foreground color, and background color.



¢

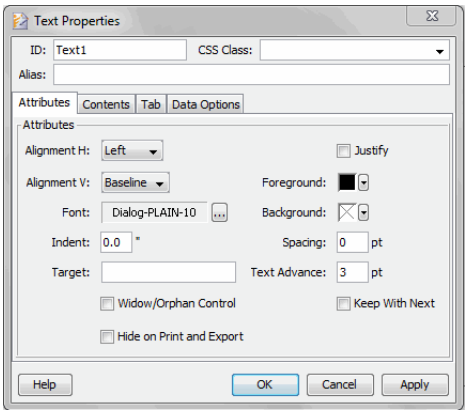
5.6 Text Element

Text is one of the most basic components of a report. Style Studio offers two elements for adding text, the Text element and the Textbox element. The following sections explain these two elements and their differences.

5.6.1 Text Element

Text elements are the basic elements for adding text. They possess relatively few properties, and these relate mostly to font and positioning.

Figure 4. Text Properties Dialog Box



When you plan to enter a large amount of text, you may find it useful to select the ‘Text Tool’ button in the toolbar. This allows you to click in the report and start typing (similar to a word processor). You can also use the ‘Text Tool’ to easily modify text elements that already exist in the report.

Adding a New Text Element

To add a text element, follow these steps:

1. Click in the report to place the insertion point at the desired location.



2. In the **Toolbox** panel, click the ‘Text’ icon. This creates a gray shaded box to the right of the last element added.



You can also bind the Text element to a data source. See *Binding Data to a Text Element*.

3. Type the desired text into the box.

   The Text element resizes itself dynamically based on the text that you enter. The familiar text formatting controls apply are available in the Style Studio toolbar.



Text Wrapping

When one or more anchored elements exist on the same line as the Text element, the Text element can wrap around the anchored elements. The wrapping style is specified on the **Layout** tab of the ‘Properties’ dialog box for the *anchored component*.

Table 3. Text Wrapping

	Flow overlaps the float element.
	Wraps around the left side of the float element.
	Wraps around the right side of the float element.
	Wraps around both sides of the float element.
	No contents allowed on either side of the float element.

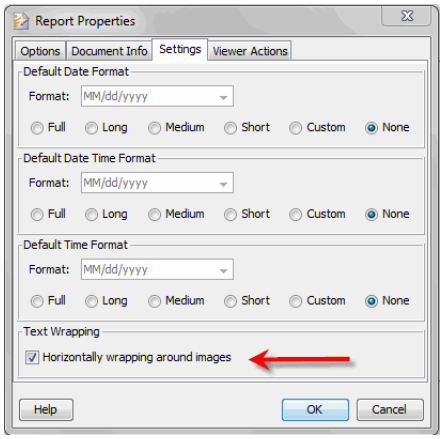
Wrapping occurs on both sides of the element (if the wrapping style allows).

To specify a vertical wrapping direction (rather than the default horizontal), follow the steps below:

1. Select ‘Report Properties’ from the Style Studio ‘File’ menu.

2. Select the **Settings** tab.

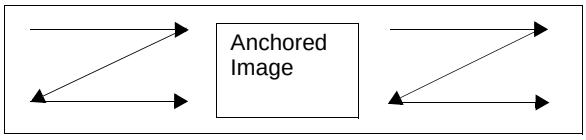
3. In the 'Text Wrapping' panel, disable the 'Horizontally wrapping around images' option.



The wrapping direction is a global attribute for the report template; therefore, it affects the wrapping of the elements in the entire report.

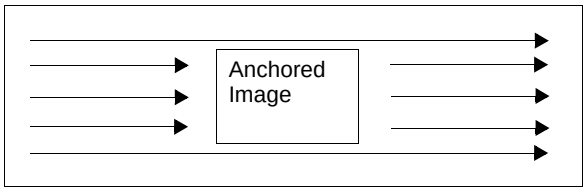
Vertical wrapping divides the area between the edges of the page and the anchored element into printable sub-areas. Text is shown in each sub-area before advancing to the next sub-area.

Figure 5. Vertical Text Wrapping for Anchored Float Element



If the wrapping direction is set to horizontal, the flow of text occurs across the sub-areas, from left to right, until the text is exhausted or the flow continues onto the next line (below the anchored element).

Figure 6. Horizontal Text Wrapping for Anchored Float Element



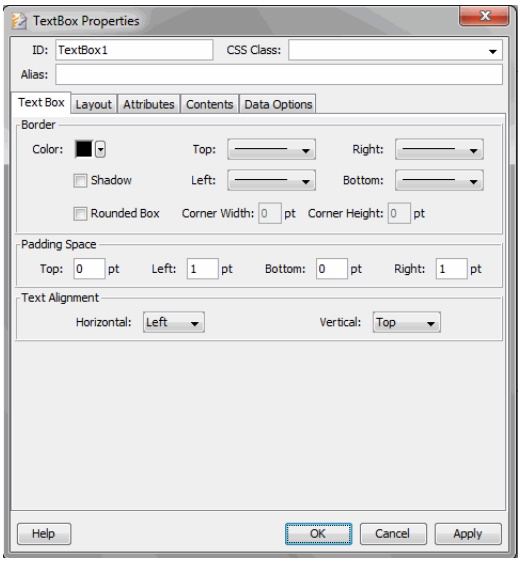
See Also

Positioning and Sizing a Report Element, for more information about anchoring.

5.6.2 TextBox

The TextBox element has a defined rectangular area around which borders can be placed. (The basic Text element does not have this feature). You can control the border style, corner style, color, and other attributes from the 'TextBox Properties' dialog box.

To open the 'TextBox Properties' dialog box, right-click the TextBox and select 'Properties' from the context menu.



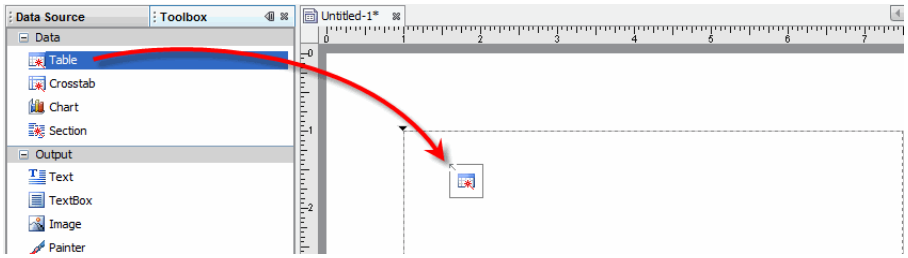
- **Border Style**: The default border uses thin lines. To change the line style, use the 'Top', 'Bottom', 'Right' and 'Left' menus under the **Text Box** tab. To provide a simple shadow effect, select the 'Shadow' option.
- **TextBox Color**: To change the TextBox border color, use the 'Color' chip under the **Text Box** tab. To change the TextBox text color or the fill color, use the 'Foreground' and 'Background' color chips under the **Attributes** tab.
- **Rounded Borders**: The default box uses square corners. To change the style to rounded corners, select the 'Rounded Box' option under the **Text Box** tab. Then enter values (in pixels) for the 'Corner Width' and 'Corner Height' fields to set the corners' horizontal and vertical radii, respectively. Larger radius values produce softer corners.
- **Padding**: You can add padding around the inner border of the TextBox so that text contents do not run too close to the border. Under the **Text Box** tab, enter the desired spacing values using the 'Padding Space' panel.

5.7 Table Element

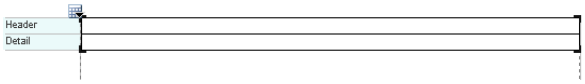
There are two elements that display tabular data: Table and Section. This section explains how to use the Table element. See *Tables vs. Sections* for a discussion of the differences between Table and Section elements.

5.7.1 Adding a Blank Table to a Report

To add a blank Table element to the report, drag the 'Table' icon from the **Toolbox** panel to a place in the report.



This adds a blank table element at the insertion point.



See *Creating a New Table Using Freehand Operations* to add data fields to the Table.

See Also
Binding Data to a Table Element, to add a Table containing data.

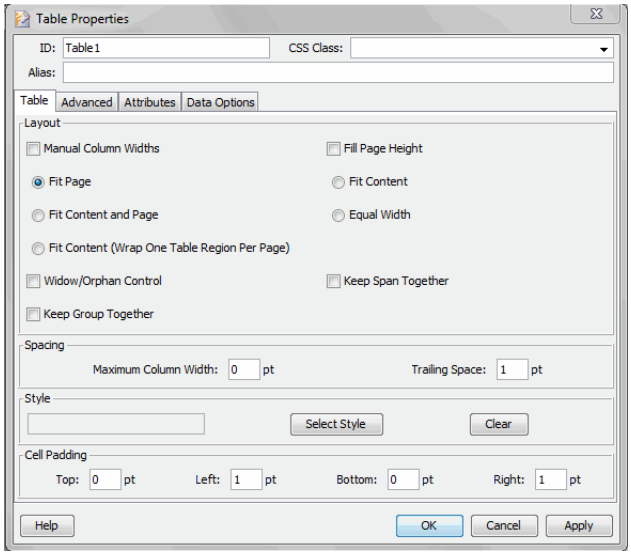
5.7.2 Table Properties

The 'Table Properties' dialog box allows you to specify layout and style for a Table. The following sections explain the available features.

Table Layout

To specify a table's layout properties, follow the steps below:

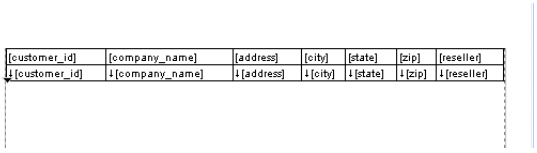
1. Right-click the table, and select 'Properties' from the context menu. This opens the 'Table Properties' dialog box.



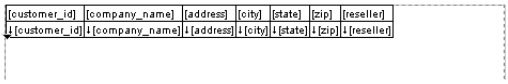
2. In the ‘Layout’ panel, select one of the available layouts, and click ‘OK’.

When the table is bound to a data source (see *Binding Data to a Table Element*), the table column widths are calculated automatically based on the size of the result set. Use the layout options in the ‘Table Properties’ to modify how these widths are calculated:

- **Fit Page**
  The table width is set to the width of the page. The column widths are approximately proportional to the size of the data they hold. If the contents are too wide to fit in the column, the size of the row is increased and/or the contents are wrapped.

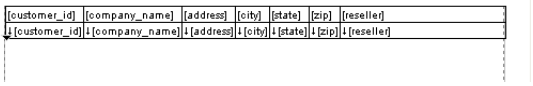


- **Fit Content**
  The column widths are sized to fit the data they contain. If a row is wider than the page, the table is wrapped around as shown below.

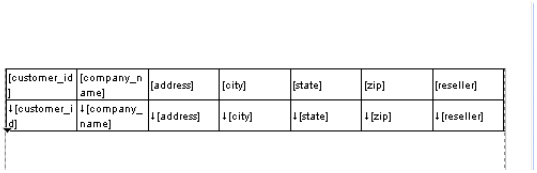


- **Fit Content and Page**
  This is a variation of the ‘Fit Content’ layout option. In this case, the last column of each table region is always adjusted to fill the entire page

area. In other words, all columns are sized to exactly fit their contents with the exception of the last column in each table region, which is sized to fit the remaining page width.

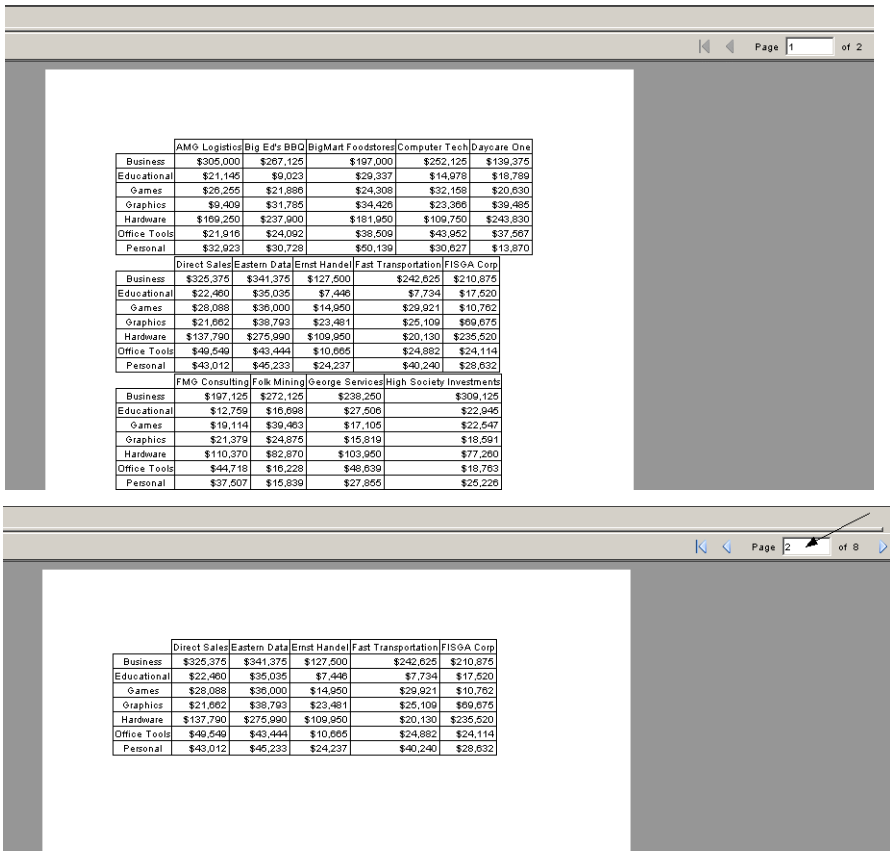


- **Equal Width**
  The table width is calculated in the same way as in the ‘Fit Page’ layout. Instead of distributing width to the columns based on their contents, all columns are assigned the same width.



- **Fit Content (Wrap One Table Region Per Page)**
  This layout option is similar to ‘Fit Content’. It is particularly useful for cross-tab (pivot) tables, where the number of column headers is dynamic. This could cause the horizontal span of the table to get quite large. If a table is too wide, the table is horizontally ‘chopped’ up into multiple regions, with each region displayed on a new page.

Figure 7. Fit Content vs. Fit Content (Wrap One Table Region Per Page)



	AMG Logistics	Big Ed's BBQ	BigMart Foodstores	Computer Tech	Daycare One
Business	$305,000	$267,125	$197,000	$252,125	$139,375
Educational	$21,145	$9,023	$29,337	$14,978	$18,789
Games	$26,255	$21,886	$24,308	$32,158	$20,630
Graphics	$9,409	$31,785	$34,426	$23,366	$39,485
Hardware	$169,250	$237,900	$181,950	$109,750	$243,830
Office Tools	$21,916	$24,092	$38,509	$43,952	$37,567
Personal	$32,923	$30,728	$50,139	$30,627	$13,870

	Direct Sales	Eastern Data	Ernst Handel	Fast Transportation	FISGA Corp
Business	$325,375	$341,375	$127,500	$242,625	$210,875
Educational	$22,460	$35,035	$7,446	$7,734	$17,520
Games	$28,088	$36,000	$14,950	$29,921	$10,762
Graphics	$21,662	$38,793	$23,481	$25,109	$69,675
Hardware	$137,790	$275,990	$109,950	$20,130	$235,520
Office Tools	$49,549	$43,444	$10,665	$24,882	$24,114
Personal	$43,012	$45,233	$24,237	$40,240	$28,632

	FMG Consulting	Folk Mining	George Services	High Society Investments
Business	$197,125	$272,125	$238,250	$309,125
Educational	$12,759	$16,698	$27,506	$22,946
Games	$19,114	$39,463	$17,105	$22,547
Graphics	$21,379	$24,875	$15,819	$18,591
Hardware	$110,370	$82,870	$103,950	$77,260
Office Tools	$44,718	$16,228	$48,639	$18,763
Personal	$37,507	$15,839	$27,855	$25,226

	Direct Sales	Eastern Data	Ernst Handel	Fast Transportation	FISGA Corp
Business	$325,375	$341,375	$127,500	$242,625	$210,875
Educational	$22,460	$35,035	$7,446	$7,734	$17,520
Games	$28,088	$36,000	$14,950	$29,921	$10,762
Graphics	$21,662	$38,793	$23,481	$25,109	$69,675
Hardware	$137,790	$275,990	$109,950	$20,130	$235,520
Office Tools	$49,549	$43,444	$10,665	$24,882	$24,114
Personal	$43,012	$45,233	$24,237	$40,240	$28,632

'Keep Span Together' applies to tables with multiple levels of headers, and ensures that parent-child header combinations are displayed on the same page. The 'Keep Group Together' option applies to tables with grouping, and uses forced page breaks to keep all detail rows of a group together on the same page if possible.

Note: there is a global maximum of 450 which can be changed by setting *preference.trail.max* in the .stylereport file.

It may be useful to set the maximum width of the table columns, particularly when one of the 'Fit Content' layout types is selected. The 'Table Properties' dialog box allows the maximum width in points to be entered.

To resize a table column manually, simply drag the table cell border with the mouse. (This enables the 'Manual Column Widths' option in the 'Table Properties' dialog box, and disables the automatic layout specification).

Matching Column Sizes in Two Tables

```
Column matching is not available for Crosstabs.
```

You can match the column widths of one table to the column widths of another table. For example, assume that you have created a freehand table

with customized column widths, and you want to match the widths of another table's columns to this table. You can set up an exact match one-to-one column matching, or configure a more customized matching, such as matching the width of multiple columns in one table to a single column in the other.

To match column sizes between two tables, follow the steps below:

1. Right-click the table whose column size you wish to change. Select 'Properties' from the context menu to open the 'Table Properties' dialog box.

2. Select the **Advanced** tab.

3. In the 'Column Width' panel, select 'Match Column Width'.

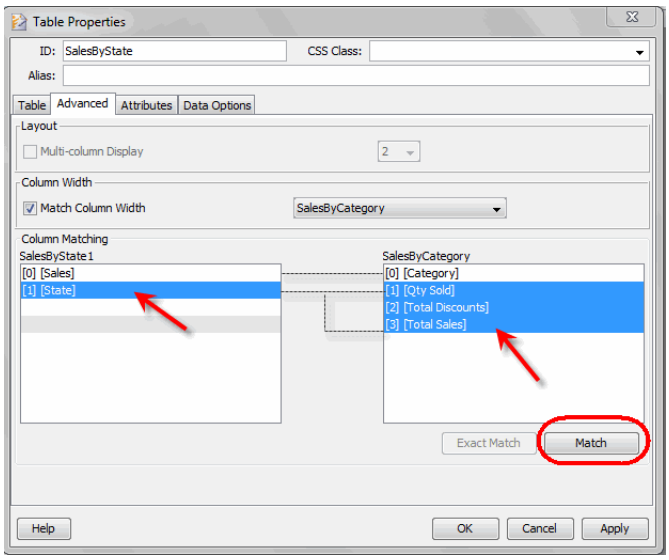
4. If the two tables contain the same number of columns, and you wish to perform a one-to-one sequential column match, click the 'Exact Match' button.

5. If the two tables contain different numbers of columns, follow the steps below to perform a custom match:

   a. In the 'Column Matching' panel, click to select a column in the *left* list (Ctrl-click or Shift-click to select multiple columns.)

   b. Click (Ctrl-click or Shift-click) to select the column or columns in the *right* list that should be matched to the selected columns in the left list.

   c. Click the 'Match' button.



d. Repeat the above steps to match additional sets of columns.

6. Click ‘OK’ to close the dialog box.

Adding a Table Style

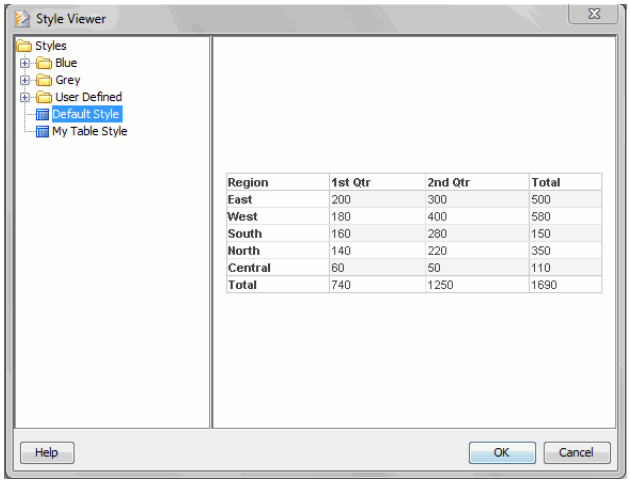
A *table style* is a set of predefined table presentation elements, such as colors, borders, fonts, and other visual attributes. Table styles allow you to set a visual treatment for the table as a whole.

To apply a table style to a table, follow the steps below:

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

2. Select the **Table** tab.

3. In the ‘Style’ panel, click ‘Select Style’. This opens the ‘Style Viewer’ dialog box, which displays a hierarchical list of styles.



4. Select a style from the list, and click ‘OK’.

See Also

Table Formatting, for information on styling individual cells, rows, or columns.

Creating a Custom Table Style

In addition to using the predefined table styles (see *Adding a Table Style* or *Table Properties* in *Advanced Dashboard Design*), you also can create your own custom table styles. A custom table style is a reusable component that can be applied to multiple tables in reports or Viewsheets.

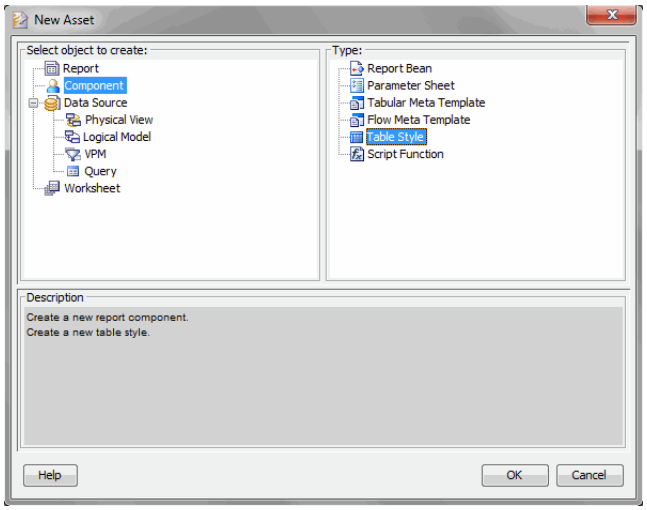
To create a custom table style, follow the steps below:

You do not need to open a report or Viewsheet in order to create a table style.

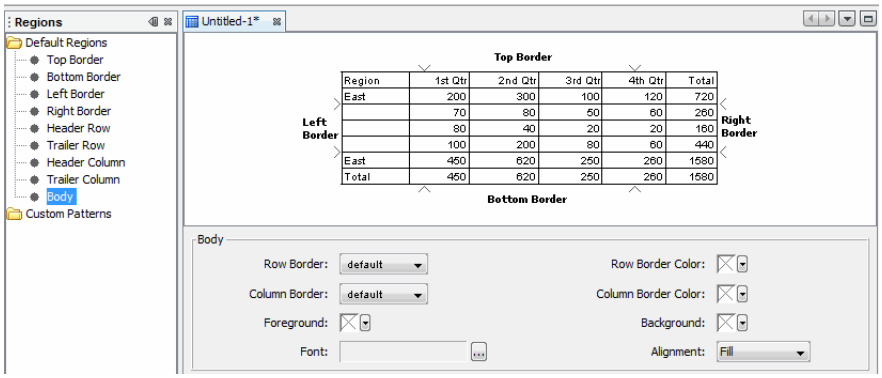
1. Click the ‘New’ button in the Style Studio toolbar. This opens the ‘New Asset’ dialog box.

2. In the left panel, select the ‘Component’ node.

3. In the right ‘Type’ panel, select the ‘Table Style’ node.



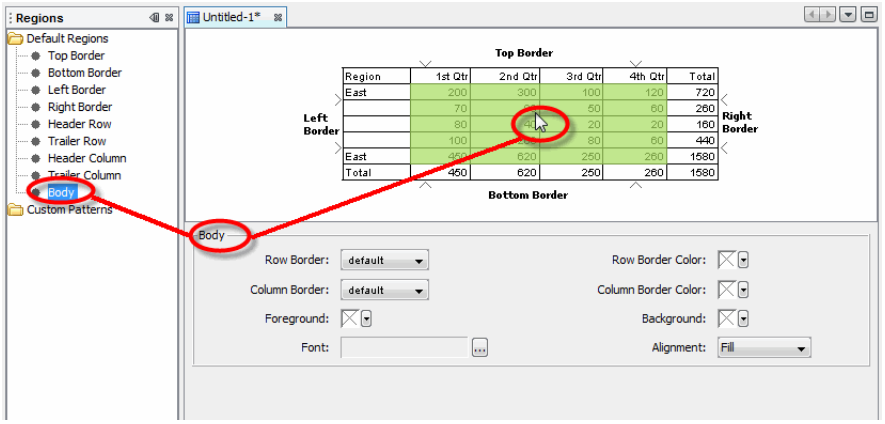
4. Click ‘OK’. This opens the Table Style Editor.



The table diagram in the top panel is an interactive interface that has several selectable regions. See *Table Style Regions* for a list of the different areas that you can style.

5. (Optional) If desired, create a custom region to style. See *Creating a Custom Style Pattern* for more details.

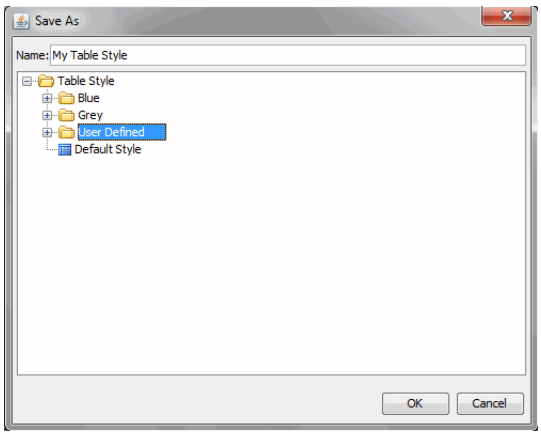
6. Click on the region of the table diagram that you wish to style (or click the region name in the **Regions** panel). The bottom panel displays the visual property settings for the region you selected.



7. Modify the properties to achieve the desired appearance for the selected region of the table.

8. Click the ‘Save’ button in the Style Studio toolbar. This opens the ‘Save’ dialog box.

9. Enter a name for the new table style, and select a category (folder) into which the style should be placed.



10. Click ‘OK’. This saves the table style into the library (*stylereport.srl.d*).

You can use the new style in the same way that you use the predefined styles. See see *Adding a Table Style* or *Table Properties* in *Advanced Dashboard Design* for details.

Creating a Table Style Folder

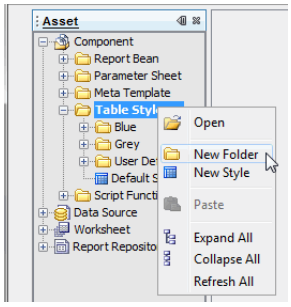
The existing table styles are stored in a predefined set of folders (e.g., ‘Bands’, ‘Columns’, ‘Light’, etc.). You can store your custom styles in any of these predefined folders, or you can define your own folders.

To define a new folder for your custom table styles, follow the steps below:

1. Expand the Style Studio **Asset** panel. (Select 'Asset Pane' from the Style Studio 'Window' menu.)

2. Expand the 'Component' node, and then expand the 'Table Style' node.

3. Right-click the 'Table Style' node or any sub-folder, and select 'New Folder' from the context menu.



4. When prompted, enter a name for the new folder, and click 'OK'.

Deleting a Table Style

To delete a table style, follow the steps below:

1. Expand the Style Studio **Asset** panel. (Select 'Asset Pane' from the Style Studio 'Window' menu.)

2. Expand the 'Component' node, and then expand the 'Table Style' node.

3. Right-click on the table style that you wish to delete, and select 'Delete' from the context menu.

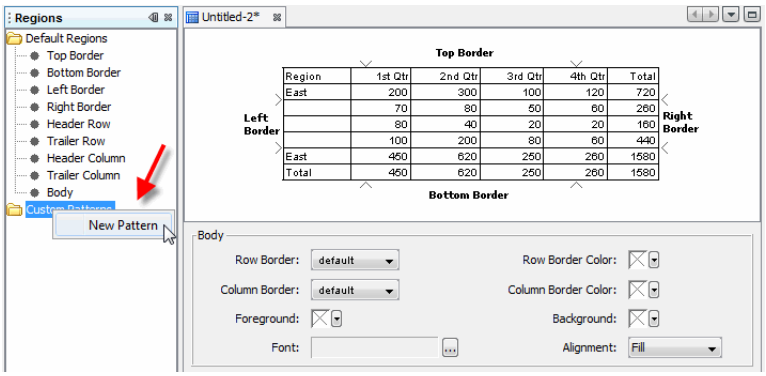
Creating a Custom Style Pattern

In *Creating a Custom Table Style*, you learned how to specify your own table style by applying visual formats to different regions of the table. You can create more complex visual styles by specifying a *row/column pattern* for the table styles. The *pattern* is a mathematical rule that determines the rows or columns to which the formatting is applied. For example, you can add a background color to *every other row*, or you can add a heavy border to *every 3rd column*.

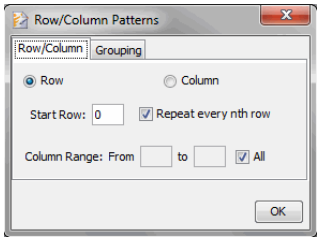
To create a pattern for use in a table style, follow the steps below:

1. Create a new table style and open the Table Style Editor as described in *Creating a Custom Table Style*.

2. In the **Regions** panel, right-click the 'Custom Patterns' node, and select the 'New Pattern' option.



This opens the 'Row/Column Pattern' dialog box.



3. On the **Row/Column** tab, select 'Row' to designate a row-based pattern rule, or select 'Column' to designate a column-based pattern rule.

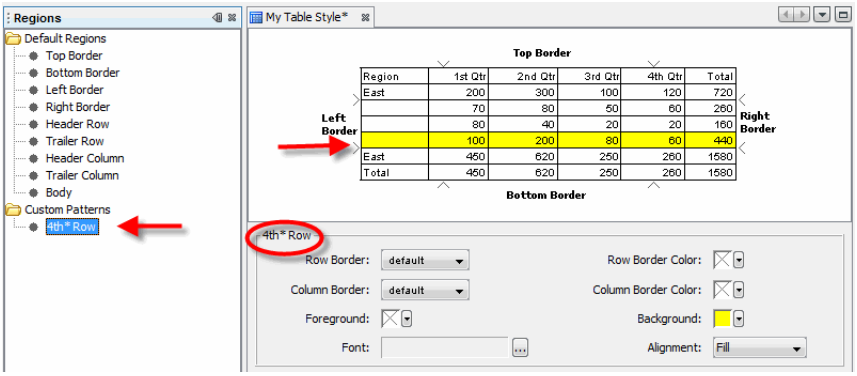
4. In the 'Row' or 'Column' text field, enter an integer value to the indicate the interval at which the style should be applied. For example, a value of "4" indicates that the style will be applied to the 4th row or column.

5. Select the 'Repeat' option to repeat this pattern for subsequent rows/columns. For example, this would repeat the formatting in the previous step to *every* 4th row or column in the specified range.

6. In the 'Cell Range' field, indicate whether the specified pattern should be in effect for 'All' rows or columns, or only for a specified range of cells.

7. (Optional) Click the **Grouping** tab to specify formatting pattern for group headers and footers.

8. Click ‘OK’ to save the style pattern. The new pattern is now listed under the ‘Custom Patterns’ node in the **Regions** panel.

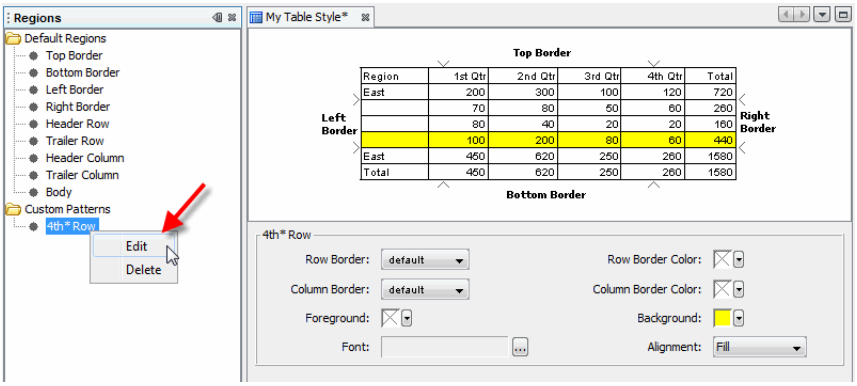


Editing a Custom Style Pattern

To edit a pattern for a custom table style, follow these steps:

1. Right-click the pattern name under the ‘Custom Patterns’ node in the **Regions** panel.

2. Select ‘Edit’ from the context menu.



This opens the ‘Row/Column Pattern’ dialog box. See *Creating a Custom Style Pattern* for further details.

Table Style Regions

The table is divided into nine regions, each of which has its own set of attributes.

Note: When there is a conflict between row and column settings, row settings take precedence over column settings. For instance, if you set the background of the *Header Row* to green and the background of the *Header Column* to red, the top-left cell of the table which is common to both will have a green background (adopting the *Row* setting).

Table 4. Table Regions

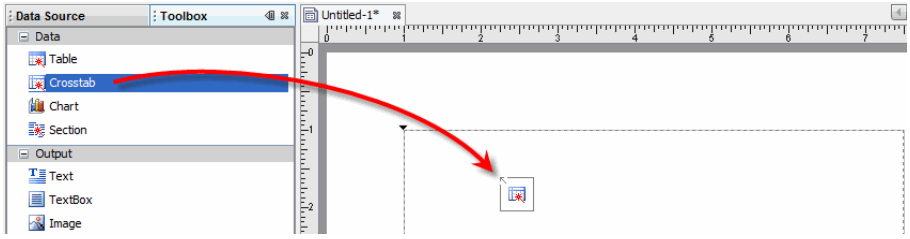
REGION	SELECTION
Header Row	Click on the first row.
Header Column	Click on the first column.
Trailer Row	Click on the last row.
Trailer Column	Click on the last column.
Body	Click in the middle of the table (cells excluding header/ trailer).
Top Border	Click on the Top Border arrow.
Left Border	Click on the Left Border arrow.
Right Border	Click on the Right Border arrow.
Bottom Border	Click on the Bottom Border arrow.

Complex Table Layout

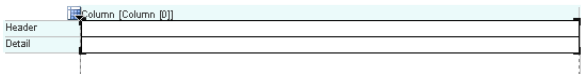
You can construct more complex table layouts, requiring specific formatting at the cell level. Please see *Setting Cell and Row Visual Properties* in the *Advanced Report Scripting* for more information.

5.8 Crosstab Element

A Crosstab presents a compact summary of data in a tabular format. To add a blank Crosstab element to the report, drag the 'Crosstab' icon from the **Toolbox** panel to an empty location in the report.



This adds a blank Crosstab element at the insertion point.



See *Creating a Crosstab Using Freehand Operations* for information on how to add data fields to the Crosstab.

To specify layout and style for a Crosstab, right-click the Crosstab element and select 'Properties' from the context menu. This opens the 'Table Properties' dialog box, which provides largely the same features as those available for a regular Table element. See *Table Properties* for more details.

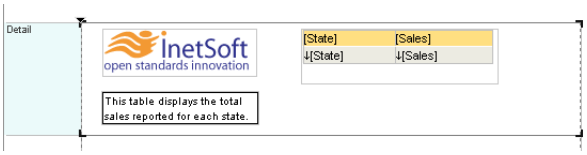
See Also

Binding Data to a Crosstab Element, to add a Crosstab containing data.

5.9 Section Element

There are two elements that display tabular data: Table and Section. This section explains how to use the Section element. See *Tables vs. Sections* for a discussion of the differences between Table and Section elements.

The *Section* element behaves as a container of other elements, and is very useful for application requiring fixed positioning of element, such as forms. The Section provides a canvas in which you can freely place and position any other elements, including other Sections. For example, the Section below contains a TextBox element, an Image element, and a Table.

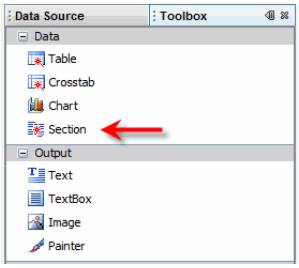


The following sections explain how to create a Section, and how to manually add components to a Section.

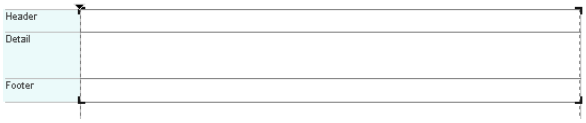
You can also bind a Section to a data source so that the Section displays a *pseudo-table*. When you use a Section as a pseudo-table, Text elements within the Section display data from fields of the data source, which produces a tabular data display. For information on how to bind a Section to a data source to produce a pseudo-table, see *Binding Data to a Section Element*.

5.9.1 Adding a Section Element

To add a Section element to the report, press the 'Section' button in **Toolbox** panel.



This adds a blank Section at the insertion point.



See *Creating a Section Using Freehand Operations* to add data fields to the Section.

The section consists of a set of bands. (The default bands are 'Header', 'Detail', and 'Footer'.) Each band can contain other fixed-position elements as sub-elements. The sub-elements of a band are positioned relative to the top-left corner of the band. Each section band can be decorated with borders and a background color. (See *Setting Global Section Options*.)

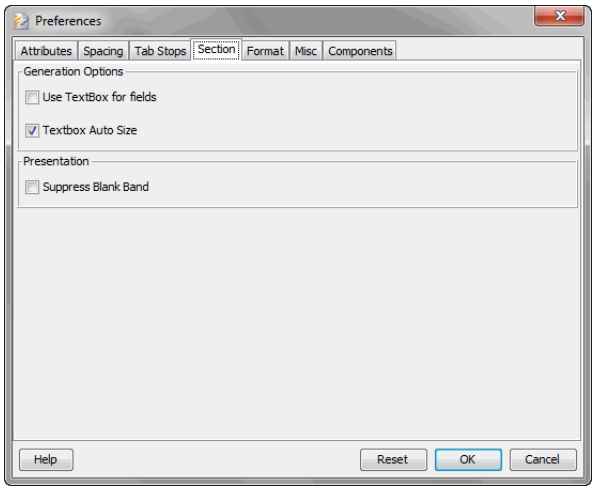
When you use a Section as a pseudo-table to display grouped data, the Section may display multiple levels of headers and footers. These header and footer bands display group labels and group summaries, while the 'Detail' band always show the detail records. See *Grouping and Summarization in Data Binding Wizard* for more details.

See Also

Binding Data to a Section Element, to create a new Section containing data.

5.9.2 Setting Global Section Options

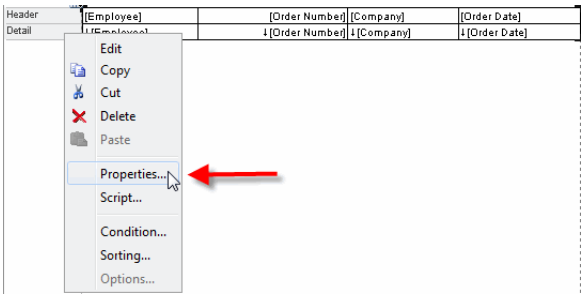
To specify global default settings for all Sections in a report, select 'Preferences' from the 'Report' menu. This opens the 'Preferences' dialog box. Click the **Section** tab.



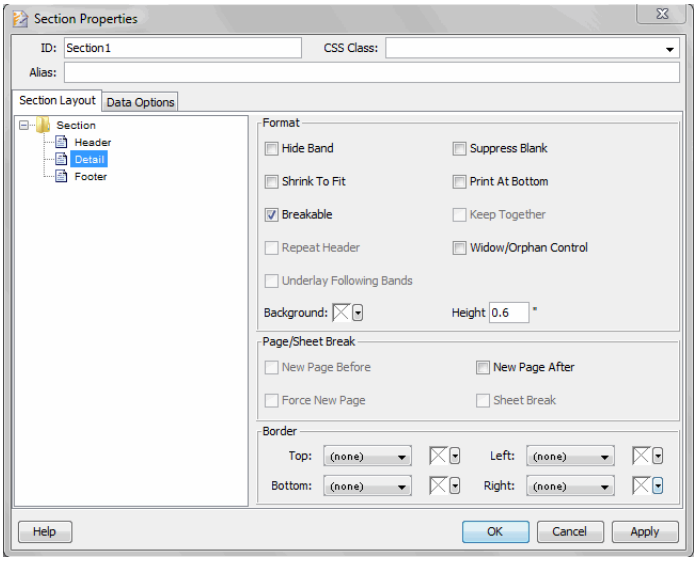
The settings here specify the defaults for the 'Section Options' dialog box. See *Section Options for a Freehand Section* for details about the available properties.

5.9.3 Section Properties

Most Section properties are associated with the individual *bands* in the Section. To set the properties for a Section's bands, right-click in the left margin of the Section, and select 'Properties' from the context menu.



This opens the 'Section Properties' dialog box. In the 'Section Properties' dialog box, the various section bands are listed in a tree structure. For a simple section, there are three bands under the root node: 'Header', 'Detail', and 'Footer'.



Click on a band in the tree to view the corresponding in band properties. The following properties are available:

- **Hide Band** makes the specified band invisible. When a band is hidden, it is not used when rendering a page. However, all elements in the band are still intact, and are restored when the band is made visible again.
- **New Page Before** forces the corresponding band to always be displayed at the top of a new page, thus changing the pagination of the flow. The pagination is enforced so that either the band is the first band of a section, or in the middle of a section.

- **New Page After** forces a new page following the displayed band. These attributes are ignored for the top-level header band because they would cause an infinite loop in the layout.
- **Force New Page** forces a *page break* rather than an *area break* when 'New Page Before' or 'New Page After' is in effect. This operation is only available for *flow layout*. (See *Flow Report Layout* for more information about flow reports.)
- **Sheet Break** causes a new sheet to be created in Excel exports whenever a new page is created by the 'New Page Before' or 'New Page After' properties.
- **Shrink to Fit** collapses any space used by hidden elements in a band so there is no white space in the middle of a band if elements are hidden (through script or other means).
- **Breakable** causes the specified band to break across pages when it cannot be displayed in its entirety on the current page. If this property is disabled, the entirely of the band is pushed to the following page.
- **Print at bottom** causes the section band to be printed at the bottom of the page. If the band repeats, such as for the detail rows of the content band, each repeated band is placed on a separate page.
- **Suppress Blank** hides the section band when it contains no content. If some of the elements in the band are bound to data, the band is considered to have no content when all of the bound elements for a particular row are blank. If binding is not defined for any of the elements in the band, the band is considered to have no content only if all the elements in the band are invisible.
- **Underlay following bands** causes fields in one band to be printed on the same line as fields in the following band. As a result, multiple rows are written over one another. An example application is when the first band contains a picture of an item and the second band contains the item description. These can be displayed on a single line by setting 'Underlay following bands'.
- **Keep Together**, for a group header, causes the report to attempt to keep all of the group's bands on the same page. If this is not possible, the contents are moved to the top of the next page. This setting is not compatible with 'Repeat Header'. (See note below.)
- **Widow/orphan control**, for any header band or for the innermost footer band, prevents a single header band from being printed at the bottom of a page or a single footer band from being printed at the top of a page.
- **Background Color** sets the fill for a section band.
- **Borders** adds borders to any of the four sides of the specified band.

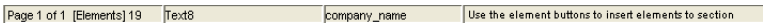
- **Height** sets the height of the specified section band. This can also be manipulated by manually dragging the section band borders.

```
Note: When 'Repeat Header' is set for a section or group header,
this disables the ‘Keep Together’ option for all subordinate
groups.
```

- **Repeat Header** displays the section or group header on each page. This setting is not compatible with ‘Keep Together’.

5.9.4 Viewing Information for Elements Inside a Section

To view information about an element contained within a Section, click on the desired element. The Style Studio status bar (at the bottom of the screen) displays the relevant information, including any data binding infomrtaion.



5.9.5 Manually Adding Elements to a Section

You can manually add any element from the **Toolbox** panel into a Section band. To add an element to a Section, use one of the following methods:

- Drag an element from the **Toolbox** panel into the Section.
- Select the section, right-click, and choose ‘Insert’ from the context menu. Then select the desired element.

Adding a Table or Chart to a Section

A table contained within a section can dynamically grow within the bounds of that section. By setting the position and layout properties for the table, you can control how it spans across multiple pages, wraps, etc.

For charts, the data binding must be associated with the chart *after* the chart is placed in the section. If you place a chart into a Header band in a section, the chart will only reflect data in the current group for the corresponding data binding.

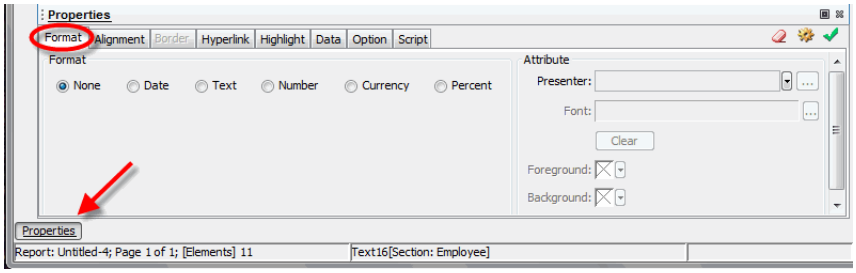
Adding Text Elements to a Section

If you add a Text or Textbox element to a section, the element immediately enters editing mode, allowing you to enter the desired text. The height of the Text element is a multiple of the height of a single text row, and therefore depends on the current font settings.

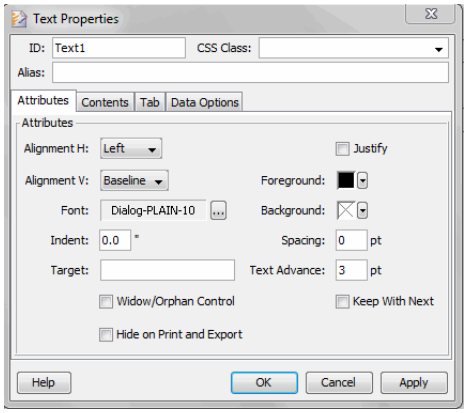
To change the format of a Text element, follow these steps:

1. Click the Text element in the Section to select it.
2. Expand the **Properties** panel (at bottom of Style Studio window).

3. Select the **Format** tab, and enter the desired format. See *Formatting* for more details on the format settings.



To change the location and size of a Text element, simply click and drag on the element handles. Alternatively, you can right-click the element, and select 'Properties' from the context menu. This opens the 'Text Properties' dialog box. Click the **Position** tab to access position settings.



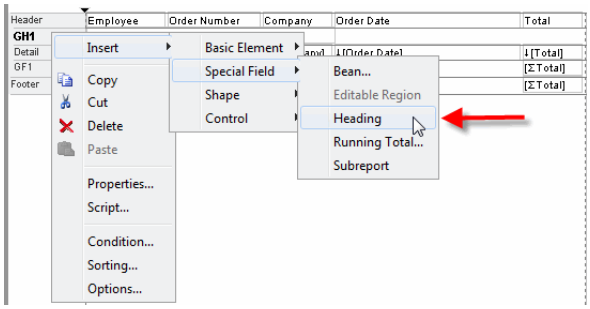
Enable 'Auto size' to allow the element to grow or shrink based on its content.

Adding Heading Elements to a Section

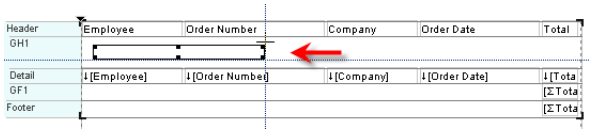
In a Section that displays grouped data, you can use the heading element to generate a table of contents (TOC) listing the different groups. To do this, add a heading element into the header band of the Section as follows:

1. Right-click in the header band of the Section that represents the group you wish to display in the TOC.

2. Select 'Insert' > 'Special Field' > 'Heading' from the context menu. This changes the pointer to a crosshair.



3. Drag the crosshair in the Section to draw the heading element.



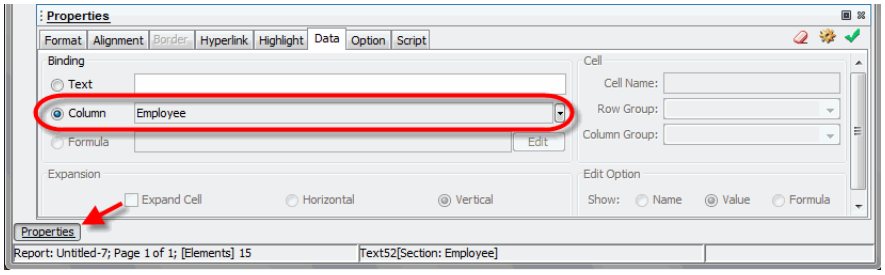
4. Bind the new heading element to the desired grouping field. Follow the steps below:

   a. Click the header element to select it.

   b. Click the 'Properties' button at the bottom of the Style Studio window to open the **Properties** panel.

   c. Select the **Data** tab on the **Properties** panel.

   d. In the 'Binding' area of the **Properties** panel, select the desired field from the 'Column' menu. This field will supply the values that the heading element will display.



   e. Click the 'Apply' button at the top right of the **Properties** panel.

5. Preview the report. The heading element now displays the group name for each group in the Section header. (Each repetition of the Section header band – for each group – displays the corresponding group name.)

You can add a TOC element to the report to list these groups within a table of contents. See *Table of Contents* for more information.

See Also
Heading, for information on how to use a heading.
Table of Contents, for information on how to use a TOC.

5.9.6 Moving and Resizing Elements in a Section

To move, resize, or copy an element in a Section, first click the element to select it. (Ctrl-click to select multiple elements.) Then move, resize, or copy the selected element(s) as described below:

- To move the selected element(s) with the mouse, drag on an element body. To constrain the movement to the horizontal or vertical direction, hold down the Shift key while dragging.
- To move the selected element(s) with the keyboard, press the arrow keys on the keyboard to move the element(s) in 5-pixel increments. Hold down the Ctrl key or Shift key while pressing the arrow keys to move in 1-pixel or 10-pixel increments, respectively.
- To resize the selected element(s), drag an element border in the desired direction.
- To copy the selected element(s) using the mouse, Ctrl-click and drag on an element. Alternatively, right-click on an element and select 'Copy' from the context menu. Then right-click in an empty region of the Section and select 'Paste' from the context menu.
- To copy the selected element(s) using the keyboard, press Ctrl-C on the keyboard to copy. Then click at the location where you want to paste the copied element(s), and press Ctrl-V on the keyboard.

To help control the position of element that you move or resize, you can turn on the grid-snap feature. This snaps the elements to the grid (whether the grid is visible or not) as you move them.

To turn on grid-snap, select 'Snap to Grid' from the Style Studio 'View' menu. See *Using Grids and Rulers* for information about configuring the report grid.

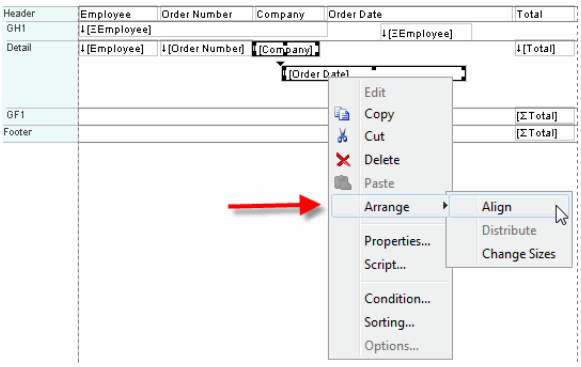
See Also
Changing Sizes of Elements in a Section, to automatically change element sizes.
Aligning Elements in a Section, for information on automatic alignment features.

5.9.7 Aligning Elements in a Section

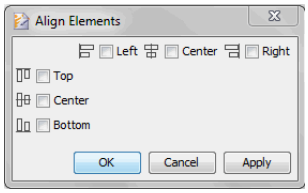
To align two or more elements in a Section, follow these steps:

1. Ctrl-click the elements desired elements to select them. (You can also drag across the elements to select them.)

2. Right-click on any one of the selected elements. From the context menu, select ‘Arrange’ > ‘Align’.



This opens the ‘Align Elements’ dialog box.



3. Select a vertical and/or horizontal alignment option, and click ‘OK’.

The selected elements are aligned according to the specified alignment options.

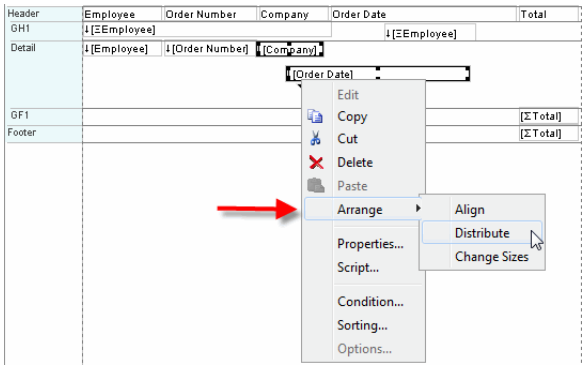
5.9.8 Distributing Elements in a Section

You can automatically space elements so that they are distributed at equal intervals (horizontally or vertically) based on their edges or centers.

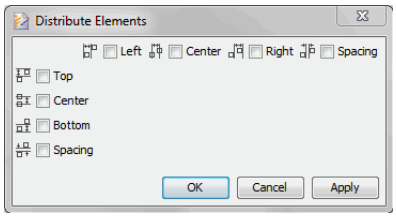
To distribute three or more elements within a section, follow the steps below:

1. Ctrl-click the desired elements to select them. (You can also drag across the elements to select them.)

2. Right-click on any one of the selected elements. From the context menu, select ‘Arrange’ > ‘Distribute’.



This opens the 'Distribute Elements' dialog box.



3. Select the appropriate spacing option. The spacing calculation can be based on the left or right edges of elements, center of elements, or the distances between elements.

4. Click 'OK'.

The elements are now spaced according to the distribution strategy that you specified.

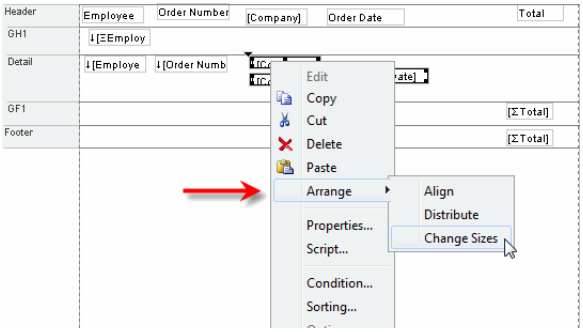
5.9.9 Changing Sizes of Elements in a Section

You can automatically change the sizes of multiple elements so that all sizes are equal. This is useful when you need to lay out data or form fields with similar-sized elements.

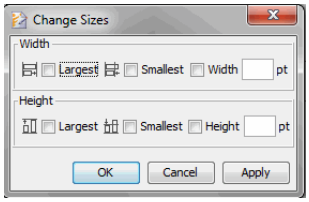
To force a set of elements to adopt the same size, follow the steps below:

1. Ctrl-click the desired elements to select them. (You can also drag across the elements to select them.)

2. Right-click on any one of the selected elements. From the context menu, select 'Arrange' > 'Change Sizes'.



This opens the 'Change Sizes' dialog box.



3. Check the desiired option to specify how the sizes should be adjusted. Select the 'Width' or 'Height' option to manually enter the exact size for the elements in points (1/72'').

5.10 Chart Element

Charts display summarized data in an easy-to-understand visual form. Style Intelligence provides over 30 built-in chart types to designed to satisfy the needs of all business users.

5.10.1 Adding a Chart Element

To add a chart element, follow the steps below:



1. Click the 'Chart' button in the **Toolbox** panel. This adds a new chart to the report, and opens the **Data Source** tab and **Binding** panel.

2. Resize the chart by dragging the resize handles. (The default size for the chart is the entire width of the page.)

3. Drag fields from the **Data Source** panel to the appropriate regions ('X', 'Y', 'Color' etc.) of the **Binding** panel to display the data on the chart. See *Binding Data to a Chart Element* for more information about displaying data.

4. To customize the appearance of the chart, right-click the chart, and select 'Properties' from the context menu. This opens the 'Chart Properties' dialog box. See *Chart Properties* for more information.

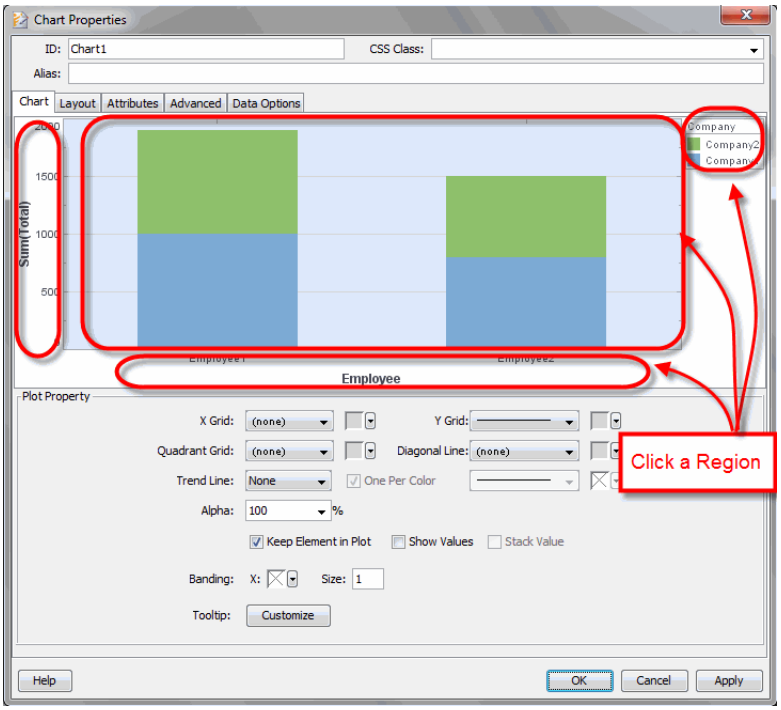
5.10.2 Chart Properties

The following sections discuss configuration settings available for charts in the ‘Chart Properties’ dialog box. To open the ‘Chart Properties’ dialog box, do one of the following:

- Right-click the chart, and select ‘Properties’ from the context menu.
- Double-click the chart.

Not all chart attributes are mutually compatible.

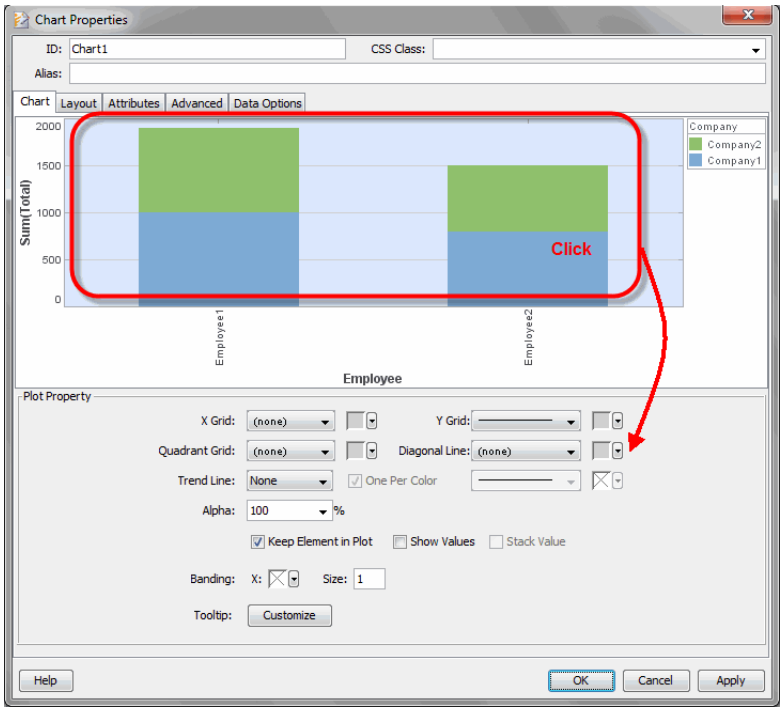
To set the properties for a particular aspect of the chart (e.g., X-axis, legend, etc.), click the desired region in the chart diagram. This displays the corresponding chart properties in the panel at the bottom.



Plot Property

In the ‘Chart Properties’ dialog box, click on the plot region on the chart graphic to open the ‘Plot Property’ panel.

Figure 8. Plot Property



Plot properties include a number of common properties for all the chart styles and also some style-specific properties. The common properties are listed in the following table.

Table 5. Plot Properties

PROPERTY	DESCRIPTION
X Grid	Chart x-axis grid line style and color.
Y Grid	Chart y-axis grid line style and color.
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.
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.)

Project Forward	Specifies the number of axis units that the trend line should extend beyond the final data point. Not available for the following chart types: Waterfall, Pie, 3D Pie, Radar, Filled Radar, Map. Not available for the following data types: String, Boolean. Projection requires that axis values be a linear and complete series of either numbers or dates.
Show Values	Label the chart elements with corresponding values. See *Showing Values on a Chart*.
Stack Values	Displays a single aggregate value for each primary (X-axis) grouping, rather than for each individual subseries grouping.
Alpha	Sets transparency for the chart elements. If disabled, the default transparency for the particular chart type is used.
Keep Element in Plot	Automatically extend the chart boundaries to display chart elements that exceed the specified axis maximum.
Tooltip	Value is displayed as tooltip when mouse hovers over corresponding chart area. To display custom HTML in the tooltip, see *Custom Tooltip*.
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 the specified axis. Select the desired band color from the color picker.
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.
Explode Pie	For the pie chart style, create a separation between the slices.

Custom Tooltip

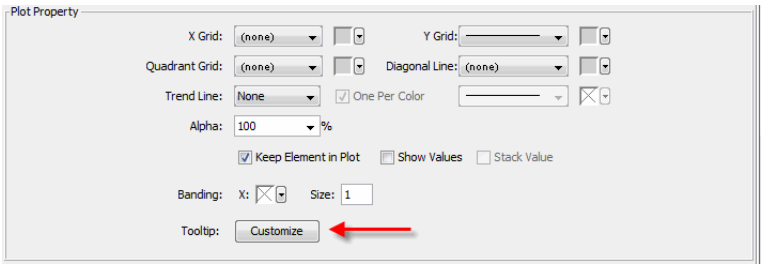
You can create a custom tooltip that incorporates data values in *java.text.MessageFormat* format.

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. Select the **Chart** tab, and click on the body of the chart.

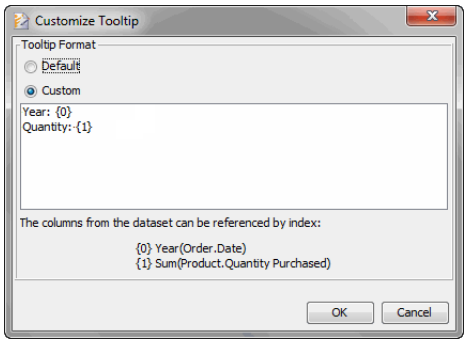
3. In the 'Plot Properties' panel, press the 'Customize' button next to 'Tooltip'.



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.



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 *Formatting* for information about format masks.

```
Date: {0, date, MMM-yy}
Sales: {1, number, $#,###.00}
```

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:

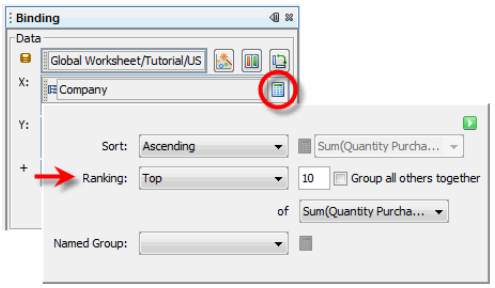
1. Create a new report, and drag a Chart element into the report. Click on the Chart to select it.

2. In the **Data Source** panel, expand the 'Global Worksheet' node and the 'Tutorial' node. Expand the 'US Sales' Worksheet.

3. Drag the 'Company' dimension to the 'X' field in the **Binding** panel.

4. Drag the 'Quantity Purchased' measure to the 'Y' field in the **Binding** panel. This creates a chart that displays quantity purchased for each company.

5. Click the 'Edit Dimension' button next to the 'Company' field, and select 'Ranking' to display only the top 10 companies. Click the 'Apply' button.



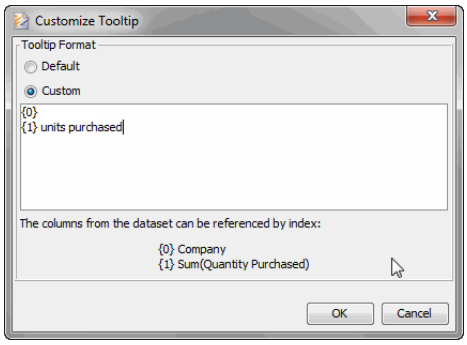
6. Right-click the Chart, and select 'Properties' from the context menu. This opens the 'Chart Properties' dialog box.

7. Under the **Chart** tab, click on the Chart body to display the 'PLot Property' panel.

8. Next to the 'Tooltip' option, press 'Customize' button. This opens the 'Customize Tooltip' dialog box.

9. Select the 'Custom' option and enter the following in the text field:

```
{0}
{1} units purchased
```



10. Press ‘OK’ to close the ‘Customize Tooltip’ dialog box, and press ‘OK’ again to close the ‘Chart Properties’ dialog box.

11. Save the report, and then view the report

When you hover the mouse over the bars on the chart, you will now see the custom tooltip you specified.



¢

See Also

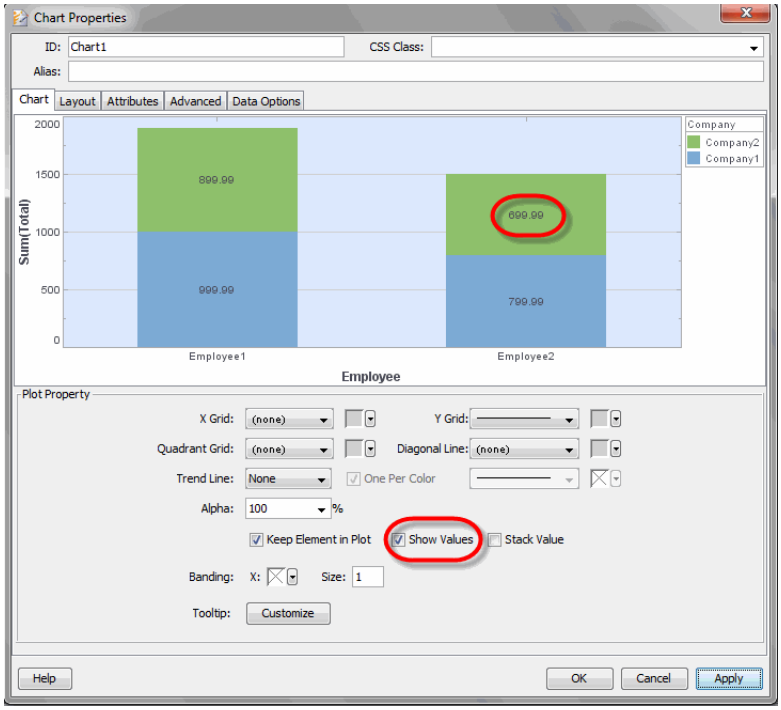
Adding a Non-Displaying Measure to a Chart, to utilize a measure in the tooltip which is not displayed on the chart.

Showing Values on a Chart

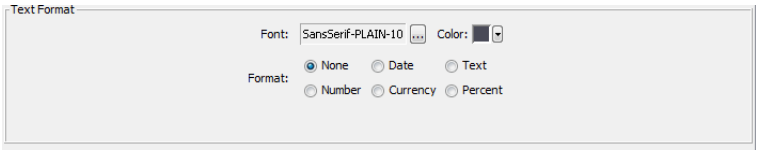
To display values in the plot area of a chart, follow the steps below.

1. In the ‘Chart Properties’ dialog box, click the plot area in the chart graphic to open the ‘Plot Property’ panel.

2. Select the ‘Show Values’ option, and click the ‘Apply’ button. This displays the values on the chart image.



3. Click a value on the chart image to display the ‘Format’ options.



4. Specify the desired display format for the values, and click ‘OK’.

See Also

Format Types, for more details on the available formats.

Legend Property

In the ‘Chart Properties’ dialog box, click on the legend region of the chart graphic to open the ‘Legend Property’ panel.

Figure 9. Legend Region

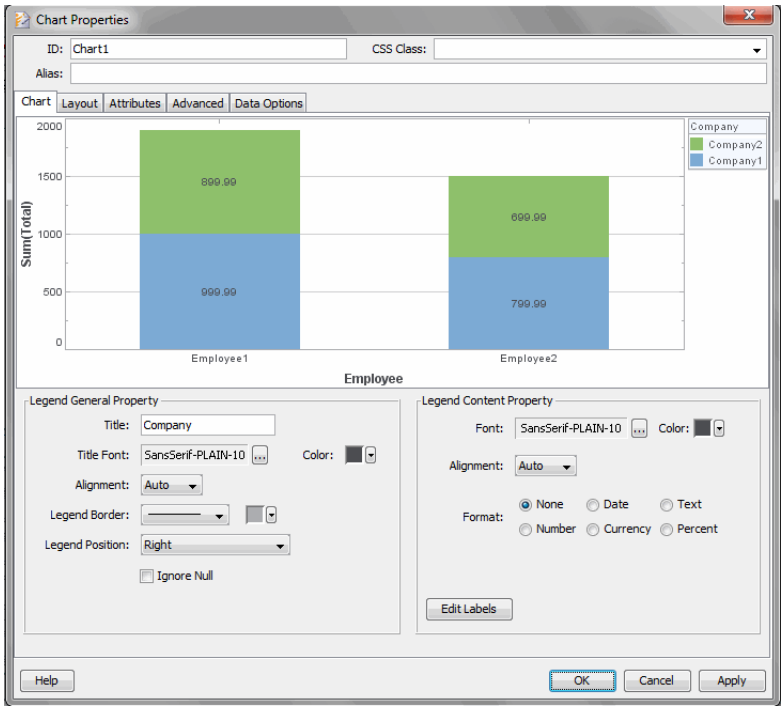


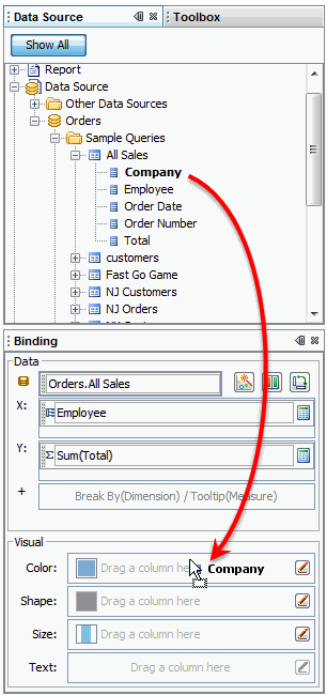
Table 6. Legend Properties

PROPERTY	DESCRIPTION
Title	The text that appears in the legend title bar.
Visible	Sets the visibility of the legend title bar.
Title Font and Color	The appearance of the legend title text.
Alignment	The alignment (left, right, or center) for legend title and content.
Dataset Label Font and Color	The appearance of the legend dataset labels
Legend Border	The line style to be used for the border around the legend. Setting it to none removes any existing border around the legend.
Dataset Label Format	Format for converting the values to strings for legend labels.
Legend Position	Sets the position of the legend on the plot. Legend position can also be changed by dragging with the mouse. See *Moving the Legend*.
Ignore Null	Ignore null values when generating the legend.

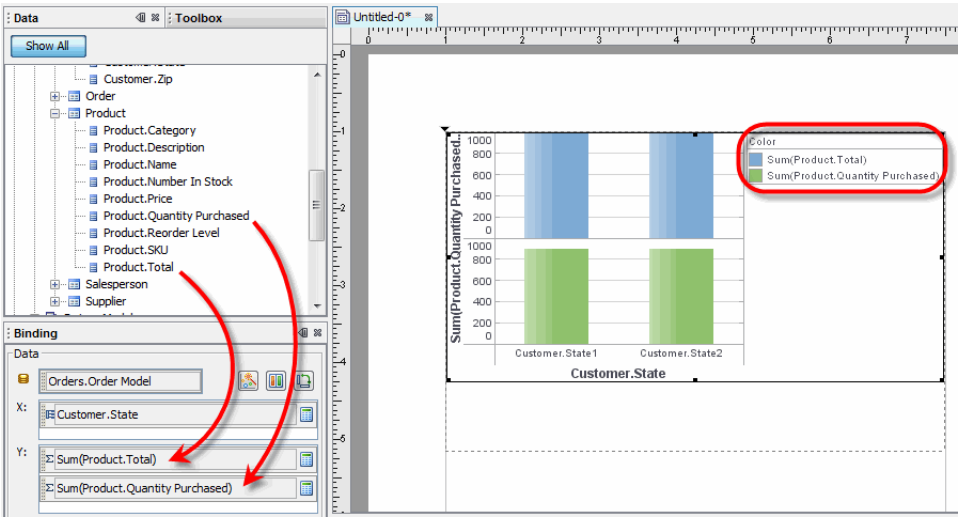
Creating a Chart Legend

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 **Binding** panel of the Chart Editor.



For information about modifying the legend, see *Legend Property*.

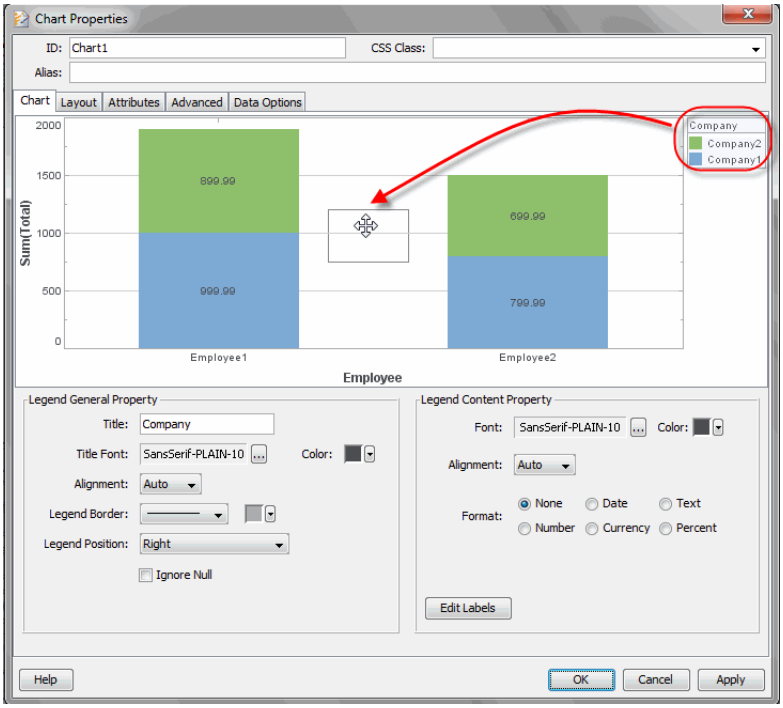
See Also

Legend Property, for information about modifying a chart legend.
Changing Legend Labels, for information on modifying the legend text.

Moving the Legend, for information about moving a chart legend.

Moving the Legend

To change the position of the legend, click the legend in the 'Properties' dialog box graphic, and drag to the desired location.



By default, the legend snaps to one of four predefined positions (Top, Bottom, Left, Right) outside the plot area. To manually position the legend within the plot region, follow the steps below:

1. From the 'Legend Position' menu of the 'Legend General Property' panel, select the 'In Place' option.

2. In the chart diagram, click-and-drag on the legend to position as desired.

To hide a legend, right-click the legend you want to hide, and select 'Hide Legend' from the context menu. To show all legends, right-click in the plot area, and select 'Show Legend'.

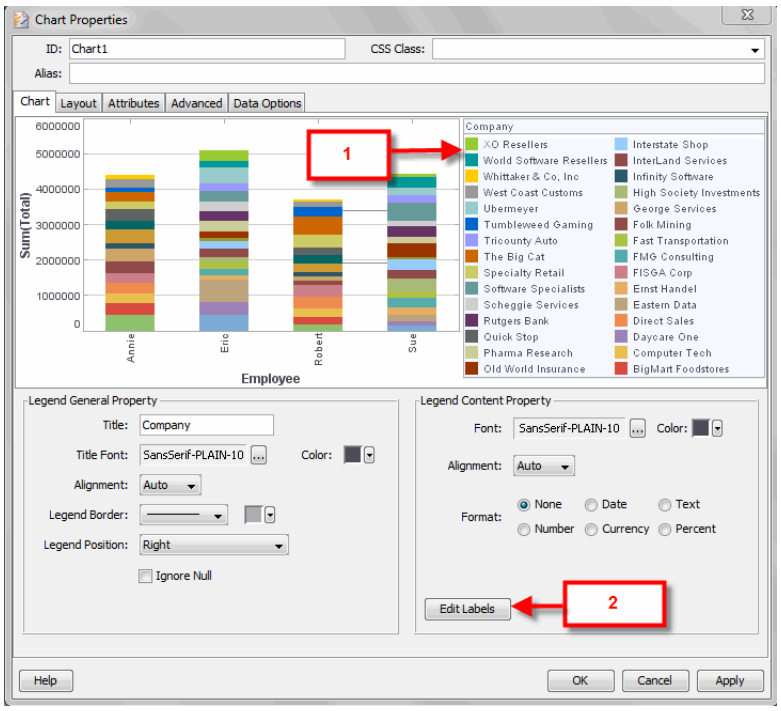
Changing Legend Labels

You can change the text of the labels displayed in the legend. To modify the labels of a legend, follow the steps below.

1. In the Style Studio toolbar, click the 'Live Edit' button to switch the report to live edit mode.

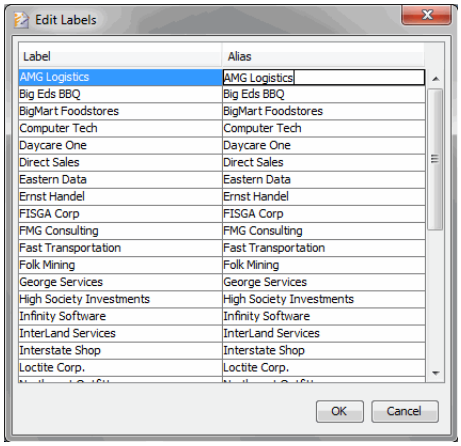
2. Right click the chart, and select 'Properties' from the context menu to open the 'Properties' dialog box.

3. Click on the legend to activate the 'Legend Content Property' panel, and click the 'Edit Legend' button.



This displays a table containing the mapping between original legend labels and new legend labels.

4. 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.



5. Click ‘OK’ to close the dialog box. This updates the legend labels with the specified replacement text.

See Also

Displaying a Report in Live Edit View, for information on live edit mode.
Adding a Sub-Series, for information about adding a sub-series and legend.

X-Axis Label Property

In the ‘Chart Properties’ dialog box, click on the X-axis label region of the chart image to open the ‘X Axis Label Property’ panel.

Figure 10. X-Axis Label Region

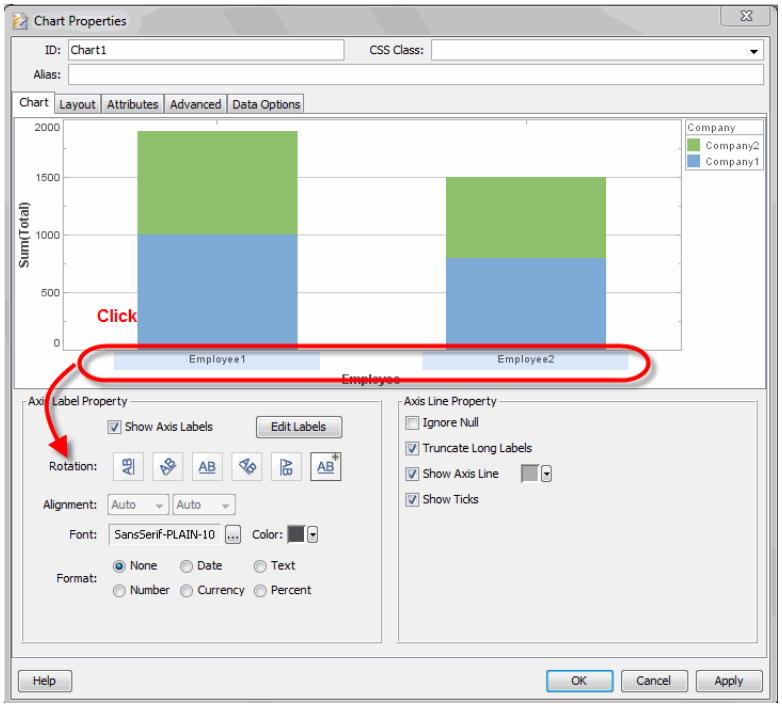


Table 7. X-Axis Label Properties

PROPERTY	DESCRIPTION
Show Axis Labels	Enables display of X-axis labels.
Rotation	Rotation angle for X-axis labels.
Font/Color	The font and text color of the X-axis labels.
Format	Format for converting the values to strings for X-axis labels.
Show Axis Line	Enables display of the horizontal X-axis line (above the labels).
Show Ticks	Enables display of X-axis tick marks.

X-Axis Title Property

In the 'Chart Properties' dialog box, click on the X-axis title region of the chart image to open the 'X Axis Title Property' panel.

Figure 11. X Axis Title Region

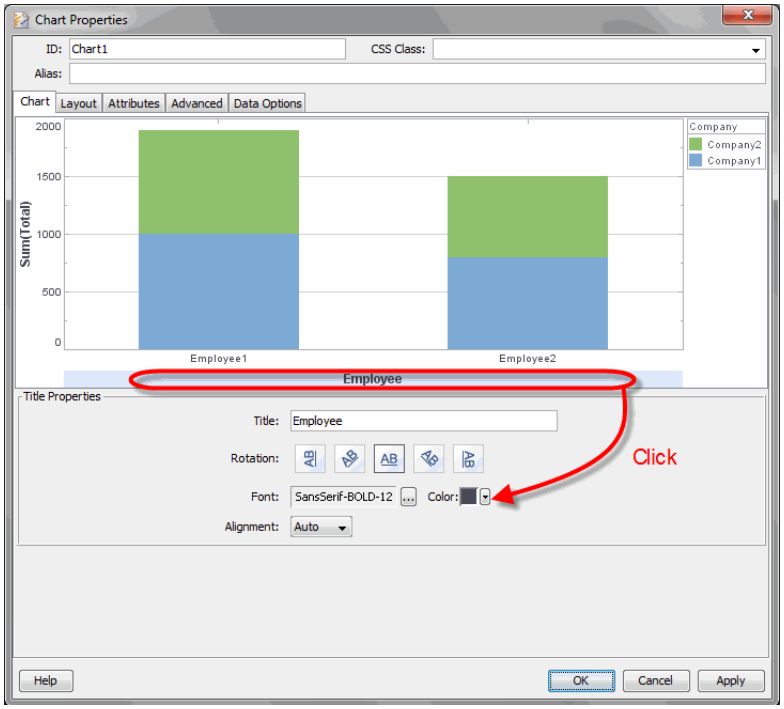


Table 8. X-Axis Title Properties

PROPERTY	DESCRIPTION
Title	Text of the X-axis title. (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 'Date' on the X-axis, you can enter a custom title such as "Breakdown by {0}" to display the following X-axis title: "Breakdown by Date".
Rotation	Rotation angle for X-axis title.
Font/Color	The font and text color of the X-axis title.
Alignment	The alignment (left, right, or center) for the X-axis title.

Y-Axis Label Property

In the 'Chart Properties' dialog box, click on the Y-axis region of the chart graphic to open the 'Y Axis Label Property' panel.

Figure 12. Y Axis Label Region

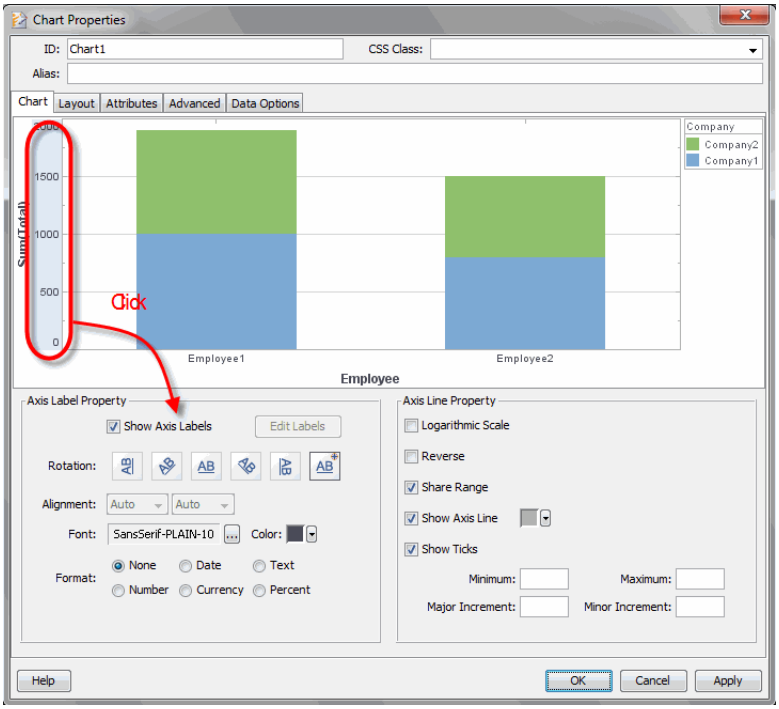


Table 9. Y-Axis Label Properties

PROPERTY	DESCRIPTION
Show Axis Labels	Enable the display of Y-axis labels.
Rotation	Rotation angle for Y-axis labels.
Alignment	The alignment (left, right, or center) for the Y-axis labels.

Font	Font and color for the Y-axis labels.
Format	Format for converting the values to strings for Y-axis labels.
Logarithmic Scale	Enables logarithmic scaling on the Y-axis.
Show Axis Line	Enables display of the vertical Y-axis line (next to labels).
Show Ticks	Enables the display of Y-axis ticks marks.
Minimum	The value at the bottom of the Y axis.
Maximum	The value at the top of the Y axis.
Major Increment	Increment value between ticks on Y axis.
Minor Increment	Minor increment on Y axis. (For log scales, a minor increment *n* generates 10/*n*–1 minor tick marks between labels. For example, *n*=5 generates one tick mark, n=2 generates four tick marks, etc. Use n=0 to suppress tick marks completely.)

Y Axis Title Property

In the 'Chart Properties' dialog box, click on the Y-axis title region of the chart image to open the 'Y Axis Title Property' panel.

Figure 13. Y Axis Title Region

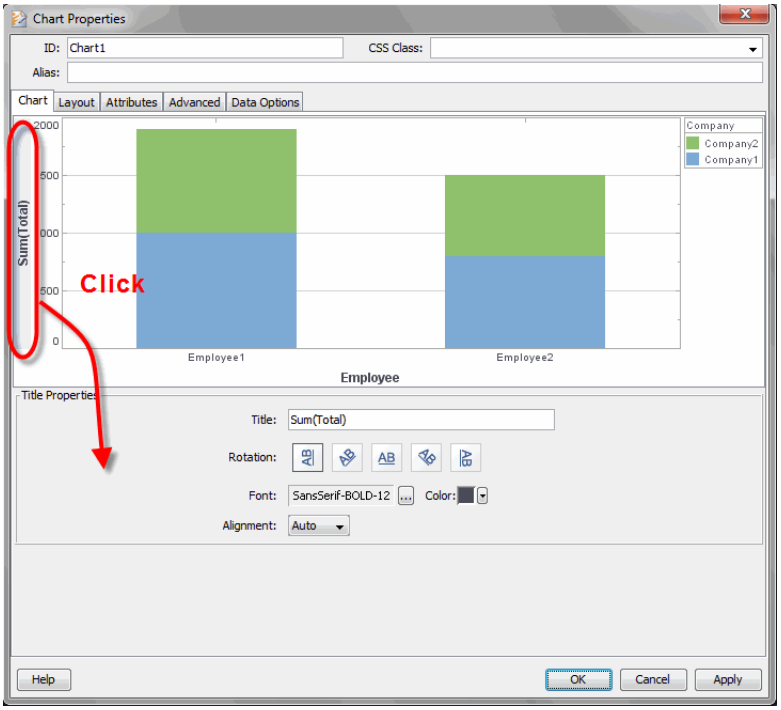


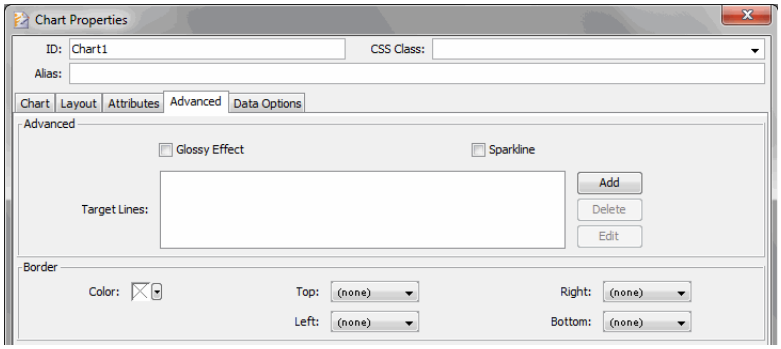
Table 10. Y Axis Title Properties

PROPERTY	DESCRIPTION
Title	Text of the Y-axis title. (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".
Rotation	Rotation angle for Y-axis title.
Font/Color	The font and text color of the Y-axis title.
Alignment	The alignment (left, right, or center) for the Y-axis title.

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.
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.
Target Lines	Specify fixed horizontal marker lines to be placed in the plot area. See *Adding a Target Line* for more information.

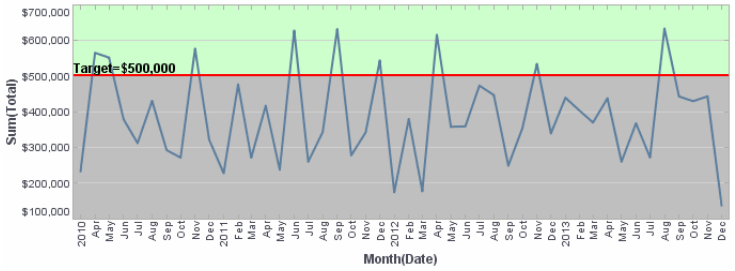


See Also

X-Axis Title Property, *Y Axis Title Property*, to change title text or hide 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:

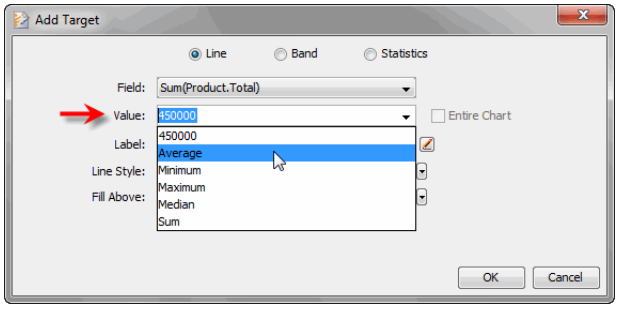
1. Right-click the Chart, and select ‘Properties’ from the context menu. This opens the ‘Chart Properties’ dialog box.

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** option.

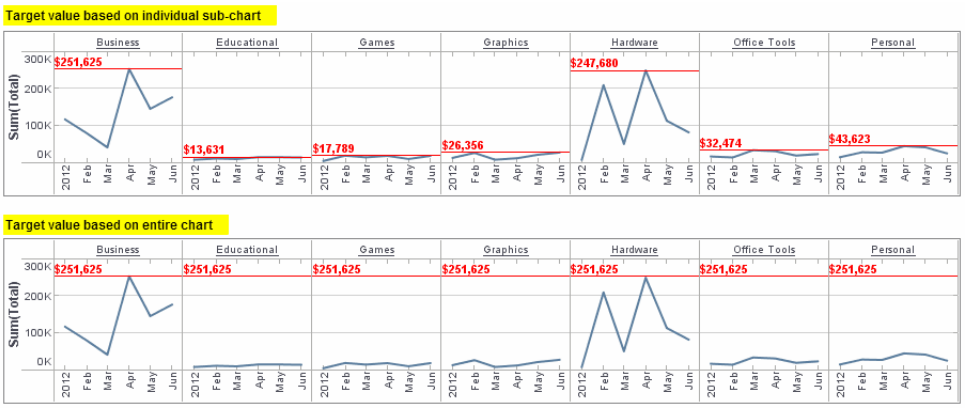
4. In the ‘Field’ menu, select the chart measure to which you want to add the target line.

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.)

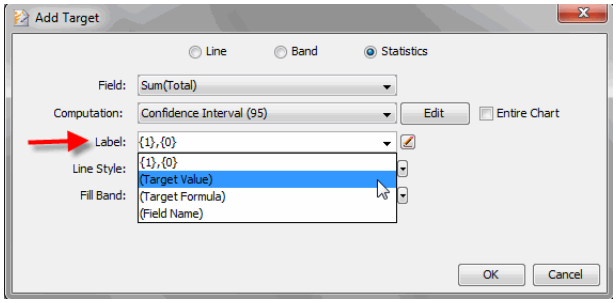


6. (Optional) If you select one of the available target line computations (‘Average’, ‘Minimum’, etc.), enable the ‘Entire Chart’ option to compute the target value based on measure data from the entire chart. Disable the ‘Entire Chart’ option to compute the target value for each sub-chart based only 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.

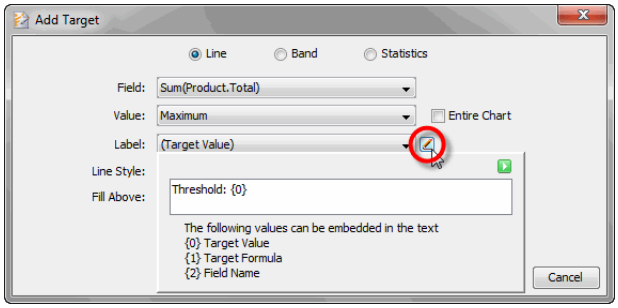
   b. Select '(Target Value)' to insert the numerical value of the target line as the label.

   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.

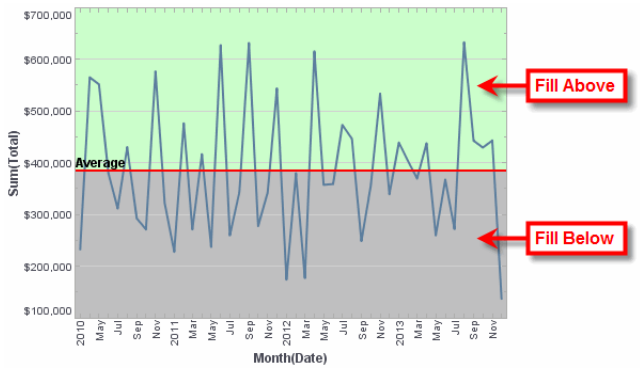


If desired, you can add the target value, target formula, and field name into the label by inserting the corresponding codes (`{0},{1},{2}`) shown at the bottom of the panel. You can apply formats to the inserted values by using the same syntax for *Custom Tooltip*. Some examples are shown below:

```
{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 ‘Add Target’ dialog box, and press ‘OK’ to close the ‘Chart Properties’ dialog box.

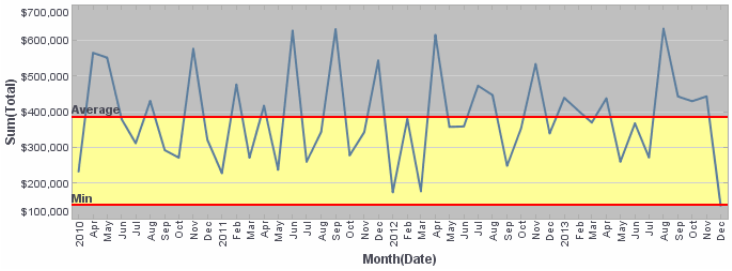
By default, the target line appears on the chart even if its value is greater than the largest data point. This may sometimes cause the data points on the chart to be compressed into a small region of the plot area, which makes the chart difficult to read. To correct this, turn off the ‘Keep Element in Plot’ option in the ‘Plot Properties’ dialog box. See *Plot Property* 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 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:

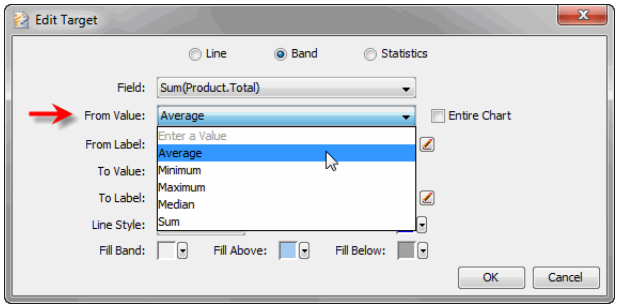
1. Right-click the Chart, and select ‘Properties’ from the context menu. This opens the ‘Chart Properties’ dialog box.

2. Select the **Advanced** tab of the ‘Chart Properties’ dialog box. In the ‘Target Lines’ panel, press the ‘Add’ button. This opens the ‘Add Target’ dialog box.

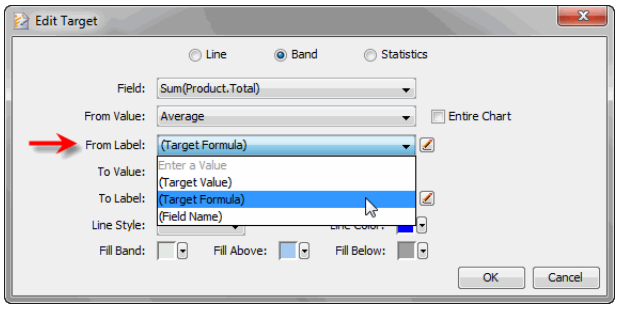
3. Select the **Band** option.

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.

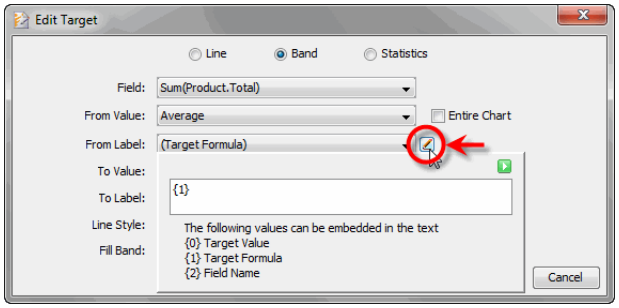
   b. Select ‘(Target Value)’ to insert the numerical value of the lower band boundary as the label.

   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.

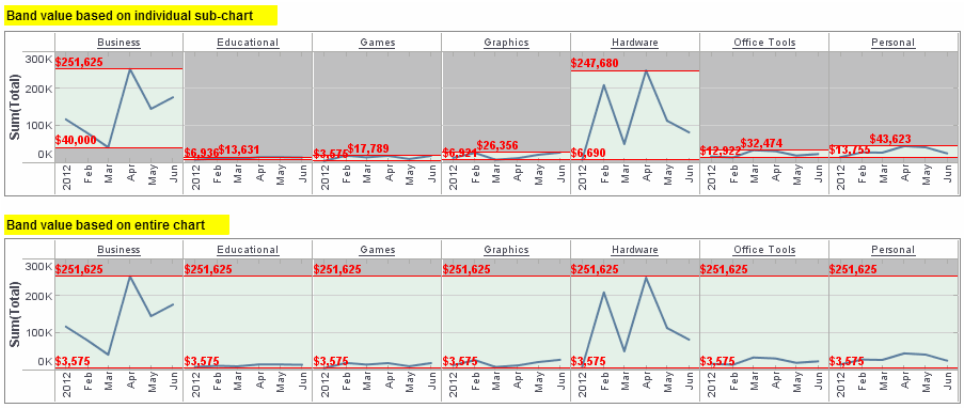


If desired, you can add the target band value, target band formula, and field name into the label by inserting the corresponding codes (`{0}`,`{1}`,`{2}`) shown at the bottom of the panel. You can format the inserted values using the same syntax as *Custom Tooltip*. Some examples are shown below:

```
{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.

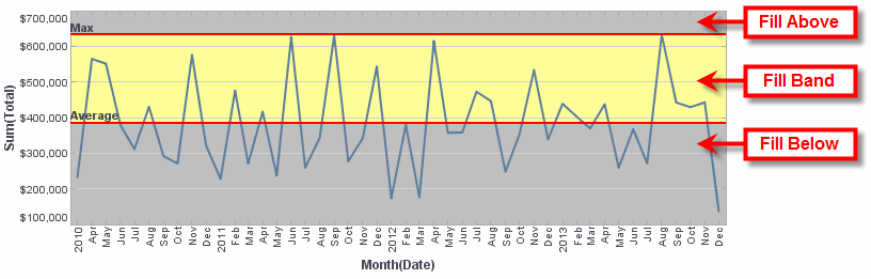
   The following illustration demonstrates the effect of the ‘Entire Chart’ setting. (In both cases, ‘From Value’ is set to ‘Minimum’ and ‘To Value’ is set to ‘Maximum’).



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 ‘Add Target’ dialog box, and press ‘OK’ to close the ‘Chart Properties’ 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 *Plot Property* for more information.

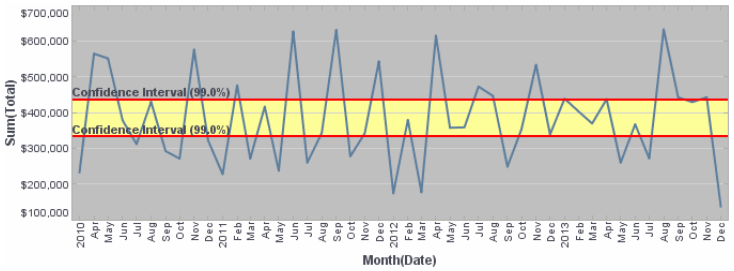
See Also

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 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.

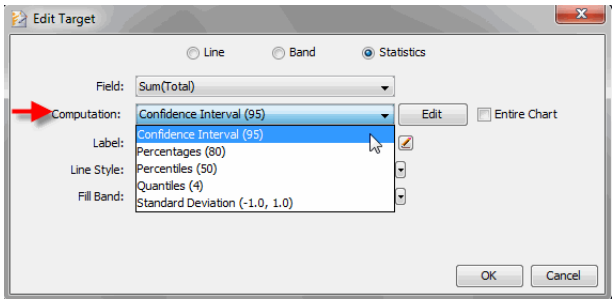
2. Select the **Advanced** tab of the ‘Properties’ dialog box.

3. In the ‘Target Lines’ panel, press the ‘Add’ button. This opens the ‘Add Target’ dialog box.

4. Select the **Statistics** option.

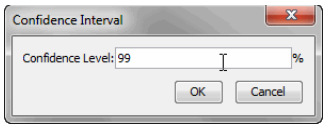
5. In the ‘Field’ menu, select the chart measure to which you want to add the statistical measure.

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.)

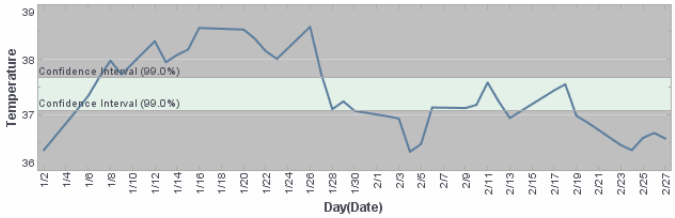


7. To modify the statistical measure, press the ‘Edit’ button. The following settings are available:

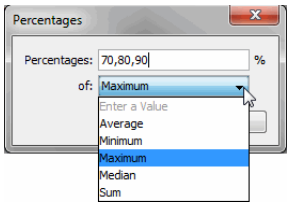
   a. **Confidence Interval:** For the ‘Confidence Interval’ option, enter a value as a percentage.



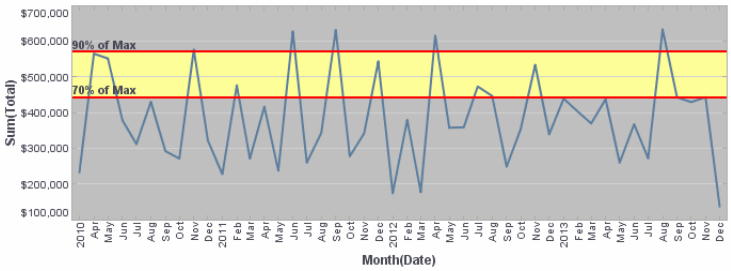
   The resulting top and bottom confidence bounds indicate the interval of values in which the “true” value is expected to be found. For example, the “true” temperature in the chart below would be expected to fall within the displayed confidence interval in 99 out of 100 such samples. (In other words, the true temperature is expected to be *outside* the confidence bounds purely by chance in 1 out of 100 samples.)



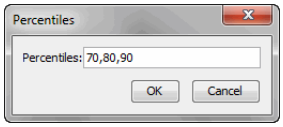
b. **Percentages:** For the ‘Percentages’ option, enter a value or comma-separated list of values as percentages. In the ‘Of’ field, specify the basis on which the percentage should be computed. You can type a fixed value or select from the following presets: ‘Average’, ‘Minimum’, ‘Maximum’, ‘Median’, ‘Sum’.



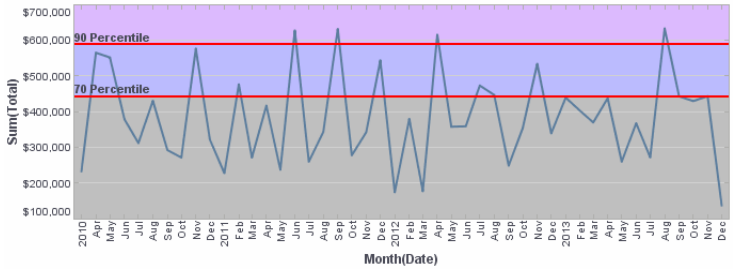
For example, to display percentage lines at 70% and 90% of the Maximum, enter “`70,90`” in the ‘Percentages’ field and select the ‘Maximum’ option from the ‘Of’ field.



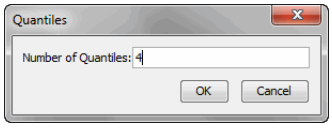
c. **Percentiles:** For the ‘Percentiles’ option, enter a value or comma-separated list of values as percentages.



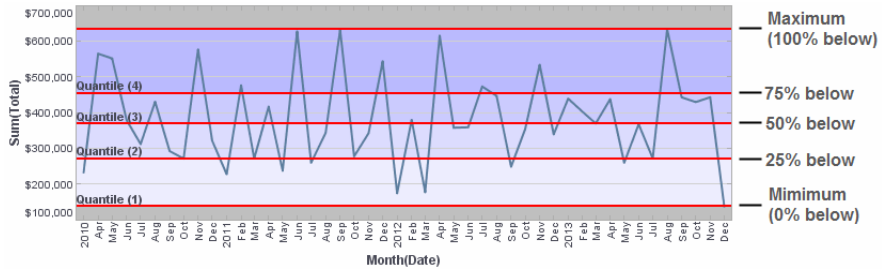
The resulting percentile lines indicate the levels below which the specified percentages of values are found. For example, percentile lines at 70% and 90% (“`70,90`” in the ‘Percentages’ field) designate the levels, respectively, below which 70% and 90% of the data are found.



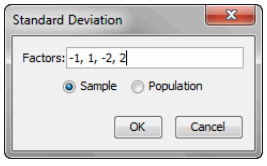
d. **Quantiles:** For the 'Quantiles' option, enter the number of quantiles to display.



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%.

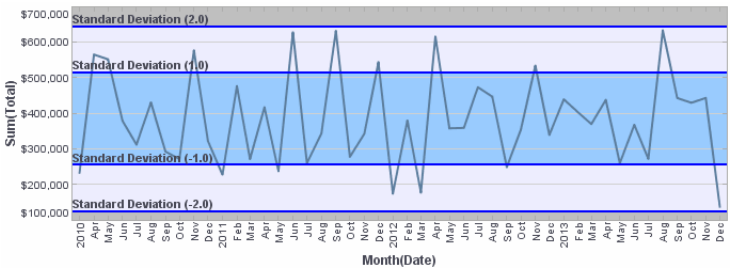


e. **Standard Deviation:** For the 'Standard Deviation' option, enter a comma-separated list of factors. Each successive pair of factors represents, respectively, the lower and upper multipliers for the standard deviation.



For example, enter "`-1,1,-2,2`" in the 'Factors' field to draw lines, respectively, at 1 standard deviation below the mean, 1

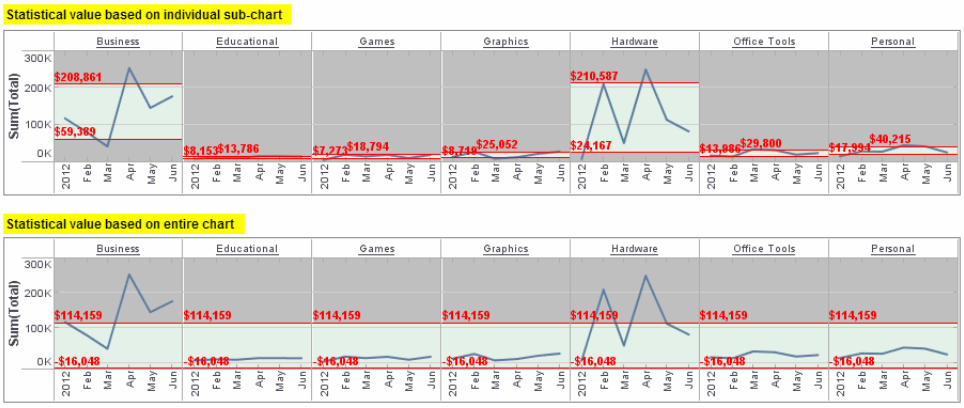
standard deviation above the mean, 2 standard deviations below the mean, and 2 standard deviations above the mean.



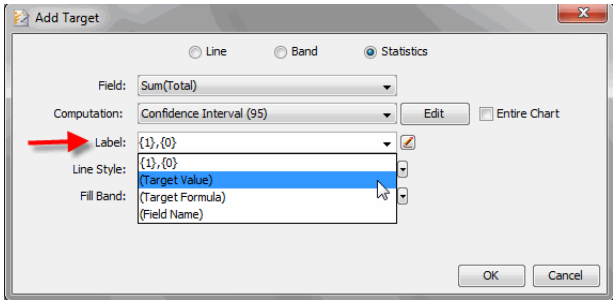
Select the ‘Sample’ option to compute the sample standard deviation or select the ‘Population’ option to compute the population standard deviation. (The distinction between sample and population standard deviation can be found in any statistical reference.)

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).

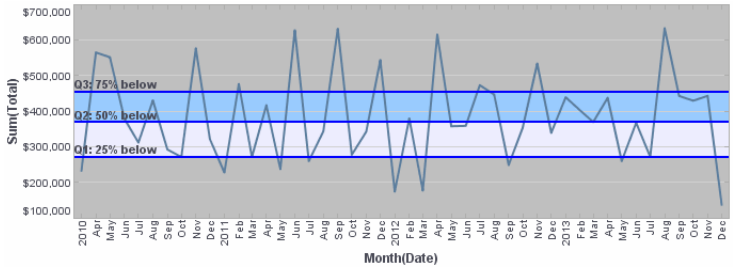


9. From the ‘Label’ menu, select one of the following label options:



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.

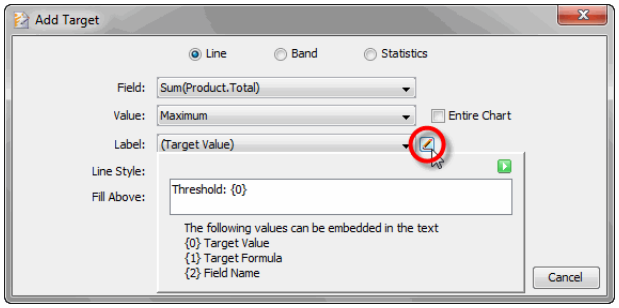
b. Select '(Target Value)' to insert the numerical value of the lines as the labels.

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").

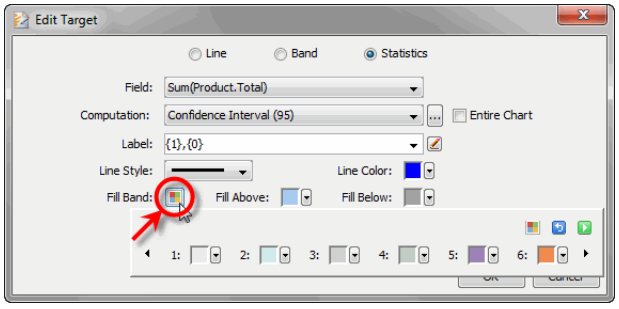
Labels for individual lines should be separated by commas. For example, if you are generating the 4-quantile (which creates three lines), enter three labels separated by commas, e.g., "`Q1: 25% below, Q2: 50% below, Q3: 75% below`".

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)"
```

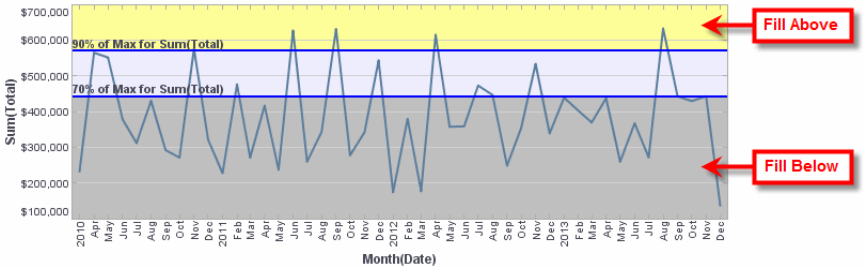
11. Select a 'Line Style' and 'Line Color' in which to display the statistical lines.

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 'Add Target' dialog box, and press 'OK' to close the 'Chart Properties' 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 *Plot Property* 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.

5.11 Special Elements

This section covers several special report elements.

5.11.1 Report Bean

A *Report Bean* is a reusable, customizable, composite report element. In other words, a bean is a group of "bundled" report elements that can be reused in multiple reports. When you design the report bean, you can decide which properties of these bundled elements will be exposed for customization in the individual report, and which properties will be fixed by the bean definition.

Like all reusable report components, report beans are stored in the *stylereport.srl.d* library in the repository. See *Configuring a Repository* in *Using Style Studio* for information on setting the location of this file.

Creating and Saving a Report Bean

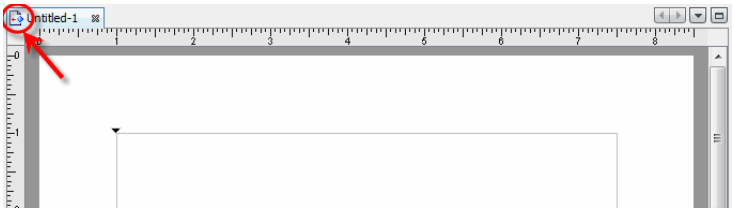
To create a report bean, follow the steps below:

1. Click the ‘New’ button in the Style Studio toolbar. This opens the ‘New Asset’ dialog box.

2. Click the ‘Component’ node in the left panel.

3. In the right ‘Type’ panel, select ‘Report Bean’ and click ‘OK’.

This create a new blank bean to which you can add report elements. The icon in the tab indicates that this asset is a *bean*, rather than a *report*.



Note that the ‘Deploy’ toolbar option is disabled, because a bean cannot be deployed independently. It can only be used within a report.

Walkthrough

In this example, you will create a composite component (bean) that consists of a text element and a chart. In order to allow report designers to customize the bean for the particular report in which they use it, you will expose certain bean properties for modification. In this case, you will expose the contents and alignment of the text element, and the chart type and legend position for the chart element.

Follow the steps below:

1. Click the ‘New’ button in the Style Studio toolbar. This opens the ‘New Asset’ dialog box.

2. Click the ‘Component’ node in the left panel.

3. In the right ‘Type’ panel, select ‘Report Bean’ and click ‘OK’. This creates a new report bean.



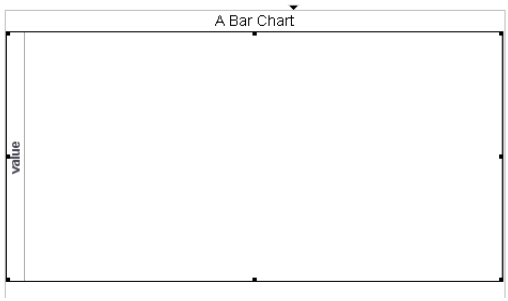
4. Click the ‘Text’ button in the **Toolbox** panel to insert a text element into the bean.

5. Enter the following text as the contents of the Text element: “A Bar Chart”.

6. Use the toolbar buttons to change the text alignment to ‘horizontal center’, and increase the font size to 14 point.



7. Click below the Text element to place the insertion point after the text element. Clicking the ‘Chart’ button in the **Toolbox** panel to add a chart.



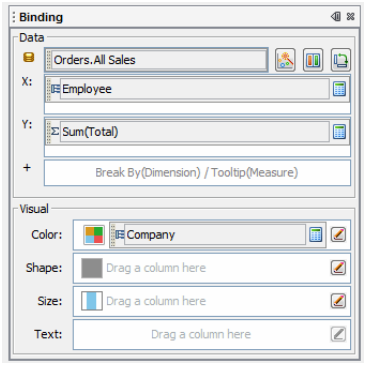
8. Click the chart to select it. This opens the **Data Source** and **Binding** panels.

9. On the **Data Source** panel, expand the ‘Orders’ data source, the ‘Sample Queries’ folder, and the ‘All Sales’ query.

10. From the **Data Source** panel, drag the ‘Employee’ field to the ‘X’ region of the **Binding** panel.

11. From the **Data Source** panel, drag the ‘Total’ field to the ‘Y’ region of the **Binding** panel.

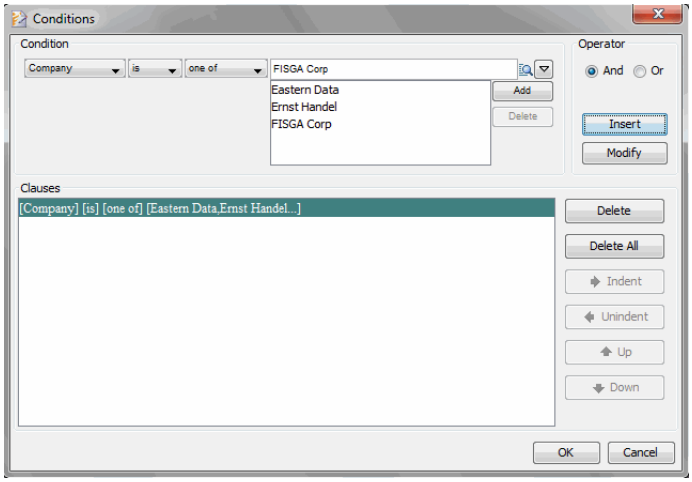
12. From the **Data Source** panel, drag the ‘Company’ field to the ‘Color’ region of the **Binding** panel.



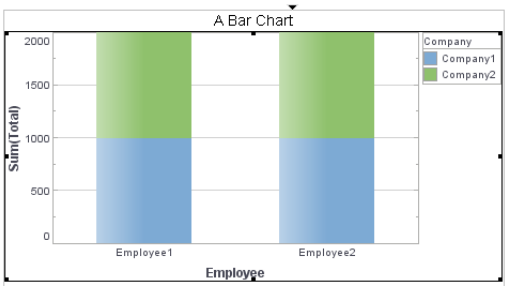
13. Right-click the chart, and select ‘Condition’ from the context menu. This opens the ‘Conditions’ dialog box.

14. In the ‘Conditions’ dialog box, enter the following condition:

```
[Company][is][one of][Eastern Data,Ernst Handel,FISGA Corp]
```

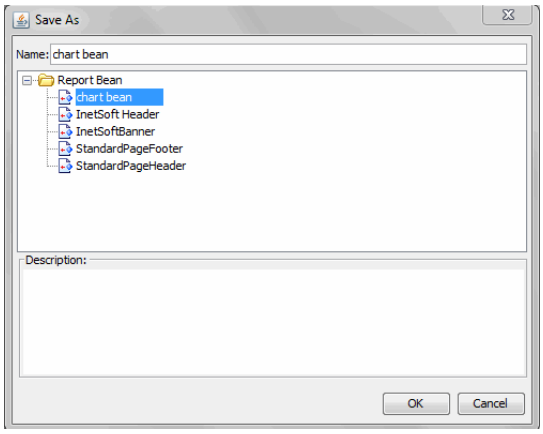


15. Click ‘OK’ to close the ‘Conditions’ dialog box.



16. Click the ‘Save’ button on the Style Studio toolbar. This opens the ‘Save As’ dialog box.

17. Enter the name “chartbean,” and type a description (if desired).



18. Click ‘OK’ to save the bean in the bean library.

This example continues in *Exposing the Properties of Report Bean Elements*.

Exposing the Properties of Report Bean Elements

Walkthrough

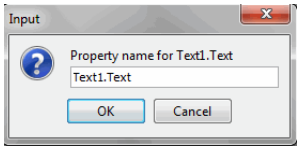
You will now expose certain properties of the report bean that you created in the previous example, *Creating and Saving a Report Bean*. This will allow report designers to customize these properties of the bean for the particular report in which they use the bean. For the text element, you will expose the text contents and the text alignment property. For the chart element you will expose the chart type and the legend position.

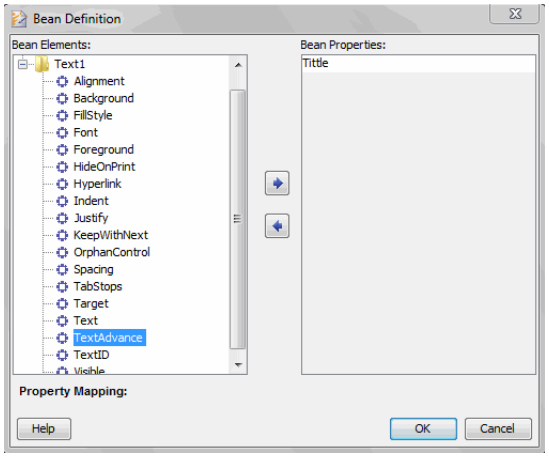
Continuing the previous example (with the bean open for editing in Style Studio), follow the steps below:

1. Select 'Bean Definition' from the Style Studio 'Report' menu. This opens the 'Bean Definition' dialog box.

2. Expand the 'Text1' node in the 'Bean Elements' pane. Select 'Text' and click the right-arrow. This opens a dialog box that allows you to name the exposed property.

3. Enter "Title" as the name, and click 'OK'. This adds the 'Text' property to the 'Bean Properties' list with the alias "Title".





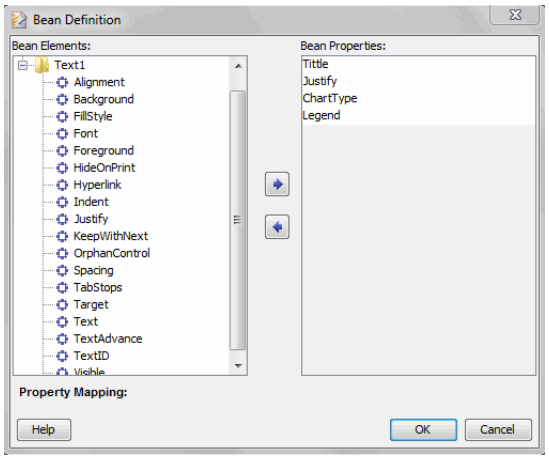
4. Select the 'Alignment' property from beneath the 'Text1' node in the 'Bean Elements' pane.

5. Click the right-arrow, and label the exposed property "Justify". Click 'OK'.

6. Expand the 'Chart1' node, select the 'ChartStyle' property, and click the right-arrow. Name the property "ChartType".

7. Select the ‘LegendPosition’ property from beneath the ‘Chart1’ node in the ‘Bean Elements’ pane.

8. Click the right-arrow, and label the exposed property “Legend”. Click ‘OK’.



9. Click ‘OK’ to close the ‘Bean Definition’ dialog box.

10. Click the ‘Save’ button in the Style Studio toolbar to save the bean.

11. Close the ‘X’ button in the bean tab, or press Ctrl-W on the keyboard.

This example continues in *Using a Report Bean in a Report*.

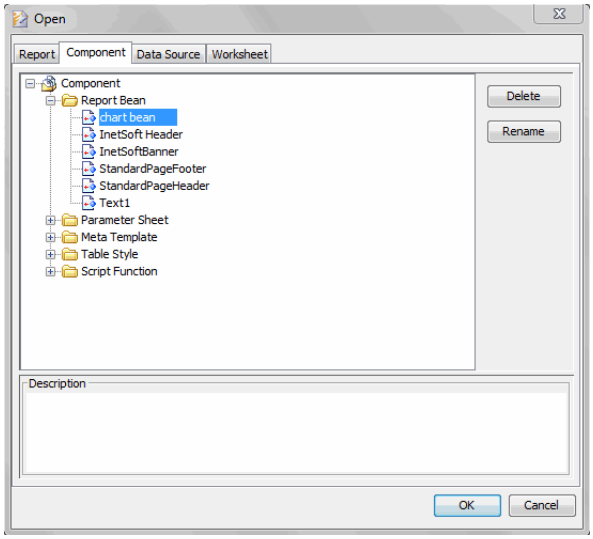
Editing a Report Bean

To open a report bean for editing, follow the steps below:

1. Click the ‘Open’ button on the Style Studio toolbar.

2. Select the **Component** tab.

3. Expand the ‘Report Bean’ node. This displays a list of all beans saved in *stylereport.srl.d*.



4. Select the bean you wish to edit and click 'OK'. This opens the bean for editing.

Using a Report Bean in a Report

Walkthrough

You will now insert the bean you created earlier into a report. This example continues from *Exposing the Properties of Report Bean Elements*.



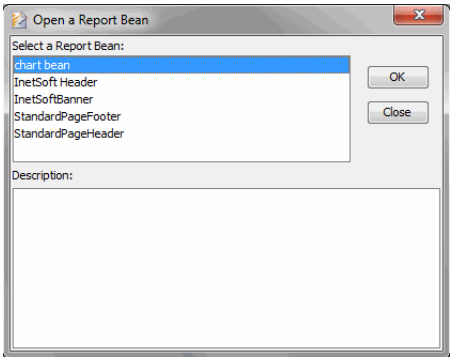
1. Click the 'New' button in the Style Studio toolbar. This opens the 'New Asset' dialog box.

2. In the left panel, select 'Report'. In the right 'Type' panel, select 'Blank Tabular Report' and click 'OK'. This opens a new report for editing.

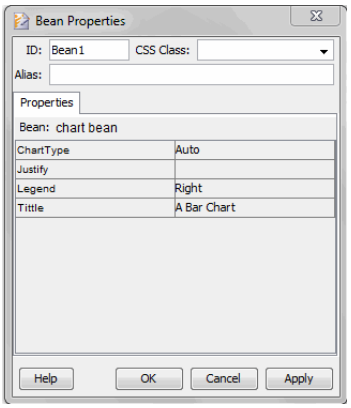


3. Click the 'Bean' component in the 'Special' region of the **Toolbox** panel. This opens the 'Open a Report Bean' dialog box.

4. Select "chartbean" and click 'OK'. This places the bean into the report. (The bean contains a Text element and a Chart element.)



5. Double-click anywhere on the inserted bean. This opens the 'Bean Properties' dialog, which lists the bean's exposed properties. (See *Exposing the Properties of Report Bean Elements*.)



6. Make the following modifications:

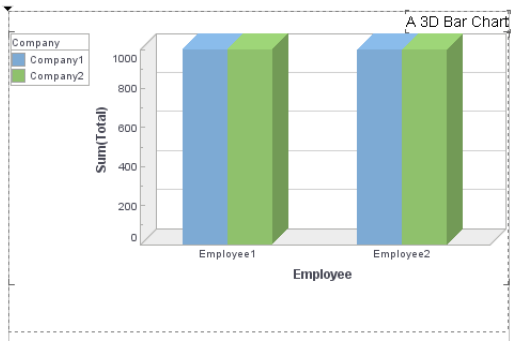
   a. From the 'ChartType' menu, select '3D Bar'.

   b. From the 'Justify' menu, select 'Right Center'.

   c. From the 'Legend' menu, select 'Left'.

   d. In the 'Title' field, type "A 3D Bar Chart".

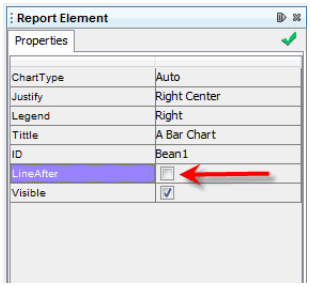
7. Click 'OK'. Observe the changes to the report bean.



8. Save the report.

If you close the report and then reopen it, you will observe that the property modifications that you made have been saved with the report. These properties are no longer referenced from the bean definition, so changes to the bean definition do not affect these properties. All other properties of the bean (i.e., those that were not exposed) are referenced from the original bean definition.

By default, elements that you insert following the bean will appear on a new line. To permit these elements to appear on the same line as the bean, open the **Report Element** panel, and deselect the 'LineAfter' option from the **Properties** tab.



5.11.2 Subreport

A subreport is a report embedded within another report. A subreport can only be embedded within a Section element. See *Subreports* for more information.

5.11.3 Heading

A Heading element is a special kind of text element, which has an additional *heading level* attribute. The heading level has two effects on the element:

- The report engine automatically generates a heading number and (optionally) prints the number to the left of the text.
- The report engine uses the heading level to generate a Table Of Contents (TOC) if a TOC element exists in the report. (See *Table of Contents* for details.)

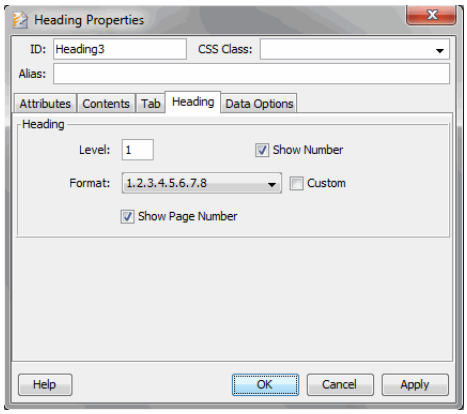
To add a Heading element to a report, follow these steps:

1. Click in the report to place the cursor at the location where you want to add the new heading.
2. Click the 'Heading' component in the **Toolbox** panel. This opens a popup menu containing the available level options.
3. Select the heading level from the popup menu. This inserts a new Heading element at the cursor location. (The heading level can be changed later.)
4. Enter the desired heading text into the text field. (See the *Text Element* section for details about text elements.)

5. Right-click the heading element, and select ‘Properties’ from the context menu. This opens the ‘Heading Properties’ dialog box.

6. Select the **Heading** tab in the ‘Heading Properties’ dialog box.

7. Enter the ‘Level’ of the heading, and select a heading ‘Format’.



   The dot-separated formats, such as “1.2.3.4.5.6.7.8” and “a.b.c.d.e.f.g.h,” indicate that the number or letter of the selected heading will be prefixed with the numbers/letters of its superordinate headings. The single-character formats, such as “8” or “h,” indicate that the number of the selected heading will be displayed without any prefixed digits. To define a custom heading format, select the ‘Custom’ option, and specify a class that implements the `inetsoft.report.HeadingFormat` class.

8. Select ‘Show Number’ to display the heading number to the left of the heading text (default).

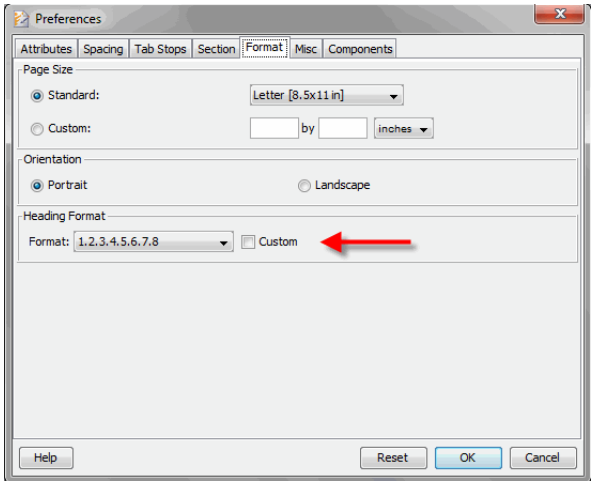
9. Select ‘Show Page Number’ to display the heading page number in a generated TOC (default).

To set a default heading numbering style, follow these steps:

1. Select ‘Preferences’ from the Style Studio ‘Report’ menu.

2. Select the **Format** tab.

3. Enter the default ‘Format’ to be used for newly created headers.



See Also

Adding Heading Elements to a Section, to use heading elements to generate a TOC of group names.

5.11.4 Table of Contents

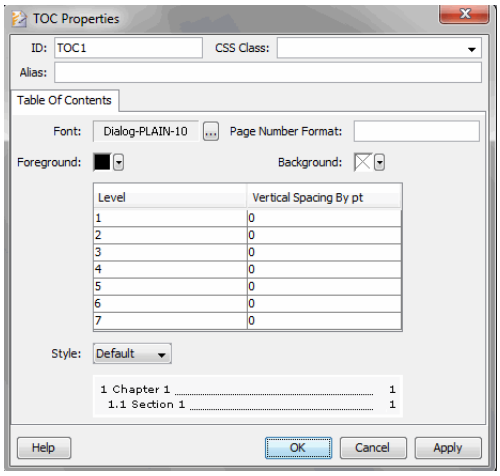
To add a TOC element, follow these steps:

1. Click in the report to place the cursor at the location where you want the Table of Contents to be inserted.



2. Click the 'TOC' component in the **Toolbox** panel. This automatically generates a TOC based on the headings already defined in the report.

3. Right-click the TOC, and select 'Properties' from the context menu. This opens the 'TOC Properties' dialog box.



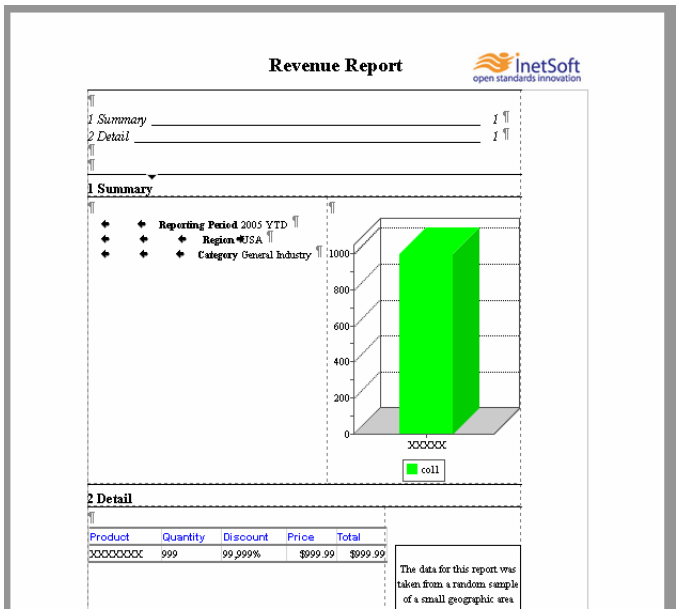
Additional styles can be created using the Java API.

4. Select a TOC style from the ‘Style’ menu. There are seven built-in styles.

5. Format the page number display in the TOC by using the ‘Page Number Format’ option.

   There are two special tags available for use in formatting: The “{0}” tag inserts the section number on the TOC, and the “{1}” tag inserts the actual page number. For example, the format “*Topic {0} found on page {1}*” would translate to something similar to “*Topic 1.2 found on page 5.*”

Figure 14. A Report with Table of Contents (toc1.srt)



5.11.5 Painter

To add a painter, click the ‘Painter’ component in the **Toolbox** panel. This opens the ‘Painter Properties’ dialog, where you can set attributes such as the anchor, text wrapping, margin, and rotation for the Painter. The Painter appears as a gray box area on the report, indicating where the actual Painter will be drawn. You need to associated a Painter object with a report programmatically.

5.11.6 Image

The image element is commonly used to decorate a page (e.g., adding a corporate logo or fancy report header).

Adding an Image Element

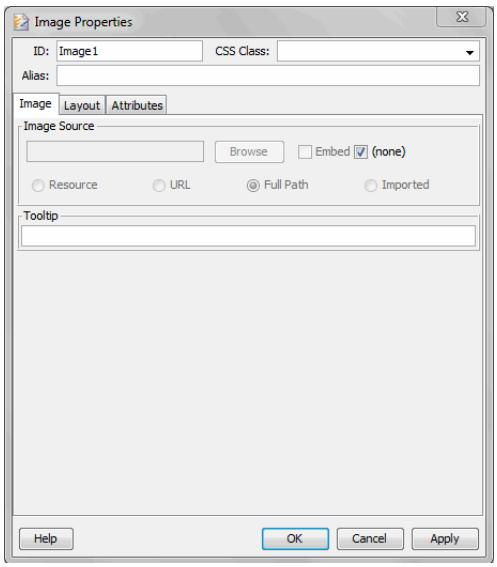
To add an image to a report, follow the steps below:

1. Click in the report to place the cursor at the location where you want to place the image.

You can also simply copy-and-paste an image onto the report page. This creates an embedded image of type 'Imported'.

2. Click the Image component in the **Toolbox** panel. This opens the 'Image Properties' dialog box.

3. Uncheck the '(none)' option, and enter the image location. See *Specifying the Image Location* below.



4. Select 'Embed Image' to embed the image data in the report template as an XML object (convenient for smaller images). See *Specifying the Image Location* below.

5. Click 'OK' to close the dialog box.

Specifying the Image Location

You can specify the location of image files using any of the following methods:

- Java resource file loaded by the class loader
- URL
- Relative file path on local file system
- Absolute file path on local file system

The specification for the image location depends on how the application is deployed. The types of supported image files are the same as the Java implementation, which include GIF, JPEG, and PNG (with JDK 1.3 and higher).

When you ‘Embed’ an image in a report, the image data is saved in the report template as an XML object. Subsequently, the image is loaded from the template directly without consulting the specified image path.

Image embedding is very convenient for smaller images, eliminating the need to manage the deployment of image files. If the image is very large, however, the resulting template file will be large. If multiple template files share the image, it is best to deploy the image as a resource file; thus the overall size of the template files remain small.

5.12 Decorative Elements

This section discusses various decorative report elements.

5.12.1 Tab



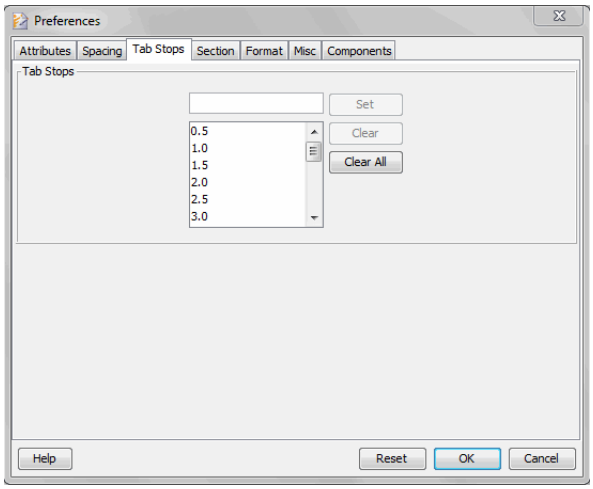
A Tab is a simple mechanism for aligning elements vertically at specified tab stop positions. This is useful for creating forms, where field labels and values are lined up vertically in multiple rows. One disadvantage of the tab is that if the size of an element causes the cursor to move past the tab stop, it automatically advances to the next tab stop. This results in the vertical alignment of the elements being incorrect. You can avoid this by increasing the size of the default tab stops.

Defining a Tab Stop

To set a Tab stop, follow these steps:

1. Open the ‘Report’ menu in the Style Studio toolbar, and select ‘Preferences’. This opens the ‘Preferences’ dialog box.

2. Select the **Tab Stops** tab. (The default tab stops are placed every half inch across the page.)



3. (Optional) To clear all default tabs, click ‘Clear All’. Click ‘Reset’ to restore the default tabs.

4. Enter a value for the new tab location in the text field, and click the ‘Set’ button (or press ‘Enter’ on the keyboard).

5. Click ‘OK’ to close the dialog box.

   This setting will be inherited by all new Tab elements that you add to the report.

Adding a Tab



To insert a Tab, click the ‘Tab’ component on the **Toolbox** panel or press ‘Tab’ on the keyboard.

Creating a Right-Aligned Tab

Text elements are left-aligned by default. To align components on their right edges, you can use right-aligned tabs.

To create a right-aligned tab, follow the steps below:

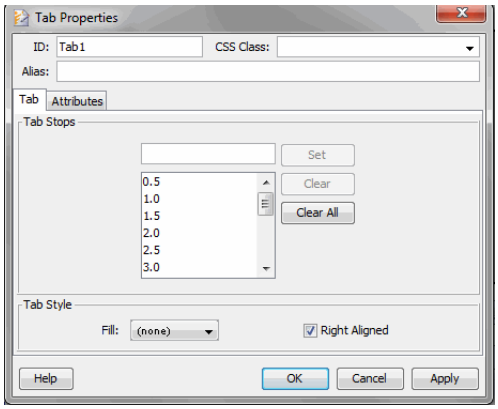


1. Add the Tab to the page (click the ‘Tab’ component on the **Toolbox** panel or press ‘Tab’ on the keyboard).

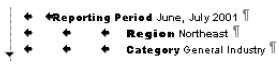
2. Right-click the Tab element and select ‘Properties’ from the context menu. This opens the ‘Tab Properties’ dialog box.

3. Select the **Tab** tab.

4. Check the ‘Right Aligned’ option, and click ‘OK’.



The tab arrows will now point to the left and the text element will be right-aligned.

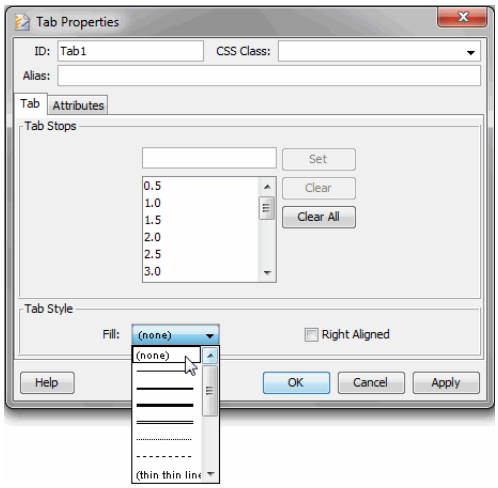


Adding a Tab Leader (Filling)

The space occupied by a Tab can be filled with a line (leader). The leader line is placed at the bottom of the tab.

To create a tab with leader, follow the steps below:

1. Add the Tab to the page (click the 'Tab' component on the **Toolbox** panel or press 'Tab' on the keyboard).
2. Right-click the Tab element and select 'Properties' from the context menu. This opens the 'Tab Properties' dialog box.
3. Select the **Tab** tab.
4. Select the desired leader style from the 'Fill' menu, and click 'OK'.

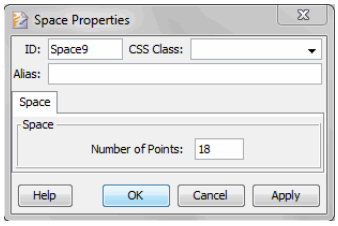


5.12.2 Space

A Space element adds horizontal space between the element preceding the space and the element that follows. The space is specified in points.

To adjust the size of the space, follow the steps below.

1. Right-click the Space element and select 'Properties' from the context menu. This opens the 'Space Properties' dialog box.
2. Enter a value for 'Number of Points', and click 'OK'.



5.12.3 Bullet



A Bullet is a special image element that uses the internal bullet icon. Text that follows the bullet aligns flush with the bullet position.

To add a bullet and its associated text, follow the steps below:

1. Click in the report to place the cursor at the location where you want to add the bulleted text.

2. Click the 'Bullet' component in the **Toolbox** panel. This adds the bullet image to the report, as well as the associated Text element.

3. Enter the desired text in the Text element.

4. (Optional) To change the size of the bullet, follow these steps:

    a. Right-click the bullet icon (not the text), and select 'Properties' from the context menu. This opens the 'Bullet Properties' dialog box.

    b. Click the 'Font' button, and adjust the font size as desired.

In general, the bullet font should be the same as the accompanying text font to maintain appropriate typographical appearance.

5.12.4 Editable Region

An Editable Region is a container that allows you to position editable components within a meta-template. Components that appear within an editable region can be modified within the report that is based on the meta-template, and those modifications are then saved with the individual report template. See *Designing a Meta-Template* for full details on how to use Editable Regions.

5.12.5 Separator

A Separator adds a horizontal line which occupies the entire width of the printable area.

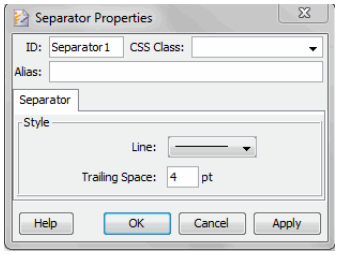
To add a Separator to the report, follow the steps below:

1. Click in the report to place the cursor at the location where you want to add the separator.

2. Click the ‘Separator’ component in the **Toolbox** panel. This adds the horizontal line to the report.

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

4. From the ‘Line’ menu, select the desired line style for the separator.



5. Click ‘OK’ to close the dialog box.

5.12.6 Newline

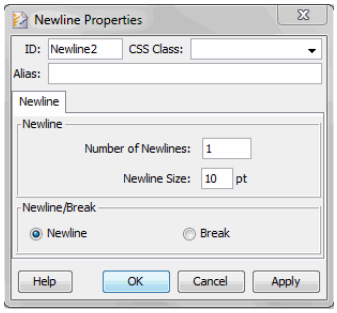
The ‘Newline’ object (carriage return) advances printing to the next line. If a Newline exists on a line by itself, the vertical advance is determined by the Newline size in points (1/72 inch).

To add a Newline, follow these steps:

1. Click in the report to place the cursor at the location where you want to add the Newline.

2. Click the ‘Newline’ component in the **Toolbox** panel. This adds the Newline element to the report.

3. Right-click the Newline element, and select ‘Properties’ from the context menu. This opens the ‘Newline Properties’ dialog box.



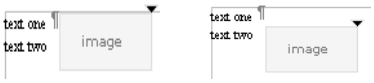
4. Specify the ‘Number of Newlines’ to add, as well as the ‘Newline Size’ or vertical displacement.

5. (Optional) Select ‘Break’ to mark the new line as a ‘break’. (See below for details.)

6. Click ‘OK’ to close the dialog box.

When you select the ‘Break’ option from the ‘Properties’ dialog box, elements following the Newline component are not forced onto a new line. Rather, the cursor is advanced according to the ‘Newline Size’ specification. In normal flow, a ‘Break’ behaves in the same way as a ‘Newline’. However, if anchored elements are used on the same line., this changes the flow.

For example, consider a text element that is followed by a break, which is in turn followed by another text element. When an anchored element is placed after the second text element, both text elements are printed to the left of the anchored element. If you change the break to a newline, the second text element and the anchored element are printed below the first element.



With the line break, all elements are processed in one pass to determine if they fit on the same line. Standard anchored element wrapping is then applied. When the break is changed to a newline, the elements up to the newline are processed first. After the cursor is advanced by the newline, the elements following the newline are then processed. This causes the elements following a newline to always be placed below the newline element.

5.12.7 Page Break

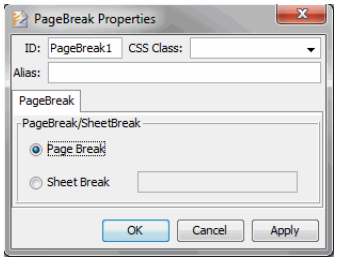
To add a Page Break, follow these steps:

1. Click in the report to place the cursor at the location where you want to add the Page Break.

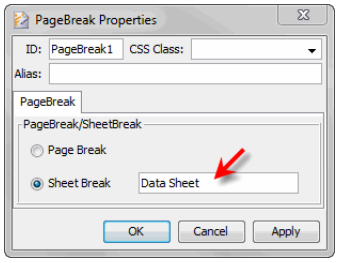
2. Click the ‘Page Break’ component in the **Toolbox** panel. This adds the Page Break element to the report.

3. Right-click the Page Break element, and select ‘Properties’ from the context menu. This opens the ‘Page Break Properties’ dialog box. There are two options for inserting page breaks.

a. Select ‘Page Break’ to immediately advance printing to the next page for reports that are exported in paginated format.



b. Select ‘Sheet Break’ to immediately advance printing to a new “sheet” for reports exported to Excel format. Enter a label for the new sheet in the adjacent text field.



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

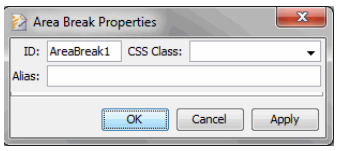
See Also

Exporting a Report, for information about presenting reports in particular formats.

5.12.8 Area Break

An area break serves a similar function to the page break (*Page Break*). However, instead of advancing the printing to the next page, an area break moves the flow to the next area on the page. If the area break is in the last area, it is the same as a page break.

To configure an Area Break element, right-click the element and select ‘Properties’ from the context menu. This opens the ‘Area Break Properties’ dialog box.

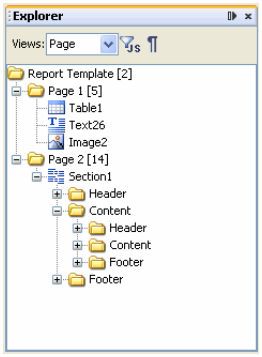


5.13 Report Explorer View

The *Report Explorer* provides a tree view of report elements to help you navigate your report design. This can be very effective when used together with the *Element Property View*.

- You can easily visualize element sequence and ordering, which may not be as obvious on the report page itself.
- You can filter the tree view to display only certain element types, such as elements containing script.
- You can select an element name in the tree to select the corresponding element on the page.

To open the **Report Explorer** panel, select 'Explorer Pane' from the Style Studio 'Window' menu.



The 'Views' menu at top of the Report Explorer provides three different displays of the report elements.

- **Page**: The tree lists each page as a node, and beneath each page all of the elements on that page.
- **Elements**: Lists all elements without pagination information.
- **Type**: Lists all elements grouped by their type, e.g., "Tables," "Charts," "Sections," etc.



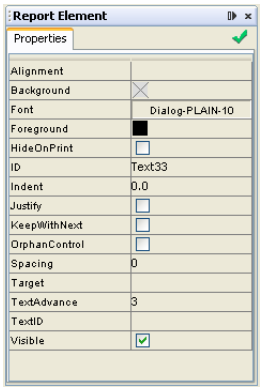
To display only elements containing script, click the 'Filter Script Elements' button.



To hide whitespace characters (spaces, tabs, pagebreaks, etc.), deselect the 'Filter Whitespace Elements' button.

5.14 Element Property View

The **Report Element** panel lists the properties of the currently selected element. To open the **Report Element** panel, select 'Element Pane' from the 'Window' menu.



Keep this panel open while you work to rapidly access element properties. This can be very effective when used together with the *Report Explorer View*.

Note that the full suite of properties available for a report element is available through the element's 'Properties' dialog box. To access the dialog box, right-click the element, and select 'Properties' from the context menu.

6 Data Binding

Data binding is the process of associating a report element with a dataset, usually provided by a data model or a query. In most cases, the binding information is stored in the report template, while the data itself is referenced from an external database. The data binding specifies the data source from which the report element draws data, as well as any filtering, sorting, grouping, or summarization that should be applied to the data prior to display.

The following report elements support data binding:

- Table (see *Binding Data to a Table Element*)
- Crosstab (see *Binding Data to a Crosstab Element*)
- Chart (see *Binding Data to a Chart Element*)
- Section (see *Binding Data to a Section Element*)
- Text and Textbox (see *Binding Data to a Text Element*)

Table, Crosstab, and Section elements offer a *Data Binding Wizard* to simplify and centralize the binding process. The following sections explain the various aspects of binding a report element to a data source.

See Also

The element's *query* property (e.g., *query* for Tables), to bind data using script.

Formula Tables, in *Advanced Report Scripting*, to populate a Table without global binding.

6.1 Data Binding Toolbar Buttons

The Style Studio toolbar contains several buttons for binding-related functions.

Table 11. Data Binding Toolbar Buttons

	**Live Edit**: Populates report elements with data using the elements' data binding information. You can continue to add and edit elements while previewing how the data will affect the layout. (See *Displaying a Report in Live Edit View* for more details.)
	**Preview**: Previews the current report by executing all queries and scripts, so that the report appears in its final generated state. (See *Displaying a Report in Live Edit View* for more details.)
	**Refresh**: Rebuilds the metadata for queries and data models from their definitions, and refreshes the report. This is needed if query or data model definitions change and Style Studio does not detect the changes.

6.2 Creating a New Table, Crosstab, or Section using a Wizard

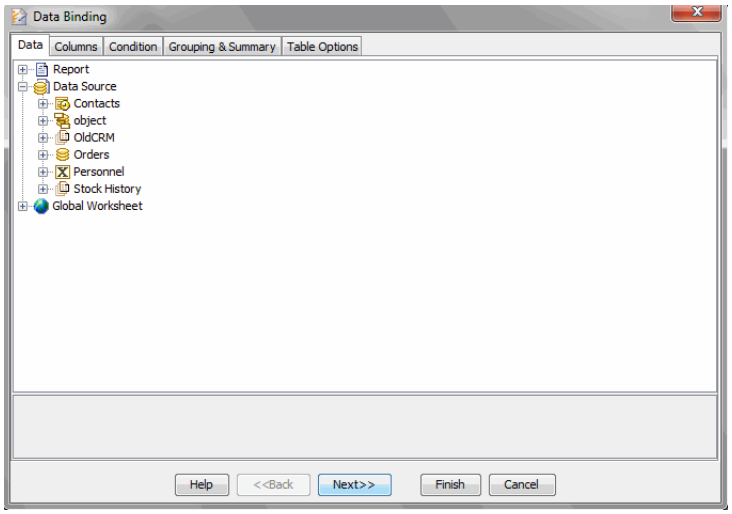
Data binding examples can be found in the *examples/docExamples/datasource* directory.

Style Studio provides a *Data Binding Wizard* for Table, Crosstab, and Section elements. This Wizard makes it easy to rapidly bind these elements to a data source. The features provided by the Wizard vary for the different elements. This section provides an outline of the general procedure for using the

To create and bind a new Table, Crosstab, or Section, follow the steps below:

1. Click in the report to place the cursor at the location where you would like to insert the new element.

2. In the **Toolbox** panel, click the element that you want to insert and bind to data: 'Table', 'Crosstab', or 'Section'. This opens the 'Data Binding' dialog box for the element.



3. Click the **Data** tab, and select the desired data source. See *Walkthrough: Choosing the Binding Data Source* for more details.

4. Click the **Columns** tab. Select the desired columns to display, and create any desired formula columns. See *Specifying Columns in Data Binding Wizard* for more details.

5. Click the **Condition** tab, and specify a filtering condition (if desired). See *Filtering Data in Data Binding Wizard* for more details.

6. Click the **Grouping and Summary** tab, and select grouping and aggregation options (if desired). See *Grouping and Summarization in Data Binding Wizard* for more details.

7. Click the **Options** or **Section Options** tab, and make additional settings as described in *Table Grouping Options*, *Crosstab Options*, or *Section Options for Data Binding Wizard*.

6.2.1 Walkthrough: Choosing the Binding Data Source

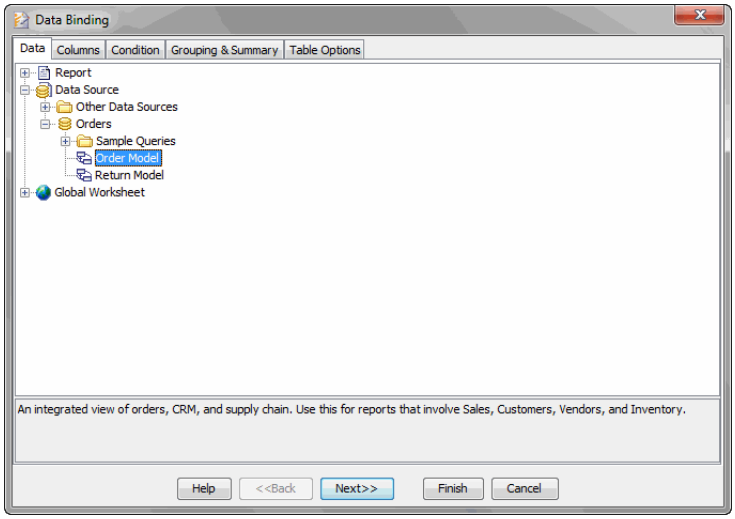
Walkthrough

Use the **Data** tab of the 'Data Binding' dialog box to specify the source of the dataset for an element binding. Follow the steps below.

1. Click the **Data** tab in the Data Binding dialog box.

2. Select the desired data source. The binding source can be a data model, query, Worksheet, or another report element.

   For this walkthrough example, select 'Orders' > 'Order Model'.



This example continues in the next section, *Walkthrough: Selecting and Creating Columns in Data Binding Wizard*.

See Also

Data Sources Available for Binding, for details regarding the data source types.
Advanced Data Modeling, for information on creating data models and queries.
Advanced Data Mashup, for information on creating Data Worksheet assets.

6.2.2 Data Sources Available for Binding

There are several types of data sources available for data binding:

- **Query**: A global query stored in *asset.dat.d*.
- **Local Query**: A query embedded within a report template.

- **Data Model**: An entity-attribute model providing a broad view of the database
- **Report Data**: Data bound to another report element (chart, table, section).
- **Worksheet**: A Data Block created in a Data Worksheet.
- **Local Worksheet**: A Data Block embedded within a report template.
- **Embedded Data**: Data that you enter into the report manually.
- **Parameter**: A named input parameter for the report.

The data binding process is similar for the most data sources, but parameter binding uses a different procedure. See *Binding Data to a Text Element* for an example of parameter binding.

Queries and data models are listed under their respective data sources in the tree, worksheets are listed under the 'Worksheet' node, and report data and parameters are listed underneath the 'Report' node.

Data source information (by default) is also embedded in the report template file. However, if a matching data source exists in the *asset.dat* file or *asset.dat.d* directory, the information will be drawn from there rather than from the embedded data.

Local Queries and Local Worksheets

Local queries are listed under the 'Report' > 'Local Query' node in the Style Studio **Asset** panel. Local queries are regular queries that are stored within the report template rather than in the global query registry. See *Creating a Local Query* in *Advanced Data Modeling* for more information.

Local Worksheets are listed under the 'Report' > 'Local Worksheet' node in the Style Studio **Asset** panel. Local Worksheets are regular Data Worksheets that are stored within the report template rather than in the *asset.dat* file or *asset.dat.d* directory. See *Creating a Local Worksheet* in *Advanced Data Mashup* for more information.

Because local queries and local Worksheets are not saved in the global *asset.dat* file or *asset.dat.d* directory, they are not available to reports other than the parent report. However, you can export a local query or local Worksheet to the global registry (*asset.dat.d*) to make it publicly available. See *Exporting a Local Query to the Global Repository* in *Advanced Data Modeling* and *Exporting a Local Worksheet to the Global Repository* in *Advanced Data Mashup* for more information.

See Also

Independent Query, in *Advanced Data Modeling*, for information on queries.

Creating a Local Query, in *Advanced Data Modeling*, for information on embedded queries.
Creating a Local Worksheet, in *Advanced Data Mashup*, for information on embedded Worksheets.
Data Models, in *Advanced Data Modeling*, for logical models.
Creating a Data Block, in *Advanced Data Mashup*, for Worksheet queries.
Parameterization, for information on named input parameter.

6.2.3 Specifying Columns in Data Binding Wizard

In the 'Data Binding' dialog box, the **Columns** tab allows you to select the fields from data source to include in the binding, as well as to create derived columns (i.e., *formula/expression columns*) and to specify sorting.

Selecting Columns in Data Binding Wizard

The fields that you select from the **Columns** tab will be returned by the data source and displayed by the report element. To select columns from the data source, follow the steps below:

1. In the 'Data Binding' dialog box, select the **Columns** tab. (See *Creating a New Table, Crosstab, or Section using a Wizard* for information on how to access the 'Data Binding' dialog box.)

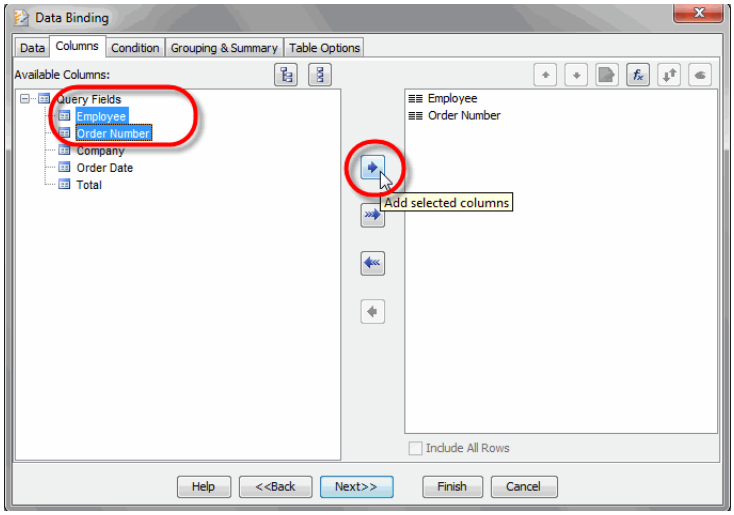
2. Add the desired columns from the 'Available Columns' panel to the right panel. You can do this by any of the following methods:

   a. Double-click a column in the 'Available Columns' panel to add it to the right-side panel.

   b. Click to select a column (or Shift/Ctrl-click to select multiple columns) in the 'Available Columns' panel. Drag the selected column(s) to the right-side panel.

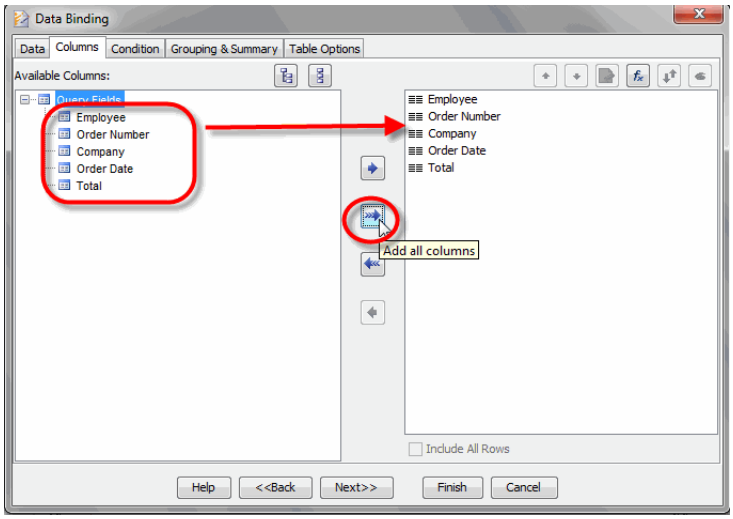
   c. Click to select a column (or Shift/Ctrl-click to select multiple columns) in the 'Available Columns' panel. Click the right-arrow button to add the selected column(s) to the right-side panel.

The left arrow removes a selected field.



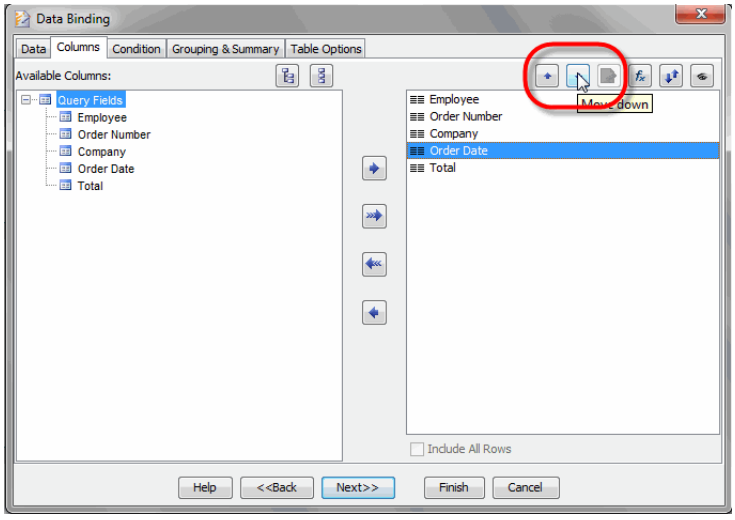
The left "multi-arrow" removes all of the selected fields.

d. Click the right "multi-arrow" to select all of the fields in the 'Available Columns' panel.



e. For a data model, click to select an entity (or Shift/Ctrl-click to select multiple entities) in the 'Available Columns' panel. Click the right-arrow button to add *all* of the selected attributes from the selected entity to the right-side panel.

3. To change the ordering of a selected column, click to highlight the column, and then press the 'Move up' and 'Move down' buttons.



4. Click ‘Finish’ to close the ‘Data Binding’ dialog box, or continue with other desired modifications.

To remove a field from the right-side panel, select the field and click the left-arrow button. In general, fields that are not among the selected fields (in the right-side panel) are removed from the query that is sent to the database. However, in cases where the data binding specifications cannot be merged into the original query, the full query is executed and all query fields are returned. Nevertheless, only the selected fields are displayed in the report element.

Sorting Columns in Data Binding Wizard

You can sort columns in original, ascending, or descending order. (For string data, ascending order is alphabetical, and descending order is reverse-alphabetical.)

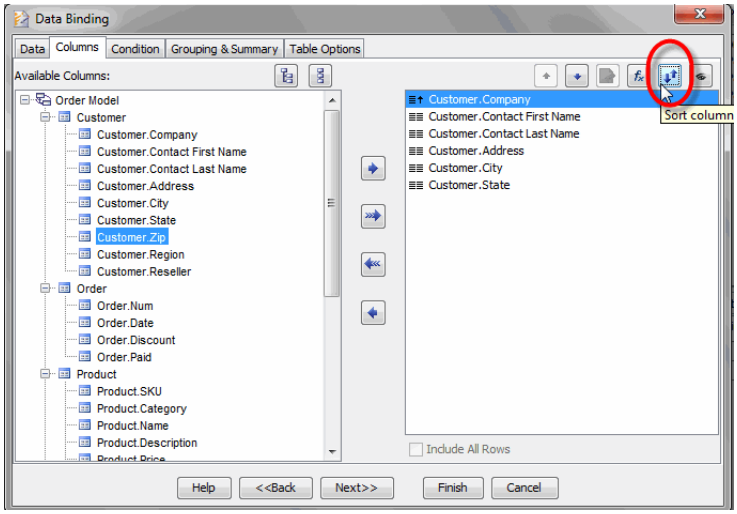
To sort a column, follow the steps below:

1. In the ‘Data Binding’ dialog box, select the **Columns** tab. (See *Creating a New Table, Crosstab, or Section using a Wizard* for information on how to access the ‘Data Binding’ dialog box.)

2. Click the desired column in the right-side panel to select it.

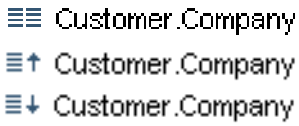


3. Click the ‘Sort column’ button once to set ascending order. Click a second time to set descending order, and a third time to restore the original sorting.



The sort order is indicated by the icon to the left of the field name. An “up-arrow” indicates ascending order, while a “down-arrow” indicates descending order.

Figure 15. Column sorted in original, ascending, and descending order



Hiding Columns in Data Binding Wizard

To remove columns from the query entirely, see *Selecting Columns in Data Binding Wizard*.

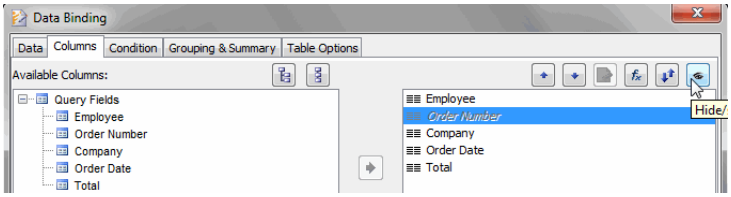
The **Columns** tab of the ‘Data Binding’ dialog box allows you to hide particular columns so that these columns are not displayed by the report element. Hidden columns are still included in the query that is sent to the database, which means that you can still access these fields for purposes of grouping and summarization (see *Grouping and Summarization in Data Binding Wizard*).

To hide a column follow the steps below:

1. In the ‘Data Binding’ dialog box, select the **Columns** tab. (See *Creating a New Table, Crosstab, or Section using a Wizard* for information on how to access the ‘Data Binding’ dialog box.)

2. Click the desired column in the right-side panel to select it.

3. Click the ‘Hide/show column’ button to hide the column. (Click a second time to show the column.)



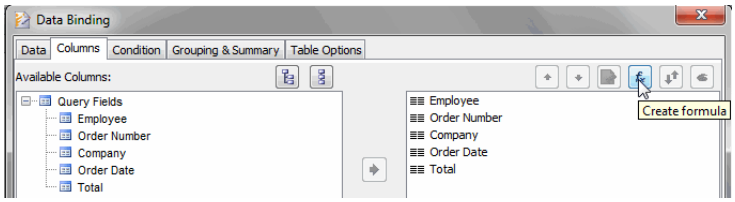
Creating a Formula Column in Data Binding Wizard

In addition to selecting existing columns directly from a data source (query, data model, etc.) you can also create new columns that are derived from existing columns. The values in these derived columns are computed by using SQL or JavaScript commands, and are called *formula columns* or *expression columns*.

To create a formula column, follow the steps below:

1. In the 'Data Binding' dialog box, select the **Columns** tab. (See *Creating a New Table, Crosstab, or Section using a Wizard* for information on how to access the 'Data Binding' dialog box.)

2. Click the 'Expression' button (labeled '*fx*') above the right-side panel. This opens the 'Formula' dialog box.



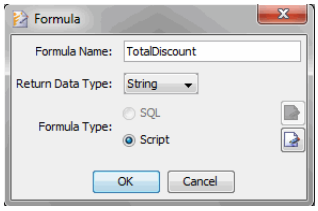
3. With the 'Formula' dialog box open, follow the steps below:

   a. In the 'Formula Name' field, enter a name for the new column.

   b. For 'Return Data Type', select the data type of values that will populate the new column. This data type should match the type of value returned by your script.

Note: In cases where the data binding specifications cannot be merged into the query, the 'SQL' option is not available. (This may be due to the presence of grouping and aggregation at the query level.)

   c. For 'Formula Type' select 'SQL' or 'Script', and then click the corresponding 'Edit' button. This opens the Script Editor.



4. Enter the desired script, using the syntax (SQL or JavaScript) that you specified in the previous step. To data values, use the following syntax:

   a. To access the value in another column of the *current* row, use the "`field['field name']`" syntax.

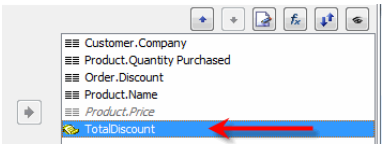
   b. To access the value in another column of a row at a certain offset (i.e., relative position) from the current row use the following syntax:

      ```
      field[relative position]['field name'];
      ```

      For example, `field[-1]['Product.Total']` contains the value in the `Product.Total` field for the previous record.

   For more information on scripting, see *Advanced Report Scripting*. For a simple example of a formula column script, see *Walkthrough: Selecting and Creating Columns in Data Binding Wizard*.

5. Close the Script Editor, and click 'OK' to close the 'Formula' dialog box. This adds the new formula column (indicated by "script" icon) to the list of selected columns.



6. Click 'Finish' to close the 'Data Binding' dialog box, or continue with other desired modifications.

To edit an existing formula column, select the desired column, and then click the 'Edit' button. This will reopen the Script Editor. You can also hide, sort, reorder, or remove a formula column in the same way as other columns. See the appropriate section in *Specifying Columns in Data Binding Wizard* for instructions.

See Also

Selecting Columns in Data Binding Wizard, for information on selecting existing columns.

Walkthrough: Selecting and Creating Columns in Data Binding Wizard

Walkthrough

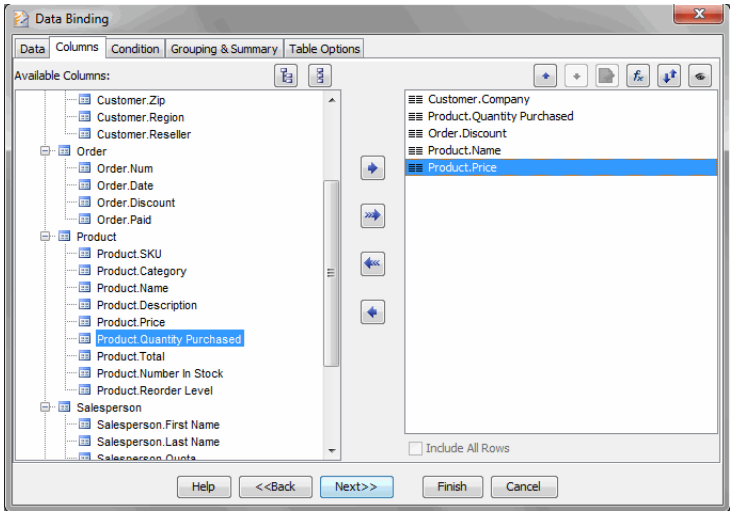
This example is continued from the previous section, *Walkthrough: Choosing the Binding Data Source*.

1. In the 'Data Binding' dialog box, select the **Columns** tab. (See *Creating a New Table, Crosstab, or Section using a Wizard* for information on how to access the 'Data Binding' dialog box.)

2. Add the following columns from the tree on the left by selecting the column and clicking on the right-arrow button:

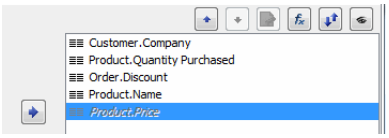
   'Customer.Company'
   'Product.Quantity Purchased'
   'Order.Discount'
   'Product.Name'
   'Product.Price'

You can remove any selected fields by using the left-arrow button.



   For data models only, if you want the generated query to use full-outer joins rather than the model's predefined joins, select 'Include all rows', beneath the right panel. (Note: This may return a very large amount of data.)

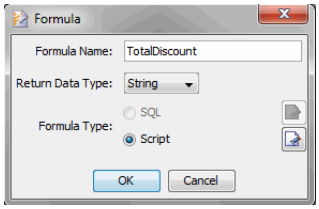
3. Select 'Price' and click the 'Hide/show column' button to hide the column.





4. Select the ‘Create formula’ button.

5. Enter the name “TotalDiscount” and select ‘Float’ for the return data type.

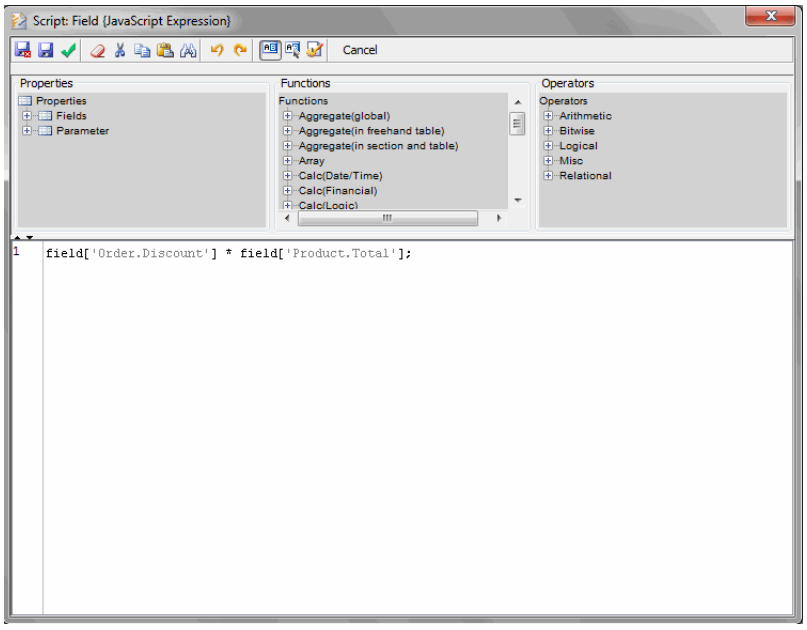




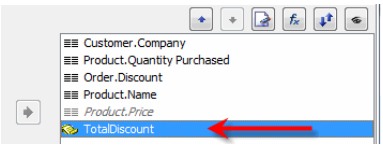
6. Make sure ‘Script’ is selected and click on the ‘Edit’ icon.

7. Enter the following formula either by typing directly or by selecting the appropriate fields from the GUI:

```
field['Order.Discount'] * field['Product.Total'];
```



8. Click the ‘Save and Close’ button, and click ‘OK’ in the ‘Formula’ dialog box to add the formula column to the columns list.



This example is continued in the following section, *Walkthrough: Adding Conditions in Data Binding Wizard*.

6.2.4 Filtering Data in Data Binding Wizard

Use the **Condition** tab to add filtering conditions to the result set. You can compare field values to a specified fixed value or to a parameter (variable). The **Condition** tab also enables the use of nested conditions (defining an explicit evaluation order).

See Also

Parameterizing a Condition, for details on using a variable in a condition.

Walkthrough: Adding Conditions in Data Binding Wizard

Walkthrough

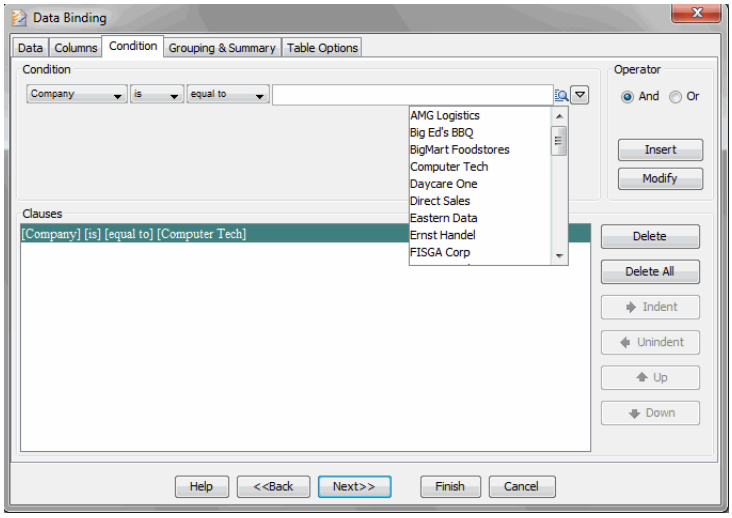
This example is continued from the previous section, *Walkthrough: Selecting and Creating Columns in Data Binding Wizard*. Assume you want to list all the orders for the "Computer Tech" *or* "Eastern Data" company which have a quantity greater than or equal to a parameter 'minimum order size'.

1. In the 'Data Binding' dialog box, select the **Columns** tab. (See *Creating a New Table, Crosstab, or Section using a Wizard* for information on how to access the 'Data Binding' dialog box.)

2. Create the following condition by selecting the appropriate entries in the drop-down menus:

   ```
   [Customer.Company][is][equal to][Computer Tech]
   ```

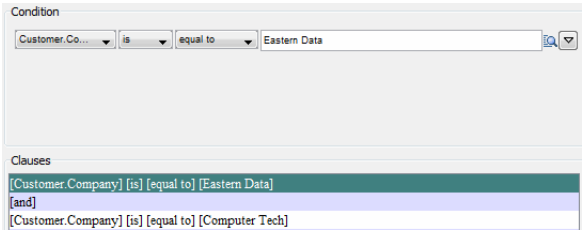
3. Click the 'Append' button. The newly defined condition is now displayed in the 'Clause' panel.



4. Enter a second condition clause as follows:

   `[Customer.Company][is][equal to][Eastern Data]`

5. Select ‘Or’ from the ‘Operator’ panel, and click ‘Insert’.



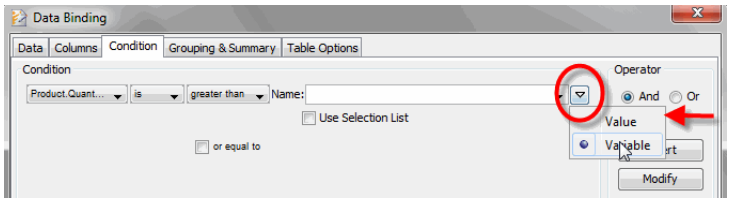
6. Enter the third condition as follows:

   a. Make the following menu choices:

      `[Product.Quantity Purchased][is][greater than]`

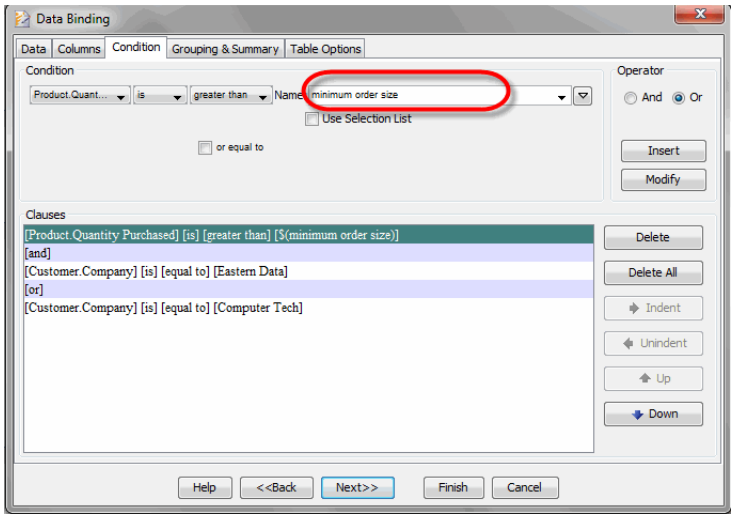
   b. Select the ‘or equal to’ checkbox.

   c. Click the ‘arrow’ button at the right side of the condition, and select ‘Variable’ from the menu.

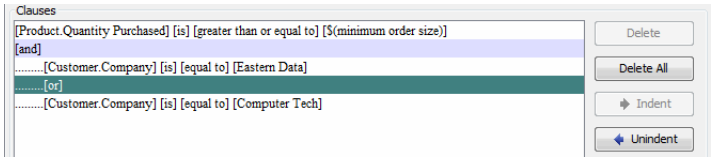


   d. In the ‘Label’ field, enter “minimum order size” (without quotes).

e. Select ‘And’ from the ‘Operator’ panel, and click ‘Insert’.



7. In the ‘Clauses’ panel, select the “`[or]`” between the two Company conditions, and click on the ‘Indent’ button to create the proper nesting.



This example is continued in the next section, *Walkthrough: Grouping and Summary in Data Binding Wizard*.

Removing or Modifying a Condition in Data Binding Wizard

To remove or modify a condition under the **Conditions** tab of the ‘Data Binding’ dialog box, select the desired clause and click the ‘Delete’ or ‘Modify’ button. The ‘Delete All’ button removes all the conditions listed in the ‘Clauses’ window. A condition can also be repeated by selecting the condition and clicking on the ‘Insert’ button. Use the ‘Indent’ and ‘Unindent’ buttons to modify nested conditions.

Reserved Parameter Names

Conditions for date-time fields can utilize several standard date values for comparison:

`_TODAY`
`_BEGINNING_OF_WEEK`
`_END_OF_WEEK`
`_BEGINNING_OF_MONTH`
`_END_OF_MONTH`

_BEGINNING_OF_QUARTER
_END_OF_QUARTER
_BEGINNING_OF_YEAR
_END_OF_YEAR

To use one of these values, select the ‘Parameter’ checkbox, and choose the desired value from the menu.

In addition, conditions can use system parameters `_USER_` and `_ROLES_` for the current user’s name and roles respectively. See *RepletParameters Object* in *Advanced Report Scripting* for the complete list.

Dynamically Dropping a Filter Condition in the Data Binding Wizard

A parameterized filter condition can be dynamically dropped by passing `null` as the parameter value. To allow a condition to be dropped this way, follow the steps below:

1. Specify the desired parameterized condition using the **Conditions** tab (see *Filtering Data in Data Binding Wizard*), and exit the ‘Data Binding’ dialog box.
2. Select the ‘Parameter’ option from the Style Studio ‘Report’ menu. This opens the ‘Parameter Definition’ dialog box.
3. In the ‘Parameter List’ panel, select the desired parameter name.
4. Click the ‘Set Selection List’ button.
5. Under the **List** tab, click the ‘Add’ button.
6. In the ‘Label’ column, enter the text “All” (without quotes) or something similar.
7. In the ‘Value’ column, enter “`null`” (without quotes).
8. Click ‘OK’ to close the ‘Selection List’ dialog box.
9. Click ‘OK’ to close the ‘Parameter Definition’ dialog box.

You can achieve the same effect by using the *onLoad* script of a report by using the following syntax.

```
parameter.{parameter name} = null;
```

For example, “`parameter.state = null`” drops any condition that uses the ‘`state`’ parameter. See *onLoad Handler* in *Advanced Report Scripting* for more details.

As described above, setting the value of a parameter (in a parametrized filter condition) to `NULL` effectively drops the filtering condition. In order to explicitly filter null values or empty string values, you should use the key words. '`NULL_VALUE`' and '`EMPTY_STRING`'.

6.2.5 Grouping and Summarization in Data Binding Wizard

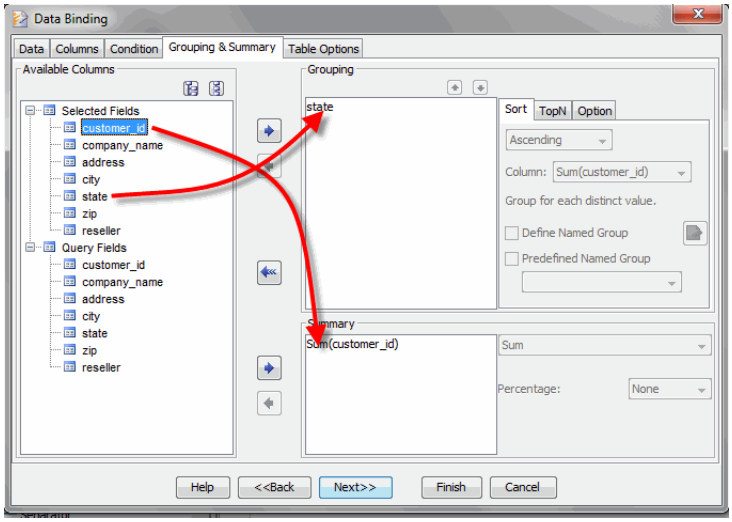
When you *group* a given field in a Table or Section, this causes rows with the same value for the field to be positioned consecutively. When you *summarize* a given field, this adds a row containing the summary or aggregate value of the field to the end of the grouped records. You can apply summarization to individual groups or to the dataset as a whole (e.g., grand total).

To add grouping and summarization to a Table or Section using the Data Binding Wizard, follow these steps:

1. In the 'Data Binding' dialog box, select the **Grouping & Summary** tab. (See *Creating a New Table, Crosstab, or Section using a Wizard* for information on how to access the 'Data Binding' dialog box.)

2. Drag the desired grouping field from the 'Available Columns' panel to the 'Grouping' panel.

3. Drag the desired summary field from the 'Available Columns' panel to the 'Summary' panel.



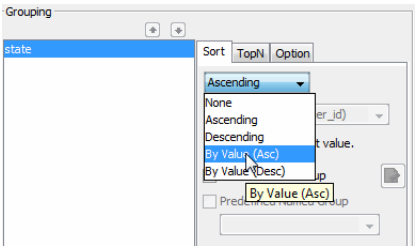
4. Repeat the above steps to add additional grouping and summary fields.

5. (Optional) To customize sort order, ranking, and granularity for a given *group*, follow the steps below:

   a. Click the desired group in the 'Grouping' panel to select it.

   b. Click the **Sort** tab in the 'Grouping' panel.

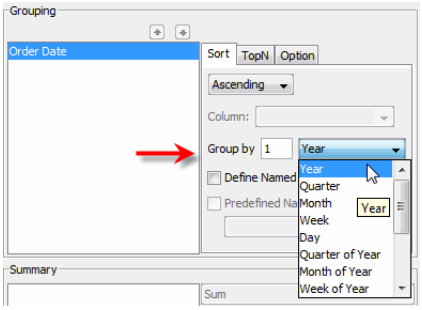
   c. From the top menu, select the desired sort order for the group.




```
The 'By Value(Asc)' and 'By Value(Desc)' options are only available
when a measure is specified in 'Summary' panel.
```

   The 'Ascending' option orders the groups alphabetically, while 'Descending' orders the groups reverse-alphabetically. The 'By Value(Asc)' and 'By Value(Desc)' options rank the groups according to the summarized measure specified by the 'Column' menu.

   d. For a Date-type field, use the 'Group by' field to specify the granularity of the date binning: day, month, year, etc. If the dates are already grouped by the database, select 'None'.



   e. To specify a coarser grouping granularity by defining more inclusive grouping categories, select 'Define Named Group', and click the 'Edit' button. See *Named Groups* for information on how to specify the groups.

   f. To use a named group that you previously created in a Data Worksheet, select 'Predefined Named Group' and choose the desired grouping from the menu. See *Creating a Named*

Grouping in *Advanced Data Mashup* for information on creating such groupings.

6. (Optional) To filter and rank a specified number from among the top or bottom groups (according to a given summarized measure) follow these steps:

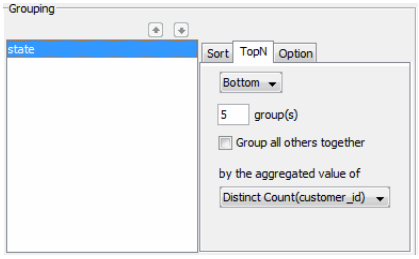
   a. Click the desired group in the 'Grouping' panel to select it.

   b. Select the **TopN** tab in the 'Grouping' panel.

   c. Select 'Top' or 'Bottom' from the menu.

   d. Choose the desired number of groups, and specify the aggregated measure that should be used to determine the ranking.

   e. Select 'Group all others together' to include as a separate category groups that are not amongst the specified top or bottom groups. Otherwise, these groups do not appear in the Table or Section.



7. (Optional) To selectively summarize certain groups and force group-based page breaks, follow these steps:

   a. Click the desired group in the 'Grouping' panel to select it.

   b. Select the **Options** tab in the 'Grouping' panel.

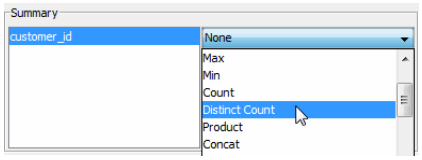
   c. Select 'Summarize Group' to display the aggregate value specified in the 'Summary' panel for the selected group. Deselect this option to suppress the summary for this group.

   d. Select 'Page Break After Group' to force a page break after the final record in the selected group (or after the summary, if one is displayed).

8. (Optional) To customize the calculation method for a given *aggregate* or *summary*, follow the steps below:

a. Click the desired measure in the ‘Summary’ panel to select it.

b. Select the desired summarization method. See *Summarization Formulas* for an explanation of the available options.



c. To automatically represent values as a percentage, make a selection in the ‘Percentage’ menu. The percentage can be computed on the basis of the parent ‘Group’ (if multiple grouping levels are specified) or on the basis of the ‘GrandTotal’.

Summarization Formulas

The following standard summarization formulas are available for use in the ‘Summary’ field of the data binding.

Table 12. Summarization Formulas

FORMULA	DESCRIPTION
Sum	Calculating the sum total of numbers.
Average	Calculating the average (mean) of numbers.
Count	Count the number of elements.
Max	Find the largest number.
Min	Find the smallest number.
Distinct Count	Count the number of distinct elements.
First	Find the first value for the measure when sorted based on the values in a second column.
Last	Find the last value for the measure when sorted based on the values in a second column.
Product	Calculate the product (multiplication) of numbers.
Standard Deviation	Calculate the standard deviation of the number series.
Variance	Variance is a measure of dispersion. The mean of the square of the deviations is called the variance
Population Variance	This is the average of squared differences between the mean and each item in the population.
Population Standard Deviation	Standard deviation is a measure of dispersion. It is the positive square root of the variance. See also variance.

Correlation	The correlation coefficient indicates the degree of linear relationship between two variables. The correlation coefficient always lies between -1 and +1. -1 indicates a perfect negative linear relationship between two variables, +1 indicates a perfect positive linear relationship, and 0 indicates a lack of any linear relationship.
Covariance	The covariance between two random variables X and Y is the expected value of the product of the variables' deviations from their means. If there is a high probability that large values of X go with large values of Y and small values of X go with small values of Y, then the covariance between X and Y will be positive; if there is a high probability that small values of X go with large values of Y and large values of X go with small values of Y, then the covariance will be negative.
Weighted Average	A weighted average is a modified version of an arithmetic mean. An average of 5 and 7 is 12/2=6, but you can count 5 twice so that a weighted average is 17/3=5.67, etc.
Median	Calculates the value that is in the middle of a list. The median of a population is the point that divides the distribution of scores in half.
pth Percentile	The pth percentile of a data set is defined as that value where p percent of the data is below that value and (1-p) percent of the data is above that value. For example, the 50th percentile is the median.
nth Largest	Returns the nth largest number in a list.
nth Smallest	Returns the nth smallest number in a list.
Mode	Returns the value that occurs most frequently.
nth Most Frequent	Returns the nth most frequent value in a list.
Concat	Concatenates values into a comma-separated list.

Walkthrough: Grouping and Summary in Data Binding Wizard

Walkthrough

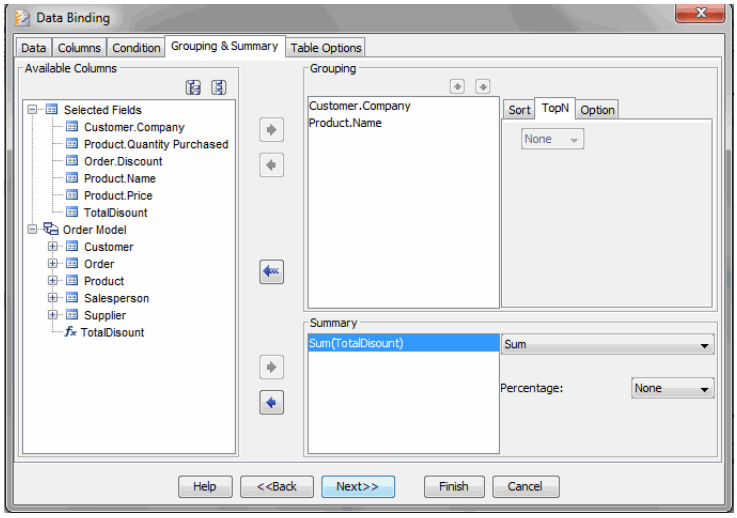
This example is continued from the previous example, *Walkthrough: Adding Conditions in Data Binding Wizard*.

1. In the ‘Data Binding’ dialog box, select the **Grouping & Summary** tab. (See *Creating a New Table, Crosstab, or Section using a Wizard* for information on how to access the ‘Data Binding’ dialog box.)

2. Drag the ‘Customer.Company’ field from the ‘Available Columns’ panel to the ‘Grouping’ panel.

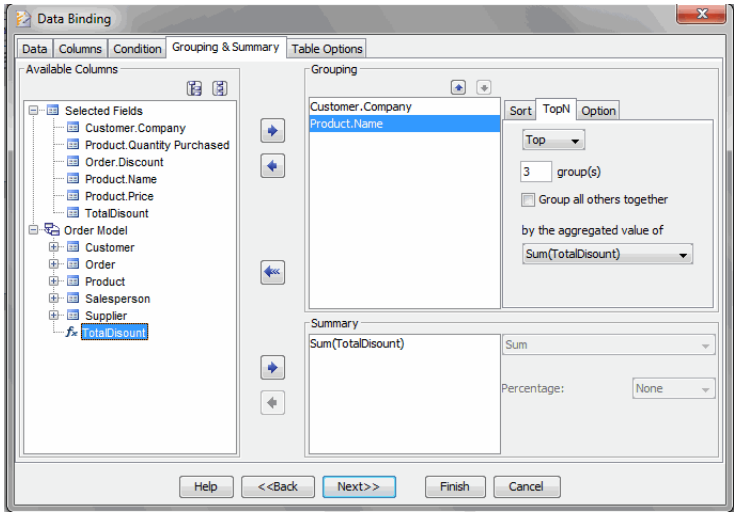
3. Drag the ‘Product.Name’ field from the ‘Available Columns’ panel to the ‘Grouping’ panel. Drop it below ‘Customer.Company’.

4. Drag the ‘TotalDiscount’ field from the ‘Available Columns’ panel to the ‘Summary’ panel.



5. In the ‘Grouping’ panel, select ‘Product.Name’ and click the **Top N** tab.

6. Select ‘Top’ from the drop-down list, enter “3” for the number of groups, and select ‘Sum(TotalDiscount)’ from the second list.



7. Click ‘Finish’ to complete the data binding.



8. Click ‘Preview’ in the Style Studio toolbar. When prompted, enter “10” for the ‘minimum order size’ filter parameter.

Notice how the data now displays only records with quantities greater than or equal to 10, *and* where the customer company was either ‘Computer Tech’ *or* ‘Eastern Data’. Additionally, the data only displays records for the top three products based on summarized total.

6.2.6 Data Binding Options

The settings in the **Options** tab of the 'Data Binding' dialog box are specific to particular report element types, and are discussed together with the corresponding report elements.

See Also
Binding Data to a Table Element, for Table options.
Binding Data to a Section Element, for Section options.

6.3 Binding Data to a Text Element

You can bind a Text/Textbox element to one of the following data sources:

```
A Text/Textbox element cannot be bound to embedded (manually
entered) data.
```

- Query
- Data model
- Data Worksheet
- Report parameter

6.3.1 Binding a Text Element to Data

Walkthrough

The following example shows how to bind a Text or TextBox element to data (the sample 'Order details' query in this example) and use the Text or TextBox to display a summarized measure. The procedure is the same for binding data models and Data Worksheets.

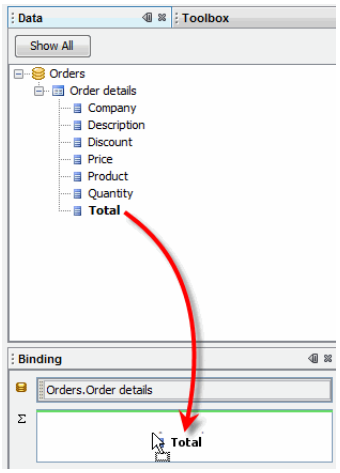
Follow the steps below:

1. Add a Text element to the report, and type some text into the element. (See *Text Element* for instructions.)

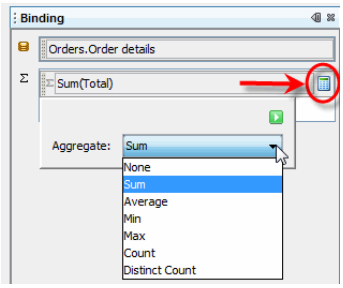
2. Click the Text element to select it. This opens the **Data Source** panel and **Binding** panel.

3. In the **Data Source** panel, expand the 'Orders' data source, the 'Sample Queries' folder, and then the 'Order details' query.

4. Drag the 'Total' column from the 'Order details' query to the 'Aggregate' region of the **Binding** panel.



5. Click the 'Edit Measure' button next to the 'Aggregate' region. This opens a pop-up panel.



6. In the pop-up panel, choose the desired aggregation formula, and press the 'Apply' button (green arrow). See *Summarization Formulas* for more information about the aggregation operators.

   In general, since a Text/TextBox element bound to data is intended to display an aggregated value, you should select an appropriate aggregation formula. If you select 'None' as the aggregation formula, the Text/TextBox element displays the value in the first column on the first row of the bound data source and no aggregation is performed.

7. (Optional) To apply a filtering condition to the Text or TextBox element, right-click the element, and select 'Condition' from the context menu. This opens the 'Condition' dialog box. The 'Condition' dialog box is the same as that for Tables. See *Filtering Data in a Table* for further details.

Note that the Text or TextBox element does not display meta-data information (unlike a Table or Section element that is bound to data). To preview the result of the Text element binding, press the 'Preview' button in the Style Studio toolbar.

6.3.2 Binding a Text Element to a Parameter

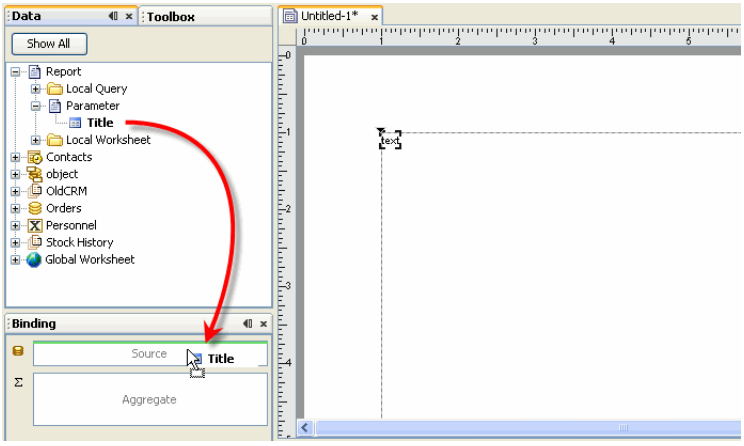
To bind a text element to a report parameter, follow these steps:

1. Define the report parameter to which the text element will be bound. (See *Defining and Editing Report Parameters* for more information.)

2. Click the Text element to select it. This opens the **Data Source** panel and **Binding** panel.

3. In the **Data Source** panel, expand the 'Report' node and then expand the 'Parameter' node.

4. From the 'Parameter' node listing, drag the desired parameter to the 'Source' region of the **Binding** panel.



The Text element will now display the value of the specified report parameter.

See Also

Defining and Editing Report Parameters, for information on report parameters.

6.4 Binding Data to a Table Element

The following sections explain how to create and bind data to a Table. In general, to create a Table follow the steps below:

1. Use the Data Binding Wizard to help you create the desired table in a step-by-step manner. See *Creating a Table Using the Data Binding Wizard*.

You can also create a Table using freehand operations. See *Creating a New Table Using Freehand Operations*.

2. (Optional) To further modify the appearance and contents of the Table, use additional freehand operations. See *Creating a Table Using Freehand Operations*.

3. (Optional) To create more complex structures with custom column and row interdependencies, convert the Table to a Formula Table. See *Formula Tables* for more details.

6.4.1 Creating a Table Using the Data Binding Wizard

When you bind a Table to a data model or a query is, the result set meta-data is displayed in the Table element. The meta-data contains the definition for each table column, including its type and name, which helps designers to specify appropriate column widths, column formats, alignments, etc.

Walkthrough: Selecting Data Source Columns using the Wizard

Walkthrough

This example creates a simple report to demonstrate how you can use the Data Binding Wizard to populate a Table with data.



1. Click the ‘New’ button in the Style Studio toolbar. This opens the ‘New Asset’ dialog box.

2. In the left panel, select ‘Report’. In the right ‘Types’ panel, select ‘Blank Tabular Report’, and click ‘OK’. This creates a new blank report.

3. Expand the Style Studio **Toolbox** panel. (Select ‘Toolbox Pane from the Style Studio ‘Window’ menu.)



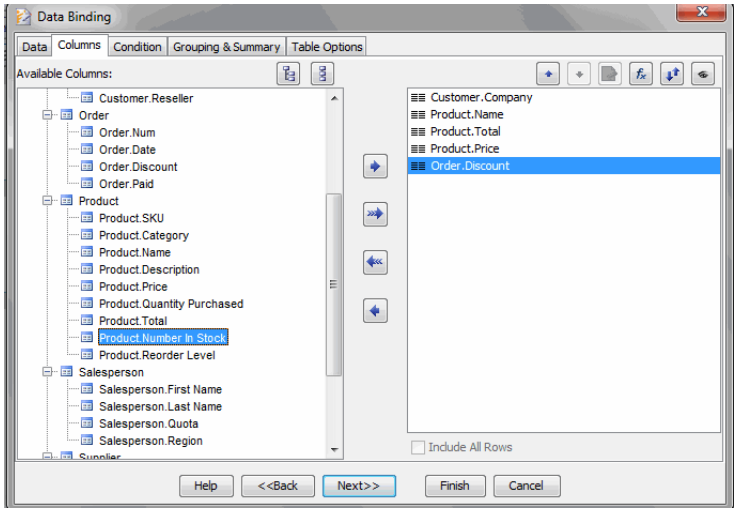
4. Click the ‘Table’ button in the **Toolbox** panel. This opens the ‘Data Binding’ dialog box for the Table.

5. Under the **Data** tab, expand the ‘Orders’ data source and select the ‘Order Model’ data model.

Note: The ‘Clear’ button can be used to remove the data binding from an element.

6. Select the **Columns** tab. Drag the following columns from the ‘Available Columns’ panel to the right panel:

   ‘Customer.Company’
   ‘Product.Name’
   ‘Product.Total’
   ‘Product.Price’
   ‘Order.Discount’



This example continues in *Walkthrough: Adding Table Grouping using the Wizard*.

Walkthrough: Adding Table Grouping using the Wizard

Walkthrough

This example continues from *Walkthrough: Selecting Data Source Columns using the Wizard*. Now you will add grouping to the table.

1. In the 'Data Binding' dialog box, select the **Grouping & Summary** tab. (See *Creating a New Table, Crosstab, or Section using a Wizard* for information on how to access the 'Data Binding' dialog box.)

To remove grouping, click the 'Clear All' button (the dashed left-arrow at the center of dialog box).

2. Drag the 'Customer.Company' field from the 'Available Columns' panel into the 'Grouping' panel.

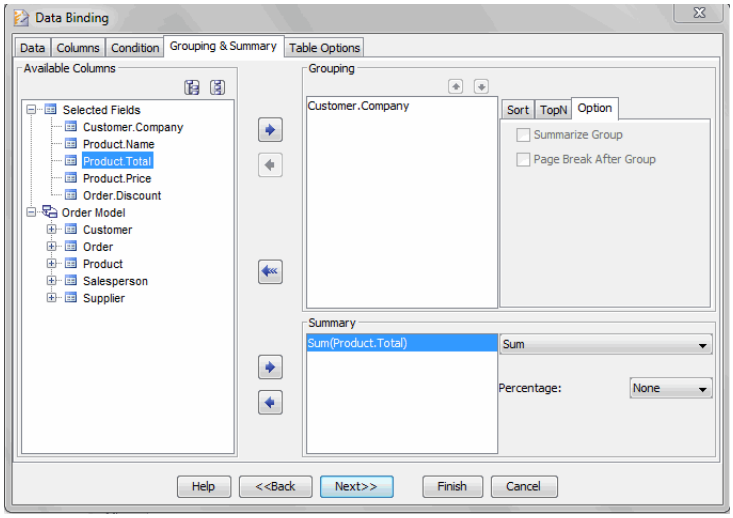
3. (Optional) Click the 'Customer.Company' field in the 'Grouping' panel to select it. Click the **Option** tab in the 'Grouping' panel.

   a. Check 'Summarize Group' to create a summary row (subtotal) after every group (if a summary measure has been defined).

   b. Check 'Page Break After Group' to force each group to appear on a separate page.

4. Drag the 'Product.Total' field from the 'Available Columns' panel into the 'Summary' panel.

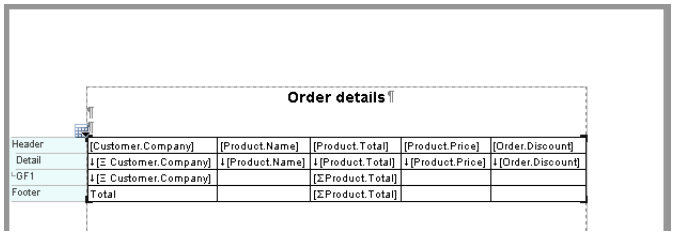
5. Click the 'Product.Total' field in the 'Summary' panel, and confirm that the selected aggregation method is 'Sum'.



6. (Optional) Click the **Options** tab in the 'Data Binding' dialog box, and specify any additonal grouping options. See *Table Grouping Options* for further details.

```
Note: After you click 'Finish', you cannot return to the 'Data
Binding' dialog box.
```

7. Click 'Finish' to close the 'Data Binding' dialog box. The table in should appear as shown:



Note that the table columns only display the table meta-data in *Design* mode.

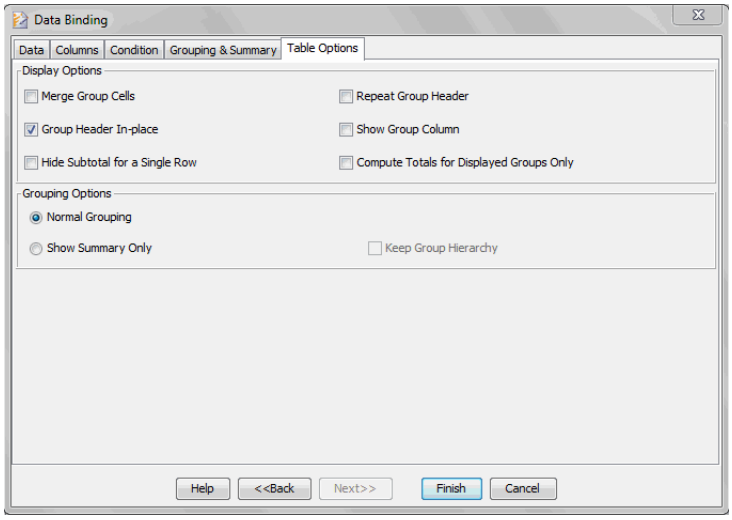


8. Click the 'Preview' button in the Style Studio toolbar to see the data displayed in the table. Close the preview.

9. Click the 'Save' button in the Style Studio toolbar. Save the report in the *templates* directory as "*sample1.srt*".

Table Grouping Options

You can set a number of different display options under the **Table Options** tab of the 'Data Binding' dialog box.



The figures below illustrate the effects of the 'Group Header in Place' and 'Show Group Columns' options.

Figure 16. Group Header In-Place

Company	Name	Total	Price	Discount
AMG Logistics	Wireless Keyboard	$800	$40.00	0%
	Fast Mail	$507	$39.00	0%
	Fancy Menus	$343	$49.00	0%
	Barbie's Fashion	$237	$79.00	0%
	Animal World	$531	$59.00	0%
	Web Bridge	$315	$105.00	0%
	Barbie's Fashion	$1,501	$79.00	0%
	NetStorage	$34,200	$2,850.00	0%
	Wireless Keyboard	$400	$40.00	0%
	MeToo AppServer	$21,000	$3,000.00	0%
	InsideView	$1,125	$125.00	0%
	Fancy Menus	$980	$49.00	0%
	Fast Go Game	$2,400	$200.00	0%
	Easy Chess	$350	$35.00	0%
	Learn ABC	$225	$75.00	0%
	Green Planet	$450	$75.00	0%
	Animal World	$767	$59.00	0%
	Xconnect Server	$4,500	$2,250.00	0%
	WebCalendar	$1,736	$248.00	0%
	Animal World	$944	$59.00	0%
	Wireless Keyboard	$520	$40.00	0%
	NetStorage	$54,150	$2,850.00	0%

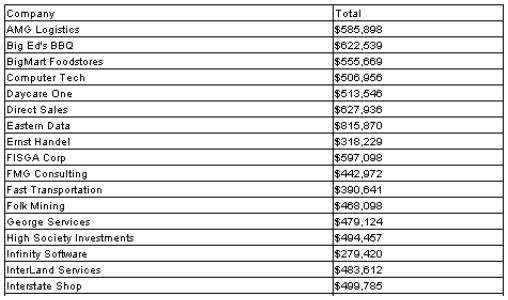
Figure 17. Show Group Column

Company	Name	Total	Price	Discount
AMG Logistics				
AMG Logistics	Wireless Keyboard	$800	$40.00	0%
AMG Logistics	Fast Mail	$507	$39.00	0%
AMG Logistics	Fancy Menus	$343	$49.00	0%
AMG Logistics	Barbie's Fashion	$237	$79.00	0%
AMG Logistics	Animal World	$531	$59.00	0%
AMG Logistics	Web Bridge	$315	$105.00	0%
AMG Logistics	Barbie's Fashion	$1,501	$79.00	0%
AMG Logistics	NetStorage	$34,200	$2,850.00	0%
AMG Logistics	Wireless Keyboard	$400	$40.00	0%
AMG Logistics	MeToo AppServer	$21,000	$3,000.00	0%
AMG Logistics	InsideView	$1,125	$125.00	0%
AMG Logistics	Fancy Menus	$980	$49.00	0%
AMG Logistics	Fast Go Game	$2,400	$200.00	0%
AMG Logistics	Easy Chess	$350	$35.00	0%
AMG Logistics	Learn ABC	$225	$75.00	0%
AMG Logistics	Green Planet	$450	$75.00	0%
AMG Logistics	Animal World	$767	$59.00	0%
AMG Logistics	Xconnect Server	$4,500	$2,250.00	0%
AMG Logistics	WebCalendar	$1,736	$248.00	0%
AMG Logistics	Animal World	$944	$59.00	0%
AMG Logistics	Wireless Keyboard	$520	$40.00	0%
AMG Logistics	NetStorage	$54,150	$2,850.00	0%

Other options are described below:

- **Repeat Group Header**: Repeats the group header at the top of each page spanned by the group.
- **Merge Group Cells**: Removes the borders between cells in the grouping column, which can make the grouping more easily visible.
- **Hide Subtotal for a Single Row**: Suppresses the summary row when there is only a single detail record.
- **Show Summary Only**: Displays only the grouping and summary rows and not the detail rows.
- **Keep Group Hierarchy**: Maintains the group hierarchy in the summary.
- **Compute Totals for Displayed Groups Only**: Computes the totals based only on the groups that are actually displayed in the table, i.e., groups selected by any Top- or Bottom-N filtering that you have specified. Disable this option to aggregate the totals based on all groups, including those groups that are not displayed in the table.

Figure 18. Show Summary Only.



Company	Total
AMG Logistics	$585,898
Big Ed's BBQ	$622,539
BigMart Foodstores	$555,669
Computer Tech	$506,956
Daycare One	$513,546
Direct Sales	$627,936
Eastern Data	$815,870
Ernst Handel	$318,229
FISGA Corp	$597,098
FMG Consulting	$442,972
Fast Transportation	$390,641
Folk Mining	$468,098
George Services	$479,124
High Society Investments	$494,457
Infinity Software	$279,420
InterLand Services	$483,612
Interstate Shop	$499,785

6.4.2 Creating a Table Using Freehand Operations

After you have created a Table with the Table Wizard (see *Creating a New Table, Crosstab, or Section using a Wizard*), you can continue to make additional modifications by using *freehand operations*. You can also create a Table using freehand operations. The following sections explain how to use freehand operations to create or modify an existing table.

For greater control of individual cell bindings and table layout, you can convert the Table to a Formula Table. See *Formula Tables* for more information.

See Also

Binding Data to a Section Element, to create more flexible layouts with arbitrary element positioning.

Creating a New Table Using Freehand Operations

In *Creating a Table Using the Data Binding Wizard*, you learned how to use the Data Binding Wizard to bind a Table to a data source. You can also create a new Table bound to a data source by using freehand operations.

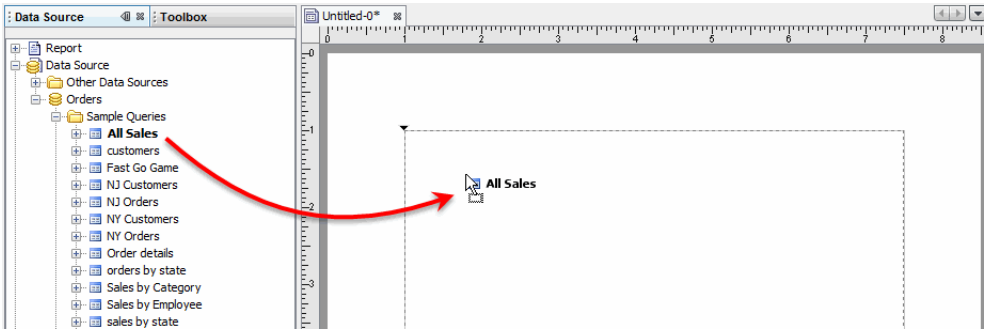
Display All Query Fields

To create a new Table that displays the results from an *entire* query, data model entity, or Data Block, follow the steps below:

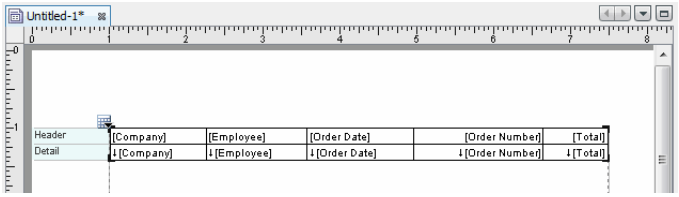
1. Click in the report to place the insertion point at the desired location.

2. Open the **Data Source** panel. (Select 'Data Source Pane' from the 'Window' menu.)

3. Drag a query, data model entity, or Data Block from the **Data Source** panel into the report.



This creates a new Table that is bound to the selected query, data model entity, or Data Block, and displays the *entire* result set.



4. Modify this table as desired using freehand operations. See *Creating a Table Using Freehand Operations*.

Display a Subset of Query Fields

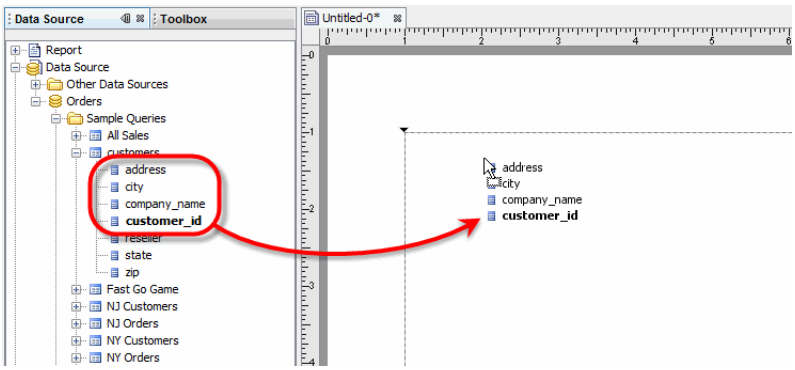
To create a new Table that displays a subset of fields from a query, data model entity, or Data Block, follow the steps below:

1. Click in the report to place the insertion point at the desired location.

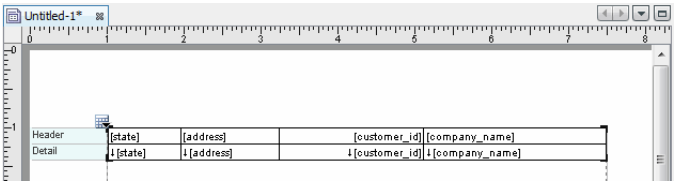
2. Open the **Data Source** panel. (Select 'Data Source Pane' from the 'Window' menu.)

3. Expand a query, data model entity, or Data Block in the **Data Source** panel.

4. Click to select a field (or Ctrl-click to select multiple fields) from the data source, and drag the field(s) into the report.



This creates a new Table that is bound to the selected data source, and displays only the selected fields from result set.



5. Modify this table as desired using freehand operations. See *Creating a Table Using Freehand Operations*.

Creating a Blank Table

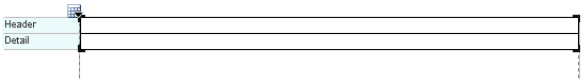
To create a blank Table, with no data fields, follow the steps below:

1. Click in the report to place the insertion point at the desired location.

2. Open the **Toolbox** panel. (Select 'Toolbox Pane' from the 'Window' menu.)

If you click (instead of drag) the 'Table' component, this opens the Data Binding Wizard. See *Creating a Table Using the Data Binding Wizard* for more details.

3. Drag the ‘Table’ component onto the report. This creates a new blank Table.



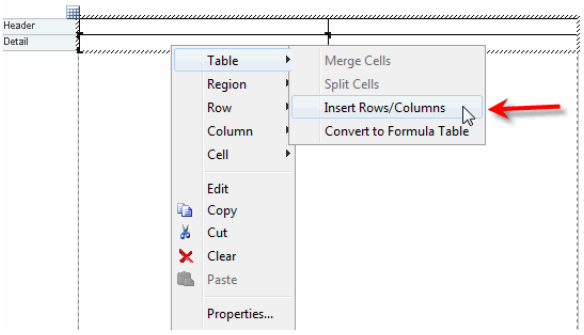
4. Modify this table as desired using freehand operations. See *Creating a Table Using Freehand Operations*.

Inserting Rows and Columns into a Table

To insert an arbitrary number of rows or columns into a table, follow the steps below:

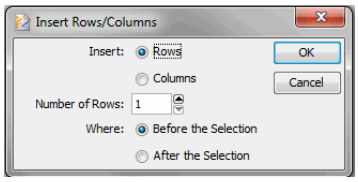
1. Click in the Table to select a cell, row, or column. (See *Selecting a Cell, Row, or Column of a Table*.)

2. Right-click and select ‘Table’ > ‘Insert Rows/Columns’ from the context menu.



This opens the ‘Insert Rows/Columns’ dialog box.

3. Select the number of rows or columns to insert, and whether the insertion should take place ‘Before the Selection’ (i.e., to the left or top of the selected cells) or ‘After the Selection’ (i.e., to the right or bottom of the selected cells).



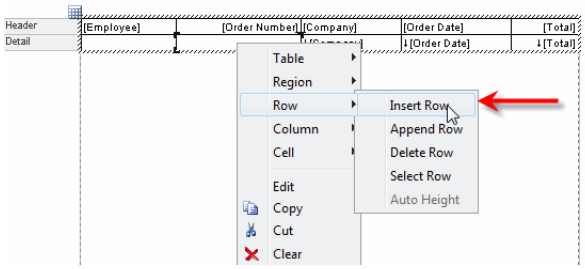
4. Click ‘OK’. This inserts the specified number of rows or columns into the table.

To insert a single row or column into a table, follow the steps below:

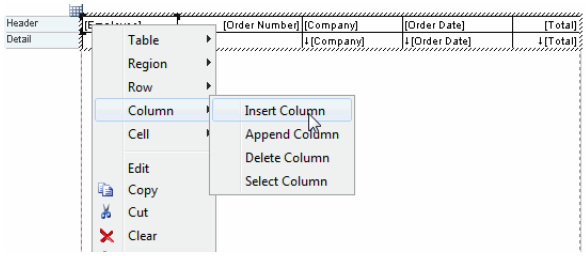
1. Click in the Table to select a cell, row, or column. (See *Selecting a Cell, Row, or Column of a Table*.)

2. Right-click and select one of the following options from the context menu:

   a. 'Row' > 'Insert Row' to add a new row above the selection, or 'Row' > 'Append Row' to add a new row below the selection.



   b. 'Column' > 'Insert Column' to add a new column to the left of the selection, or 'Column' > 'Append Column' to add a new column to the right of the selection.



See Also

Inserting New Columns with Data Binding, to add a new column together with binding.

Changing a Table Cell Data Binding, to change the data binding for a cell or column.

Deleting Rows and Columns from a Table

To delete a row or column from a Table, follow the steps below:

1. Select the rows or columns you wish to delete (or select a set of cells within those row or column). See *Selecting a Cell, Row, or Column of a Table*.

2. Right-click on the selected elements, and make one of the following choices from the context menu:

   a. 'Row' > 'Delete Row' to delete the corresponding rows.

b. ‘Column’ > ‘Delete Column’ to delete the corresponding rows, or ‘Row’ > ‘Append Row’ to add a new row below the selection.

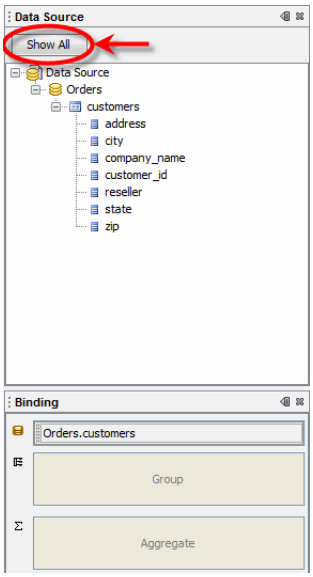
This deletes the selected elements from the table.

Inserting New Columns with Data Binding

You can insert one or more new columns that display data from fields of the data source to which the Table is bound. To add a new column or columns to a table, follow the steps below:

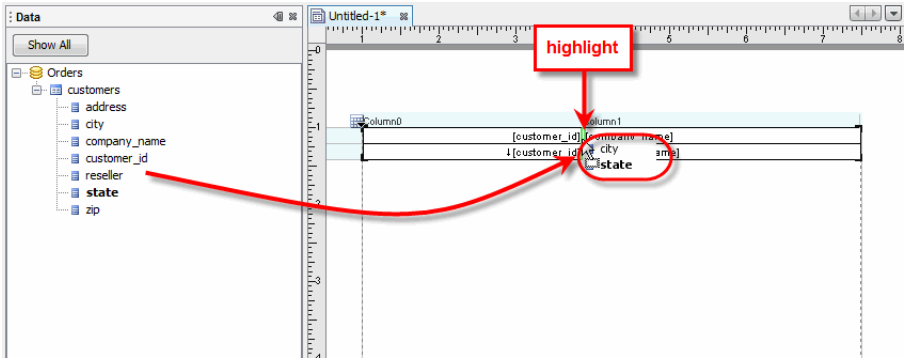
1. Click on the Table to select it. This opens the **Data Source** panel and **Binding** panel.

2. (Optional) To see only fields from the data source to which the Table is bound, disable ‘Show All’ in the **Data Source** panel.



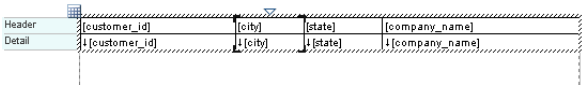
3. Expand the data source (query, data model entity, or Data Block) to which the Table is bound.

4. Click to select a field (or Ctrl-click to select multiple fields) from the data source, and drag the field(s) into the Table.



As you drag the fields over the Table, a green highlights indicates the available positions for placement.

5. Drag the fields until the green highlight appears *between* the columns where you want to place the new fields, and drop the fields at that location.



This adds the new columns with their data bindings at the specified location.

See Also

Inserting Rows and Columns into a Table, to add blank rows or columns.
Changing a Table Cell Data Binding, to change the data binding for a cell or column.

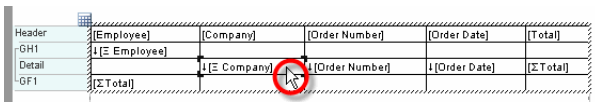
Selecting a Cell, Row, or Column of a Table

This section explains how to select various regions of a Table.

Cell in a Table

```
You cannot select a discontiguous set of cells, rows, or columns.
```

To select a cell of a table, click on the desired cell.

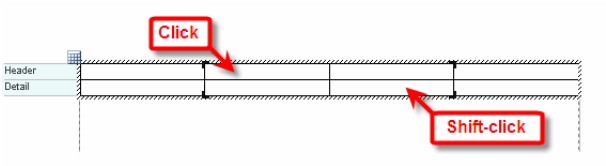


Range of Cells in a Table

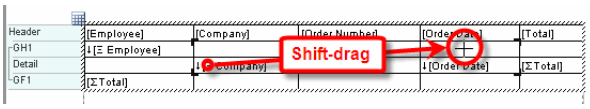
To select a range of cells in the table, do one of the following:

The selected range is indicated by square brackets.

1. Click to select a cell to start the range. Shift-click a cell to end the range. This select all cells in between the chosen cells.



2. Hold down the Shift key, and drag across the desired range of cells.

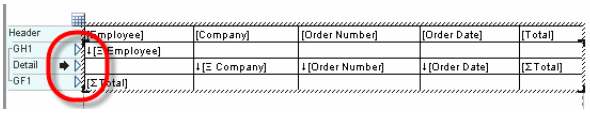


Rows in a Table

To select one or more rows of the table, do one of the following:

1. Select one or more cells in the table. Right-click the selection, and choose ‘Row’ > ‘Select Row’.

2. Move the mouse over the left margin of the table so that the pointer displays as an “arrow”. Click to select a row to start the range. Shift-click to select a row to end the range. This selects all rows between the chosen rows.



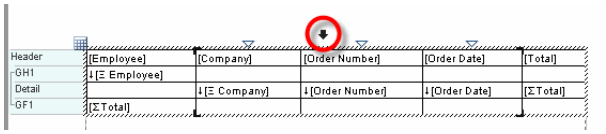
3. Move the mouse over the left margin of the table so that the pointer displays as an “arrow”. Hold down the Shift key, and drag across the desired range of rows to select them.

Columns in a Table

To select one or more columns of the table, do one of the following:

1. Select one or more cells in the table. Right-click the selection, and choose ‘Column’ > ‘Select Column’.

2. Move the mouse over the top margin of the table so that the pointer displays as an “arrow”. Click to select a column to start the range. Shift-click to select a column to end the range. This selects all columns between the chosen columns.



3. Move the mouse over the top margin of the table so that the pointer displays as an “arrow”. Hold down the Shift key, and drag across the desired range of columns to select them.

Splitting and Merging Table Cells

To create a “span cell,” i.e., a cell that spans multiple columns, you can merge multiple cells into one. Follow the steps below:

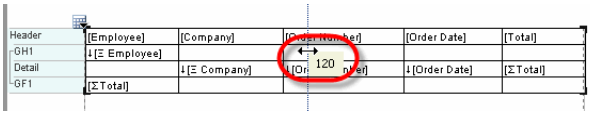
1. Select the range of cells that you wish to merge. See *Selecting a Cell, Row, or Column of a Table*.

2. Right-click in the selected region, and select ‘Table’ > ‘Merge Cells’ from the context menu. This merges the cells together into a single *span cell*.

To undo the merge operation, right-click on the *span cell*, and select ‘Table’ > ‘Split Cells’ from the context menu.

Resizing Table Rows and Columns

To manually resize rows and columns, click and drag the border between cells.



To automatically resize table columns, follow the steps below:

1. Right-click the table and select ‘Properties’ from the context menu. This opens the ‘Properties’ dialog box.

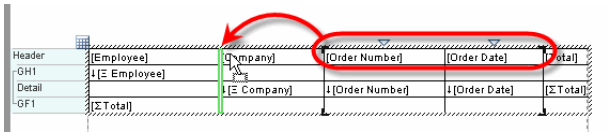
2. Select an automatic layout option from ‘Layout’ panel (under the **Table** tab). See *Table Layout* for a description of the options.

Changing the Table Column Order

To change the position of a column or range of columns, follow the steps below:

1. Select the column or range of columns you wish to move. See *Selecting a Cell, Row, or Column of a Table*.

2. Drag the columns to the desired position in the table. A green highlight indicates the new placement.



Changing a Table Cell Data Binding

You can alter the binding of a table cell to display static text, values from a different field of data, or values computed by a formula. To modify a cell binding, follow the steps below:

1. Click the desired cell in the table to select it.

1. Click the ‘Properties’ link at the bottom of the Style Studio window. This expands the ‘Properties’ panel.

2. Click the **Data** tab. In the ‘Binding’ panel, make one of the following selections:

   a. To enter static text in the cell, select the ‘Text’ option, and entered the desired contents.

   b. To bind the cell to a different data field, select the ‘Column’ option, and choose the desired data field from the adjacent menu.

   c. To enter an expression, select ‘Formula’, and click the ‘Edit’ button. This opens the Script Editor, which allows you to create and save the desired formula. See *Formula Tables* for more details about formulas.

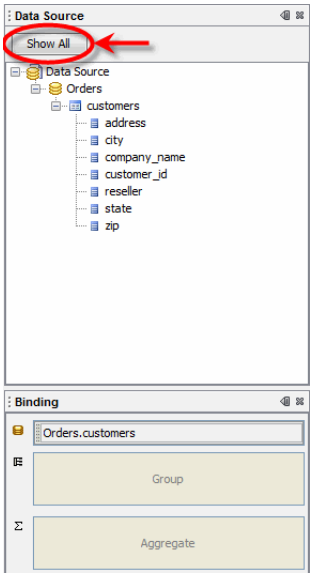
3. Click the ‘Apply’ button to submit the changes.

Changing the Table Data Source

To change the data source to which the table is bound, follow the steps below:

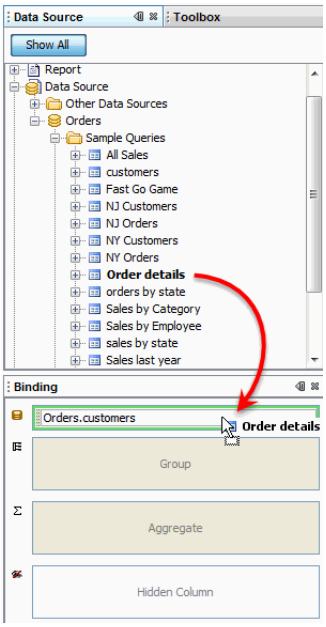
1. Click on the Table to select it. This opens the **Data Source** panel and **Binding** panel.

2. To see all available data sources, enable ‘Show All’ in the **Data Source** panel.



You can also drag fields from the new data source onto the table itself.

3. Drag the desired data source (query, data model entity, or Data Block) from the **Data Source** panel onto the top field in the **Binding** panel.



This operation will generate a warning, which advises you that changing the binding will destroy the current table configuration. Click 'Yes' to proceed.

This rebinds the table to the new data source.

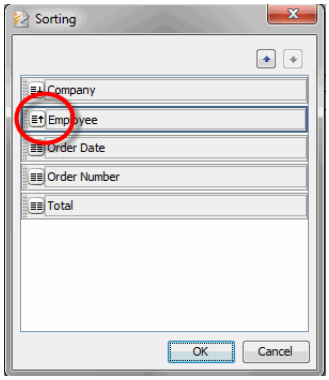
Changing the Column Sort Order

To change the sorting order for one or more columns in a table, follow the steps below:

1. Right-click the table, and select ‘Sorting’ from the context menu. This opens the ‘Sorting’ dialog box.

Grouping fields are not listed. To sort a grouping field, see *Grouping Data in a Table*.

2. In the ‘Sorting’ dialog box, click to the left of the field you wish to sort. A small arrow indicates the prevailing sort. Click once to set ascending (alphabetical) order, click a second time to set descending (reverse alphabetical) order, and click a third time to remove the sorting.



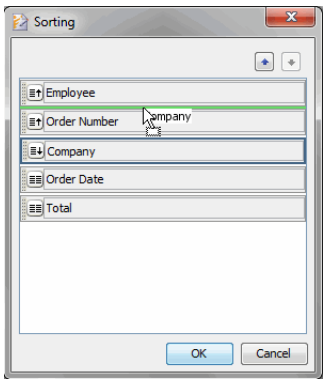
3. (Optional) Repeat the above step to simultaneously sort additional fields.

   Sorting on multiple fields creates a nested sorting. The position of a given field in the sorting hierarchy is defined by its position in the ‘Sorting’ dialog box. For example, consider a sort specified for both a ‘State’ field and ‘City’ field. If the ‘State’ field appears above the ‘City’ field in the dialog box, then the records are sorted primarily by ‘State’, and are sorted secondarily (i.e., within each state) by ‘City’.

4. (Optional) To change the nesting order for multiple-field sorting, change the order of the fields in the ‘Sorting’ dialog box. Use one of the following approaches:

   a. Click in the ‘Sorting’ dialog box to select a sorted field. Then click the “arrow” buttons at the top of the dialog box to change the field’s relative position.

b. Click on the field name, and drag the field to the desired position.



5. Click ‘OK’ to close the dialog box.

Filtering Data in a Table

To apply a filtering condition to a table, follow the steps below.

1. Right-click the table, and select ‘Condition’ from the context menu. This opens the ‘Condition’ dialog box.

2. Use the menus to enter the desired condition clause(s). See *Filtering Data in Data Binding Wizard* for information about the controls, and an example of constructing a condition.

3. Click ‘OK’ to close the dialog box, and apply the condition.

See Also
Parameterizing a Condition, for details on using a variable in a condition.

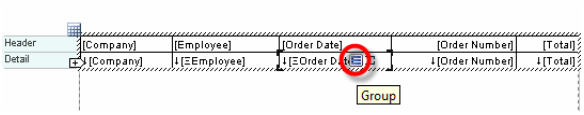
Grouping Data in a Table

There are two ways to group data in a table. You can keep the grouping field in the same row as the detail records, or you can display the grouping field in its own group region. The following sections explain the two approaches.

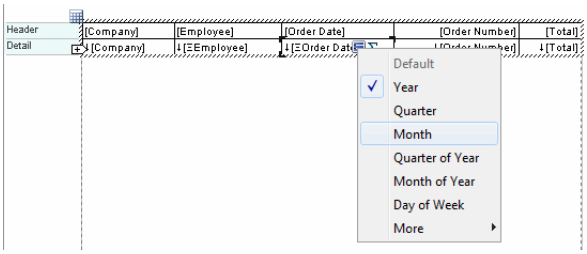
Displaying the Grouping Field in the Detail Row

To group the data in a table and display the grouping field(s) in the detail row, follow the steps below:

1. Press the ‘Group’ button in the desired grouping cell(s).



2. For a Date-type field, select a date level from the pop-up menu, such as ‘Year’ or ‘Month’.



This defines the field as a grouping field, and adds the selected field to the grouping area of the **Binding** panel. See *Modifying Table Grouping Properties* to make further adjustments to the grouping process.

To remove grouping from a field, press the ‘Group’ button a second time. For a Date-type field, after pressing the ‘Group’ button, uncheck the selected level in the pop-up menu.

Displaying the Grouping Field in a Group Region

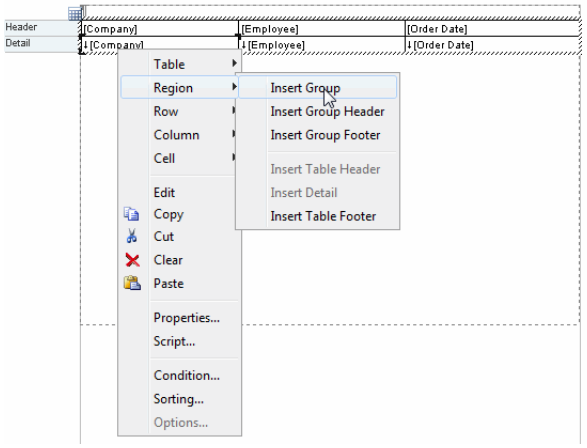
To group the data in a table and display the grouping field in a dedicated grouping region, follow the steps below:

For a categorical field, select ‘Default’. For a Date field, select a Date level such as ‘Year’ or ‘Month’ (if the dates are not already grouped by the database).

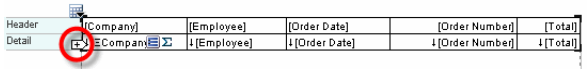
1. Right-click on the detail row, and select one of the following:

   a. ‘Region’ > ‘Insert Group’ to insert a new grouping region with both header and footer.

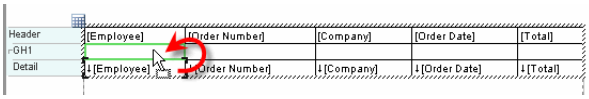
   b. ‘Region’ > ‘Insert Group Header’ to insert a new grouping region with only header.



Note: You can also insert a region by pressing the '+' button at the left side of the row.



2. Drag the desired grouping field from the detail row to a cell in the new grouping header region.

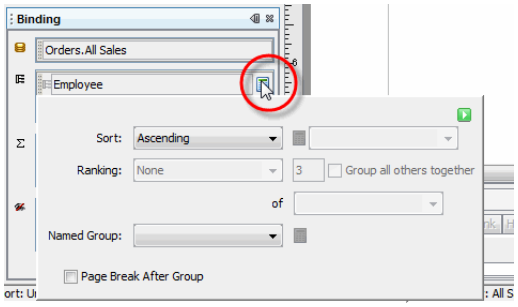


This defines the field as a grouping field in its own region, and adds the selected field to the grouping area of the **Binding** panel. See *Modifying Table Grouping Properties* to make further adjustments to the grouping process.

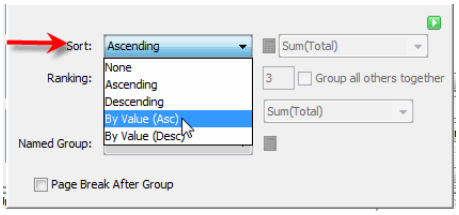
Modifying Table Grouping Properties

To customize sort order, ranking, and granularity for a given *group*, follow the steps below:

1. Click the 'Edit Dimension' button in the **Binding** panel for the desired group.



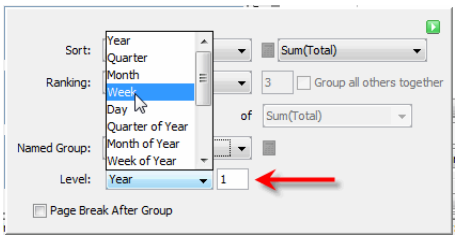
2. From the top menu, select the desired sort order for the group.



`The 'By Value(Asc)' and 'By Value(Desc)' options are only available when a measure is specified in 'Summary' panel.`

The 'Ascending' option orders the groups alphabetically, while 'Descending' orders the groups reverse-alphabetically. The 'By Value(Asc)' and 'By Value(Desc)' options rank the groups according to the summarized measure specified by the 'Column' menu.

3. For a date-time field, use the 'Level' field to specify the granularity of the date binning: day, month, year, etc.



4. To specify a coarser grouping granularity by defining more inclusive grouping categories, select the 'Customize' option in the 'Named Groups' menu, and click the 'Edit' button. See *Named Groups* for information on how to specify the groups.

5. To use a named group that you previously created in a Data Worksheet, select the desired group from the 'Named Groups' menu. See *Creating a Named Grouping* in *Advanced Data Mashup* for information on creating such groupings.

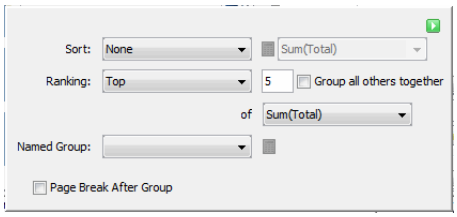
6. To filter and rank a specified number from among the top or bottom groups (according to a given summarized measure) follow these steps:

   a. Select ‘Top’ or ‘Bottom’ from the ‘Ranking’ menu.

   b. Choose the desired number of groups, and specify the aggregated measure that should be used to determine the ranking.

   c. Select ‘Group all others together’ to include as a separate category groups that are not amongst the specified top or bottom groups. Otherwise, these groups do not appear in the Table or Section.

When you specify multiple grouping fields, ‘Group all others together’ is available only for the lowest grouping level.

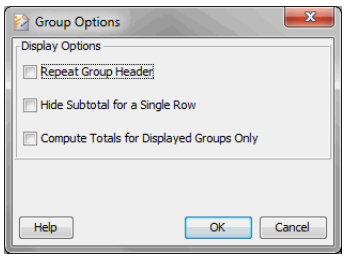


7. Select ‘Page Break After Group’ to force a page break after the final record in the selected group (or after the summary, if one is displayed).

8. Click the ‘Apply’ button (green arrow) to submit the changes.

Setting Global Grouping Options

To set global grouping options (which apply to all groups), right-click on the Table, and select ‘Options’ from the context menu. This opens the ‘Group Options’ dialog box.



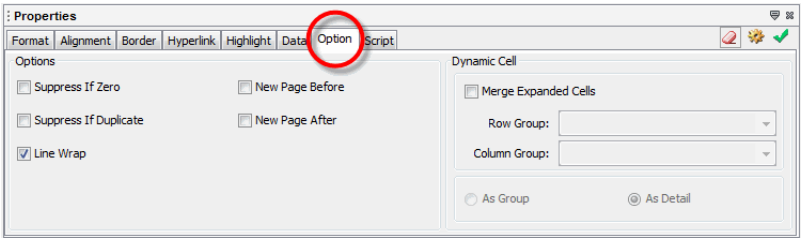
The options are as follows:

- **Repeat Group Header**: Repeat the group name at the top of each page on which the group’s records are displayed.

- **Hide Subtotal for a Single Row**: Suppress display of a group's aggregate(s) when the group contains only one detail record.

Setting Group Cell Display Options

You can set additional options for grouped table cells in the **Options** tab of the bottom **Properties** panel.



The available options are as follows:

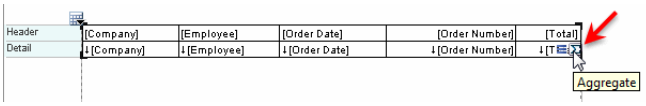
- **Suppress If Zero**: Suppress the display of a zero value, and instead display an empty cell.
- **Suppress If Duplicate**: Suppress the display of duplicate rows, and instead display just a single representative row.
- **Line Wrap**: Allow text to wrap in cell.
- **New Page Before**: Insert a page break before the cell.
- **New Page After**: Insert a page break after the cell.
- **Merge Expanded Cells**: Merge automatically expanded cells into a single cell. This is a useful way to simplify the table when expanded cells all contain the same text. The 'Row Group' and 'Column Group' properties determine which row and column grouping the merging should follow. See *Spanning Cells* for an example.
- **As Group**: Specifies that a grouped cell that appears in the 'Detail' row displays the group value only once.
- **As Detail**: Specifies that a grouped cell that appears in the 'Detail' row displays the group value once for each detail record.

Summarizing Data in a Table

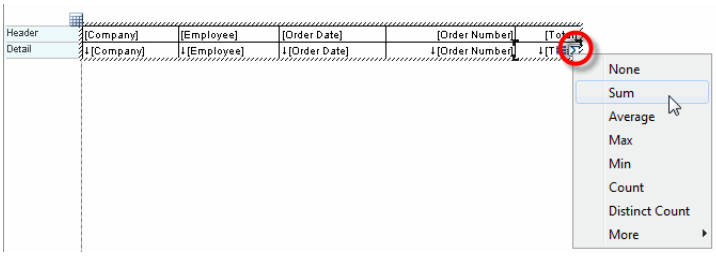
When you aggregate data in a table, you can compute summaries on a group-by-group basis and/or on a global basis (e.g., grand total).

To create a summary based on a particular grouping level, follow the steps below:

1. Press the 'Aggregate' button in the desired cell(s) that you want to summarize.



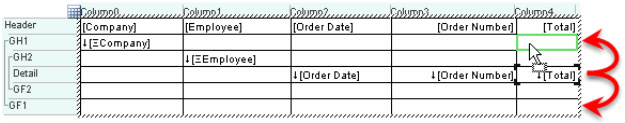
2. Choose the desired aggregation method (e.g., ‘Average’, ‘Max’, etc.)



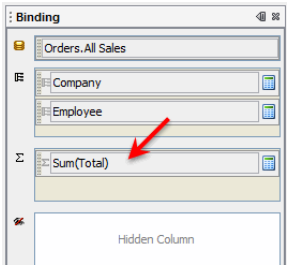
This defines the field as a summary field. Because the field resides in the detail row, it only summarizes a single detail value. To summarize a particular group, or the entire table, proceed to the next steps.

To add a grouping region, see *Displaying the Grouping Field in a Group Region*.

3. To summarize a particular group, Ctrl-drag (or copy and paste) the summary field to the corresponding grouping header or footer region.

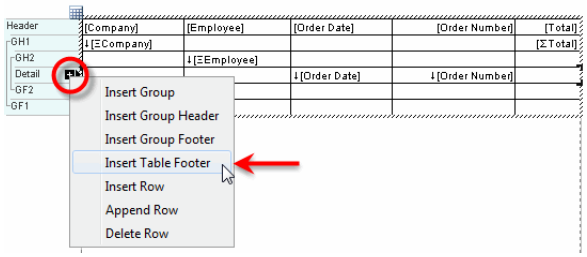


This adds the selected field to the summary area of the **Binding** panel.

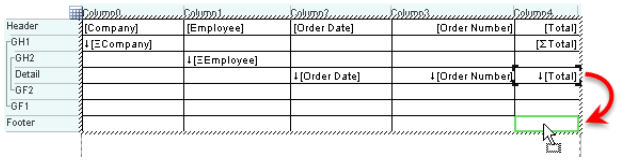


4. To summarize all records in the table, follow the steps below:

   a. Click in the left margin next to any row to select the row. Press the ‘+’ icon in the margin. Select ‘Insert Table Footer’ from the context menu.



b. Ctrl-drag (or copy and paste) the summary field to the table footer region.



This adds the selected field to the summary area of the **Binding** panel.

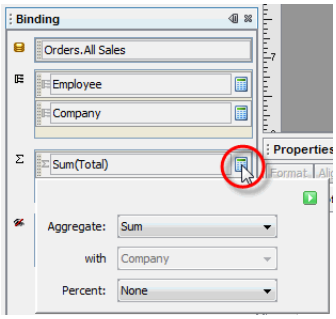
See *Modifying Summary Properties* to make further adjustments to the summarization process.

To remove summarization from a field, press the 'Aggregate' button a second time, and uncheck the selected aggregation in the pop-up menu.

Modifying Summary Properties

To customize the calculation method for a given *aggregate* or *summary*, follow the steps below:

1. Click the 'Edit Measure' button in the **Binding** panel for the desired summary measure.

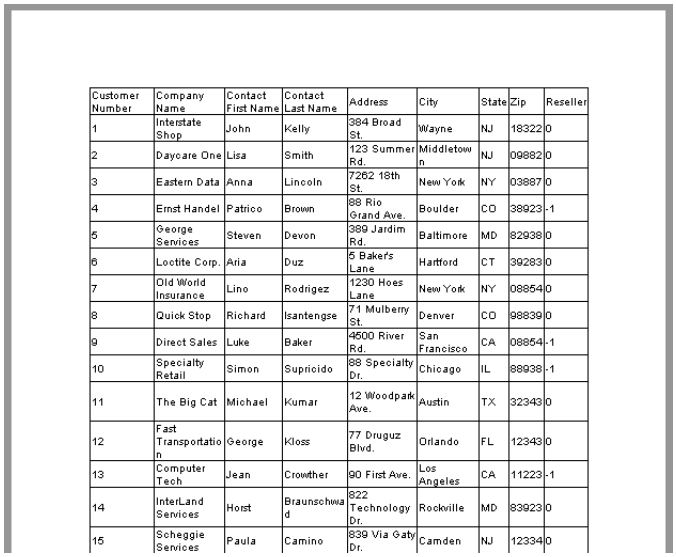


2. Select the desired summarization method. See *Summarization Formulas* for an explanation of the available options.

3. To automatically represent values as a percentage, make a selection in the ‘Percentage’ menu. The percentage can be computed on the basis of the parent ‘Group’ (if multiple grouping levels are specified) or on the basis of the ‘GrandTotal’.

Walkthrough: Freehand Operations on a Table

The following sample report illustrates these features. Consider the ‘Customers’ entity from the ‘Order Model’ in the ‘Orders’ data source. When you create the report with this data model bound to a rigid table structure, the output is shown below.



Customer Number	Company Name	Contact First Name	Contact Last Name	Address	City	State	Zip	Reseller
1	Interstate Shop	John	Kelly	384 Broad St.	Wayne	NJ	18322	0
2	Daycare One	Lisa	Smith	123 Summer Rd.	Middletown	NJ	09882	0
3	Eastern Data	Anna	Lincoln	7262 18th St.	New York	NY	03887	0
4	Ernst Handel	Patrico	Brown	88 Rio Grand Ave.	Boulder	CO	38923	-1
5	George Services	Steven	Devon	389 Jardim Rd.	Baltimore	MD	82938	0
6	Loctite Corp.	Aria	Duz	5 Baker's Lane	Hartford	CT	39283	0
7	Old World Insurance	Lino	Rodrigez	1230 Hoes Lane	New York	NY	08854	0
8	Quick Stop	Richard	Isantengse	71 Mulberry St.	Denver	CO	98839	0
9	Direct Sales	Luke	Baker	4500 River Rd.	San Francisco	CA	08854	-1
10	Specialty Retail	Simon	Supricido	88 Specialty Dr.	Chicago	IL	88938	-1
11	The Big Cat	Michael	Kumar	12 Woodpark Ave.	Austin	TX	32343	0
12	Fast Transportation	George	Kloss	77 Druguz Blvd.	Orlando	FL	12343	0
13	Computer Tech	Jean	Crowther	90 First Ave.	Los Angeles	CA	11223	-1
14	InterLand Services	Horst	Braunschwad	822 Technology Dr.	Rockville	MD	83923	0
15	Scheggie Services	Paula	Camino	839 Via Gaty Dr.	Camden	NJ	12334	0

Walkthrough

Consider a case where it is necessary to move the cell elements so that the entire address is grouped together line-by-line, one below the other. You can create this table using freehand table operations.

Follow the steps below:

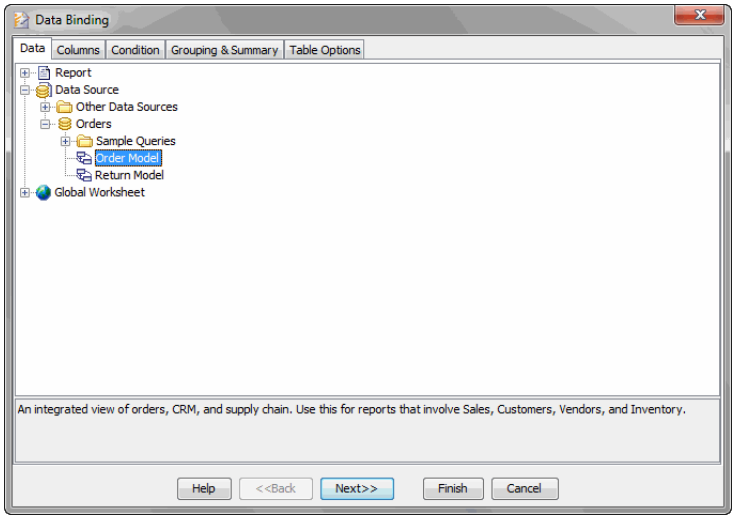
1. Click the ‘New’ button in the Style Studio toolbar. This opens the ‘New Asset’ dialog box.

2. Select ‘Report’ from the left panel, then select ‘Blank Tabular Report’ from the ‘Types’ panel, and click ‘OK’.



3. Click the ‘Table’ button in the **Toolbox** panel to open the ‘Data Binding’ dialog box for a new table.

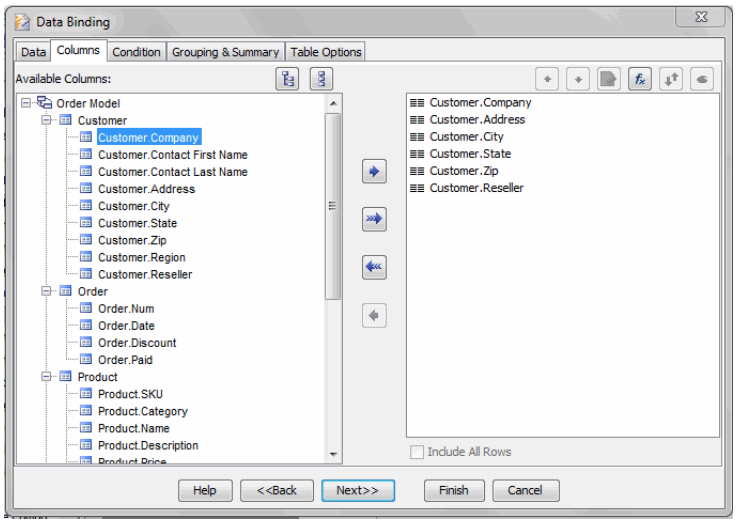
4. Under the **Data** tab, expand the ‘Orders’ data source, and select the ‘Order Model’ data model.



5. Under the **Columns** tab, select the 'Customer' node. Click the right-arrow button to add all the attributes from the 'Customer' entity.

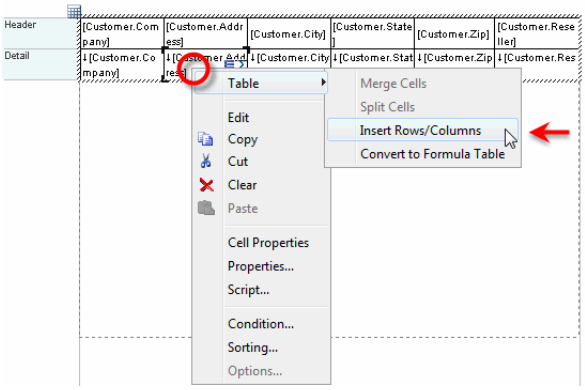
6. Remove the following attributes by using the left-arrow button:

   'Customer.Contact First Name'
   'Customer.Contact Last Name'
   'Customer.Region'

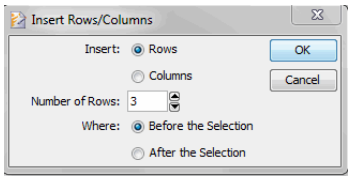


7. Click 'Finish'. This adds the new table to the report.

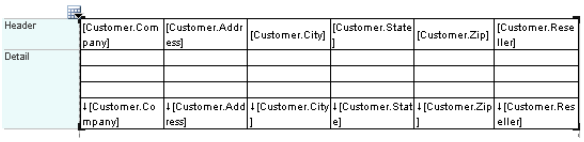
8. Select any cell in the detail row. Right-click, and select the 'Table' > 'Insert Rows/Columns' option from the context menu. This opens the 'Insert Rows/Columns' dialog box.



9. Specify that three rows should be inserted before the selection (the detail row).

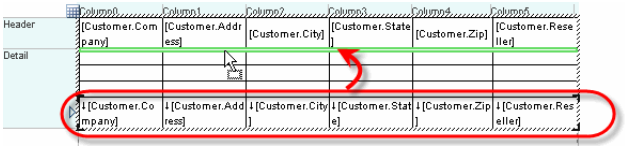


10. Press ‘OK’ to add the new rows.

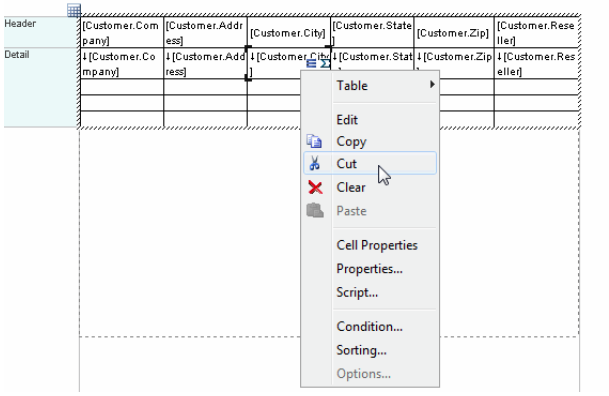


11. Click in the left margin next to the detail row to select the entire row.

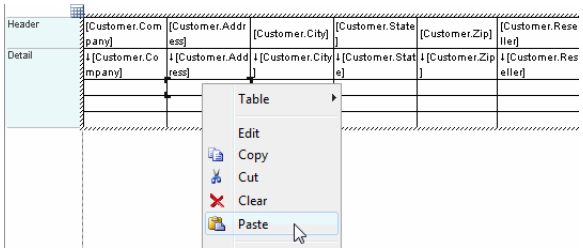
12. Click and drag the detail row to move the detail row up three rows (right below the header row).



13. Select the `[Customer.City]` cell, right-click the cell, and select ‘Cut’ from the context menu.



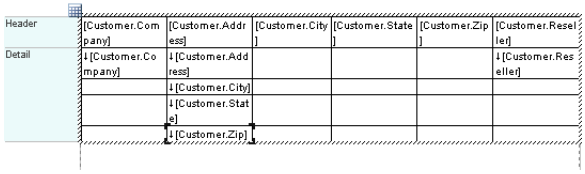
14. Right-click the empty cell below the `[Customer.Address]` cell, and select ‘Paste’ from the context menu.



This pastes the `[Customer.City]` field into that table location.

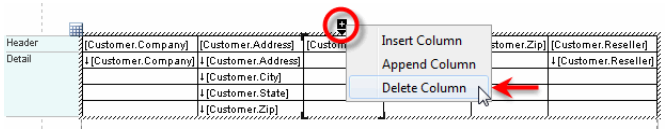
Tip: You can use the Ctrl-X and Ctrl-V keyboard shortcuts, or simply drag the fields to the desired location.

15. Repeat the above steps to cut and paste the `[Customer.State]` and the `[Customer.Zip]` fields into the empty cells below the `[Customer.City]` field.

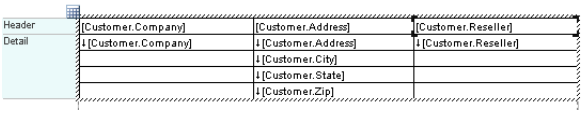


Tip: You can also click above the column to select the whole column, and then press ‘Delete’ on the keyboard.

16. Click above the ‘Customer.City’ column. Press the ‘+’ icon and select ‘Delete Column’ from the context menu.



17. Repeat the above step to delete the ‘Customer.State’ and ‘Customer.Zip’ columns.



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

19. Click the ‘Select Style’ button to open the ‘Style Viewer’ dialog box.

20. Select the ‘Blue’ > ‘Dark Header Borders’ style. Press ‘OK’ to close the ‘Style Viewer’ dialog box.

21. Press ‘OK’ to close the ‘Table Properties’ dialog box.

[Customer.Company]	[Customer.Address]	[Customer.Reseller]
↓[Customer.Company]	↓[Customer.Address]	↓[Customer.Reseller]
	↓[Customer.City]	
	↓[Customer.State]	
	↓[Customer.Zip]	

22. Save the report.



23. Preview the report and observe the table with data arranged using Freehand layout.

Company	Address	Reseller
Interstate Shop	384 Broad St.	0
	Wayne	
	NJ	
	18322	
Daycare One	123 Summer Rd.	0
	Middletown	
	NJ	
	09882	
Eastern Data	7262 18th St.	0
	New York	
	NY	
	03887	
Ernst Handel	88 Rio Grand Ave.	1
	Boulder	
	CO	
	38923	
George Services	389 Jardim Rd.	0
	Baltimore	
	MD	
	82938	

You could create a similar layout by using a *Section Element* (pseudo-table), but this would not allow you to make use of the table grid and styles.

Walkthrough: Defining Formulas

Walkthrough

The Freehand Table Layout allows you to define a formula within a cell, similar to a spreadsheet application. The example below demonstrates this feature:

1. Click the ‘New’ button in the Style Studio toolbar. This opens the ‘New Asset’ dialog box.

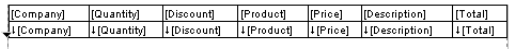
2. In the left panel, select the ‘Report’ node. From the right ‘Types’ panel, select ‘Blank Tabular Report’, and click ‘OK’

3. Expand the Style Studio **Toolbox** panel.

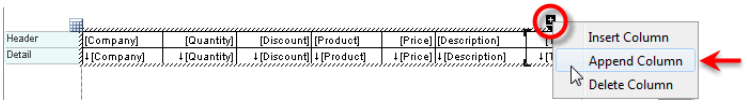


4. Click the ‘Table’ button in the **Toolbox** panel to open the ‘Data Binding’ dialog box for a new table.

5. Under the **Data** tab, bind the table to the ‘Order details’ query in the ‘Sample Queries’ folder of the ‘Orders’ data source. Under the **Columns** tab, select all of the columns. Click ‘Finish’ to perform the binding.

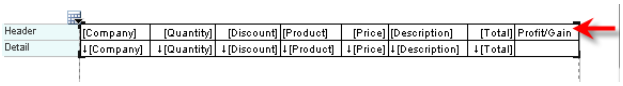


6. Select the ‘Total’ column in the table and click above the column. Press the ‘+’ icon above the column and select ‘Append Column’ from the context menu. This adds an additional column.



You will use this new column to calculate the profit/gain to a customer who has obtained a discount.

7. Select the header cell of the new column and type “Profit/Gain”.



8. Expand the Style Studio **Properties** panel.

9. Select the detail cell of the new column.

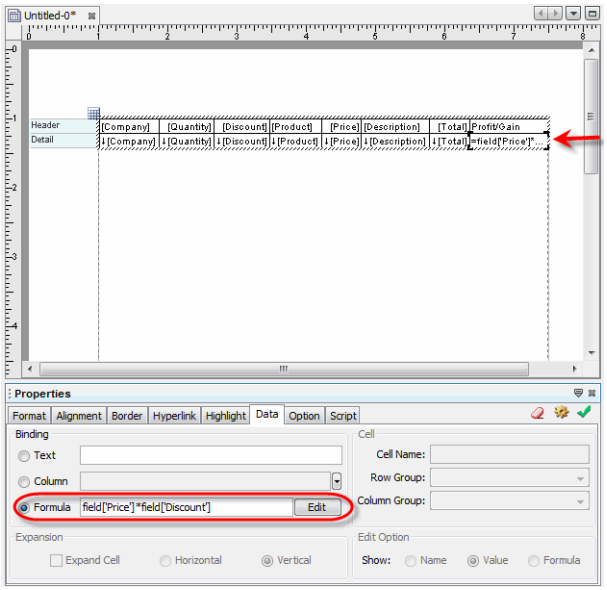
10. In the **Properties** panel, click the **Data** tab. Select the ‘Formula’ option in the ‘Binding’ panel, and click on the ‘Edit’ button to open the Script Editor.

11. In the Script Editor, enter the following formula:

```
field['Price']*field['Discount']
```



12. Click on the ‘Save and Close’ button, then click the ‘Apply’ button in the **Data** tab.



13. Preview the report to observe the effect. (Page forward in the report to see the non-zero values of 'Profit/Gain'.)

Company	Quantity	Discount	Product	Price	Description	Total	Profit/Gain
Interstate Shop	6	0	NetStorage	2850	500 GB network storage	17100	0
Interstate Shop	5	0	Animal World	59	Educational software	295	0
Interstate Shop	1	0	Mega Icons	550	Over 10,000 icons	550	0
Interstate Shop	16	0	WebCalendar	248	Web enabled calendar/org	3968	0
Interstate Shop	18	0	19 inch LCD	310	19 inch LCD monitor	5580	0
Interstate Shop	18	0	Sync Me	145	Palm power sync	2610	0
Interstate Shop	15	0	Math for Me	235	Math tutor	3525	0
Interstate Shop	16	0	Wireless Mouse	40	Wireless optical USB mouse	640	0
Interstate Shop	1	0	NetStorage	2850	500 GB network storage	2850	0
Interstate Shop	13	0	Mega Icons	550	Over 10,000 icons	7150	0
Interstate Shop	2	0	17 Inch LCD	210	17 inch LCD monitor	420	0
Interstate Shop	12	0	Green Planet	75	Personal health monitor	900	0
Interstate Shop	1	0	MeToo AppServer	3000	Application server	3000	0
Interstate Shop	7	0	Barbie's Fashion	79	Girl's game	553	0
Interstate Shop	11	0	Fast Mail	39	Lightweight email agent	429	0
Interstate Shop	3	0	Easy Chess	35	Chess game	105	0
Interstate Shop	6	0	Web Bridge	105	Bridge game	630	0
Interstate Shop	18	0	17 Inch LCD	210	17 inch LCD monitor	3780	0
Interstate Shop	16	0	Xconnect Server	2250	XML connectivity tool	36000	0
Interstate Shop	13	0	Fancy Menus	49	Collection of menus	637	0

14. Save the report.

Binding a Table to Embedded Data

In some cases you may want a Table, Chart, Crosstab, or Section to display a small dataset that you enter manually. To do this, follow the steps below:

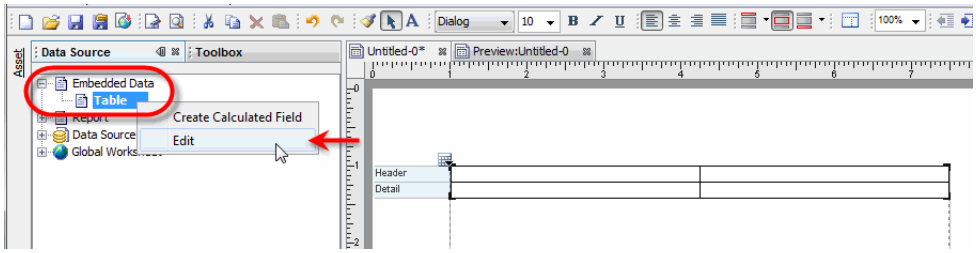
1. Create a new Table, Chart, Crosstab, or Section, as described below:

    a. Expand the Style Studio **Toolbox** panel.

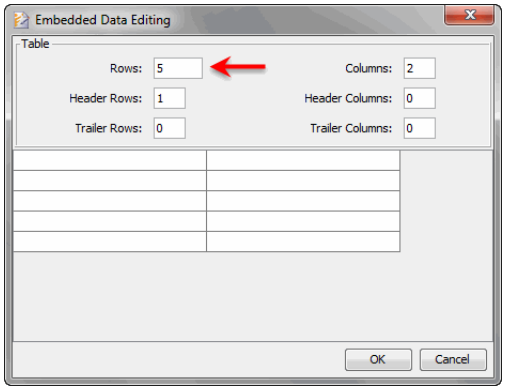
b. Drag a Table, Chart, Crosstab, or Section element onto the report page.

2. Click on the Table, Chart, Crosstab, or Section to select it.

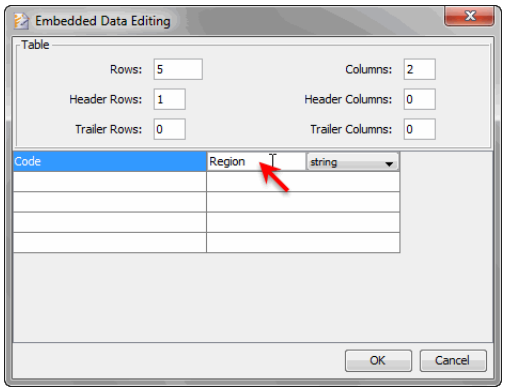
3. In the **Data Source** panel, expand the 'Embedded Data' node, right-click on the 'Table' entry, and select 'Edit' from the context menu. This opens the 'Embedded Data Editing' dialog box.



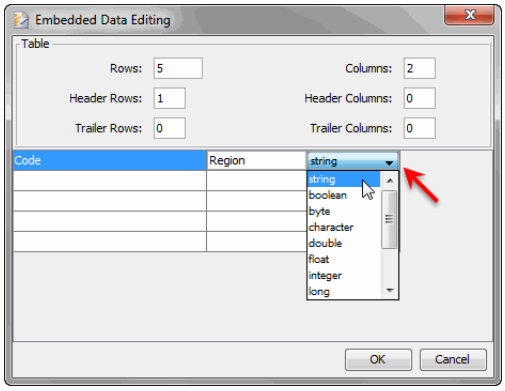
4. In the provided fields, specify the number of rows and columns for the table.



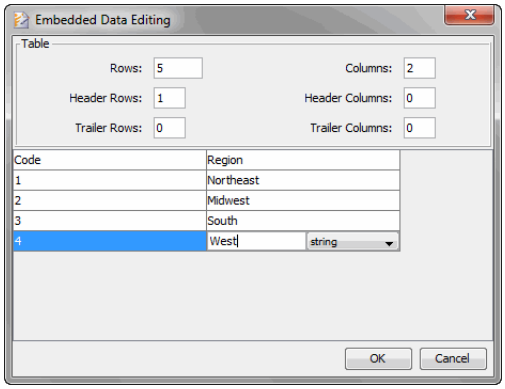
5. Click in a cell of the table, and type the desired data into the cell.



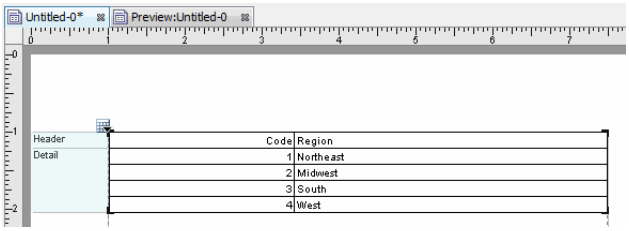
6. From the provided menu, select the data type of the entered text.



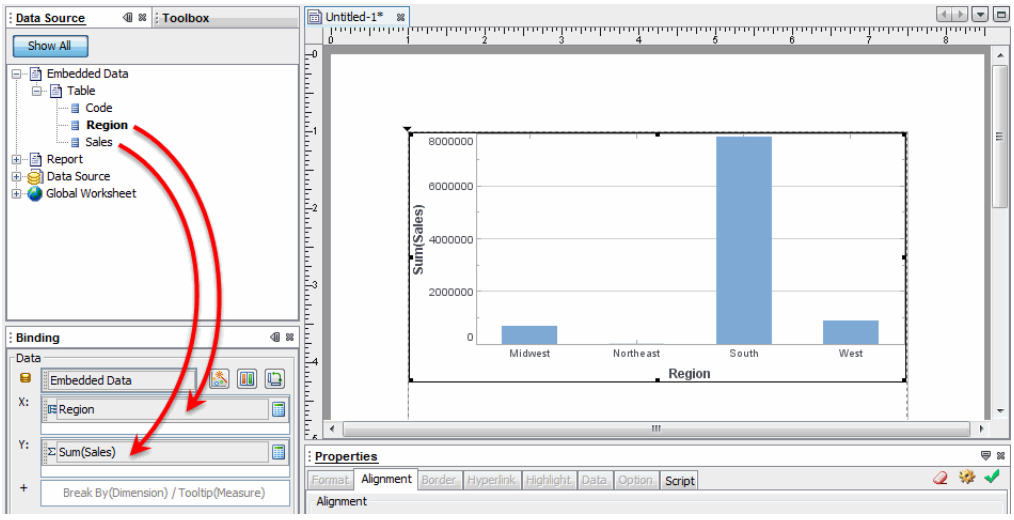
7. Repeat this process to populate the entire table.



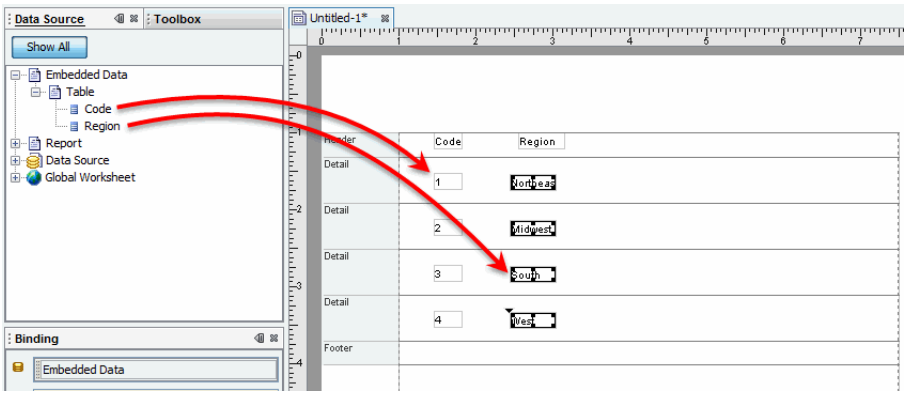
8. Press ‘OK’ to close the dialog box. This immediately populates the Table or Crosstab in the report with the embedded data that you entered.



For a Chart, proceed to bind the data fields using the Chart Editor in the usual manner.



For a Section, proceed to add the fields into the Section in the usual manner.



6.5 Binding Data to a Crosstab Element

The following sections explain how to create and bind data to a Crosstab. In general, to create a Crosstab follow the steps below:

1. Use the Data Binding Wizard to help you create the desired crosstab in a step-by-step manner. See *Creating a Crosstab Using the Data Binding Wizard*.

You can also create a Crosstab using free-hand operations. See *Creating a New Table Using Freehand Operations*.

2. (Optional) To further modify the appearance and contents of the Crosstab, use additional freehand operations. See *Creating a Crosstab Using Freehand Operations*.

3. (Optional) To create more complex structures with custom column and row interdependencies, convert the Crosstab to a Formula Table. See *Formula Tables* for more details.

6.5.1 Creating a Crosstab Using the Data Binding Wizard

A Crosstab summarizes information in a tabular format. It displays the values from one or more fields as row headers, and from one or more fields as column headers. At the intersections of the row and columns, the Crosstab displays the aggregate values computed from a measure field for the records that match the row and column headers.

For example, consider the following result set:

CUSTOMER	PRODUCT	QUANTITY
Customer1	Product1	1
Customer1	Product1	2
Customer1	Product2	30
Customer2	Product1	4
Customer2	Product2	50
Customer2	Product2	60

If you use the 'Customer' column as the row header, the 'Product' column as the column header, and the 'Quantity' column as the aggregated measure, the resulting Crosstab appears as follows:

	Product1	Product2
Customer1	3	30
Customer2	4	110

It is possible to report the same information by using a regular table with grouping on both the 'Customer' and 'Product' columns and summarization on 'Quantity' column. However, the Crosstab presentation is often more readblae.

Walkthrough: Creating a Crosstab using Data Binding Wizard

Walkthrough

In this example, you will create a Crosstab with 'Company' and 'Product' as headers, and 'Quantity' as the aggregated measure.



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

2. In the left panel, select 'Report'. In the right 'Types' panel, select 'Blank Tabular Report', and click 'OK'. This creates a new blank report.



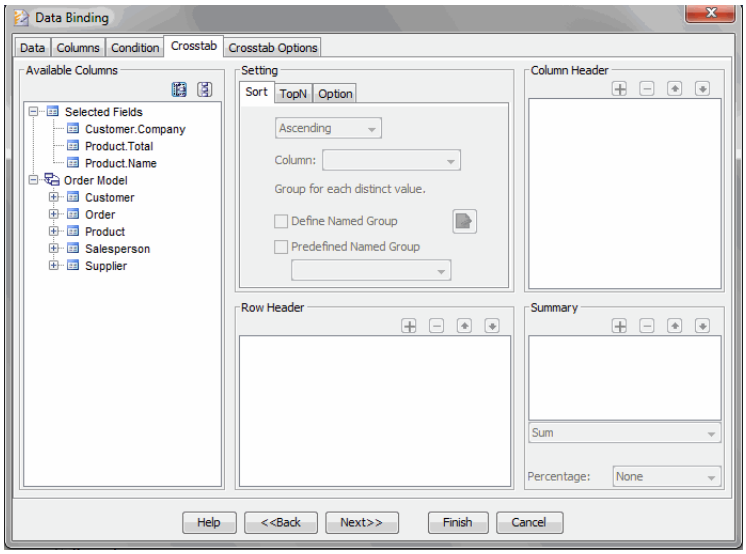
3. Press the 'Crosstab' button in the **Toolbox** panel. This opens the 'Data Binding' dialog box for the crosstab

4. Under the **Data** tab of the 'Data Binding' dialog box, expand the 'Orders' node, and select the 'Order Model' data model.

5. Press the **Columns** tab. Drag the following columns from the 'Available Columns' panel to the right panel:

   'Customer.Company'
   'Product.Total'
   'Product.Name'

6. Select the **Crosstab** tab.



   You can add multiple columns as column headers, row headers, or summary fields by selecting the filed name and clicking the appropriate 'Add' button (or by dragging the field to the desired region). The 'Percentage' option allows a column to be calculated as a percentage of some other row.

7. Drag 'Product.Name' from the 'Available Column' panel into the 'Column Header' panel.

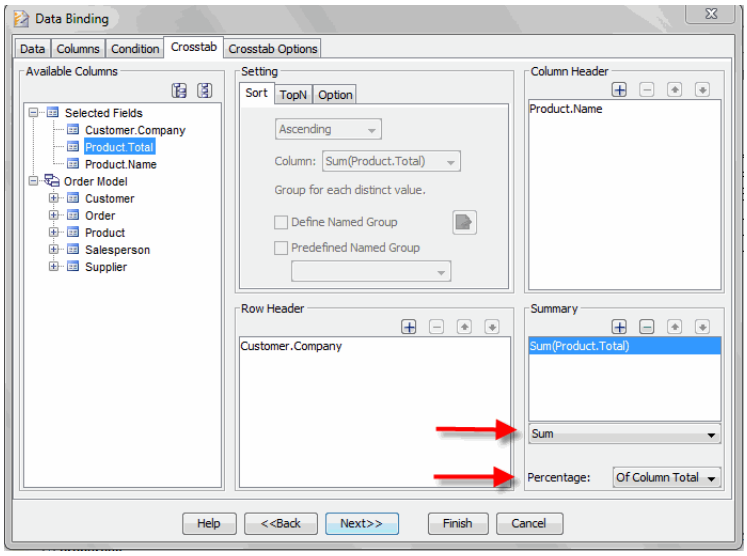
8. Drag 'Customer.Company' from the 'Available Column' panel into the 'Row Header' pane.

9. Drag 'Product.Total' from the 'Available Column' panel into the 'Summary' panel.

10. Click on 'Product.Total' in the 'Summary' panel, and choose the following options:

    a. Select 'Sum' as the aggregation method in the menu.

b. In the ‘Percentage’ menu, select ‘Of Column Total’. This displays the ‘Sum(Total)’ measure as a percentage of the column total.



11. (Optional) Set any desired options for the Crosstab under the **Options** tab in the ‘Data Binding’ dialog box. See *Crosstab Options* for more details.

Walkthrough: Setting Layout for a Crosstab

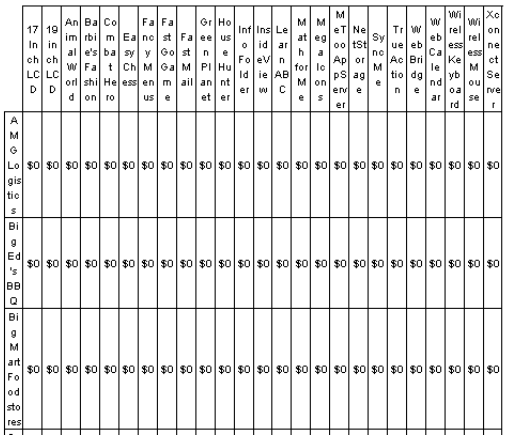
Walkthrough

This example continues from *Walkthrough: Creating a Crosstab using Data Binding Wizard*.

```
Note: After you click ‘Finish’, you cannot return to the ‘Data
Binding’ dialog box.
```

12. Click the ‘Finish’ button in the ‘Data Binding’ dialog box.

13. Preview the report.



	17 Inch LCD	19 inch LCD	Animal World	Barbie's Fashion	Combat Hero	Easy Chess	Fancy Menus	Fast Go Game	Fast Mail	Green Planet	House Hunter	Info Folder	Inside View	Learn ABC	Math for Me	Megalocos	MeTooAppServer	NetStorage	Sync Me	True Action	Web Bridge	Web Calendar	Wireless Keyboard	Wireless Mouse	Xconnect Server
AMG Logistics	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0
Big Ed's BBQ	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0
Big Mart Food stores	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0	$0

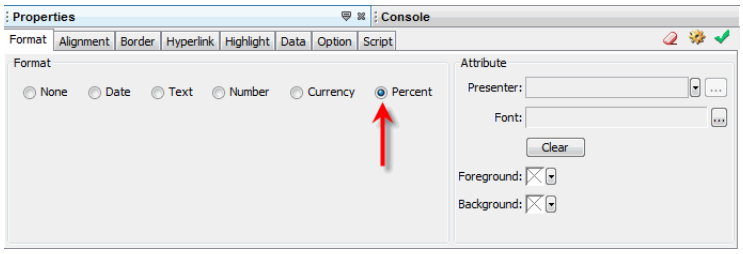
The generated crosstab has many columns, with one for each product. However, the table columns do not fit nicely on one page. To allow the table to wrap to the next page, you will change the table layout options.

14. Return to the design view. (You can keep the ‘Preview’ tab open as well.)

15. Click the Table element to select it. Click the summary cell to select the cell.

16. Click the ‘Properties’ link at the bottom of the Style Studio window. This expands the ‘Properties’ panel.

17. Click the **Format** tab, and select ‘Percent’ as the format.



18. Press the ‘Apply’ button (check mark) to finalize the settings.

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

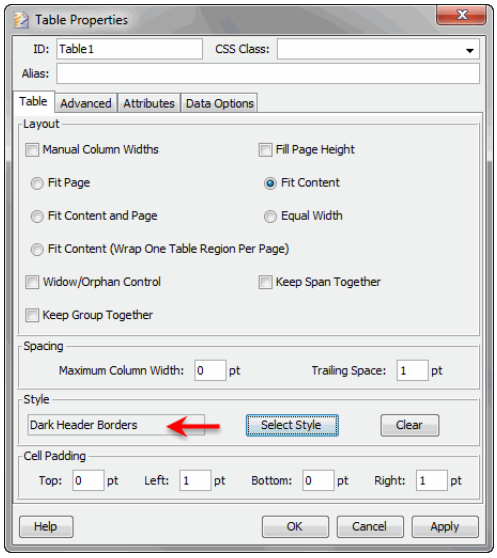
20. Click the **Table** tab, and select the ‘Fit Content’ layout option.

21. In the ‘Style’ panel, click the ‘Select Style’ button. This opens the ‘Style Viewer’ dialog box.

22. In the ‘Styles’ tree, expand the ‘Blue’ folder, and select the ‘Dark Header Borders’ table style.

23. Press ‘OK’ to save the settings and exit the ‘Style Viewer’ dialog box.

24. Press ‘OK’ to exit the ‘Table Properties’ dialog box.





25. Press ‘Preview’ in the Style Studio toolbar to preview the report.

	17 Inch LCD	19 inch LCD	Animal World	Barbie's Fashion
AMG Logistics	3%	2%	1%	1%
Big Ed's BBQ	2%	3%	0%	1%
BigMart Foodstores	1%	4%	1%	1%
Computer Tech	2%	4%	1%	1%
Daycare One	3%	4%	1%	2%
Direct Sales	4%	2%	0%	1%
Eastern Data	1%	1%	0%	1%
Ernst Handel	3%	5%	1%	1%
Fast Transportation	2%	1%	1%	1%
FISGA Corp	3%	0%	1%	1%
FMG Consulting	1%	4%	1%	1%
Folk Mining	1%	3%	0%	1%
George Services	2%	2%	1%	1%
High Society Investments	2%	1%	1%	1%

6.5.2 Creating a Crosstab Using Freehand Operations

In *Creating a Crosstab Using the Data Binding Wizard*, you learned how to use the Data Binding Wizard to bind a Crosstab to a data source. You can also bind a Crosstab to a data source by using freehand operations.

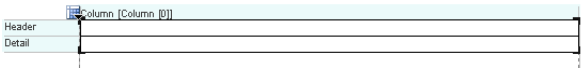
To create a new Crosstab that displays fields from a query, data model entity, or Data Block, follow the steps below:

1. Click in the report to place the insertion point at the desired location.

2. Open the **Toolbox** panel. (Select ‘Toolbox Pane’ from the ‘Window’ menu.)

`If you click (instead of drag) the ‘Crosstab’ component, this opens the Data Binding Wizard. See Creating a Crosstab Using the Data Binding Wizard for more details.`

3. Drag the ‘Crosstab’ component onto the report. This creates a new blank Crosstab.

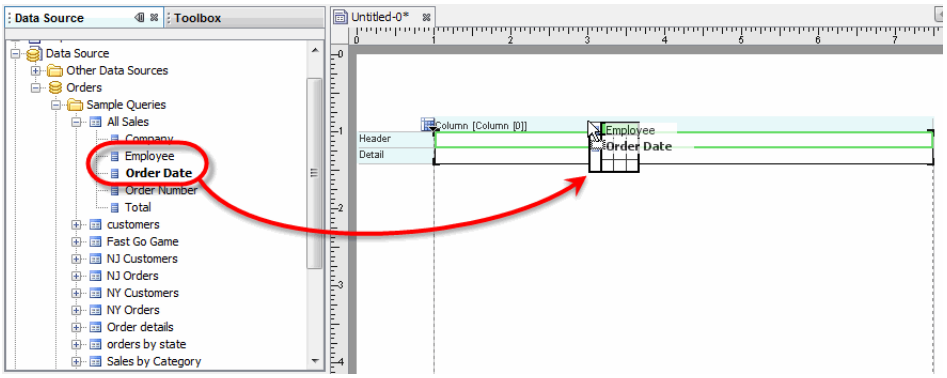


4. Open the **Data Source** panel. (Select 'Data Source Pane' from the 'Window' menu.)

5. Expand a query, data model entity, or Data Block in the **Data Source** panel.

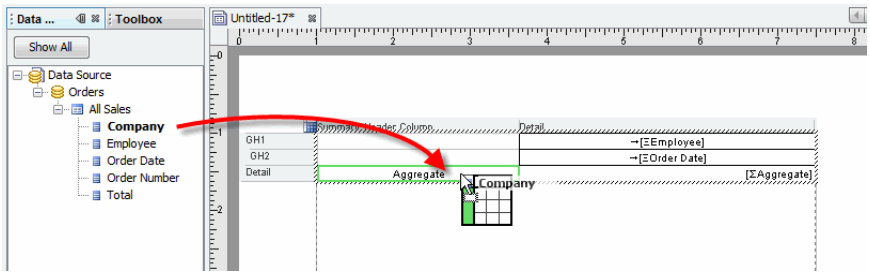
You can also drag the field to the 'Column Header' region in the **Binding** panel.

6. To specify the Crosstab column headers, click to select a field (or Ctrl-click to select multiple fields) from the data source. Drag the field(s) into the Crosstab Header region.



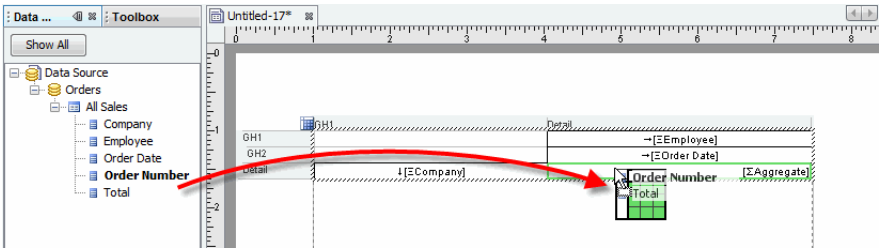
You can also drag the field to the 'Row Header' region in the **Binding** panel.

7. To specify the Crosstab row headers, click to select a field (or Ctrl-click to select multiple fields) from the data source. Drag the field(s) into the Crosstab Detail region.



You can also drag the field to the 'Aggregate' region in the **Binding** panel.

8. To specify the Crosstab aggregates, click to select a field (or Ctrl-click to select multiple fields) from the data source. Drag the field(s) into the 'Aggregate' cell.

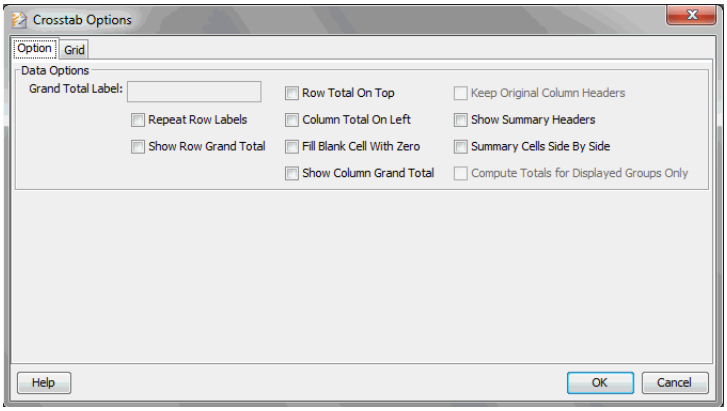


This completes the Crosstab data binding.

To change the grouping and aggregation options, see *Modifying Table Grouping Properties* and *Modifying Summary Properties*. For greater control of individual cell bindings and table layout, you can convert the Crosstab to a Formula Table. See *Formula Tables* for more information.

Crosstab Options

To specify additional attributes for a Crosstab, right-click on the Crosstab, and select 'Options' from the context menu. This opens the 'Crosstab Options' dialog box, with the following tabs: **Option** and **Grid**.



The available properties include the following:

To change the text in the summary headers, apply a text format to those cells. See *Format Types* for more information.

- **Show Summary Headers**: Displays the column header names for the summary (aggregated measure) fields.

		Annie				Eric				Robert				Sue			
		2008	2009	2010	2011	2008	2009	2010	2011	2008	2009	2010	2011	2008	2009	2010	2011
AMG Logistics	Order Number					24152	24537	24911	75746					12064	48970	12440	12605
	Total					81215	67216	104072	189916					24139	42724	47264	29352
Big Ed's BBQ	Order Number		24421	62301	63049					36140	12200	37178	37717				
	Total		84324	198368	171675					9587	21334	74746	62505				
BigMart Foodstores	Order Number	24228	36783	24903	50355					36193	36777	49699	12652				
	Total	68846	48910	86924	132962					45414	40517	96446	46650				

- **Summary Cells Side by Side**: Displays summary headers just above the data cells (otherwise, on the left).

	Annie							
	2008		2009		2010		2011	
	Order Number	Total	Order Number	Total	Order Number	Total	Order Number	Total
AMG Logistics								
Big Ed's BBQ			2	84324	5	198368	5	171675
BigMart Foodstores	2	58846	3	48910	2	86924	4	132962
Computer Tech	2	69541	1	30429	4	60327	5	121460
Daycare One								
Direct Sales	3	32441	5	142121	4	124228		

- **Show Row Grand Total**: Displays the global aggregate computed across all row groups in the last row of the Crosstab.

	Annie				Eric				Robert				Sue			
	2008	2009	2010	2011	2008	2009	2010	2011	2008	2009	2010	2011	2008	2009	2010	2011
World Software Resellers					3	1	5	1					4		3	1
					44407	12705	76646	24236					119870		173202	1645
XO Resellers					4	3	3	3					2	1	1	2
					124801	163759	22759	24801					68109	3380	4434	22333
Grand Total	37	39	48	58	43	54	54	50	32	38	36	41	33	53	46	38
	558344	947222	1029318	1480099	1319825	1348279	1245198	1196020	770329	936770	917688	1074398	881325	1416955	1260291	876793

- **Show Column Grand Total**: Displays the global aggregate computed across all column groups in the last column of the Crosstab.

	Annie				Eric				Robert				Sue				Grand Total
	2008	2009	2010	2011	2008	2009	2010	2011	2008	2009	2010	2011	2008	2009	2010	2011	
Tricounty Auto					3	2	3	2					4	6	1	1	22
					82267	41287	44728	57560					50834	131339	7702	10819	426536
Tumbleweed Gaming	2	1	2	3						5	3	2					18
	18354	8438	64142	36721						123010	83015	63940					397620
Ubermeyer					4	3	3	5					3	3	1	2	24
					123104	17624	124573	191398					67727	20280	4082	123746	672534
West Coast Customs	3	3	5	1					3	3	3	1					22
	34772	34778	135975	7786					12601	103477	14437	5636					349462

This option is not available when both row and column totals are suppressed.

- **Compute Totals for Displayed Groups Only**: Computes the totals based only on the groups that are actually displayed in the table, i.e., groups selected by any Top- or Bottom-N filtering that you have specified. Disable this option to aggregate the totals based on all groups, including those groups that are not displayed in the table.
- **Row Total on Top**: Places the row grand total (if displayed) in the first row following the columns headers, rather than in the last row.

	Annie				Eric				Robert				Sue			
	2008	2009	2010	2011	2008	2009	2010	2011	2008	2009	2010	2011	2008	2009	2010	2011
Grand Total	37	39	48	58	43	54	54	50	32	38	36	41	33	53	46	38
	558344	947222	1029318	1480099	1319825	1348279	1245198	1196020	770329	936770	917688	1074398	881325	1416955	1260291	876793
AMG Logistics	4	2	18	99	25	79	98	20	9	0	8	98	5	55	91	3
					2	2	2	6					1	4	1	1
					81215	67216	104072	189916					24139	42724	47264	29352
Big Ed's BBQ		2	5	5					3	1	3	3				
		84324	198368	171675					9587	21334	74746	62505				

- **Column Total on Left**: Places the column grand total (if displayed) in the first column following the row headers, rather than in the last column.

	Grand Total	Annie				Eric				Robert				Sue			
		2008	2009	2010	2011	2008	2009	2010	2011	2008	2009	2010	2011	2008	2009	2010	2011
AMG Logistics	19					2	2	2	6					1	4	1	1
	585898					81215	67216	104072	189916					24139	42724	47264	29352
Big Ed's BBQ	22		2	5	5					3	1	3	3				
	622539		84324	198368	171675					9587	21334	74746	62505				
BigMart Foodstores	22	2	3	2	4					3	3	4	1				
	555669	58846	48910	86924	132962					45414	40617	96446	45650				

- **Fill Blank Cell with Zero**: Place zeros in aggregate cells for which there are no corresponding records. Otherwise, the cells are left blank (default).

	Annie				Eric				Robert				Sue			
	2008	2009	2010	2011	2008	2009	2010	2011	2008	2009	2010	2011	2008	2009	2010	2011
AMG Logistics	0	0	0	0	2	2	2	6	0	0	0	0	1	4	1	1
	0	0	0	0	81215	67216	104072	189916	0	0	0	0	24139	42724	47264	29352
Big Ed's BBQ	0	2	5	5	0	0	0	0	3	1	3	3	0	0	0	0
	0	84324	198368	171675	0	0	0	0	9587	21334	74746	62505	0	0	0	0
BigMart Foodstores	2	3	2	4	0	0	0	0	3	3	4	1	0	0	0	0
	58846	48910	86924	132962	0	0	0	0	45414	40517	96446	45650	0	0	0	0
Computer Tech	2	1	4	5	0	0	0	0	0	3	2	4	0	0	0	0
	69041	30429	60327	121450	0	0	0	0	0	89338	63325	73046	0	0	0	0
Daycare One	0	0	0	0	2	7	2	3	0	0	0	0	0	0	2	2
	0	0	0	0	82726	123180	127383	41947	0	0	0	0	0	0	74785	63525

- **Keep Original Column Headers**: Places the field names for the Crosstab row headers at the top of the corresponding column.

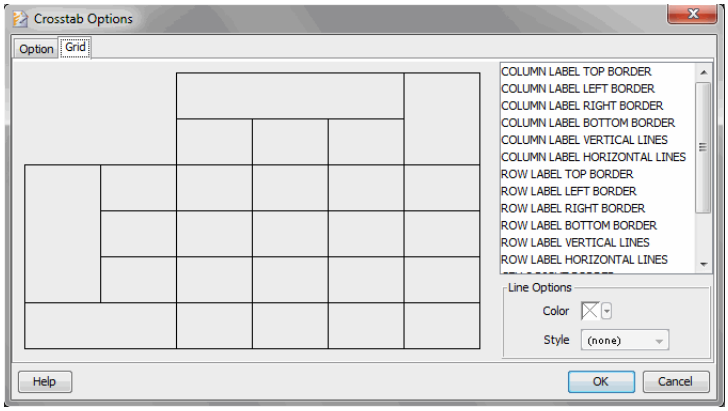
Company	Annie				Eric				Robert				Sue			
	2008	2009	2010	2011	2008	2009	2010	2011	2008	2009	2010	2011	2008	2009	2010	2011
AMG Logistics					24152	24537	24911	75746					12064	48970	12440	12605
					81215	67216	104072	189916					24139	42724	47264	29352
Big Ed's BBQ		24421	62301	63049					36140	12200	37178	37717				
		84324	198368	171675					9587	21334	74746	62505				
BigMart Foodstores	24228	36783	24903	50355					36193	36777	49699	12652				
	58846	48910	86924	132962					45414	40517	96446	45650				
Computer Tech	24142	12316	49621	63023						36740	24979	50299				
	69041	30429	60327	121450						89338	63325	73046				

- **Grand Total Label**: Sets the header text for the row grand total and column grand total, when displayed.

	Annie				Eric				Robert				Sue				Overall
	2008	2009	2010	2011	2008	2009	2010	2011	2008	2009	2010	2011	2008	2009	2010	2011	
AMG Logistics					2	2	2	6					1	4	1	1	19
					81215	67216	104072	189916					24139	42724	47264	29352	585898
Big Ed's BBQ		2	5	5					3	1	3	3					22
		84324	198368	171675					9587	21334	74746	62505					622539
BigMart Foodstores	2	3	2	4					3	3	4	1					22
	58846	48910	86924	132962					45414	40517	96446	45650					555669

- **Percentage**: Specifies the basis on which percentages are computed when you select the 'Percent' option from a measure in the **Binding** panel. (See *Modifying Summary Properties* for information on editing a measure representation.) The 'By columns' option computes the percentage so that the sum of the measure across columns in a single row is 100%, either within each group or across all groups (depending on the specified 'Percent' option in the **Binding** panel). The 'By rows' option computes the percentage so that the sum of the measure across rows in a single column is 100%, either within each group or across all groups (depending on the specified 'Percent' option in the **Binding** panel).

The **Borders** tab allows you to change the visual appearance of the Crosstab. Click a border in the 'Crosstab Grid' area to define crosstab border formatting.



Binding a Crosstab to Embedded Data

To populate a crosstab with data that you enter manually, use the 'Embedded Data' option. See *Binding a Table to Embedded Data* for more information.

6.6 Binding Data to a Chart Element

This section explains how to bind a chart to a data source, set ranking order, aggregation method, and chart style. For information on setting conditions, see *Filtering Data in a Table*.

See Also

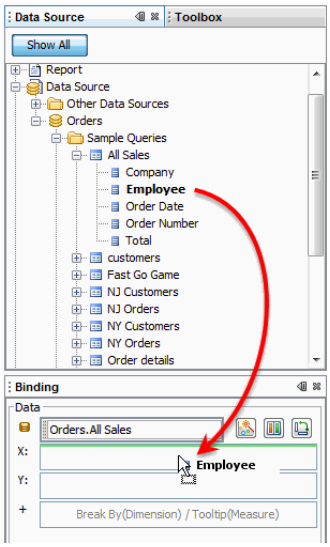
Binding Data to a Chart Element, for more details chart on data binding.
Chart Properties, for information on setting formats and controlling appearance.

6.6.1 Binding a Chart to a Data Source

After you have added a new chart element to the report (see *Adding a Chart Element*), follow the steps below to bind the chart to a data source:

1. Click on the chart to select it. This opens the **Data Source** panel and the **Binding** panel.

2. Drag a field from the desired query or data model in the **Data Source** panel onto the 'X' field in the **Binding** panel.

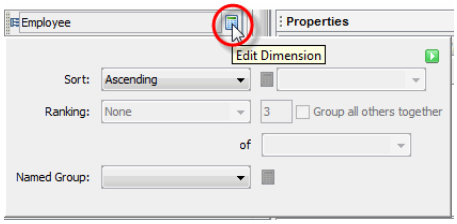




This field provides the chart's X-axis data, and the field name is automatically used as the axis label. The X-axis field should generally be configured as a dimension. See *Editing Dimensions and Measures* for more information.

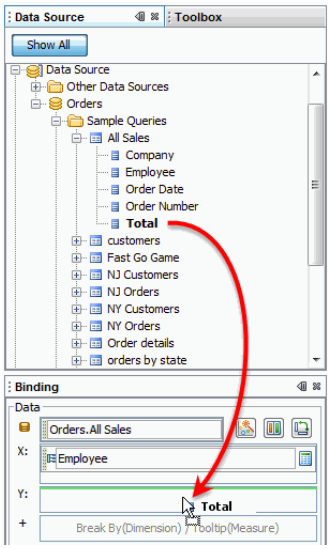
3. Click the 'Edit Dimension' button, and select the desired ordering and ranking for the X-axis data.

Ranking requires a measure to be defined (next step). To rank a dimension based on a measure that is not displayed in the chart, see *Adding a Non-Displaying Measure to a Chart*.



4. Drag a field from the desired query or data model in the **Data Source** panel onto the 'Y' field in the **Binding** panel.

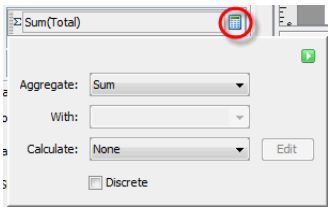
Tip: You can drag a measure field from the **Data Source** panel directly onto the Viewsheet grid to create a Range Slider component that filters the chart data.





This field provides the Chart's Y-axis data, and the field name is automatically used as the axis label. The Y-axis field should generally be configured as a measure. See *Editing Dimensions and Measures* for more information.

5. Press 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 next sections for information about changing other aspects of the chart, including visual properties, labeling, and sub-series.

Example: Chart Data Binding

In this example, you will create a chart that displays Sales vs. State.

1. Create a new blank tabular report. (See *Creating a New Blank Report*.)

2. Add a text element at the top of the report. Set the text to “Total Sales by State”. Center the text, and make the font Bold 14-point. This will be the chart title.



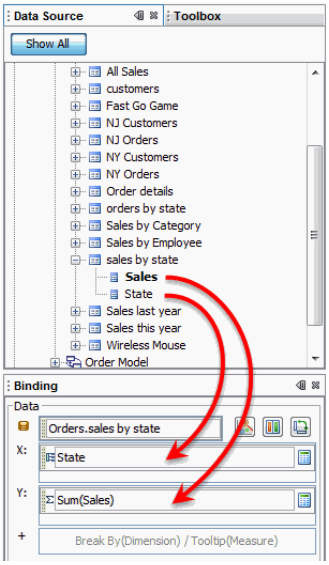
3. Click the ‘Chart’ button to add a blank chart to the report.

4. Click on the chart to select it. This opens the **Data Source** panel and the **Binding** panel.

5. Under the **Data Source** tab, expand the ‘Orders’ data source, the ‘Sample Queries’ folder, and the ‘sales by state’ query.

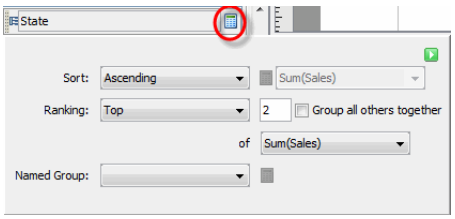
6. Drag the ‘State’ field from the ‘sales by state’ query onto the ‘X’ field in the **Binding** panel.

7. Drag the ‘Sales’ field from the ‘sales by state’ query onto the ‘Y’ field in the **Binding** panel.



8. Press the ‘Edit Dimension’ button next to the ‘State’ field.

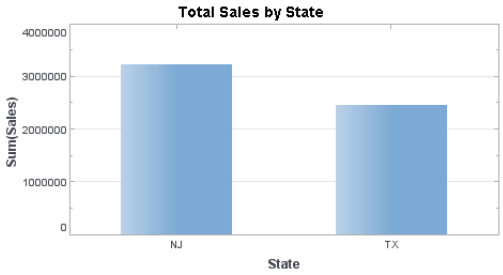
9. From the ‘Ranking’ menu, select ‘Top’, enter “2” for the number of groups, and select ‘Sum(Sales)’ for the aggregate.



This specifies that the chart should display only the top two state records, according to their sales.

10. Press the ‘Apply’ button (green arrow).

11. Click the ‘Preview’ button in the Style Studio toolbar to view the report.



¢

See Also

Text Element, for more information about text properties.

Binding a Chart to Geographical Data (Map), for instructions on representing geographic data.

Editing Dimensions and Measures, for more information on ranking and ordering.

X-Axis Label Property, *Y-Axis Label Property*, to modify chart labels.

Adding a Sub-Series, for information on using the ‘Visual’ panel.

6.6.2 Working with Multiple Measures and Subseries

To plot multiple datasets on the same chart, simply drag multiple measures into the ‘Y’ field of the **Binding** 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 how to work with multiple datasets and subseries.

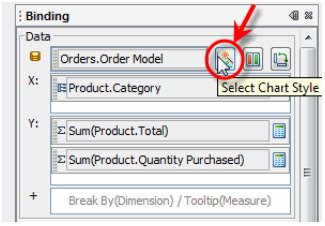
Setting a Chart Style for an Individual Dataset

To set the chart style for an individual dataset on a multi-dataset chart, follow the steps below:

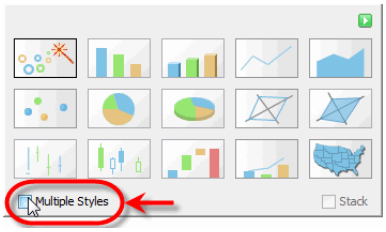
1. Click on the chart to select it. This opens the Chart Editor.

2. In the **Binding** panel of the Chart Editor, press the ‘Select Chart Type’ button. This opens the chart styles panel.

Note: The ‘Auto’ chart type automatically selects an appropriate chart type for the provided data.

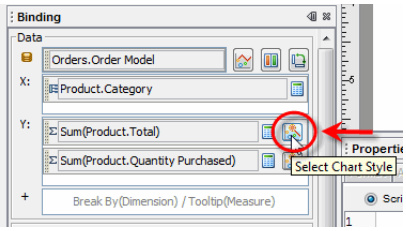


3. In the chart styles panel, enable the ‘Multiple Styles’ option (below the chart style options).

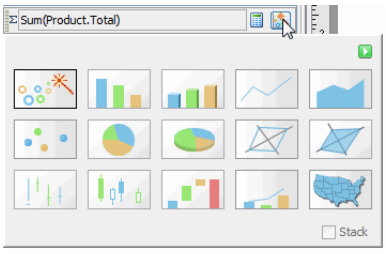


This automatically closes the panel, and enables you to set styles individually for each data set.

4. 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).

5. Click a chart type to select it, and then click the ‘Apply’ button. (You can also double-click the desired chart type.)

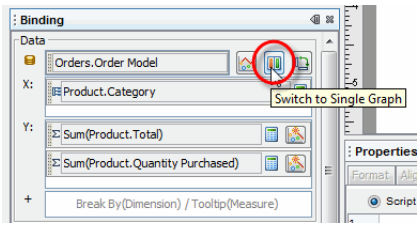
See Also

Setting the Chart Style, for information on setting a single style for all datasets.

Single Chart vs. Separate Charts

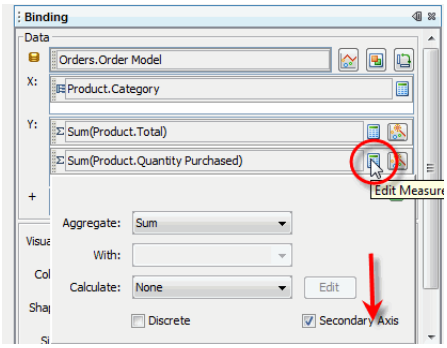
By default, when you specify two datasets on the Y-axis, the datasets are plotted on two adjacent sets of axes. If you want the two datasets to appear on the same set of axes, follow the steps below:

1. Click on the chart to select it. This opens the **Binding** panel.
2. Click the 'Switch to Single Graph' button.



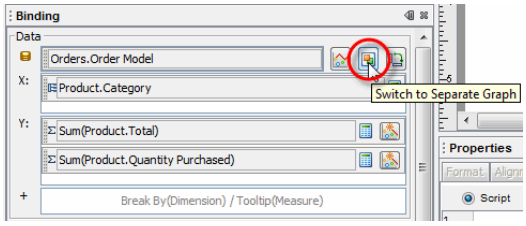
The graph must be in single-graph style to use a secondary axis.

3. (Optional) To display a dataset on the right-side secondary Y-axis, follow these steps:
   a. Click the 'Edit Measure' next to the measure that you wish to display against the secondary Y-axis.
   b. Enable the 'Secondary Axis' option at the bottom of the panel.



   c. Click the 'Apply' button (green arrow).

To return to the multiple axis style, click the corresponding 'Switch to Separate Graph' button.



Adding a Sub-Series

A sub-series represents an additional level of grouping within an individual dataset or measure. Beyond the categories represented by the X-axis labels, the subseries breaks an individual measure down into a secondary level of categories.

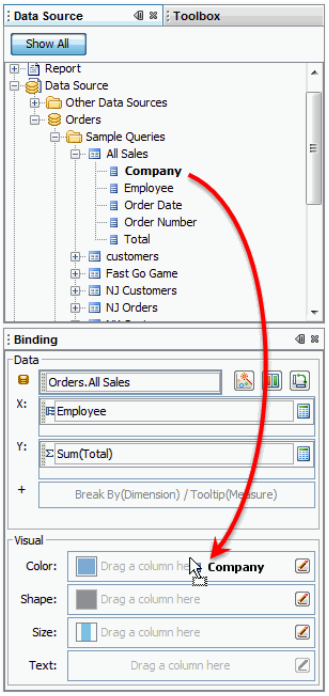
You can create a subseries with or without visual formatting.

Adding a Visually-Formatted Subseries

To add a visual-formatted subseries to an existing chart, follow the steps below:

1. Click on the chart to select it. This opens the **Data Source** panel and **Binding** panel.

2. Drag the desired subseries field from the **Data Source** panel to one or more of the fields in the 'Visual' region of the **Binding** panel.



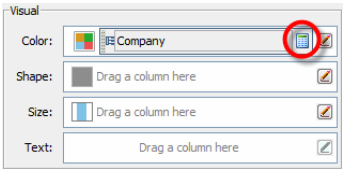
The ‘Color’ field distinguishes the subgroups by color. The ‘Shape’ field discriminates the subgroups by data point shape or fill pattern. The ‘Size’ field discriminates the subgroups by the thickness (bar and line) or size (point) of the representation element. The ‘Text’ field discriminates the subgroups by placement of appropriate text labels.

3. 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*.



Ranking requires a measure to be defined.

4. Click the ‘Edit Dimension’ button, and select the desired ordering and ranking for the subseries values. (See *Editing Dimensions and Measures* for more information.)



When you create a sub-series, this creates a corresponding legend as well. For information on controlling the legend display, see *Legend Property*. For information on further modifying the visual format of the subseries, see *Representing Data with Visual Formats*.

Example: Adding a Subseries

In this example, you will plot ‘Total Purchased’ vs. ‘State’ for a particular dataset. Additionally, you will break down the sales in each region according to ‘Category’. Therefore, ‘State’ will be the main grouping,

represented by the labels on the X-axis, and 'Category' will be the subseries.

Follow the steps below:

1. Click the Chart element in the **Toolbox** panel to add a new chart to the report.

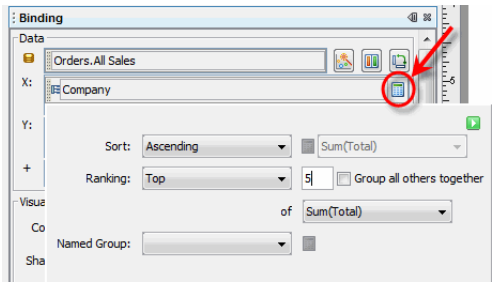
2. Click on the chart to select it. This opens the **Data Source** panel and **Binding** panel.

3. In the **Data Source** panel, expand the 'Orders' data source, the 'Sample Queries' folder, and the 'All Sales' query.

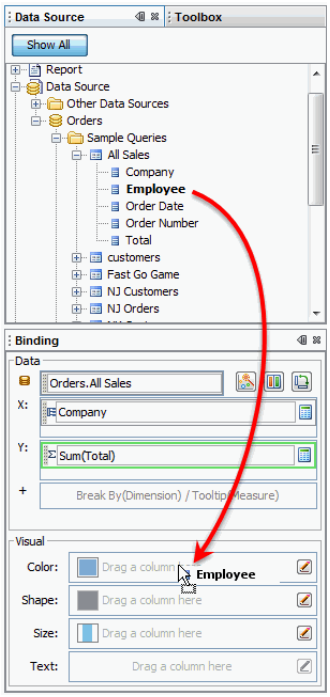
4. From the **Data Source** panel, drag the 'Company' field onto the 'X' region in the **Binding** panel.

5. From the **Data Source** panel, drag the 'Total' field onto the 'Y' region in the **Binding** panel.

6. Click the 'Edit Dimension' button next to the 'Company' field. Under Ranking, select the "Top 5 of Sum(Total)" to select just the top five companies.

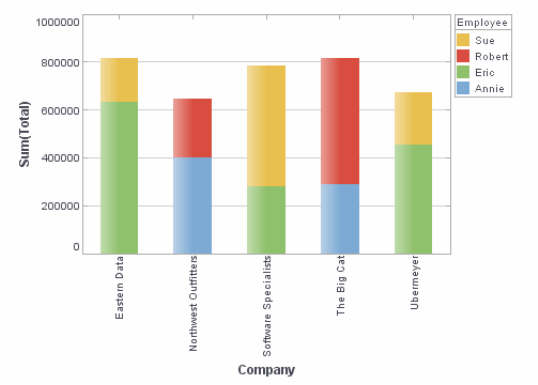


7. From the **Data Source** panel, drag the 'Employee' field onto the 'Color' region in the **Binding** panel.



This will create an 'Employee' subseries, differentiating the employees with color, and will automatically create a corresponding legend.

8. Click the 'Preview' button to display the resulting chart.



Notice that data for the four different employees is now distinguished by four different colors.

¢

See Also

Adding a Non-Formatted Subseries, for information on subseries without format.

Legend Property, for information on controlling the legend display.

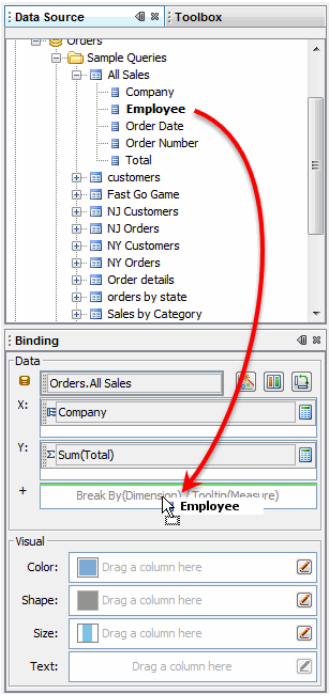
Representing Data with Visual Formats, for information about ordering styles.

Adding a Non-Formatted Subseries

To add a subseries to an existing chart without visual formatting, follow the steps below:

1. Click on the chart to select it. This opens the **Data Source** panel and **Binding** panel.

2. Drag the desired subseries field from the **Data Source** panel to the 'Break By (Dimension) / Tooltip (Measure)' region of the **Binding** 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.

See Also

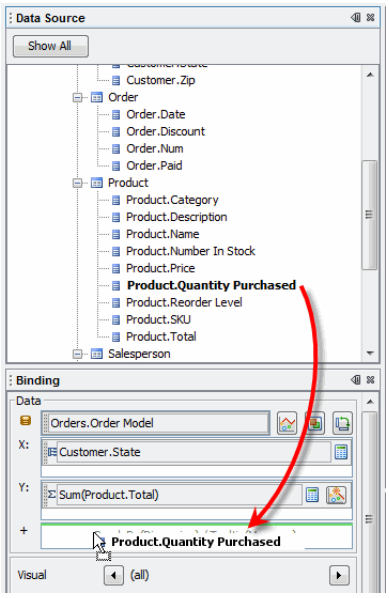
Adding a Visually-Formatted Subseries, for information on subseries with format.

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.

6.6.3 Representing Data with Visual Formats

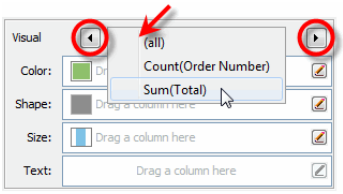
You can represent data values using the following visual formats: Color, shape/pattern, size, or text. These formats can be applied to an entire dataset or to a subseries within a dataset.

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

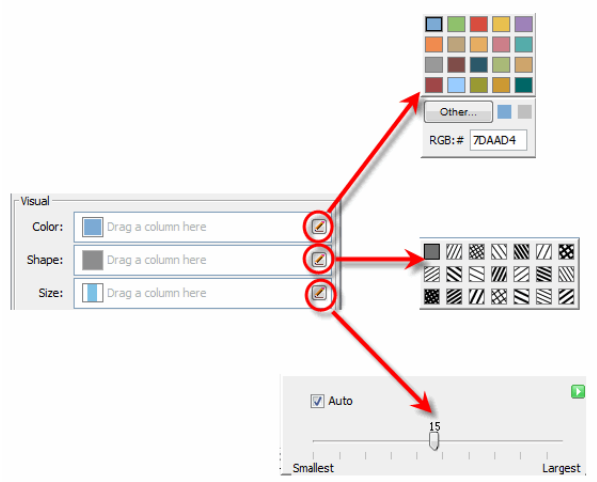
To add a fixed visual format (not keyed to data values), follow the steps below:

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. Click the ‘Edit’ button in one of the format fields in the ‘Visual’ region of the **Binding** panel: ‘Color’, ‘Shape’, ‘Size’.



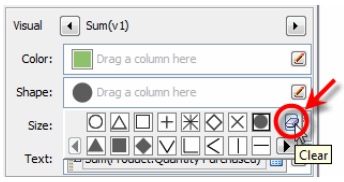
3. Press the green ‘Apply’ button to apply the format.

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 can be any size, and are 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.

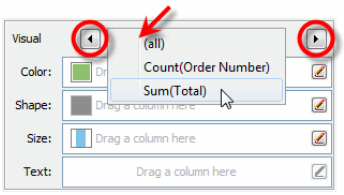
Adding Formats to Measures, for information on keying formats to data values.

Adding a Sub-Series, for information on using format to create groupings.

Adding Formats to Dimensions

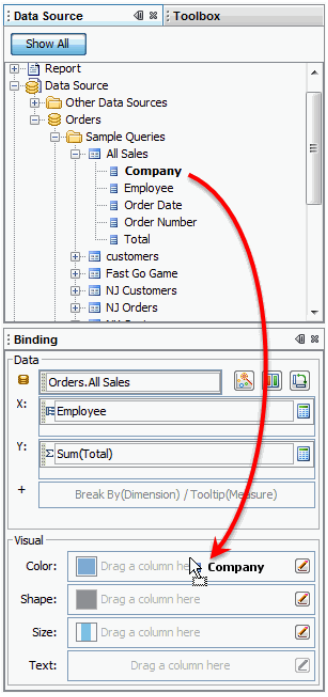
To add a visual format to a dimension, follow the steps below:

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 field from the **Data Source** panel to the desired format field in the 'Visual' region ('Color', 'Shape', 'Size', or 'Text') of the **Binding** panel. This creates a new subseries using the specified formatting.



See Also

Editing Dimensions and Measures, to convert between measure and dimension.

Setting a Chart Style for an Individual Dataset, to assign individual styles to multiple measures.

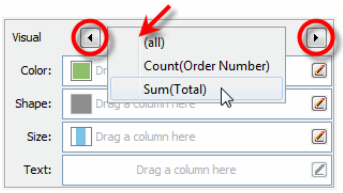
Legend Property, for information on controlling the legend display.

Adding a Sub-Series, for information on using format to create groupings.

Adding Formats to Measures

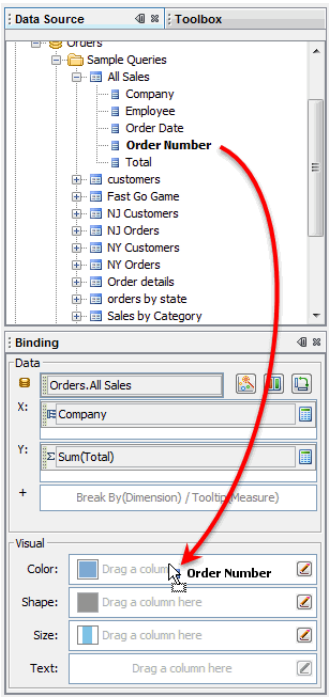
To add a visual format to a measure, follow the steps below:

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 desired field from the **Data Source** panel to the desired format field in the 'Visual' region ('Color', 'Shape', 'Size', or 'Text') of the **Binding** panel.



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

Setting a Chart Style for an Individual Dataset, to assign individual styles to multiple measures.

Editing Dimensions and Measures, to convert between measure and dimension.

Representing Data With Color

To represent a dimension or measure using color, drag a field from the **Data Source** panel to the 'Color' field in the 'Visual' region of the **Binding** panel.

See Also

Adding Formats to Dimensions, for information on using format to categorize.

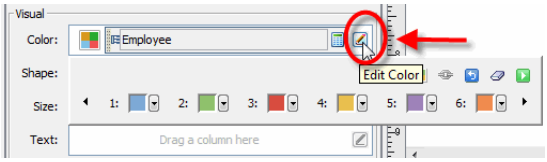
Adding Formats to Measures, for information on using format to code data.

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. Click the 'Edit' button next to the 'Color' field. This opens the color selection menu.

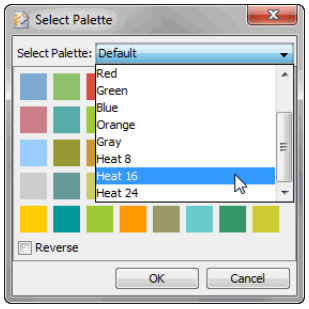


2. Press the 'Select Palette' button to open the 'Select Palette' dialog box.



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.

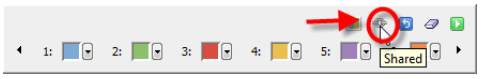


   Press 'OK'.

4. Click on the color chips to create the desired color order.

5. Click the 'Apply' button (green triangle) 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 report) 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 the working repository (*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.

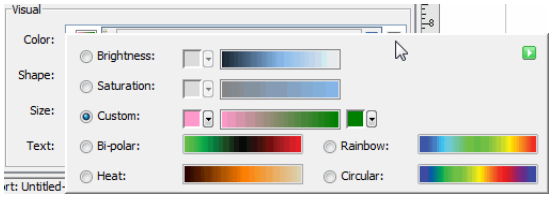
Adding Formats to Measures, for information on using format to code data.

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 measure values, follow the steps below:

1. Click the 'Edit' button next to the 'Color' region of the **Binding** panel. 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 the ‘Apply’ button (green arrow) 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 field from the **Data Source** panel to the ‘Shape’ region in the **Binding** panel.

See Also

Adding Formats to Dimensions, for information on using format to categorize.

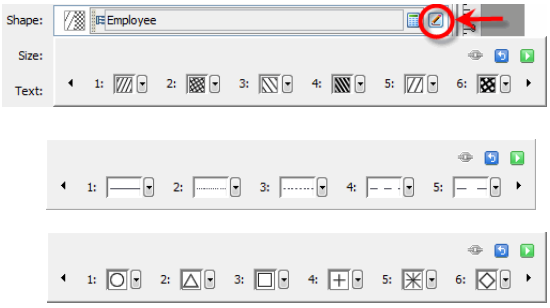
Adding Formats to Measures, for information on using format to code data.

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 depend upon 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 the ‘Apply’ button (green arrow) 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 report) which display the same dimension, click the ‘Shared’ button in the shape panel.



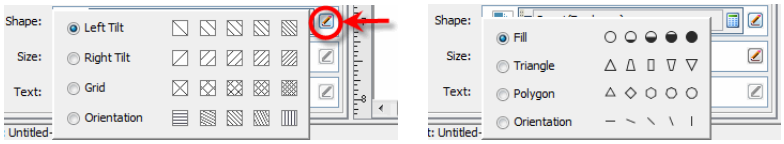
See Also

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 depend upon 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.

See Also

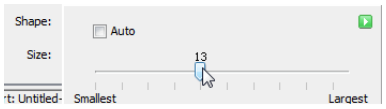
Adding Formats to Dimensions, for information on using format to categorize.
Adding Formats to Measures, for information on using format to code data.
Adding a Sub-Series, for information on using format to create groupings.

Representing Data With Size

To represent a dimension or measure using size, drag a field from the **Data Source** panel to the ‘Size’ region in the **Binding** 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 sliders 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.

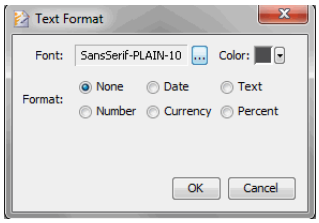
Representing Data With Text

To display dimension or measure values using text, drag the desired field from the **Data Source** panel to the ‘Text’ region in the **Binding** panel.

Note: By default, dimension and measure values are automatically displayed by tooltip when the report is 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.



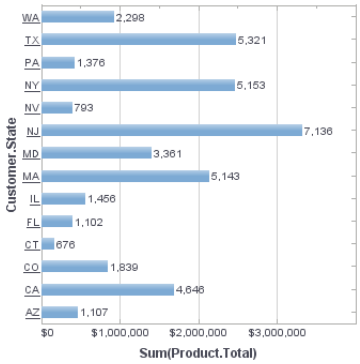
2. Select the text size, font, and color.

3. Select the format for the displayed text. See *Formatting* for an explanation of the format options.

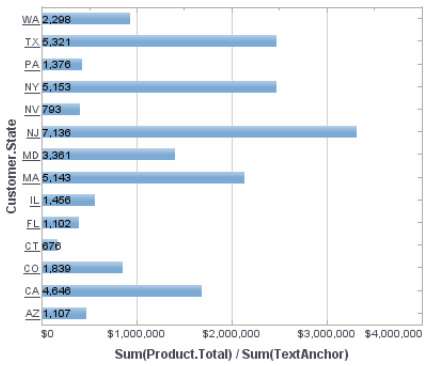
4. Click ‘OK’ to finalize the settings.

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 (in the **Visual** panel) 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:

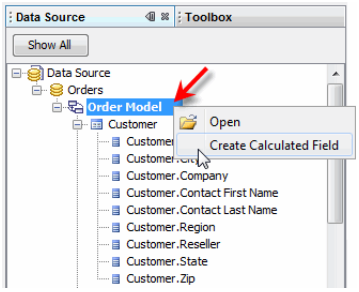
1. Create a new report and add a Chart component from the **Toolbox** panel into the report.

2. Click on the Chart to select it. In the **Data Source** panel, expand the 'Orders' data source and expand the 'Order Model'.

3. From the **Data Source** panel, drag the 'State' field from the 'Customer' entity to the 'Y' region of the **Data** panel.

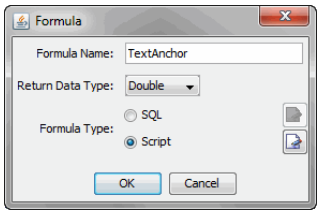
4. From the **Data Source** panel, drag the 'Total' field from the 'Product' entity to the 'X' region of the **Data** panel.

5. In the **Data Source** panel, right-click on the 'Order Model', and select 'Create Calculated Field'.



This opens the 'Formula' dialog box.

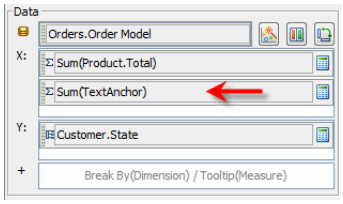
6. Name the field 'TextAnchor' and set the 'Return Data Type' to 'Double'. Press the 'Edit' button to open the Script Editor.



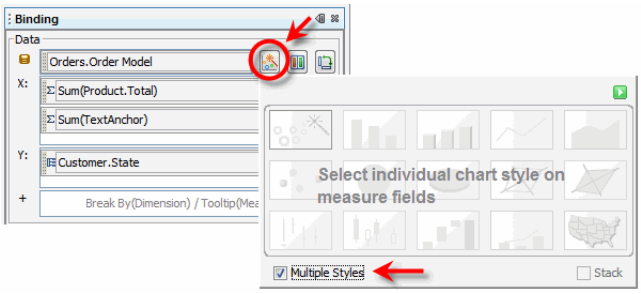
7. In the Script Editor, enter the formula "1", and press 'Save and Close'. Press 'OK' in the 'Formula' dialog box.

   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.

8. Drag the new 'TextAnchor' measure from the **Data Source** panel to the 'X' region of the **Data** panel, under the 'Total' field.



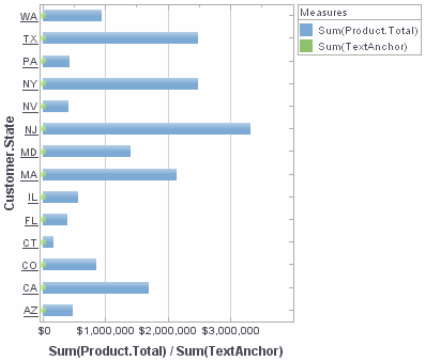
9. Press the 'Select Chart Style' button, and enable the 'Multiple Styles' option. Press the green 'Apply' button to close the panel.



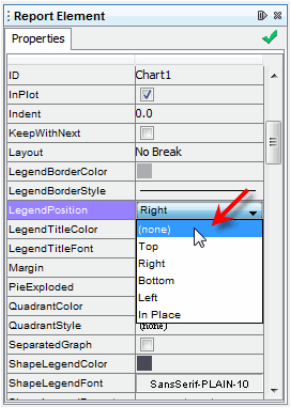
10. Press the ‘Select Chart Style’ button next to the ‘Total’ field, and switch the chart style to ‘Bar’.

11. Press the ‘Select Chart Style’ button next to the ‘TextAnchor’ field, and switch the chart style to ‘Point’.

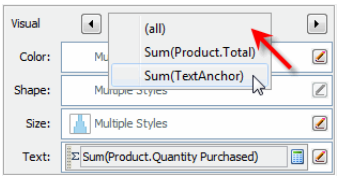
12. Press the ‘Switch to Single Graph’ button to place both data sets on the same axis.



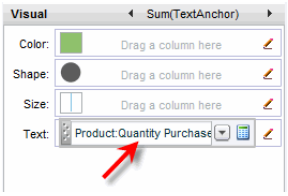
13. In the **Properties** tab of the **Report Element** panel, set ‘LegendPosition’ to ‘(none)’.



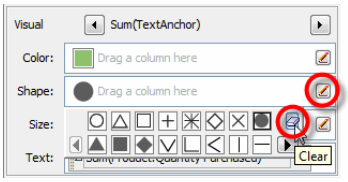
14. In the **Visual** panel, click the title bar between the arrows and select the ‘Sum(TextAnchor)’ measure.



15. From the **Data Source** panel, drag the 'Quantity Purchased' field from the 'Product' entity to the 'Text' region of the **Visual** panel.



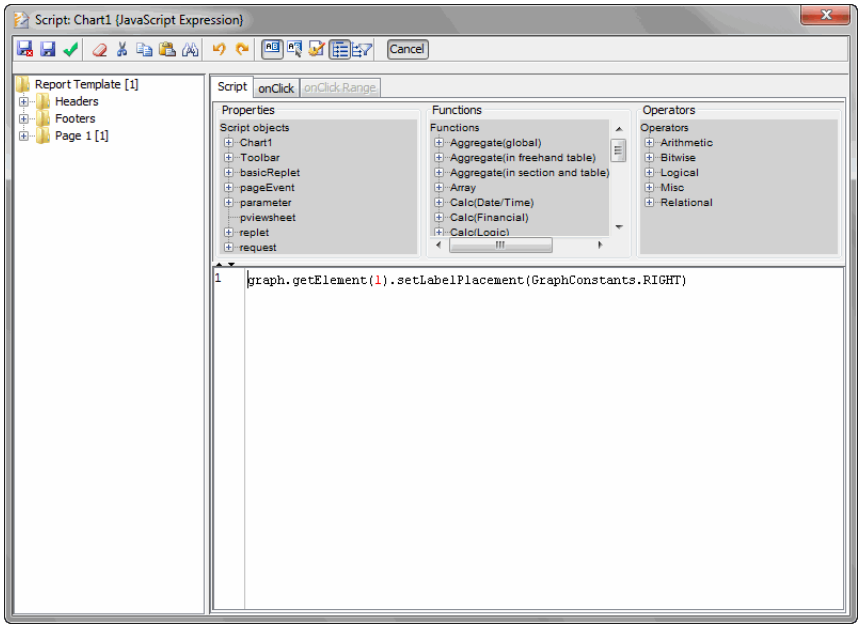
16. In the **Visual** panel, press the 'Edit' button in the 'Shape' region, and press the 'Clear' button. This hides the points for the 'TextAnchor' measure.



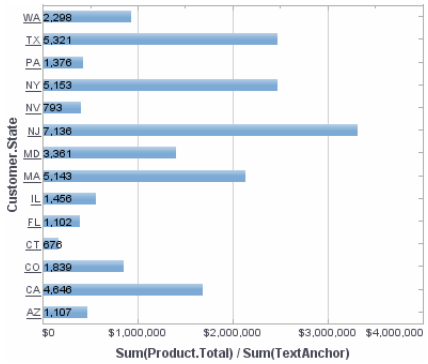
17. In the **Visual** panel, press the 'Edit' button in the 'Color' region, and set a black color for the text.

18. Right-click on the chart and select 'Script'. Enter the following script in the Editor, and press 'Save and Close'.

```
graph.getElement(1).setLabelPlacement(GraphConstants.RIGHT)
```



This sets the alignment appropriately, and yields the desired result:



¢

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.

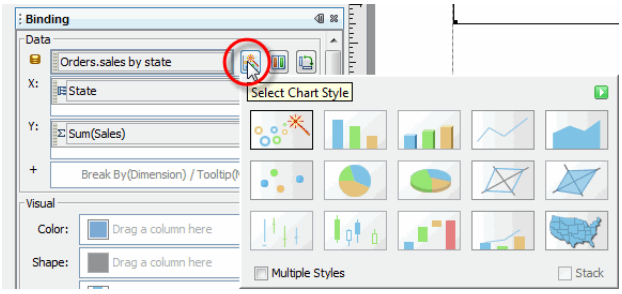
Adding a Sub-Series, for information on using format to create groupings.

6.6.4 Setting the Chart Style

To set the style of the chart (one style for all datasets), follow the steps below:

Note: The 'Auto' chart type automatically selects an appropriate chart type for the provided data.

1. In the **Binding** panel of the Chart Editor, click the 'Select Chart Type' button.



2. (Optional) Select the 'Stack' option at the bottom of the panel to view the stack-type charts.

3. Click a chart type to select it, and then click the 'Apply' button. (You can also double-click the desired chart type.)

Chart Examples

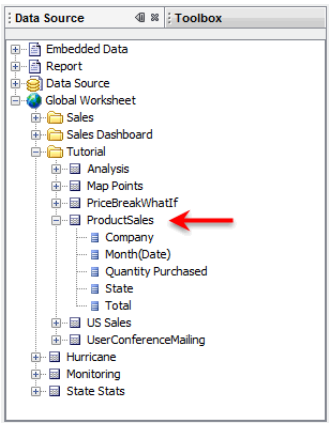
The following sections provide examples of how to create charts in different styles.

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.

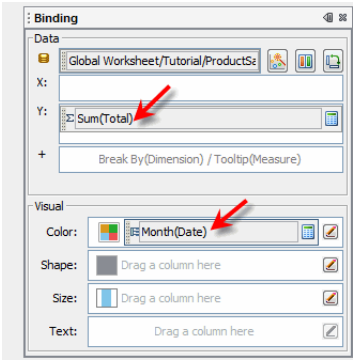
1. Create a new report, and add a blank chart element. Click the chart to open the Chart Editor.

2. In the **Data Source** panel, expand the 'Global Worksheet' node, and expand the 'Tutorial' folder and the 'ProductSales' Data Worksheet.

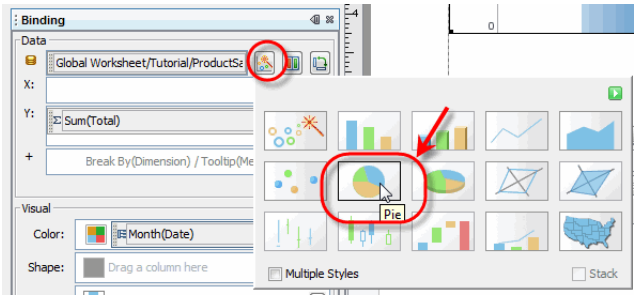


3. From 'ProductSales' Data Worksheet, drag the 'Total' field from the **Data Source** panel to the 'Y' region of the **Binding** panel.

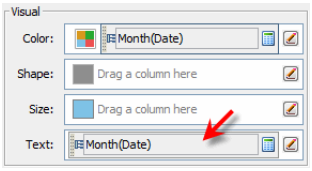
4. Drag the 'Month(Date)' field from the **Data Source** panel to the 'Color' region of the **Visual** panel.



5. Click the 'Select Chart Type' button in the **Binding** panel, and choose 'Pie'. This converts the chart into a pie-chart representation.



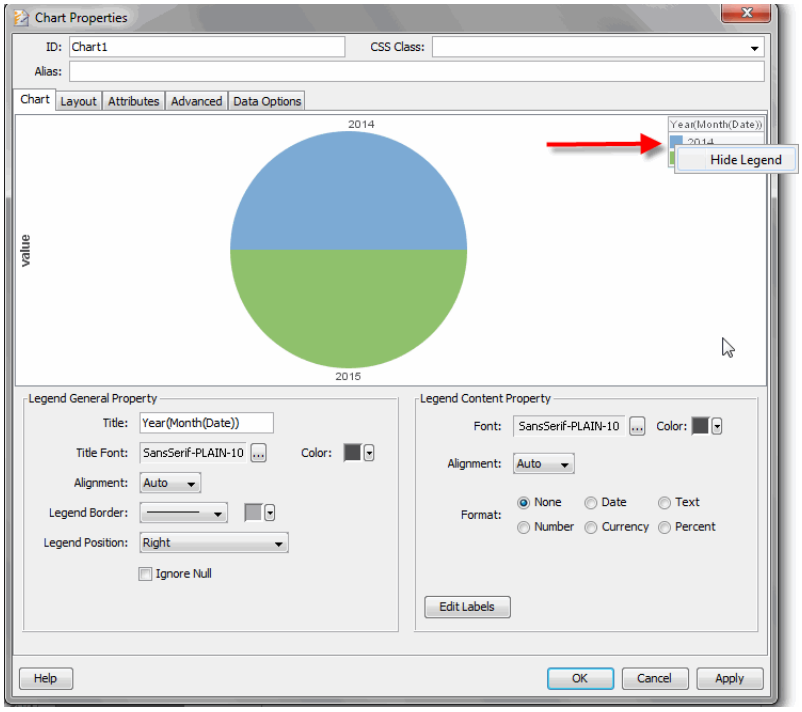
6. Add the respective year values to the pie slices: Drag the 'Month(Date)' field from the **Data Source** panel to the 'Text' region of the **Visual** panel.



7. (Optional) Remove the legend. Follow the steps below:

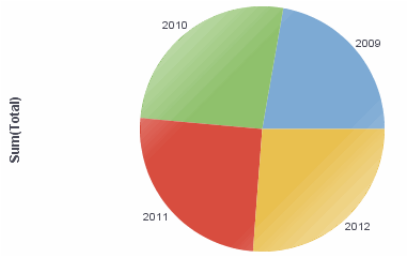
   a. Right-click the chart and select 'Properties' from the context menu. This opens the 'Chart Properties' dialog box.

   b. Right-click on the legend, and select 'Hide Legend' from the context menu.



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

8. Preview the chart.



The pie chart is now complete. To place a small gap between the slices, set the ‘Explode Pie’ plot property. See *Plot Property* for more information.

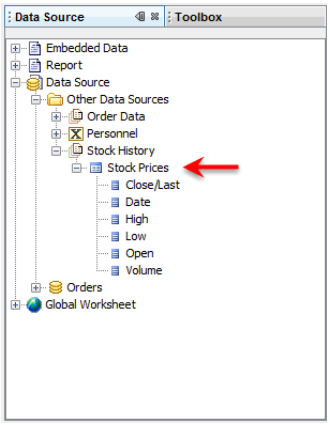
Creating a Candle Chart

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:

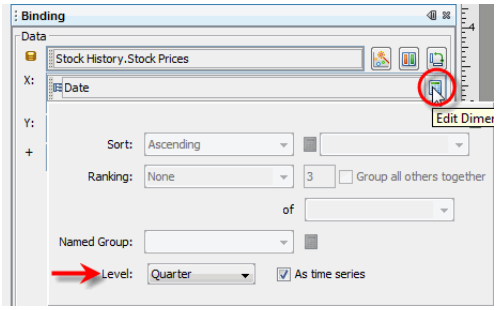
1. Create a new report, and add a blank chart element. Click the chart to open the Chart Editor.

2. In the **Data Source** panel, expand the ‘Other Data Sources’ folder, expand the ‘Stock History’ data source, and expand the ‘Stock Prices’ query.



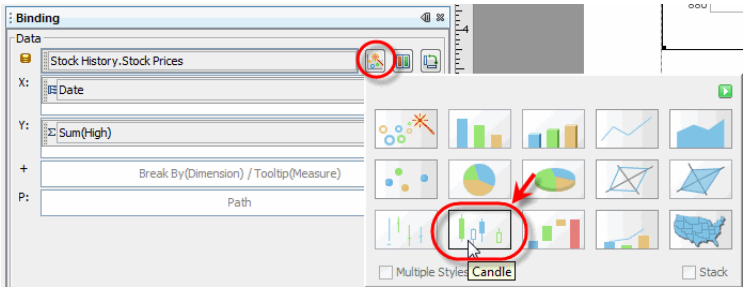
3. Drag the ‘Date’ field from the ‘Stock Prices’ query to the ‘X’ region of the **Binding** panel.

4. Press the ‘Edit Dimension’ button next to the ‘Date’ field, and choose ‘Quarter’ from the ‘Level’ menu. Then press the green ‘Apply’ button.

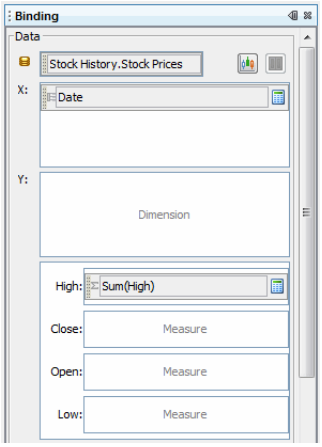


5. Drag the ‘High’ field of the ‘Stock Prices’ query to the ‘Y’ region of the **Binding** panel.

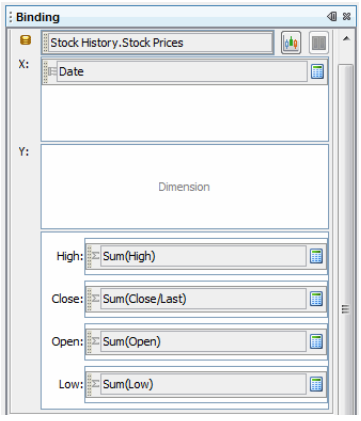
6. Press the ‘Select Chart Style’ button in the **Binding** panel, and select the ‘Candle’ style. Press the green ‘Apply’ button.



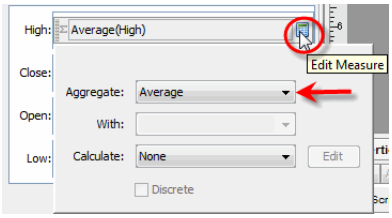
Note that the **Binding** panel now provides binding regions for 'High', 'Close', 'Open', and 'Low'.



7. Drag the 'Close/Last', 'Open', and 'Low' measures from the 'Stock Prices' data block to the appropriate regions of the **Binding** panel, as shown below.

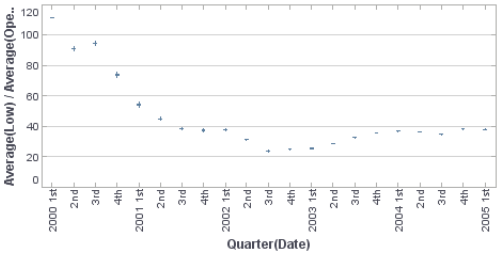


8. Press the 'Edit Measure' button next to the 'High' field, and choose 'Average' from the 'Aggregate' menu. Then press the green 'Apply' button.

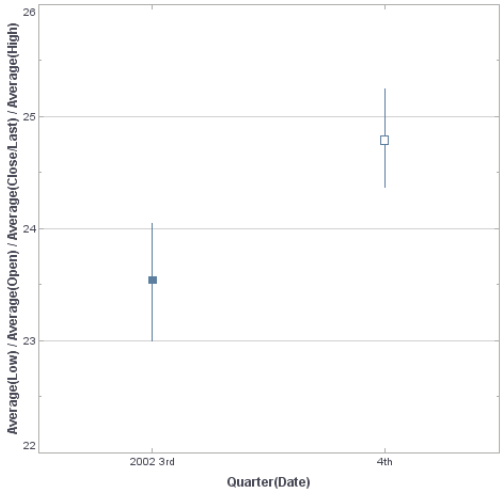


9. Repeat the above step to set the aggregate to ‘Average’ for the other measures: ‘Close/Last’, ‘Open’, and ‘Low’.

10. Preview the chart.



11. To observe the structure of the chart more closely, set a condition on the chart to limit the date range (e.g., limit to the second half of 2002), enlarge the chart element, and/or change the axis limits.



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

Binding a Chart to Geographical Data (Map), for details on binding geographical data.

Setting a Chart Style for an Individual Dataset, to set distinct styles for individual datasets.

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:

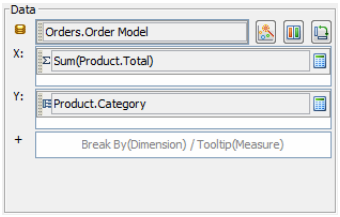
1. Create a new report. Drag a Chart element from the **Toolbox** panel into the report.

2. Click on the Chart to open the Chart Editor.

3. In the **Data Source** panel, expand the 'Data Source' node, then expand the 'Orders' data source, and then expand the 'Order Model'.

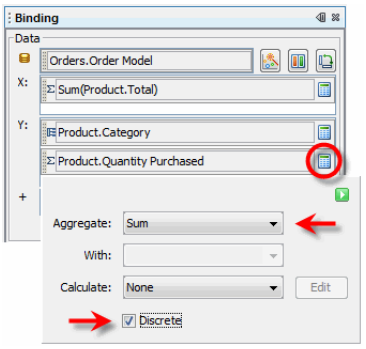
4. From the **Data Source** panel, drag the 'Total' field of the 'Product' entity to the 'X' region of the **Data** panel.

5. From the **Data Source** panel, drag the 'Category' field of the 'Product' entity to the 'Y' region of the **Data** panel.



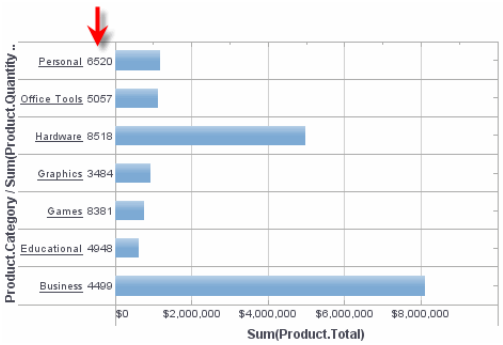
6. From the **Data Source** panel, drag the 'Quantity Purchased' field of the 'Product' entity to the 'Y' region of the **Data** panel (under 'Category').

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.



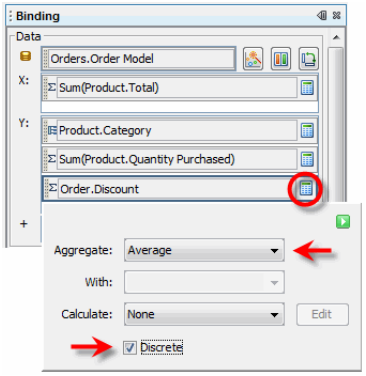
8. Press the green ‘Apply’ button.

   Preview the report and observe that the chart now displays a column of values giving the aggregated ‘Quantity Purchased’ value for each ‘Category’ group.



9. From the **Data Source** panel, drag the ‘Discount’ field of the ‘Order’ folder 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



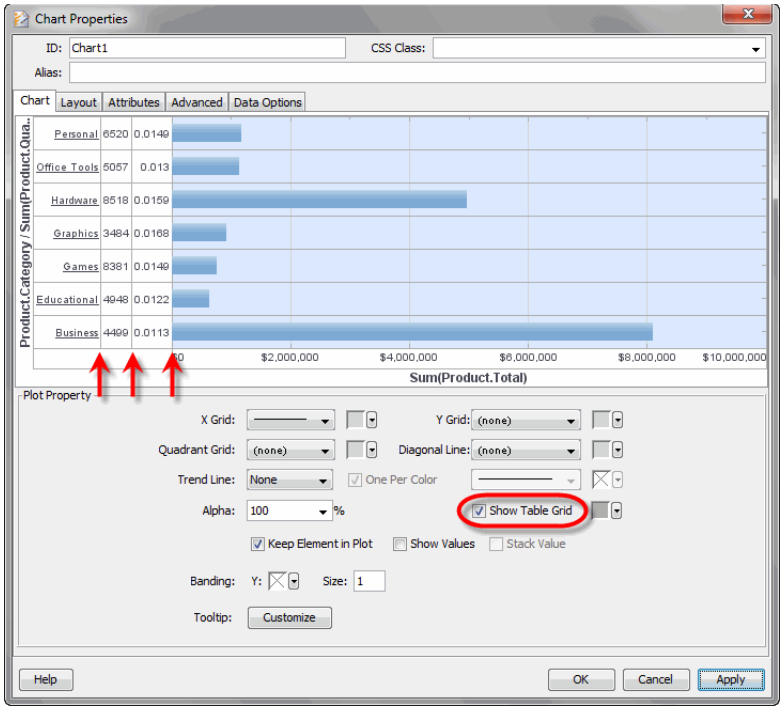
11. Press the green ‘Apply’ button.

    Preview the report and observe that the chart now displays a column of values giving the aggregated ‘Discount’ value for each ‘Category’ group.

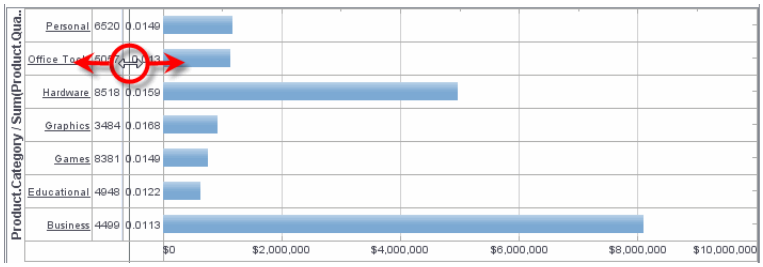
12. Right-click on the chart, and select ‘Properties’ to open the ‘Chart Properties’ dialog box.

13. Click in the plot area to open the ‘Plot Property’ panel below.

14. Enable the ‘Show Table Grid’ option, and press ‘Apply’. This displays grid lines around the plot area, and around each text column.



15. Drag on the boundaries between columns to adjust the spacing between the columns as desired.

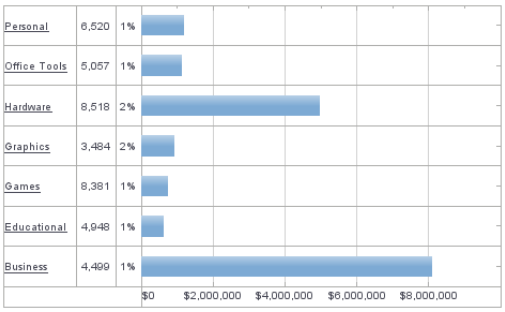


16. Click on each column of text in turn and set the desired formatting in the 'Axis Label Property' panel below. (See *Formatting* for more information about formats.)

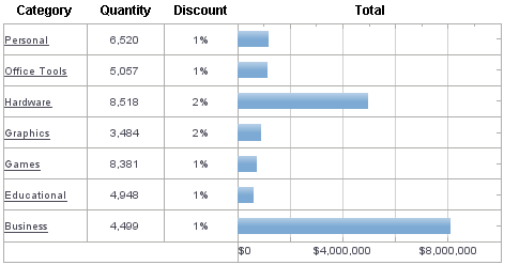
    For example, set a number format for the 'Quantity Purchased' column and set a percent format for the 'Discount' column. Center-align both of these columns, and left-align the 'Category' column.

17. Right-click the X-axis title, and select 'Hide Title' from the context menu. Do the same for the Y-axis title.

18. Press 'OK' to close the 'Chart Properties' dialog box, and preview the report.



19. Resize the chart as desired. Add text elements above the chart to provide any desired column headers. (For precise positioning, place the text elements within a *Section Element*.)



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.
```

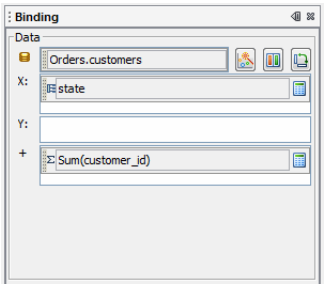
In this example, you will create a dot plot that displays the companies in each state. Follow the steps below:

1. Create a new report and add a Chart to the report. Click on the Chart to open the Chart Editor.

2. In the **Data Source** panel, expand the 'Data Source' node, the 'Orders' data source, and the 'Sample Queries' folder.

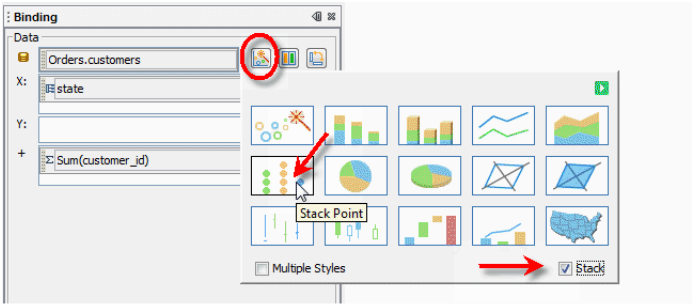
3. Expand the 'customers' query, and drag the 'state' field from the **Data Source** panel to the 'X' region of the **Data** panel.

4. Drag the 'customer_id' field from the **Data Source** panel to the 'Break By' region of the **Data** panel.

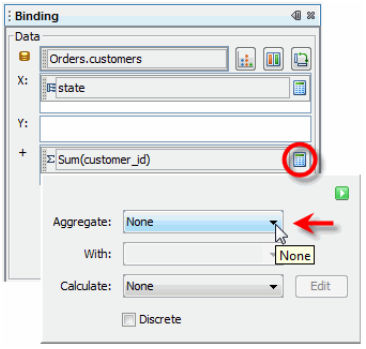


Alternatively, you can drag the 'customer_id' field to the 'Color', 'Shape', or 'Size' regions of the **Visual** panel if you wish to distinguish the data points using a visual format.

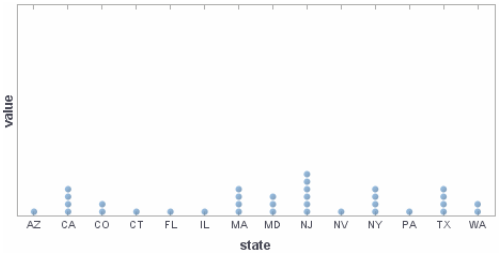
5. Press the 'Select Chart Style' button in the **Data** panel. Select the 'Point' type chart, and enable the 'Stack' option. Press the green 'Apply' button.



6. Press the 'Edit Measure' button next to 'customer_id'. Set the 'Aggregate' property to 'None', and press the green 'Apply' button.

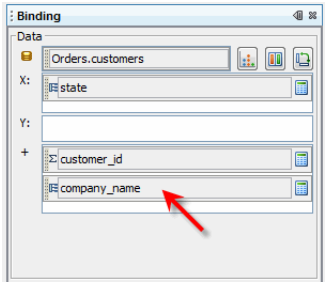


7. Preview the report. You have now created a basic dot plot.

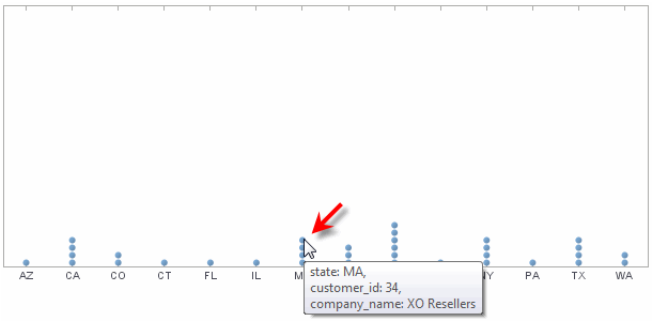


8. (Optional) Right-click on the chart and select ‘Properties’ from the context menu. This opens the ‘Chart Properties’ dialog box. In the diagram in the ‘Chart Properties’ dialog box, right-click the Y-axis title and select ‘Hide Title’ from the context menu. Press ‘OK’ to close the dialog box.

9. (Optional) Drag the ‘company_name’ field from 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. (The tooltip is available when the report is displayed in the User Portal.)



To further customize the tooltip, see *Custom Tooltip*.

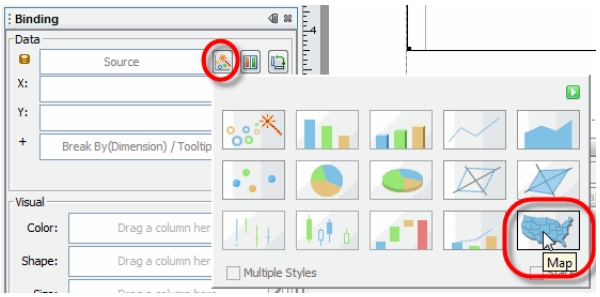
6.6.5 Binding a Chart to Geographical Data (Map)

You can bind a chart to a datasource containing geographical data to create a *map chart*. A map chart 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.

To create a map chart, first add a chart element to the report. (See *Adding a Chart Element* for more details.) Then follow the steps below to bind the chart to a data source:

1. Click on the chart to select it. This opens the **Data Source** panel and **Binding** panel.

2. In the **Binding** panel, click 'Select Chart Style', and choose the 'Map' chart type.

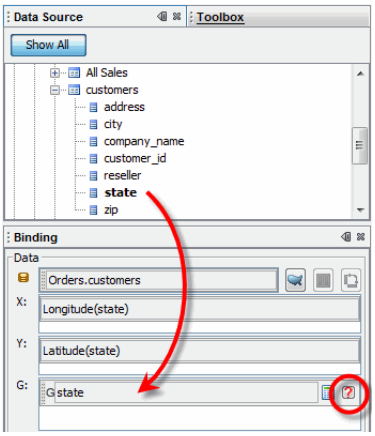


3. Click the 'Apply' button (or double-click the 'Map' icon). The **Binding** panel changes to display a 'Geographic' field.

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

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

5. To change the default map settings (default map is 'World', and default level is 'City'), click the red "?" button to open the geographic level editor.



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

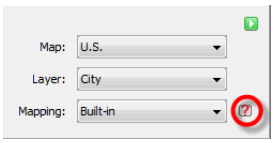
6. In the level editor, select the desired region from the ‘Map’ menu: Asia, Canada, Europe, Mexico, U.S., or World.

7. 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.

8. From the ‘Mapping’ menu, select the desired mapping definition. The options will include “Built-in” and any custom mappings.

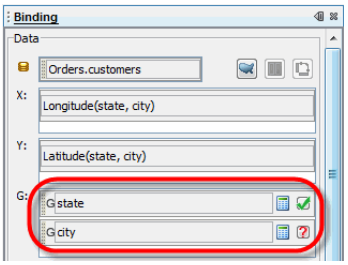
   The Chart Editor attempts to resolve values in the selected geographic field against the corresponding layer in the map database. If all data values are successfully matched, a green check mark is displayed. If some data values cannot be matched, a red “?” is displayed.



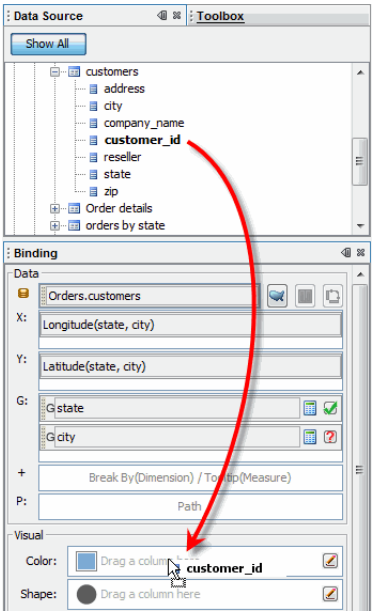
9. To resolve any unmatched names, click the red question mark. This opens the ‘Geographical Mapping’ dialog box. See *Resolving Geographical Data* for further instructions.

10. Click the green ‘Apply’ button.

11. Repeat the previous steps to bind additional ‘Geographic’ levels. For example, you can bind ‘City’ and ‘State’ fields.

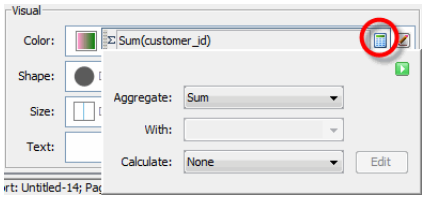


12. From the **Data Source** panel, drag a measure onto one of the fields in the 'Visual' region of the **Binding** panel ('Color', 'Shape', 'Size', or 'Text').



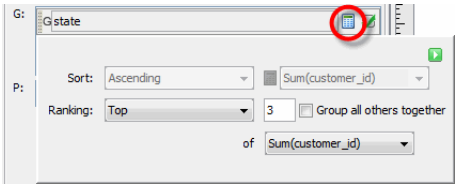
This binding will display the measure on the chart using the visual coding you selected, and automatically creates a corresponding legend. See *Adding a Sub-Series* for more information on using the 'Visual' panel.

13. Click the 'Edit Measure' button, select the desired aggregation method for the measure, and click 'Apply'.



14. (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. Click ‘Apply’.



15. (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.

See Also

Resolving Geographical Data, for information on specifying a new mapping.

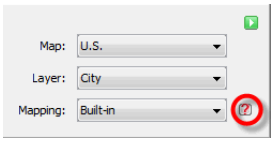
Creating a Custom Map, in *Advanced Server Administration*, to create a map based on a custom shapefile.

Chart Properties, for information on changing chart labels.

Adding a Sub-Series, for information on using the ‘Visual’ panel.

Resolving Geographical Data

When you specify a geographical field, the data binding attempts to resolve values in the 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.



When the data binding 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.

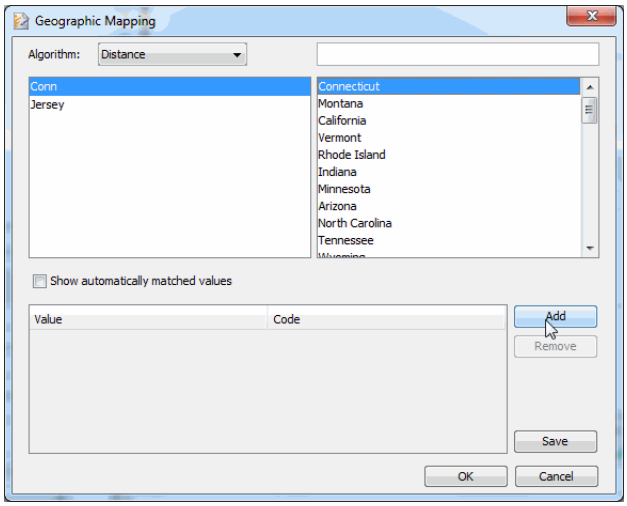
To set the mapping for geographical data, follow the steps below.

1. Click the red question mark. This opens the ‘Geographical Mapping’ dialog box.

    a. Select one of the unmatched names in the top-left panel. The right panel displays a list of candidate matches.

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.

2. Repeat the above steps until all values have been matched. (You can match multiple data values to a single geographical location.)

3. (Optional) When you have completed making all matches, press ‘Save’ to save the mapping for future use. The next time you create a map chart, you can select the saved mapping from the ‘Mapping’ menu in the geographical editing panel.

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

5. Press the ‘Apply’ button in the geographical editing panel.

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. Click the ‘Open’ button in the Style Studio toolbar.

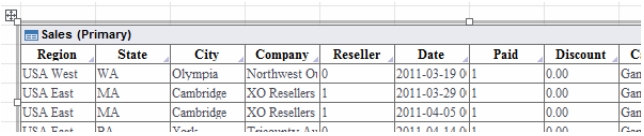
2. In the ‘Open’ dialog box, select the **Worksheet** tab.

3. Expand the ‘Tutorial Folder’ and select the ‘US Sales’ Worksheet. Click ‘OK’ to open the Worksheet.

4. Save a copy of the worksheet as ‘US Sales Copy’. (See *Advanced Data Mashup* for details on opening and saving Worksheets.)

5. In the ‘US Sales Copy’ Worksheet, create a formula column that contains variations of the state names in the ‘State’ column. Follow the steps below:

   a. Click the ‘Create Expression’ button in the title bar of the ‘Sales’ table.


Sales (Primary)

Region	State	City	Company	Reseller	Date	Paid	Discount	C
USA West	WA	Olympia	Northwest Or	0	2011-03-19 0	1	0.00	Gan
USA East	MA	Cambridge	XO Resellers	1	2011-03-29 0	1	0.00	Gan
USA East	MA	Cambridge	XO Resellers	1	2011-04-05 0	1	0.00	Gan

   b. When Style Studio 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.

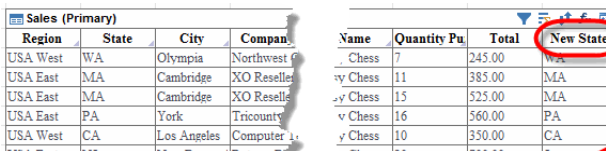
   d. In the Formula Editor, enter an expression that will change two of the names in the ‘State’ column:

```
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 (Primary)

Region	State	City	Compan
USA West	WA	Olympia	Northwest C
USA East	MA	Cambridge	XO Reselle
USA East	MA	Cambridge	XO Reselle
USA East	PA	York	Tricounty
USA West	CA	Los Angeles	Computer

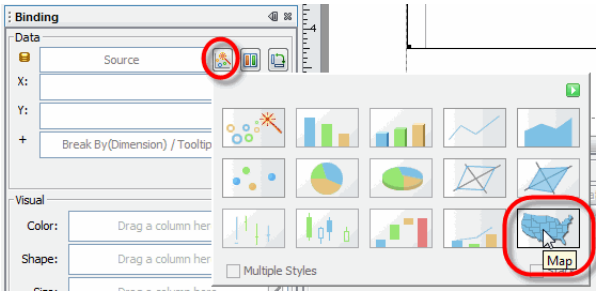
Name	Quantity Pu	Total	New State
Chess	7	245.00	WA
y Chess	11	385.00	MA
y Chess	15	525.00	MA
v Chess	16	560.00	PA
y Chess	10	350.00	CA

6. Save the Worksheet.

7. Create a new report, and add a chart.

8. Click the chart to select it. This opens the **Data Source** panel and **Binding** panel.

9. In the **Binding** panel, click 'Select Chart Style', and choose the 'Map' chart type. Click the 'Apply' button (or double-click the 'Map' icon).



The **Binding** 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 **Binding** panel.

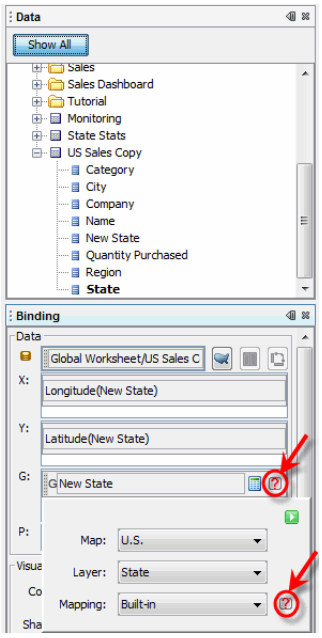
10. In the **Data Source** panel, expand the 'Global Worksheet' node, and expand the 'US Sales Copy' Worksheet.

11. Drag the 'New State' dimension onto the 'Geographic' region in the **Binding** panel.

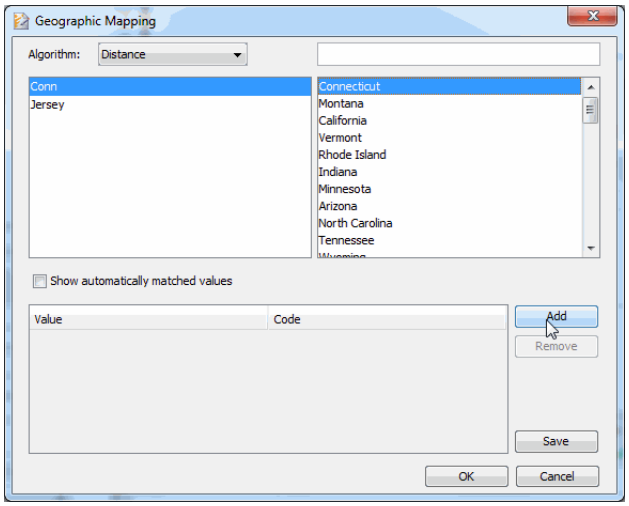
    Note the red "?" that appears beside the field name. This indicates that at least some geographical names in that field could not be successfully matched.

12. Click the red "?" to open the geographical editing panel.

13. Select 'U.S.' from the 'Map' menu, and select 'State' from the 'Layer' menu.



14. Click the red "?" next to the 'Mapping' field 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'.

15. 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'. The right panel 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.

c. Click ‘Jersey’ in the left panel. The right panel suggests a set of likely matches.

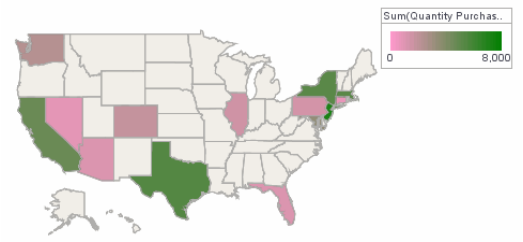
d. Select the correct match (‘New Jersey’), and click the ‘Add’ button. This creates a new mapping between the value ‘Jersey’ and the location ‘New Jersey’, and removes ‘Jersey’ from the left panel.

16. Since all unmatched locations have now been resolved, click ‘OK’ to close the ‘Geographic Feature Names’ dialog box.

17. Click ‘Apply’ in the geographical editing panel for the ‘New State’ field.

18. In the **Data Source** panel, drag the ‘Quantity Purchased’ measure from Worksheet to one of the ‘Visual’ regions (‘Color’, ‘Shape’, etc.) in the **Binding** panel.

19. Preview the report to display the completed map 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

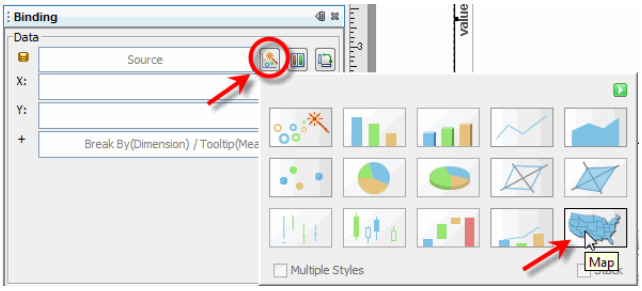
In this example, you will label a map with a list of landmarks along Historic Route 66. The locations of the landmarks are provided by a Worksheet Data Block containing raw latitude and longitude. Follow the steps below:

1. Click the ‘New’ button in the Style Studio toolbar. This opens the ‘New Asset’ dialog box.

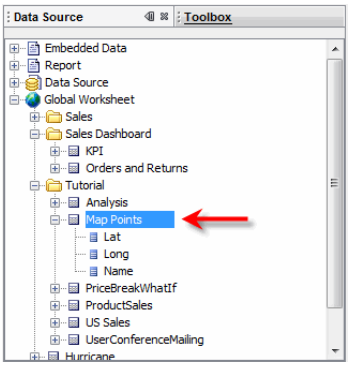
2. In the left panel, select ‘Report’. In the right ‘Types’ panel, select ‘Blank Tabular Report’, and click ‘OK’. This opens a new report for editing.

3. From the Style Studio **Toolbox** panel, drag a Chart component into the report. This opens the **Data Source** and **Binding** panels.

4. In the **Binding** panel, click the ‘Select Chart Style’ button. From the pop-up panel, select the ‘Map’ style, and click the green ‘Apply’ button.



5. In the **Data Source** panel, expand the ‘Global Worksheet’ node. Expand the ‘Tutorial’ > ‘Map Points’ Worksheet node.

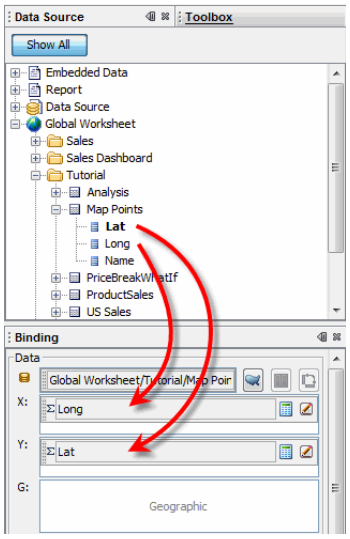


The ‘Long’ field contains longitude values.

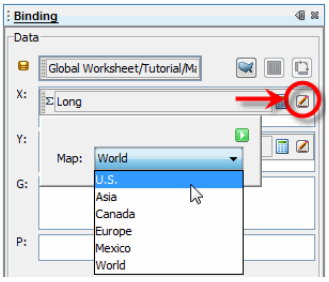
6. From the ‘Map Points’ Worksheet node, drag the ‘Long’ field to the ‘X’ region of the **Binding** panel.

The ‘Lat’ field contains latitude values.

7. From the ‘Map Points’ Worksheet node, drag the ‘Lat’ field to the ‘Y’ region of the **Binding** panel.



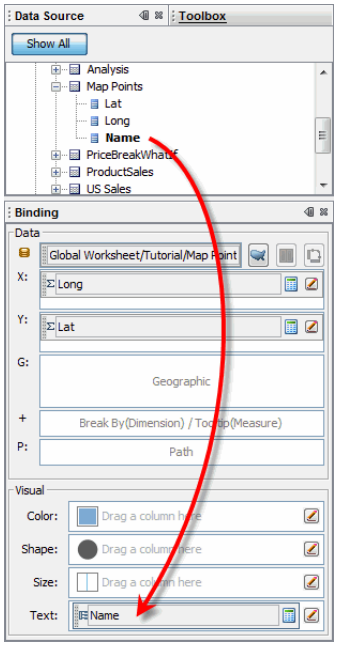
8. In the **Binding** panel, click the 'Edit Map Type' button next to the 'Long' field. From the 'Map' menu, select 'U.S.', and click the green 'Apply' button



9. Preview the report. Observe how data points now mark the locations specified by the latitude and longitude data, and outline the progression of Historic Route 66.



10. To add the landmark labels to the points on the map, return to the design view. Drag the 'Name' field from the **Data Source** panel to the 'Text' field in the **Binding** panel.



11. Preview the report again.

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 19. 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 20. Transformation for Hawaii

$$\begin{bmatrix} long' \\ lat' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 50.0 \\ 0 & 1 & 5.0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} long \\ lat \\ 1 \end{bmatrix}$$

6.6.6 Editing Dimensions and Measures

In a Chart data binding, fields are classified as "measures" and "dimensions." Typically, dimensions contain categorical data and are

plotted against the X-axis, while measures contain numerical data and are plotted against the Y-axis data.

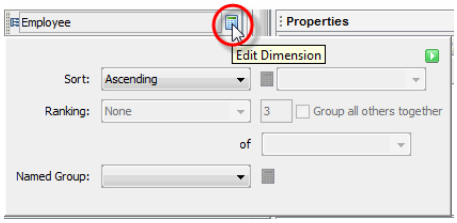
Measures and dimensions have different properties. Dimension data can be sorted and ranked, while measure data can be aggregated by various methods. The following sections explain how to edit dimensions and measures.

Editing a Dimension

To edit the properties of a dimension in a chart, follow the steps below:

1. Click on the chart to select it. This opens the **Data Source** panel and the **Binding** panel.

2. Click the 'Edit Dimension' button next to the desired field in the **Binding** panel, and select the desired sorting or ranking option.



3. Click the 'Apply' button (green arrow).

Note: A measure must be defined in order to rank or sort by value.

The following settings are available:

Sort	Change the ordering of the X-axis labels: **None**: Use the default label order. **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. This automatically creates a custom named grouping with the desired ordering. To remove the manual sort, simply clear the named grouping. (See below.)
Ranking	Filter the axis labels by aggregate measure value. **None**: No ranking of X-axis labels. **Top**: Select the top N labels based on the selected measure. **Bottom**: Select the bottom N labels based on the selected measure.

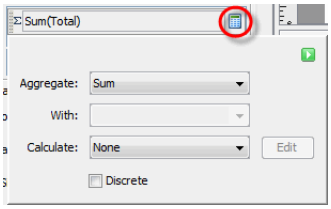
Named Group	Select a predefined named group from a Data Worksheet (see *Creating a Named Grouping* in *Advanced Data Mashup*), or specify a custom named grouping. See *Named Groups* for information on how to design a named grouping.
Level	Set the grouping level for dates. Note the distinction between 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.
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 *Plot Property*.

Editing a Measure

To edit the representation of a measure, follow the steps below:

1. Click on the chart to select it. This opens the **Data Source** panel and the **Binding** panel.

2. Click the 'Edit Measure' button next to the desired field in the **Binding** panel, and select the desired measure options.



3. Click the 'Apply' button (green arrow).

The following settings are available:

Aggregate	The method by which measure data should be aggregated to generate Y-axis values. See *Summarization Formulas* for a description of the methods.
With	For bivariate aggregations (such as 'Covariance' and 'Correlation'), the second operand.
Percentage of	For univariate aggregations, the value on which percentage representations should be based.
Calculate	The computation that determines how values are represented on the chart (i.e., moving average, percentage, difference, etc.). See *Calculating a Measure Representation*.

6.6.7 Filtering Data in a Chart

To apply a filtering condition to a chart, follow the steps below.

1. Right-click the chart, and select 'Condition' from the context menu. This opens the 'Condition' dialog box.

2. Use the menus to enter the desired condition clause(s). See *Filtering Data in Data Binding Wizard* for information about the controls, and an example of constructing a condition.

3. Click 'OK' to close the dialog box, and apply the condition.

See Also

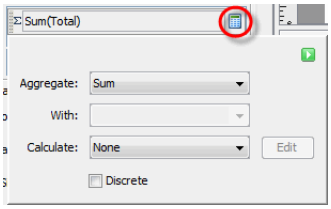
Parameterizing a Condition, for details on using a variable in a condition.

6.6.8 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.

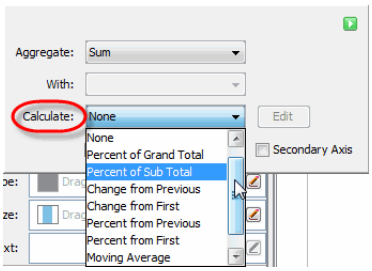
To use one of these representations, set the appropriate *calculation* method. Follow the steps below:

1. In the **Binding** panel, click the 'Edit Measure' button next to a measure.



Select 'None' (default) to use the raw aggregate. Select 'Custom' to create a new calculation method.

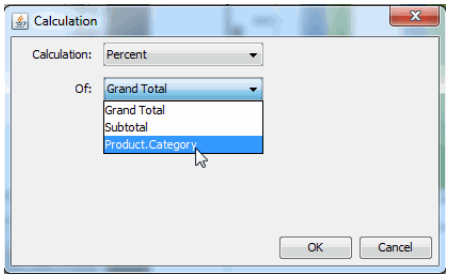
2. Select one of the calculation methods from the 'Calculate' menu.



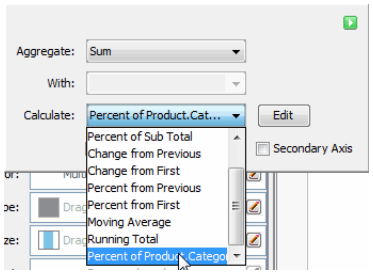
3. Click the ‘Edit’ button. This opens the ‘Calculation’ dialog box, which allows you to modify the properties of the calculation method. To do this, follow the steps below:

   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.



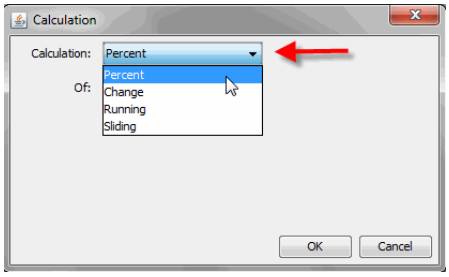
4. Click the green ‘Apply’ button.

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.
Chart Element, for step-by-step instructions on creating a chart.

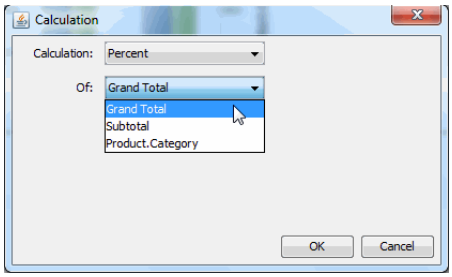
6.6.9 Calculation Methods

There are three basic types of calculations: ‘Percent’, ‘Change’, ‘Running’, and ‘Sliding’. These methods are described in greater detail below.



Percent Calculation

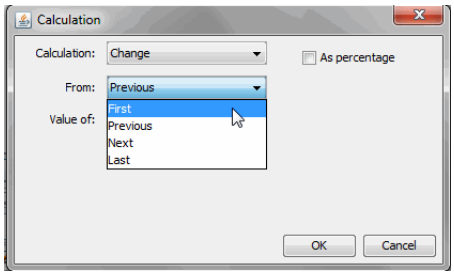
The 'Percent' calculation allows you to express the measure based on 'Grand Total', 'Subtotal', or a particular dimensional group.



- **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 Y-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.

Change Calculation

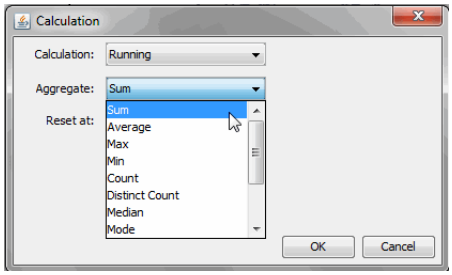
The 'Change' calculation allows you to express the group aggregate in terms of its deviation (or percent deviation, if 'As percent' is selected) from the preceding, succeeding, first, or last group in the series. The 'From' menu specifies the baseline value.



- **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 *Summarization Formulas* for more about the different methods.)

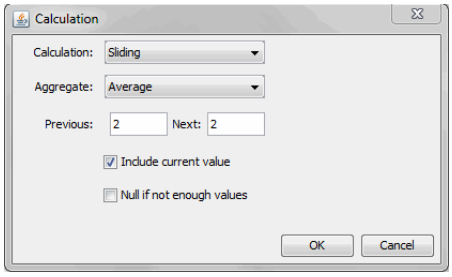


The 'Reset at' option, available for date fields, allows you to specify the date interval (e.g., year, quarter, week, etc.) at which the accumulation should be cleared.

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 *Summarization Formulas* 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.

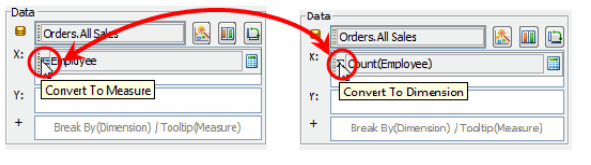
See Also
Running Calculation, to accumulate values from the start of the series.

Converting between a Measure and Dimension

By default, dimensions are displayed as discrete values, whereas measures are aggregated and displayed as a continuous range. To toggle a field between dimension and measure representations, follow the steps below:

1. Click on the chart to select it. This opens the **Binding** panel.

2. In the **Binding** panel, click the 'Convert to Measure' or 'Convert to Dimension' button next to the field name. The icon changes to indicate the representation.



To specify that a particular measure be displayed using discrete values (rather than a continuous spectrum), see *Displaying a Measure as Discrete Values*.

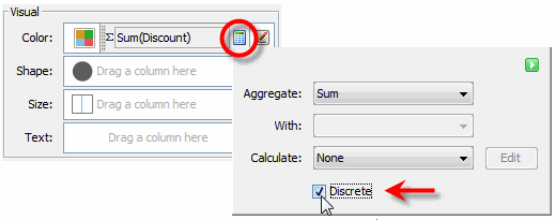
Displaying a Measure as Discrete Values

By default, dimensions are displayed as discrete values, and measures are displayed as a continuous range.

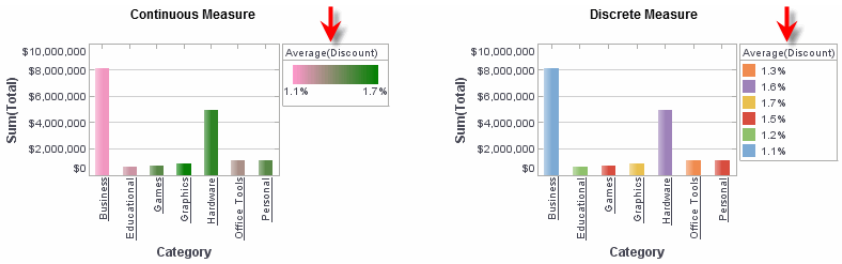
To force a measure to display as discrete values, follow the steps below:

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.

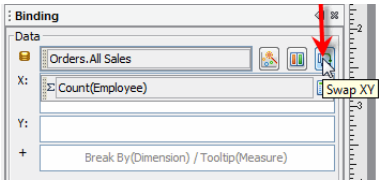


To convert a measure to a dimension, see *Converting between a Measure and Dimension*.

6.6.10 Rotating a Chart

To rotate a chart (switch the X- and Y-axes), follow the steps below:

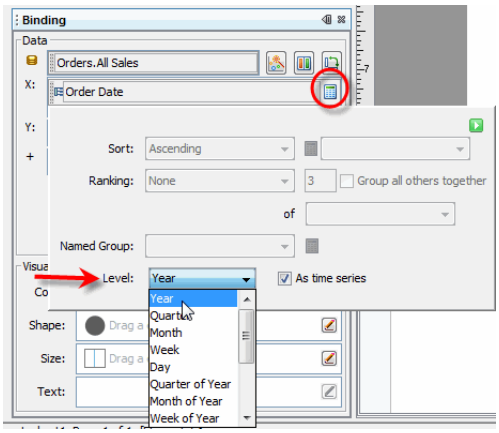
1. Click on the chart to select it. This opens the **Binding** panel.
2. Click the 'Swap XY' button in the **Binding** panel.



This will position the X-axis vertically, and position the Y-axis horizontally.

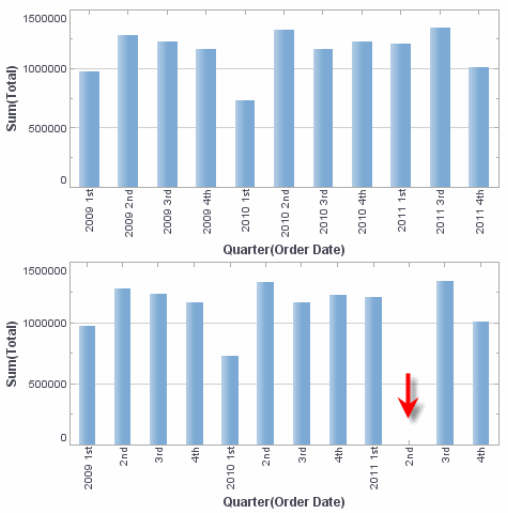
6.6.11 Setting Date Grouping Properties

When you use a date field as a dimension, you can specify the level of date grouping (month, year, etc.). To do this, click the 'Edit Dimension' button for the desired date field, and choose the desired grouping from the 'Level' menu.



If the data represents a series of measurements at uniformly-spaced time intervals, select 'As Time Series'. This forces the chart to display all measurement periods, even those for which no corresponding data is available. Otherwise, these "empty" periods are suppressed.

Figure 21. Effect of 'TimeSeries' option on display of missing data



It is important to set the format of the labels to the match the grouping level. For example, dates grouped by year should have a format such as "yyyy" to display just the year component of the date. For information on setting chart label formats, see *Chart Properties*.

6.6.12 Binding a Chart to Embedded Data

To populate a chart with data that you enter manually, use the 'Embedded Data' option. See *Binding a Table to Embedded Data* for more information.

6.7 Binding Data to a Section Element

Sections and Tables can both present tabular data. However, in a Section, each section row, or "band," is just a container for other elements. To display tabular data, each Text element in a section is bound to an appropriate data field. By adding Text elements to a Section and manually binding these elements, you can create very precise layouts.

The following sections explain how to create and bind data to a Section.

6.7.1 Creating a Section Using the Data Binding Wizard

Style Studio also allows you to automatic generate and bind section elements in the form of a pseudo-table, which you can then tailor to suit your needs.

Walkthrough

The simplest way to generate a Section is by using the Section Wizard. The Section Wizard lets you select the data binding, define grouping and summarization, and generate a section with appropriate section structure and elements. After the wizard generates the Section, you can adjust the element positions and sizes.

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 ‘Report’ node. In the right ‘Types’ panel, select ‘Blank Tabular Report’ and click ‘OK’.



3. Click the ‘Section’ button in the **Toolbox** panel to open the ‘Data Binding’ dialog box for a new Section.

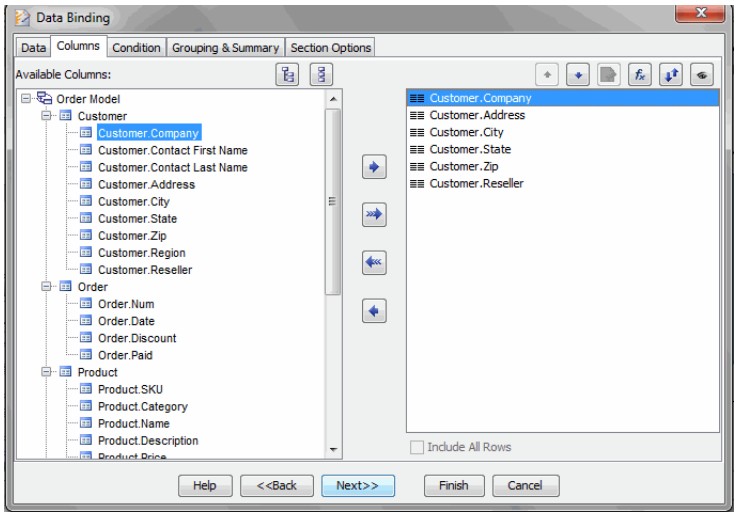
You cannot bind a Section to a Crosstab.

4. Under the **Data** tab, expand the ‘Orders’ data source, and select the ‘Order Model’ data model.

5. Under the **Columns** tab, select the ‘Customer’ node and click on the right-arrow button to add all the attributes from the ‘Customer’ entity.

6. *Remove* the following columns by using the left-arrow button:

   ‘Customer.Contact First Name’
   ‘Customer.Contact Last Name’
   ‘Customer.Region’

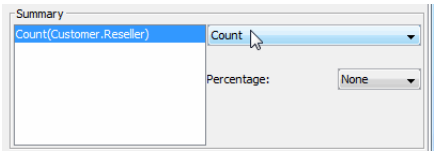


7. Click the **Grouping & Summary** tab.

8. Drag ‘Customer.State’ from the ‘Available Columns’ panel to the ‘Grouping’ panel.

9. Drag ‘Customer.Reseller’ from the ‘Available Columns’ panel to the ‘Summary’ panel.

10. Select the ‘Customer.Reseller’ field in the ‘Summary’ panel, and choose ‘Count’ for the summarization formula.

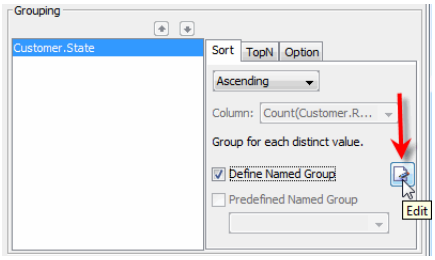


    This will provide a count of the customers in each state. (In this case, it does not matter which field is chosen for the summarization, because the count will be the same for all.)

You will now create a Named Group that groups data from states NJ and NY together into a single group called ‘Local’. (See *Named Groups* for more information about this kind of grouping.)

11. Click the ‘Customer.State’ field in the ‘Grouping’ pane, and select the ‘Define Named Group’ option.

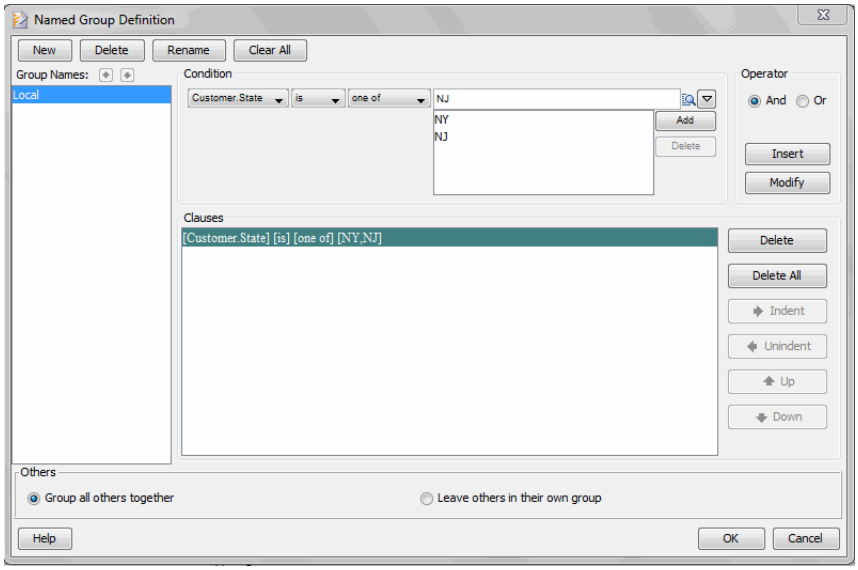
12. Click the ‘Edit’ button to open the ‘Named Group Definition’ dialog box.



13. In the ‘Named Group Definition’ dialog box, click ‘New’ and enter “Local” as the group name.

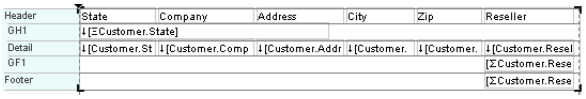
14. Specify the following condition:

```
[Customer.State][is][one of][NJ, NY]
```



15. In the ‘Others’ panel at bottom, select ‘Leave others in their own group’ and click ‘OK’.

16. Click ‘Finish’ to generate the section.



The Section Wizard generates a new Section based on the data binding and grouping definitions, with nested bands corresponding to the grouping levels. All generated Section elements are bound to the appropriate query fields.

The top-level header band contains text elements for the column labels. Header bands in each subsection contain fields for the group header, and footer bands in each subsection contain fields for the summarization columns. Detail fields appear in the detail band.

Finish the example by adjusting the appearance of the Section:

17. Delete the ‘Customer.State’ field from the Header band and from the Detail band.

18. Right-click on the ‘Customer.Company’ field in the Header band, and select ‘Edit’ from the context menu. Change the text to “Company”.

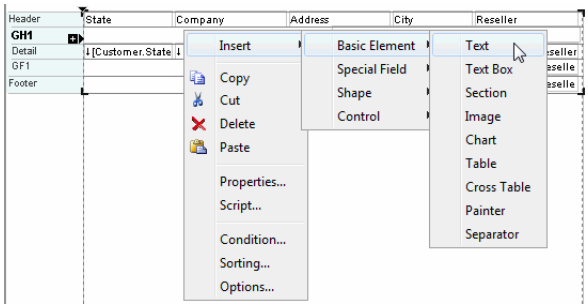
19. Repeat the above step to change the text of the ‘Customer.Address’, ‘Customer.City’, ‘Customer.Zip’, and

'Customer.Reseller' fields to "Address", "City", "Zip", and "Reseller", respectively.

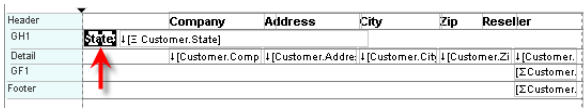
20. Ctrl-click all the fields in the Header band to select them, and use the Style Studio toolbar to change the font to 11pt Bold.

21. Drag the 'Customer.State' field in the GH1 band slightly to the right (to create additional space on the left side).

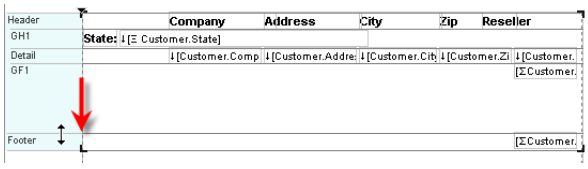
22. Right-click in the space to the left of the 'Customer.State', and choose 'Insert' > 'Basic Element' > 'Text' from the context menu. This changes the pointer to a crosshair.



23. Use the crosshair to draw a small Text element to the left of the 'Customer.State' field. Edit the Text element to display the text "State", and set the font to 11pt Bold.



24. Drag the bottom border of the GF1 band down to create additional space. (It is often easiest to move a band border by dragging in the left margin of the Section, to avoid accidently dragging an element within the Section.)



25. From the **Toolbox** panel, drag a Text element into the GF1 footer band. Edit the Text element to display the text "Count for ".

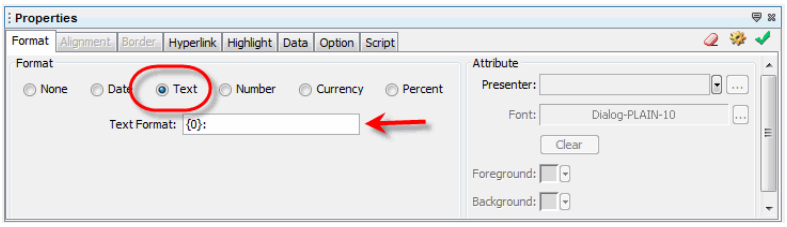
26. Ctrl-click on the 'Customer.State' field in the GH1 header band, and (while holding down Ctrl key) drag the field into the GF1

footer band. This creates a copy of the 'Customer.State' group field in that band.

27. Expand the Style Studio **Properties** panel.

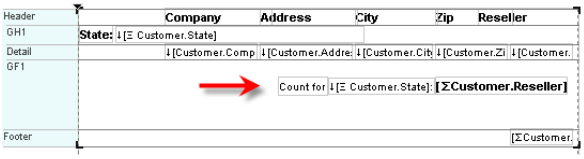
28. Select the 'Customer.State' field in the GF1 footer band.

29. Select the **Format** tab of the **Properties** panel. Select the 'Text' format, and enter the following (without quotes): "`{0}:`"



30. In the GF1 footer band, select the 'Customer.Reseller' element. Use the Style Studio toolbar to set the font to 11pt Bold.

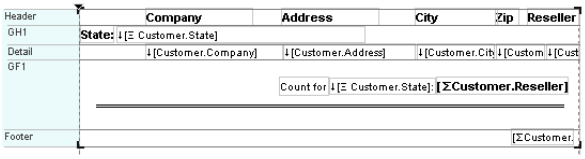
31. Position the three Text elements in the GF1 footer band as shown below.



32. From the **Toolbox** panel, drag a Separator element below the three Text elements in the GF1 footer band. Adjust the length as desired.

33. Right-click the Separator element, and select 'Properties' from the context menu. In the 'Properties' dialog box, select a double-line style, and click 'OK'.

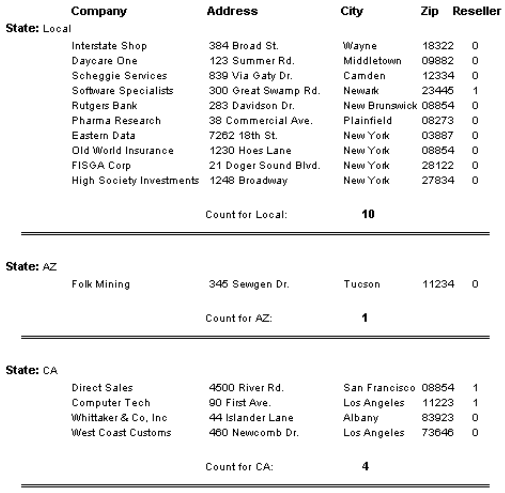
34. Resize and reposition other elements (as needed) to improve layout.



35. Right-click in the left margin of the Section, and select 'Properties'. This opens the 'Section Properties' dialog box.

36. In the left panel tree, select ‘Group Header’ under the ‘Group #1’ node. In the right panel, select ‘Keep Together’, and click ‘OK’. This will keep records from the same group together on the same page.

37. Preview your changes.



	Company	Address	City	Zip	Reseller
State: Local					
	Interstate Shop	384 Broad St.	Wayne	18322	0
	Daycare One	123 Summer Rd.	Middletown	09882	0
	Scheggie Services	839 Via Gaty Dr.	Camden	12334	0
	Software Specialists	300 Great Swamp Rd.	Newark	23445	1
	Rutgers Bank	283 Davidson Dr.	New Brunswick	08854	0
	Pharma Research	38 Commercial Ave.	Plainfield	08273	0
	Eastern Data	7262 18th St.	New York	03887	0
	Old World Insurance	1230 Hoes Lane	New York	08854	0
	FISGA Corp	21 Doger Sound Blvd.	New York	28122	0
	High Society Investments	1248 Broadway	New York	27834	0
		Count for Local:	**10**		
State: AZ					
	Folk Mining	345 Sewgen Dr.	Tucson	11234	0
		Count for AZ:	**1**		
State: CA					
	Direct Sales	4500 River Rd.	San Francisco	08854	1
	Computer Tech	90 First Ave.	Los Angeles	11223	1
	Whittaker & Co, Inc	44 Islander Lane	Albany	83923	0
	West Coast Customs	460 Newcomb Dr.	Los Angeles	73646	0
		Count for CA:	**4**		

See Also

Creating a Section Using Freehand Operations, to build a Section without the Wizard.

Named Groups

The default grouping procedure (see *Grouping and Summarization in Data Binding Wizard*) partitions a data 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 column partition by using *Named Grouping*. A Named Grouping is a set of custom conditions (i.e., rules) that partition the data set into desired groups. For example, you can create the named groups “East Coast,” “Central,” and “West Coast,” and define the conditions that determine which states belong in each of the named groups.

Named Groups in Data Binding Wizard

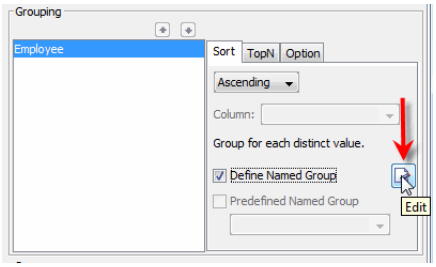
To define a Named Grouping for the data bound to a report element using the Data Binding Wizard, follow the steps below:

1. Click the **Grouping & Summary** tab.

2. If a grouping column has not yet been specified, add the desired grouping column to the 'Grouping' panel by clicking the right-arrow.

3. Select the column in the 'Grouping' panel for which you want to define named groups, and choose the 'Define Named Group' option.

4. Click the 'Edit' button to open the 'Named Group Definition' dialog box.



5. In the 'Named Group Definition' dialog box, click 'New' and enter a name for the first group.

6. In the 'Condition' panel, enter a condition that assigns particular column elements to the specified named group. (See *Filtering Data in Data Binding Wizard* for more information about specifying conditions.)

7. Click 'Append' to add the condition. Repeat the above two steps to add additional named groups and conditions.

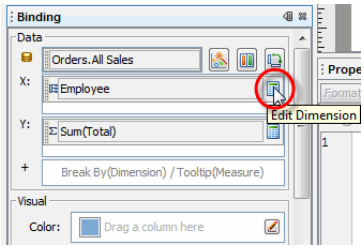
8. In the 'Others' panel at the bottom, specify how to treat column elements that do not satisfy any of the grouping conditions. 'Group all others together' groups these elements into a single catch-all group called "Others." 'Leave others in their own group' allows each of these distinct values to define its own group.

See the *Creating a Section Using the Data Binding Wizard* section for an example of data binding using named grouping.

Named Groups in Chart Binding

To define a Named Grouping for data bound to a Chart, follow the steps below:

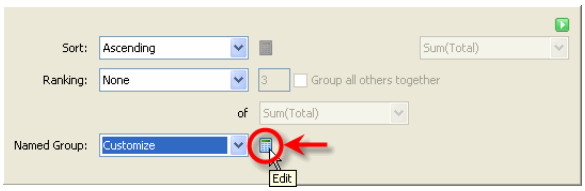
1. In the **Binding** panel of the Chart Editor, press the 'Edit Dimension' button next to the field for which you want to define named grouping.



See *Creating a Named Grouping* in *Advanced Data Mashup* for instructions on how to create a reusable named group.

This opens the 'Edit Dimension' panel. If a named grouping already exists in a Data Worksheet, you can simply select it from the 'Named Group' menu. However, to define a custom named grouping, continue with the steps below.

2. In the 'Named Group' menu, select 'Customize' and press the 'Edit' button. This opens the 'Named Group Definition' dialog box.



3. In the 'Named Group Definition' dialog box, press 'New' and enter a name for the first group.

4. In the 'Condition' panel, enter a condition that assigns particular column elements to the specified named group. (See *Filtering Data in Data Binding Wizard* for more information about specifying conditions.)

5. Click 'Append' to add the condition. Repeat the above two steps to add additional named groups and conditions.

6. In the 'Others' panel at the bottom, indicate how the chart should display column elements that do not satisfy any of the specified grouping conditions:

   a. 'Group all others together' groups these elements into a single catch-all group with the label "Others."

   b. 'Leave others in their own group' displays each of these elements as its own group.

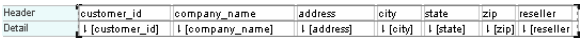
7. Press 'OK' to close the dialog box, and press the 'Apply' button (green arrow) in the 'Edit Dimension' panel to apply the grouping.

See the *Creating a Section Using the Data Binding Wizard* section for an example of data binding using named grouping.

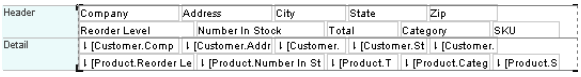
Section Options for Data Binding Wizard

The **Section Options** tab in the 'Data Binding' dialog box provides the following options:

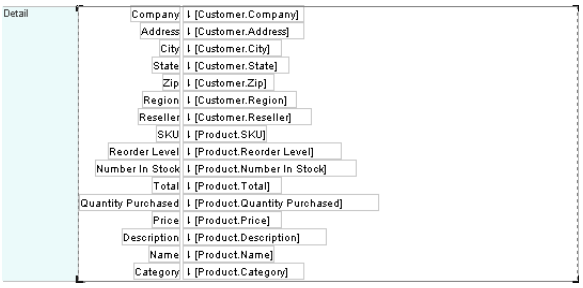
- **Use TextBox for fields**: Use Textbox elements for fields, instead of Text elements.
- **Display contents in table**: Display the content band data as a table contained within the content band, rather than as individual text elements.
- **Vertical Gap**: Vertical spacing between adjacent fields.
- **Horizontal Gap**: Horizontal spacing between adjacent fields.
- **Fit One Row**: Place all text fields on a single row.



- **Fit Multiple Rows**: Allow text fields to wrap across multiple rows.



- **As a Form**: Remove header band, and stack fields vertically in content band, side-by-side with their corresponding labels.



- **Suppress Blank Band**: Do not display empty bands.

See Also

Section Options for a Freehand Section, to set options for a freehand Section.

6.7.2 Modifying a Section Using Freehand Operations

After you have created a Section with the Section Wizard (see *Creating a New Table, Crosstab, or Section using a Wizard*), you can continue to make additional modifications by using *freehand operations*. You can also

create a Section using freehand operations. The following sections explain how to use freehand operations to create or modify an existing Section.

See Also

Binding Data to a Table Element, to create more tabular layouts with optional global styles.

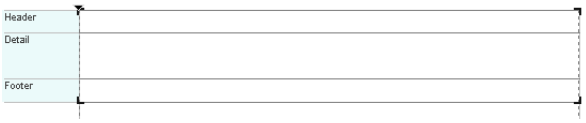
Creating a Section Using Freehand Operations

In *Creating a Section Using the Data Binding Wizard*, you learned how to use the Data Binding Wizard to bind a Section to a data source. You can also bind a Section to a data source by using freehand operations.

To create a new Section that displays a set of fields from a query, data model entity, or Data Block, follow the steps below:

1. Expand the Style Studio **Toolbox** panel.

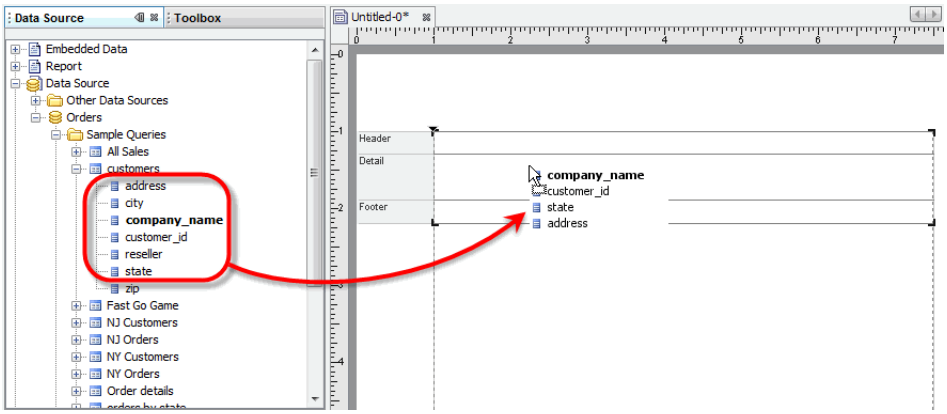
2. Drag a 'Section' element from the **Toolbox** panel to the desired location in the report. This creates a blank Section in the report.



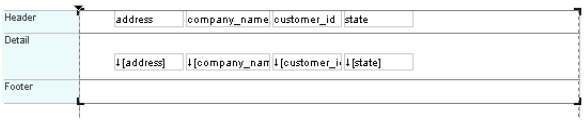
3. Open the Style Studio **Data Source** panel.

4. Expand a query, data model entity, or Data Block in the **Data Source** panel.

5. Click to select a field (or Ctrl-click to select multiple fields) from the data source, and drag the field(s) into the Detail band of the Section.



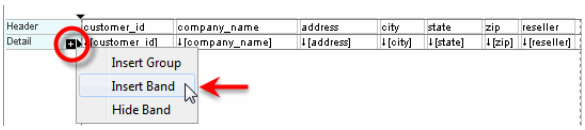
This binds the Section to the selected data source, and displays only the selected fields from result set. Detail records are displayed in the Detail band, and field headers are displayed in the Header band.



6. Modify this Section as desired using freehand operations. See *Modifying a Section Using Freehand Operations*.

Inserting Bands into a Section

To insert a new band into a Section, click in the left margin next to an existing band. Press the '+' icon and select 'Insert Band' from the context menu.



This inserts a new band below the selected band.

See Also

Inserting New Fields with Data Binding, to add a new column together with binding.

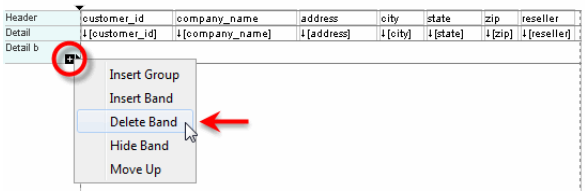
Changing a Section Element Data Binding, to change the data binding for a cell or column.

Deleting/Hiding a Section Band

To delete a Section band or hide a Section band, click in the left margin next to the desired band and press the '+' icon. Select one of the following from the context menu:

- 'Delete Band'
- 'Hide Band'

You cannot delete a band that is part of a group, although you can hide the band. To remove a grouping level entirely, see *Grouping Data in a Section*.

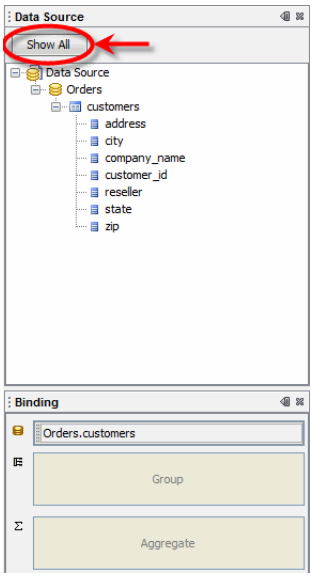


Inserting New Fields with Data Binding

You can insert one or more new fields into the Section to display data from the data source to which the Section is bound. To add a new field or fields to a Section, follow the steps below:

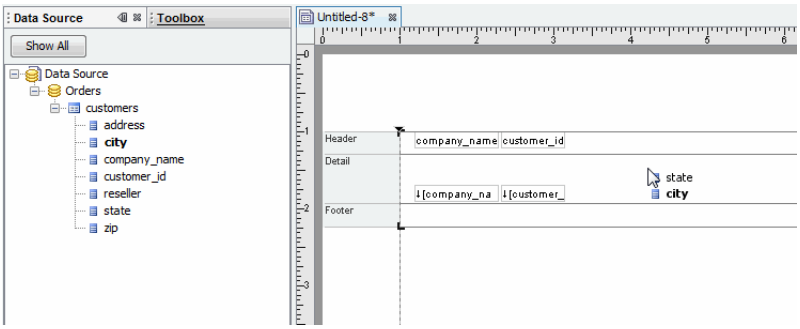
1. Click on the Section to select it. This opens the **Data Source** panel and **Binding** panel.

2. (Optional) To see only fields from the data source to which the Table is bound, disable 'Show All' in the **Data Source** panel.



3. Expand the data source (query, data model entity, or Data Block) to which the Section is bound.

4. Click to select a field (or Ctrl-click to select multiple fields) from the data source, and drag the field(s) into the Detail band of the Section.



Drop the field(s) in the desired location. This adds the new fields with their data bindings at the specified location.

See Also
Inserting Bands into a Section, to add blank bands.
Changing a Section Element Data Binding, to change the data binding for a field.

Moving Fields within a Section

To move text fields within a Section, see *Moving and Resizing Elements in a Section*. When you move or copy a bound Text element from one band to another, the data displayed by the Text element is determined by the band in which the Text element is placed.

For example, if you move a Text field from the Detail band to the GF 1 (grouping footer) band, the Text element will be automatically adapted to display aggregate data. (To change the aggregation method, see *Modifying Summary Properties*.) If you move a Text field from the Detail band to the GH 1 (grouping header) band, the Text element will be automatically adapted to display group data. (To change the grouping properties, see *Modifying Table Grouping Properties*.)

Changing a Section Element Data Binding

You can alter the binding of a Text element in a Section to display static text or values from a different field of data. To modify a Text binding, follow the steps below:

1. Expand the Style Studio **Properties** panel. (Click the 'Properties' link at the bottom of the Style Studio window.)

2. Click the desired Text element in the Section to select it.

3. Click the **Data** tab of the **Properties** panel. In the 'Binding' panel, make one of the following selections:

    a. To enter static text in the field, select the 'Text' option, and enter the desired contents.

    b. To bind the field to a different data column, select the 'Column' option, and choose the desired data column from the adjacent menu.

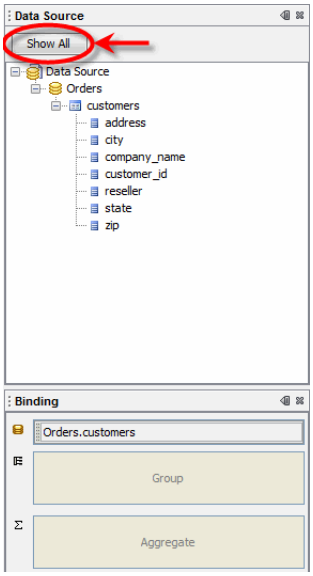
4. Click the 'Apply' button to submit the changes.

Changing the Section Data Source

To change the data source to which the Section is bound, follow the steps below:

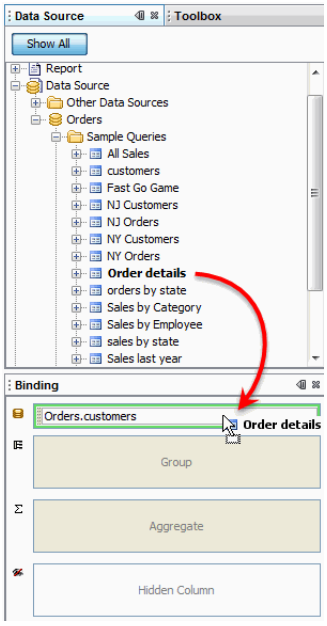
1. Click on the Section to select it. This opens the **Data Source** panel and **Binding** panel.

2. To see all available data sources, enable ‘Show All’ in the **Data Source** panel.



You can also drag fields from the new data source onto the Section itself.

3. Drag the desired data source (query, data model entity, or Data Block) from the **Data Source** panel onto the top field in the **Binding** panel.



This operation will generate a warning, which advises you that changing the binding will destroy the current Section configuration. Click ‘OK’ to proceed.

This rebinds the Section to the new data source.

Changing the Element Sort Order

To change the sorting order for fields in a Section, right-click the Section and select ‘Sorting’ from the context menu. This opens the ‘Sorting’ dialog box. The ‘Sorting’ dialog box is the same for Sections and for Tables. See *Changing the Column Sort Order* for further details.

Filtering Data in a Section

To apply a filtering condition to a Section, right-click the Section, and select ‘Condition’ from the context menu. This opens the ‘Condition’ dialog box. The ‘Condition’ dialog box is the same for Sections and for Tables. See *Filtering Data in a Table* for further details.

Grouping Data in a Section

There are two ways to group data within a Section. You can keep the grouping field in the same row as the detail records, or you can display the grouping field in its own group region. The following sections explain the two approaches.

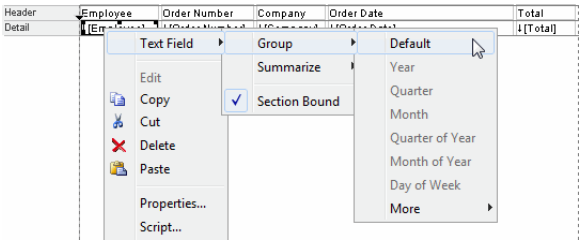
Displaying the Grouping Field in the Detail Row

To group the data in a Section and display the grouping field in the detail row, follow the steps below:

For a categorical field, select ‘Default’. For a Date field, select a Date level such as ‘Year’ or ‘Month’ (if the dates are not already grouped by the database).

1. Right-click on the Text element in the Section that represents the grouping field.

2. From the context menu, select ‘Text Field’ > ‘Group’, and choose the desired grouping.



This defines the field as a grouping field, and adds the selected field to the grouping area of the **Binding** panel. See *Modifying Table Grouping Properties* to make further adjustments to the grouping process.

Displaying the Grouping Field in a Group Region

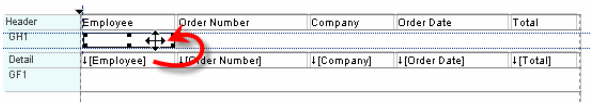
To group the data in a Section and display the grouping field in a dedicated grouping region, follow the steps below:

1. Click in the left margin next to the Detail band, and click the '+' icon. Select 'Insert Group' from the context menu.



   This inserts a new grouping region with both header and footer bands.

2. Drag the desired grouping field from the Detail band to the new grouping header region.



This defines the field as a grouping field in its own region, and adds the selected field to the grouping area of the **Binding** panel. See *Modifying Table Grouping Properties* to make further adjustments to the grouping process.

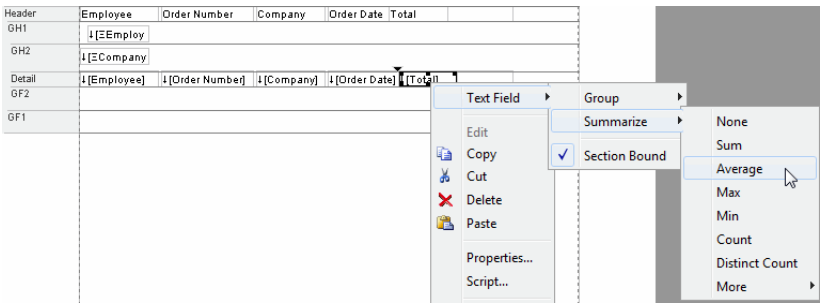
Summarizing Data in a Section

When you aggregate data in a Section, you can compute summaries on a group-by-group basis and on a global basis (e.g., grand total).

To create a summary based on a particular grouping level, follow the steps below:

1. Right-click on the Text element displaying the field that you want to summarize.

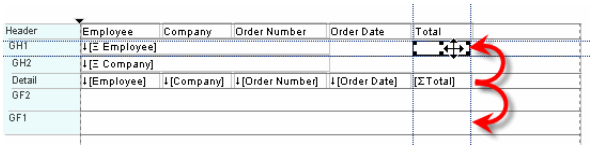
2. From the context menu, select 'Text Field' > 'Summarize', and choose the desired aggregation method (e.g., 'Average', 'Max', etc.)



This defines the field as a summary field and adds it to the summary area of the **Binding** panel. Because the field resides in the detail row, it only summarizes a single detail value. To summarize a particular group, or the entire table, proceed to the next steps.

To add a grouping region, see *Displaying the Grouping Field in a Group Region*.

3. To summarize a particular group, Ctrl-drag (or copy and paste) the summary field to the corresponding grouping header or footer region.

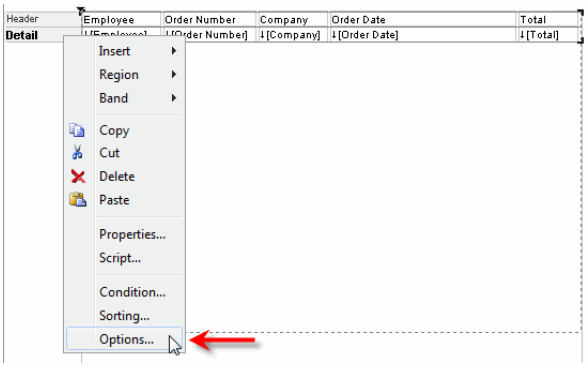


If the desired footer region is not visible, open the 'Section Properties' dialog box, and enable the footer visibility. See *Section Properties* for more information.

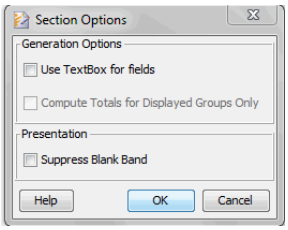
See *Modifying Summary Properties* to make further adjustments to the summarization process.

Section Options for a Freehand Section

To set options for an existing Section, right-click on the Section, and select 'Options' from the context menu.



This opens the 'Section Options' dialog box.



The ‘Section Options’ dialog box provides the following options:

- **Use TextBox for fields**: Use Textbox elements for fields, instead of Text elements.
- **Suppress Blank Band**: Do not display empty bands.

The default values of these settings are specified in the ‘Preferences’ dialog box. See *Setting Global Section Options*.

Binding a Section to Embedded Data

To populate a Section with data that you enter manually, use the ‘Embedded Data’ option. See *Binding a Table to Embedded Data* for more information.

6.7.3 Running Total

Walkthrough

A ‘Running Total’ has much in common with a summary field. The difference is that a ‘Running Total’ can be maintained independently of any predefined grouping characteristics. You can use running totals to show how a field increments row by row, or apply a formula to a value irrespective of the grouping and summarization.

Consider the following example:

1. Click the ‘New’ button in the Style Studio toolbar. This opens the ‘New Asset’ dialog box.

2. In the left panel, select the ‘Report’ node. In the right ‘Types’ panel, select ‘Blank Tabular Report’ and click ‘OK’

3. Expand the Style Studio **Toolbox** panel.



4. Click the ‘Section’ button in the **Toolbox** panel to open the ‘Data Binding’ dialog box for a new Section.

5. Under the **Data** tab, select the ‘Order Model’ data model from the ‘Orders’ data source. Under the **Columns** tab, select the following columns:

   ‘Customer.Company’
   ‘Product.Name’
   ‘Product.QuantityPurchased’
   ‘Product.Price’
   ‘Product.Description’
   ‘Order.Discount’

6. Under the **Grouping & Summary** tab, group on the 'Customer.Company' field, and summarize on the 'Product.Price' field.

7. Click 'Finish'.

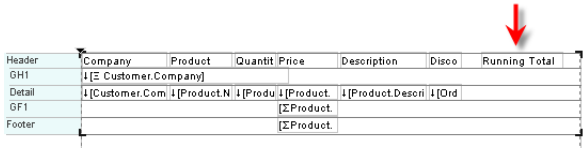
A 'Running Total' can also be used with a summarized table.

8. Create space for the new running total column:

   a. Rename the header fields as follows:
   'Product.Quantity Purchased' > 'Quantity'
   'Product.Name' > 'Product'
   'Customer.Company' > 'Company'
   'Product.Price' > 'Price'
   'Product.Description' > 'Description'
   'Order.Discount' > 'Discount'

   b. Move the existing header and content fields to the left to create empty space on the right side of the section.

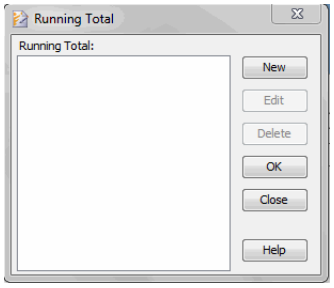
   c. In the space you created, add a text element in the header with text "Running Total".



   To do this, right-click the Section, and choose 'Insert' > 'Basic Element' > 'Text' from the context menu. Then click in the Header band with the crosshair to place the Text element in the desired location.

9. Define the running total by following the steps below:

   a. Right-click the Section, and choose 'Insert' > 'Special Field' > 'Running Total' from the context menu. This opens the 'Running Total' dialog box.



b. In the ‘Running Total’ dialog box, click the ‘New’ button. This opens the ‘Running Total Property’ dialog box.

To edit the running total at a later time, click the ‘Running Total’ button in the **Toolbox** panel. This opens the ‘Running Total’ dialog box. Click ‘Edit’ to modify an existing running measure.

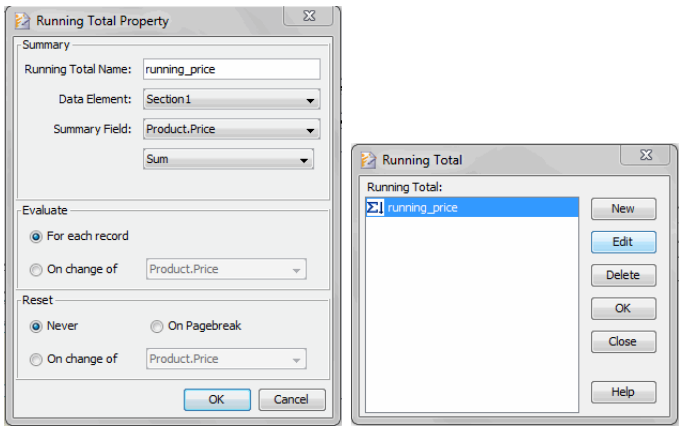
c. In the ‘Running Total Name’ field, enter “running_price”.

d. In the ‘Data Element’ menu, select “Section1”. This is the report element with which the running total should be associated.

e. From the ‘Summary Field’ menu, select ‘Product.Price’. Set ‘Sum’ as the formula.

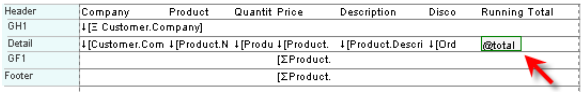
(Optional) In the ‘Evaluate’ panel, you can choose ‘For each record’ to evaluate the running total increment for each record. Choose ‘On change of’ to evaluate the running total increment at each group change. Similar options are available to reset the running total.

f. Click ‘OK’. The running total is now listed in the ‘Running Total’ dialog box.

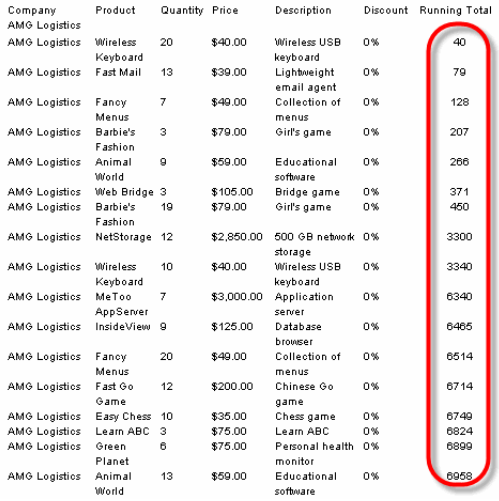


g. Click to select the ‘running_price’ measure, and click ‘OK’. The pointer now turns into a crosshair.

10. Click with the crosshair in the Detail band under the ‘Running Total’ header. This adds the running total to the Detail band. Resize and format the Text element as desired.



11. Preview the report.



Company	Product	Quantity	Price	Description	Discount	Running Total
AMG Logistics						
AMG Logistics	Wireless Keyboard	20	$40.00	Wireless USB keyboard	0%	40
AMG Logistics	Fast Mail	13	$39.00	Lightweight email agent	0%	79
AMG Logistics	Fancy Menus	7	$49.00	Collection of menus	0%	128
AMG Logistics	Barbie's Fashion	3	$79.00	Girl's game	0%	207
AMG Logistics	Animal World	9	$59.00	Educational software	0%	266
AMG Logistics	Web Bridge	3	$105.00	Bridge game	0%	371
AMG Logistics	Barbie's Fashion	19	$79.00	Girl's game	0%	450
AMG Logistics	NetStorage	12	$2,850.00	500 GB network storage	0%	3300
AMG Logistics	Wireless Keyboard	10	$40.00	Wireless USB keyboard	0%	3340
AMG Logistics	MeToo AppServer	7	$3,000.00	Application server	0%	6340
AMG Logistics	InsideView	9	$125.00	Database browser	0%	6465
AMG Logistics	Fancy Menus	20	$49.00	Collection of menus	0%	6514
AMG Logistics	Fast Go Game	12	$200.00	Chinese Go game	0%	6714
AMG Logistics	Easy Chess	10	$35.00	Chess game	0%	6749
AMG Logistics	Learn ABC	3	$75.00	Learn ABC	0%	6824
AMG Logistics	Green Planet	6	$75.00	Personal health monitor	0%	6899
AMG Logistics	Animal World	13	$59.00	Educational software	0%	6958

6.7.4 Subreports

A subreport is a report that is embedded inside another report. The subreport can only be embedded within a section element, and is displayed within this section band when the report is generated. Because a section band is repeated for every row in the query that the section is bound to, the subreport is also repeated as part of the band repetition.

A subreport is often used to present detail information for a band. For example, if a section is showing the product sales summary, the subreport could be used to present the sales details. This means a subreport needs to accept parameters from the band, so each subreport would present different information when it is repeated.

To design a subreport, follow the general steps below:

Note: Report paper size is ignored when a report is printed inside another report as a subreport.

1. Design the report that will be used as the subreport. If desired, a report bean can also be used as a subreport. See *Walkthrough: Designing the Subreport*.
2. Design the master report.
3. Embed the subreport within a Section in the master report. See *Walkthrough: Embedding the Subreport*.
4. Define the parameter links from the Section band to the subreport.

In general, do not use headers, footers, or decorative elements in subreports.

Walkthrough: Designing the Subreport

Walkthrough

In this example, the subreport will contain a Table bound to the 'customers' query. The 'customers' query accepts a 'state' parameter, and lists all customers from the specified state.

1. Click the 'New' button in the Style Studio toolbar. This opens the 'New Asset' dialog box.

2. In the left panel, select the 'Report' node. In the right 'Types' panel, select 'Blank Tabular Report' and click 'OK'.

3. Expand the Style Studio **Toolbox** panel.



4. Click the 'Table' button in the **Toolbox** panel to open the 'Data Binding' dialog box for a new Table.

5. Under the **Data** tab, bind the table to the 'customers' query from the 'Sample Queries' folder of the 'Orders' data source.

6. In the **Columns** tab, click the left-arrow to *remove* the following columns from the 'Columns' pane:

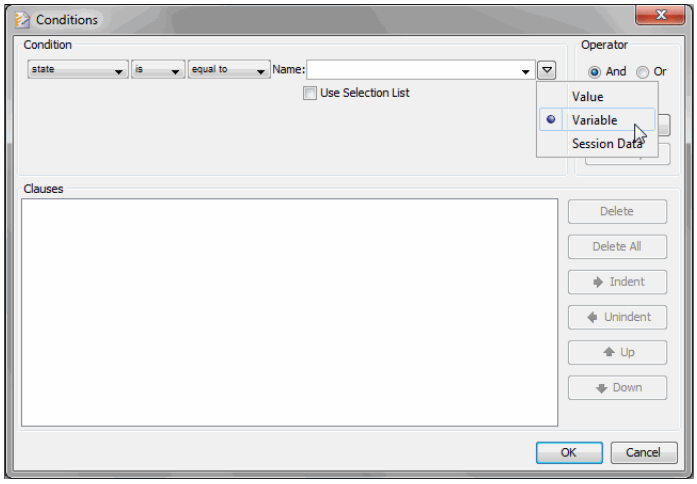
   'customer_id'
   'state'
   'zip'

Note: The parameter for the subreport must be set prior to its insertion into the main report.

7. In the **Condition** tab, specify the beginning of the condition:

   ```
   [state][is][equal to]
   ```

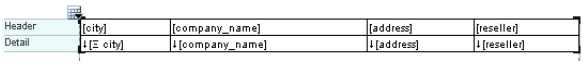
8. Select the 'Variable' option in the menu at right.



9. Type “state” into the ‘Label’ field, and click ‘Append’ to insert the complete condition:

   ```
   [state][is][equal to][$(state)]
   ```

10. In the **Grouping & Summary** tab, drag ‘city’ into the ‘Grouping’ pane, and click ‘Finish’.



11. Save this subreport as “*customer_subreport.srt*”. The example continues in *Walkthrough: Embedding the Subreport*.

Walkthrough: Embedding the Subreport

Walkthrough

This example continues from *Walkthrough: Designing the Subreport*. Now you will design the master report, and then embed the subreport in the master report.

1. Click the ‘New’ button in the Style Studio toolbar. This opens the ‘New Asset’ dialog box.

2. In the left panel, select the ‘Report’ node. In the right ‘Types’ panel, select ‘Blank Tabular Report’ and click ‘OK’.

3. Expand the Style Studio **Toolbox** panel.



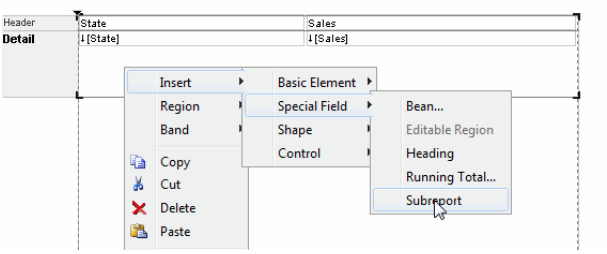
4. Click the ‘Section’ component in the **Toolbox** panel to open the ‘Data Binding’ dialog box for a new section.

5. Under the **Data** tab, bind the section to the ‘sales by state’ query in the ‘Sample Queries’ folder of the ‘Orders’ data source.

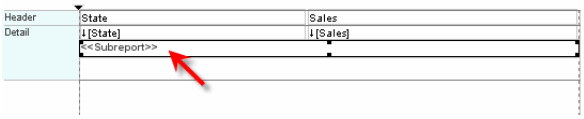
6. Click ‘Finish’ to exit the ‘Data Binding’ dialog box.

7. Click the Section to select it. Drag the bottom border of the section downward to create more space within the content band.

8. Right-click the section, and select ‘Insert’ > ‘Special Field’ > ‘Subreport’ from the context menu.

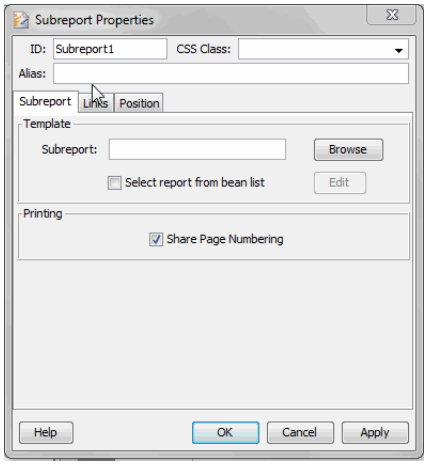


9. Click in the content band of the Section to embed the subreport.



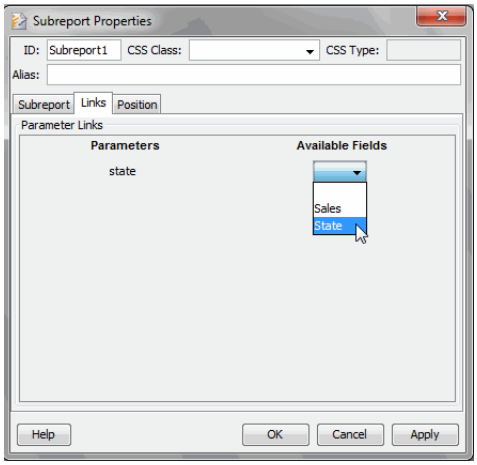
10. Right-click the Subreport element and select ‘Properties’ from the context menu. This opens the ‘Subreport Properties’ dialog box.

11. Select the **Subreport** tab, and click the ‘Browse’ button in the ‘Template’ tab. Select the subreport you previously created, *customer_subreport.srt*.



(Note: To open the selected subreport for editing, click the ‘Edit’ button. Enable the ‘Share Page Numbering’ option to continue the subreport page numbering set by “`{P}`” and “`{N}`” header/footer tags across multiple instances of the rendered subreport. When this option is disabled, numbering restarts at each subreport occurrence.)

12. Click the **Links** tab, and select ‘State’ in the ‘Available Fields’ drop-down list.



This associates the 'state' parameter of the subreport with the 'State' field of the main report. As a result, the subreport that appears within each band contains only the table rows corresponding to the 'State' value of that particular band.

13. Preview the report. The master report now displays a section with the subreport embedded in each band.

State Sales
AZ 456333.85

city	company_name	address	reseller
Tucson	Folk Mining	345 Sewgen Dr.	0

CA 1665880.05

city	company_name	address	reseller
Albany	Whittaker & Co, Inc	44 Islander Lane	0
Los Angeles	West Coast Customs	460 Newcomb Dr.	0
	Computer Tech	90 First Ave.	1
San Francisco	Direct Sales	4500 River Rd.	1

CO 838303.2

city	company_name	address	reseller
Boulder	Ernst Handel	88 Rio Grand Ave.	1
Denver	Quick Stop	71 Mulberry St.	0

CT 161689.5

city	company_name	address	reseller
Hartford	Loctite Corp.	5 Baker's Lane	0

FL 377534.8

city	company_name	address	reseller
Orlando	Fast Transportation	77 Druguz Blvd.	0

IL 558702.8

city	company_name	address	reseller
Chicago	Specialty Retail	88 Specialty Dr.	1

MA 2126975.2

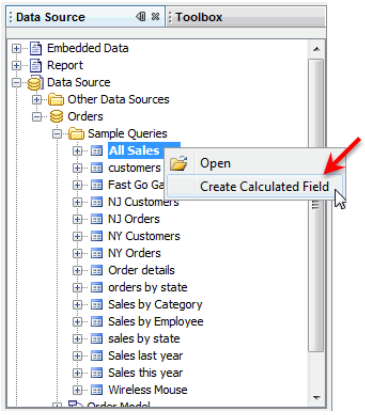
city	company_name	address	reseller
Boston	World Software Resellers	563 Uptown Dr.	1
	Ubermeyer	8300 Cleveland St.	0

6.8 Calculating a New Query Field

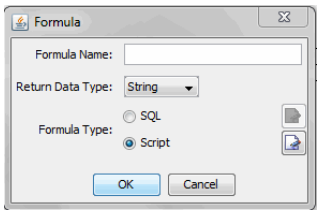
You can create a new query field (also known as a *formula column*) by performing a calculation on existing query fields. To create a new field, follow the steps below:

1. Expand the Style Studio **Data Source** panel.

2. Right-click on the query to which you want to add the new field, and select ‘Create Calculated Field’ from the context menu.



This opens the ‘Formula’ dialog box.



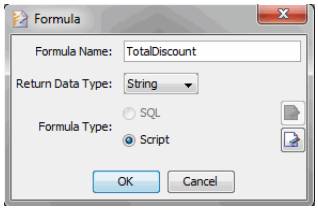
3. With the ‘Formula’ dialog box open, follow the steps below:

   a. In the ‘Formula Name’ field, enter a name for the new column.

   b. For ‘Return Data Type’, select the data type of values that will populate the new column. This data type should match the type of value returned by your script.

Note: In cases where the data binding specifications cannot be merged into the query, the ‘SQL’ option is not available. (This may be due to the presence of grouping and aggregation at the query level.)

   c. For ‘Formula Type’ select ‘SQL’ or ‘Script’, and then click the corresponding ‘Edit’ button. This opens the Script Editor.



4. Enter the desired script, using the syntax (SQL or JavaScript) that you specified in the previous step. To data values, use the following syntax:

   a. To access the value in another column of the *current* row, use the "`field['field name']`" syntax.

   b. To access the value in another column of a row at a certain offset (i.e., relative position) from the current row use the following syntax:

      ```
      field[relative position]['field name'];
      ```

      For example, `field[-1]['Product.Total']` contains the value in the `Product.Total` field for the previous record.

   For more information on scripting, see *Advanced Report Scripting*. For a simple example of a formula column script, see *Walkthrough: Selecting and Creating Columns in Data Binding Wizard*.

5. Close the Script Editor, and click 'OK' to close the 'Formula' dialog box. This adds the new formula column (indicated by the "*fx*" icon) to the query.

To edit an existing calculated field, right-click the desired column, and select 'Edit' from the context menu. This reopens the Script Editor, where you can proceed to edit the expression.

See Also

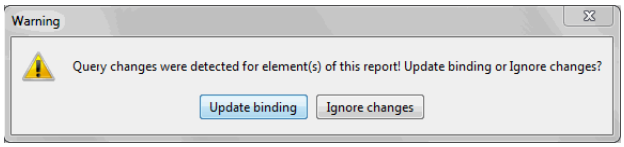
Creating a Formula Column in Data Binding Wizard, to create fields at Wizard level.

Creating a New Query, in *Advanced Data Modeling*, to create fields at the query level.

Defining Expression Attributes, in *Advanced Data Modeling*, to create fields at the data model level.

6.9 Updating Data Binding After Query Modification

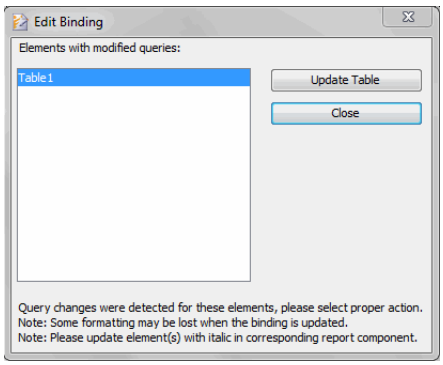
If the structure of the underlying data source (query, Worksheet, etc.) changes, any existing bindings that utilize the data source will generate the following alert when you open the report in Style Studio.



To proceed with the original binding, press the ‘Ignore changes’ button. To adapt the binding to the changes in the data source, press the ‘Update binding’ button. This opens the ‘Edit Binding’ dialog box.

Follow the steps below to update the bindings:

1. In the ‘Edit Binding’ dialog box, select the element you wish to update in the ‘Elements with modified queries’ list.



2. Press the ‘Update Table’ button to add or remove columns from the table as necessary to match the altered query.

3. Repeat the above steps for any elements listed the ‘Elements with modified queries’ panel.

4. Press ‘Close’ to dismiss the dialog box.

7 Advanced Topics

This section discusses various advanced topics, some of which are available only in the Enterprise products (Style Intelligence and Style Report EE).

7.1 Form Design

A form is a collection of input fields into which user can enter information. User interactions with the form elements are generally handled by server-side Java objects.

The following sections present some basic information about the available form elements.

See Also

Interactive Forms, in *Advanced Report Scripting*, for information about processing.

Parameter Sheets, for common ways of using form elements to acquire user input.

7.1.1 Accessing the Form Controls

To access the form controls, expand the Style Studio **Toolbox** panel. The controls are available in the **Form** panel. To add a form element to the report, simply drag the element from the **Toolbox** panel to the desired location.

Form fields do not need to be consecutive in the element flow. However, when designing a form, it is beneficial for ease of positioning to place form elements within a *Section Element*. It is also useful to add text elements next to form fields to provide labeling.

7.1.2 Form Elements

Each form consists of one or more fields. Fields are on-screen controls which can be used by end users to interact with the report. They are most commonly used in *Parameter Sheets*, but can be placed anywhere within a report as well.

Table 13. Form Controls

ELEMENT	FUNCTION
Button	Submits the form to the server when activated.
ImageButton	Submits the form to the server when activated.
CheckBox	Enters boolean values.
RadioButton	Make mutually exclusive selections.
Choice	Select choice from a list.
List	Select one or more choices from a list.

DateCombo	Specify date.
TextField	Enter single line text data.
TextArea	Enter multi-line text data.

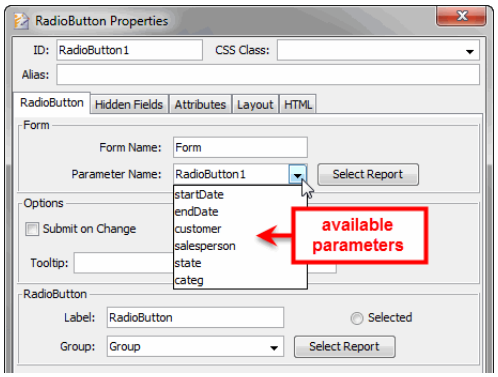
All form field elements are *float* elements. This means they share the layout properties of float elements, and can be positioned using an anchor.

To access the properties of a form element, right-click the element and select 'Properties' from the context menu. This opens the relevant 'Properties' dialog box.

A 'Submit' button submits only values from form elements which have the same 'Form Name' as the 'Submit' button itself.

Every form field element has 'Form Name' and 'Parameter Name' properties. The 'Form Name' is used to group fields into one form. All elements sharing the same 'Form Name' belong to the same form, and are packaged into a single *repletRequest* object to be forwarded to the server. The *RepletRequest* object contains values from the fields in the form.

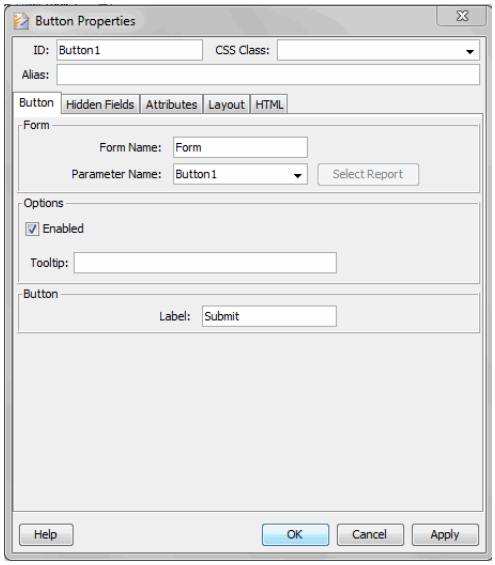
The field values are each identified by the 'Parameter Name'. The 'Parameter Name' should exactly match the name of the report parameter into which the value will be passed. Therefore, it is best to select the desired report parameter from the list provided by the 'Parameter Name' menu.



Button



The Button element allows the user to submit the selections they have made on a form. To configure the Button, right-click the element, and select 'Properties' from the context menu. This opens the 'Button Properties' dialog box.



Use the 'Label' attribute under the **Button** tab to specify the label that appears on the button.

Image Button



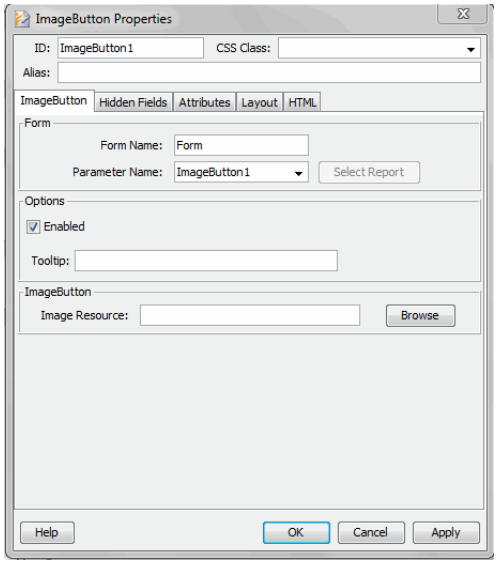
The Image Button is functionally the same as a *Button*, but displays a user-supplied image. To configure the Image Button, follow the steps below:

1. Right-click on the Image Button, and select 'Properties' from the context menu. This opens the 'ImageButton Properties' dialog box.

2. Select the **ImageButton** tab.

3. In the 'Image Resource' field, specify the image file by resource name.

   The image resource file is loaded from the classpath (specified on the **Classpath** tab of the 'Style Studio Configuration' dialog box). Make sure that the directory which contains the image is on the classpath. (See *Configuring Style Studio* for more information).

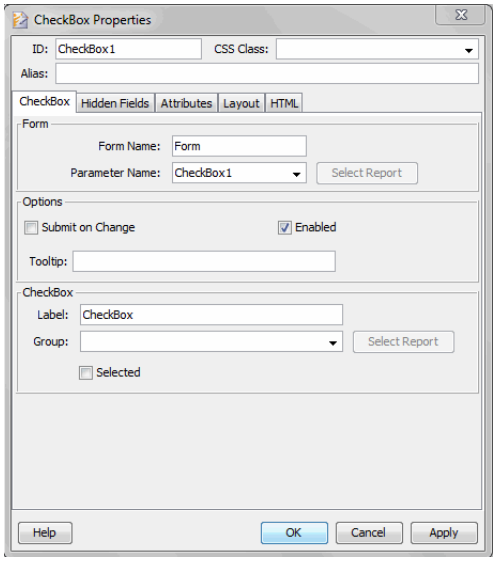


Note that the Image Button does not add any additional decoration to the image. The look-and-feel of the Image Button is completely determined by the image that you speicfy.

Check Box

A CheckBox field represents a Boolean value. To configure the Check Box, right-click the element, and select 'Properties' from the context menu. This opens the 'CheckBox Properties' dialog box.

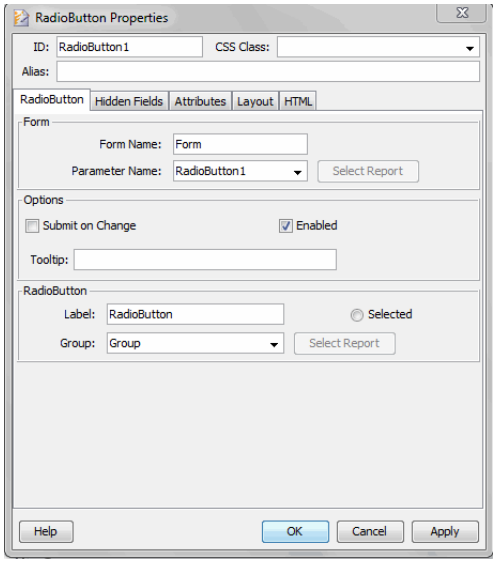
The Check Box has a 'Label' property, which specifies the text at the right side of the CheckBox icon.



Radio Button



The Radio Button has a similar appearance to a *Check Box*. To configure the Radio Button, right-click the element, and select 'Properties' from the context menu. This opens the 'RadioButton Properties' dialog box.



In addition to the 'Label' and default state attributes, each Radio Button has a 'Group' name. The Radio Button 'Group' is used to group radio buttons in the same form, with the result that Radio Buttons in the same group make mutually exclusive selections. When one Radio Button is selected, all other Radio Buttons in the same group are automatically deselected.

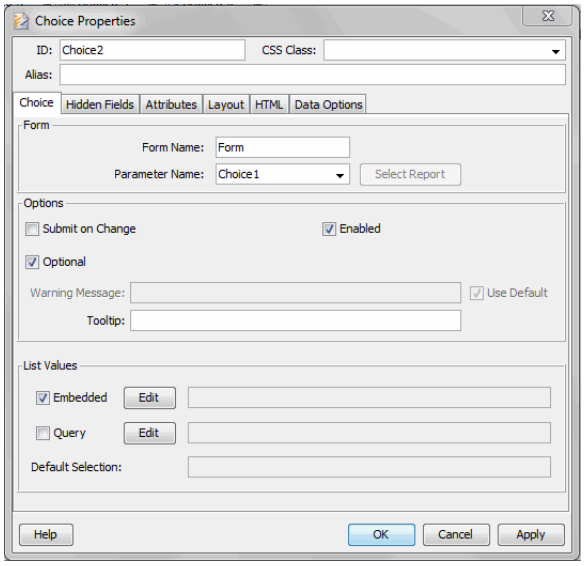
Choice

A Choice element is a menu from which the user can select a single item. To add a Choice element and populate the menu with values, follow the steps below:



1. Expand the Style Studio **Toolbox** panel. Drag a Choice component from the **Toolbox** panel to the desired location in the report.

2. Right-click the Choice element, and select 'Properties' from the context menu. This opens the 'Choice Properties' dialog box.



3. Select the **Choice** tab.

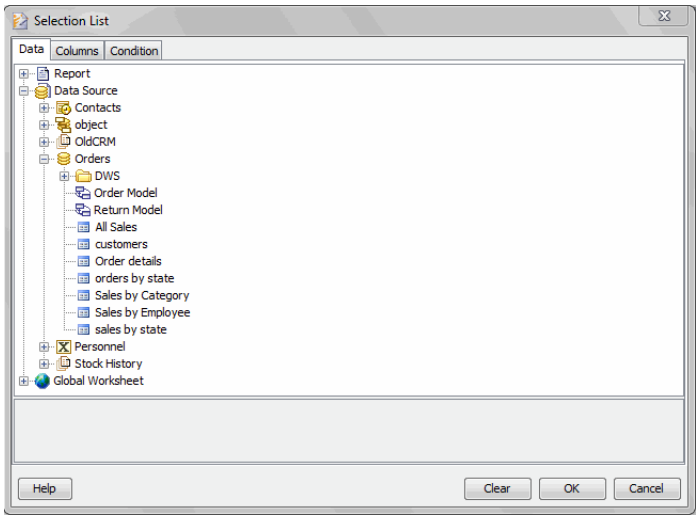
4. To populate the menu options from an existing data source, follow the steps below:

If you select the 'Query' option in addition to the 'Embedded' option, the query results are merged with the embedded list values.

a. In the 'List Values' panel, select the 'Query' option and press 'Edit'. This opens the 'Selection List' dialog box.

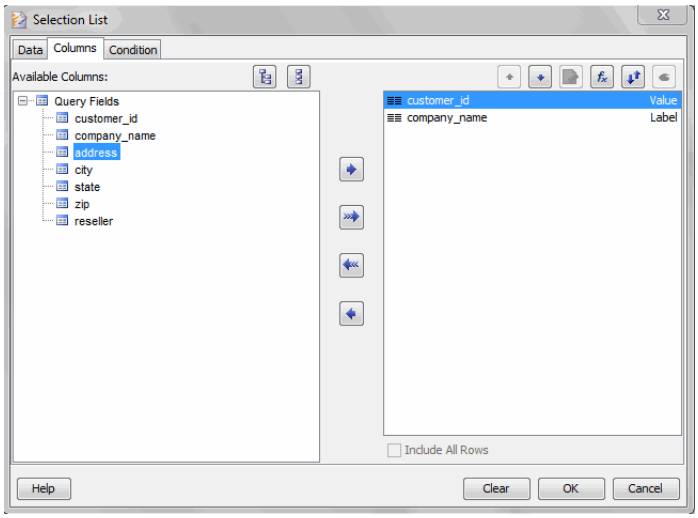


b. Select the **Data** tab. Choose the data source that will provide the labels and values for the Choice element.



If you want the labels and values to be the same, select only a single column.

c. Select the **Columns** tab. Choose one column to provide the labels for the Choice element, and one column to provide the values for the Choice element.



The *labels* determine the choices that are displayed to the user in the menu. The *values* determine the corresponding data that is submitted when the user makes a selection. To swap the 'label' field with the 'value' field, click the 'Move up' or 'Move down' arrow above the right panel. For information about the other buttons, see *Binding Data to a Table Element*.

To adapt the menu choices based on user input, specify a parameterized condition. See *Parameterizing a Condition*.

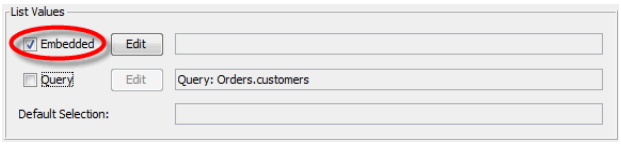
d. (Optional) Select the **Condition** tab. Specify a condition to filter the options (labels/values) displayed by the Choice element. See *Filtering Data in Data Binding Wizard* for more information on specifying a filter.

e. Press 'OK' to close the 'Selection List' dialog box.

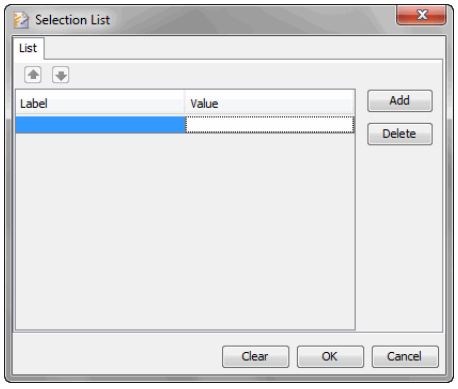
5. To populate the menu options by manually entering the label/value pairs, follow the steps below:

If you select the 'Embedded' option in addition to the 'Query' option, the embedded values are merged with the query results.

a. In the 'List Values' panel, select the 'Embedded' option and press 'Edit'. This opens the 'Selection List' dialog box.



b. Press the 'Add' button to add a new label/value pair. Double-click on a label or value to edit the text.



The *labels* determine the choices that are displayed to the user in the menu. The *values* determine the corresponding data that is submitted when the user makes a selection.

c. Check the 'Selected' box to make the corresponding label/value pair the default option in the Choice box.

d. Repeat the above steps to add additional labels.

e. To reorder the label/value pairs, press the arrow buttons above the label/value list.

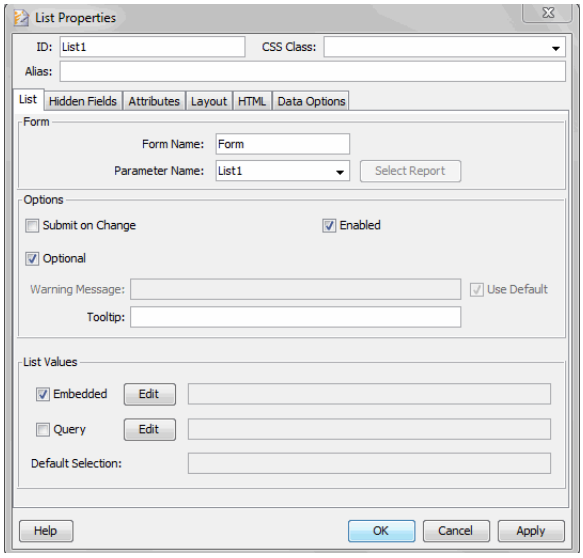
f. Press 'OK' to close the 'Selection List' dialog box.

6. (Optional) To enable the Choice element to submit the user's selection immediately (without requiring a 'Submit' button), enable the 'Submit on Change'. This is especially useful in conjunction with parameter staging, because it allows the user's selection to enable the next control in the input sequence. See *Staged Parameters* for more information.

7. Press 'OK' to close the 'Choice Properties' dialog box.

List

The List element is similar to the *Choice* element, but allows the user to make multiple selections. To configure the List element, right-click the element, and select 'Properties' from the context menu. This opens the 'List Properties' dialog box.



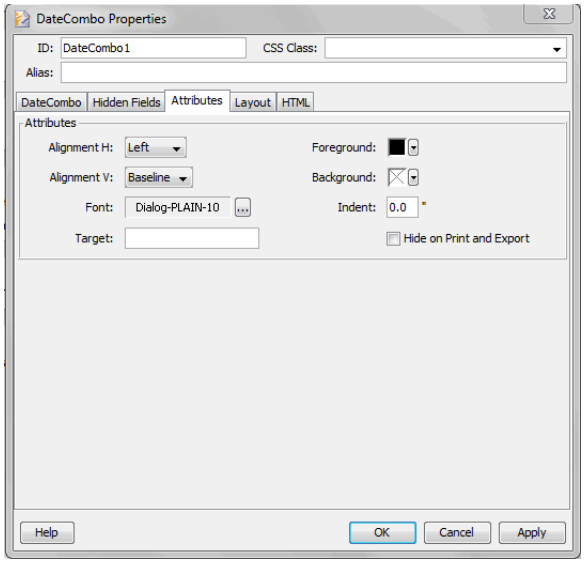
See the *Choice* element for information about how to display selections in the list by specifying a query or by entering a manual list in the 'Selection List' dialog box.

Date Combo



A Date Combo element allows the user to select a date using a calendar widget. To configure a Date Combo element, follow the steps below:

1. Right-click the Date Combo element, and select 'Properties' from the context menu. This opens the 'DateCombo Properties' dialog box.



2. (Optional) Click in the 'Default' field, and enter the default date setting.

3. (Optional) Select the ‘Prompt Time’ options to prompt the user to enter a time.

4. (Optional) Select the ‘Submit on Change’ option to allow the user to submit a date by simply clicking the desired day on the calendar popup (without clicking the popup’s ‘OK’ button).

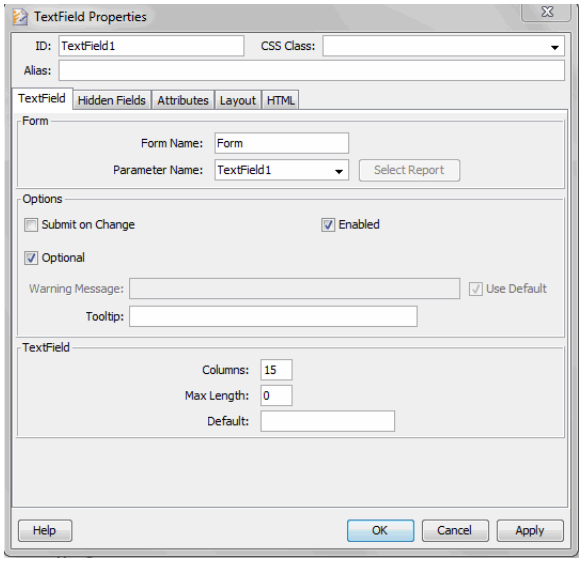
5. Click ‘OK’ to close the dialog box.

Text Field



The Text Field element is a simple control for entering a single line of text. To configure the control, follow these steps:

1. Right-click on the element and select ‘Properties’ from the context menu. This opens the ‘TextField Properties’ dialog box.



2. Select the **TextField** tab.

3. In the ‘TextField’ panel, enter a value for the ‘Columns’ attribute to set the default size for the Text Field.

4. Enter a value for ‘Max Length’ to limit the maximum number of characters that the user can enter in the Text Field.

5. In the ‘Default’ field, enter the desired default text to appear in the Text Field.

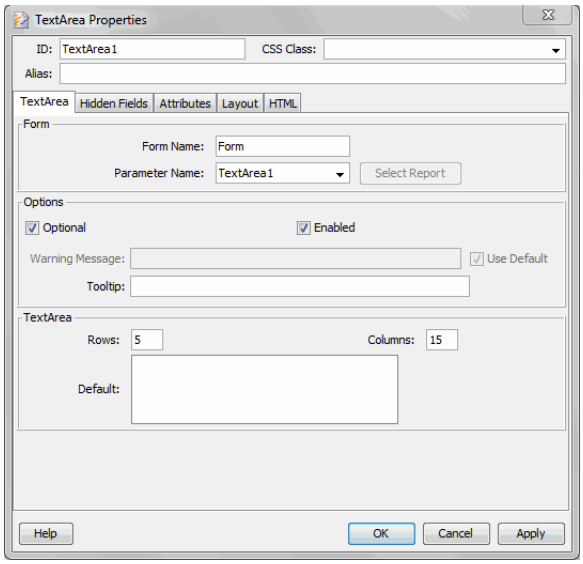
6. Click ‘OK’ to close the dialog box.

Text Area



The Text Area element is a multi-line text editor. To configure the control, follow these steps:

1. Right-click on the element and select 'Properties' from the context menu. This opens the 'TextArea Properties' dialog box.



2. Select the **TextArea** tab.

These values do not limit the amount of the text that the user can enter.

3. In the 'TextArea' panel, enter values for the 'Rows' and 'Columns' fields to specify the default size of the Text Area.

4. In the 'Default' field, enter the default text to appear in the Text Area.

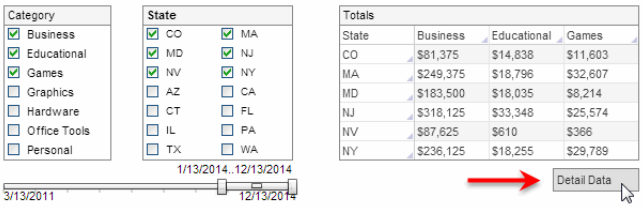
5. Click 'OK' to close the dialog box.

7.2 Sharing Filters with a Dashboard

If users will access a report by drilling-down from a dashboard, you should configure shared filters to synchronize the data displayed in the report with the selections that the user has made in the dashboard.

For example, in the dashboard below the user has made the following selections:

- 'Category' is one of {Business, Educational, Games}
- 'State' is one of {CO, MA, MD, NJ, NV, NY}
- 'Date' is in year 2014



See *Sharing Filters Between Dashboards and Reports* in *Advanced Dashboard Design* for information on configuring the Viewsheet.

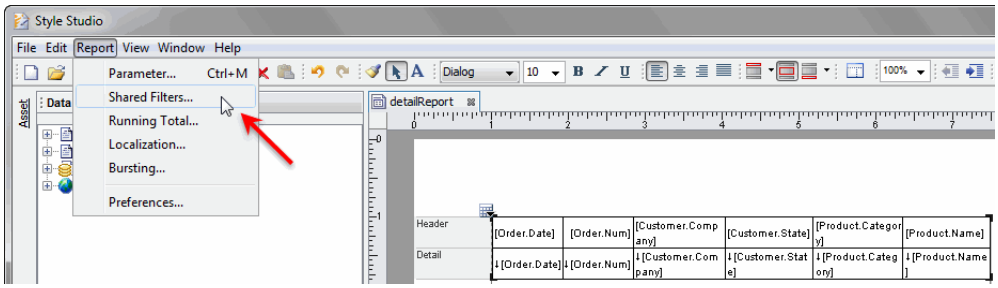
The 'Detail Data' button in this dashboard is configured to hyperlink to a report that provides a table of detail data, such as shown below:

Date	Num	Company	State	Category	Name
4/30/2011	12028	Interstate Shop	NJ	Office Tools	WebCalendar
4/30/2011	12028	Interstate Shop	NJ	Graphics	Mega Icons
4/30/2011	12028	Interstate Shop	NJ	Educational	Animal World
4/30/2011	12028	Interstate Shop	NJ	Hardware	NetStorage
6/08/2011	12051	Interstate Shop	NJ	Hardware	19 inch LCD
11/22/2011	12128	Interstate Shop	NJ	Personal	Green Planet
11/22/2011	12128	Interstate Shop	NJ	Office Tools	Fast Mail

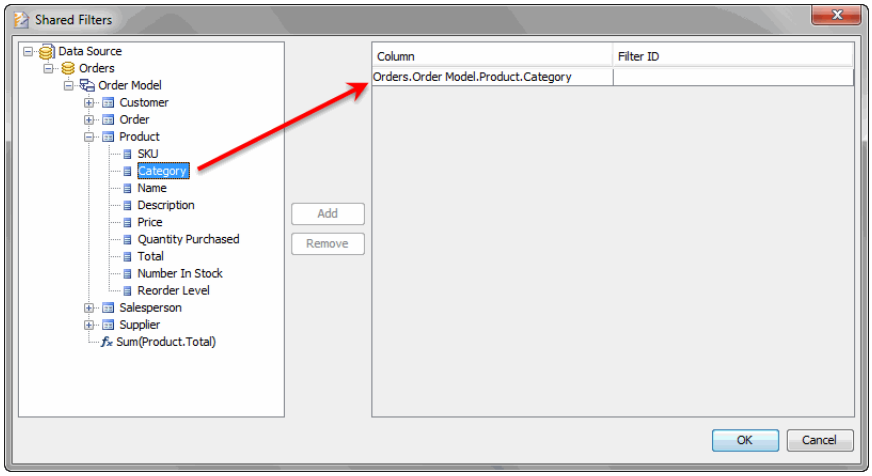
To ensure that the report displays *only* data corresponding to the selections that the user has made in the dashboard for 'Category', 'State', and 'Date', you should create a shared filter. Follow the steps below:

1. Open the report that displays the detail data.

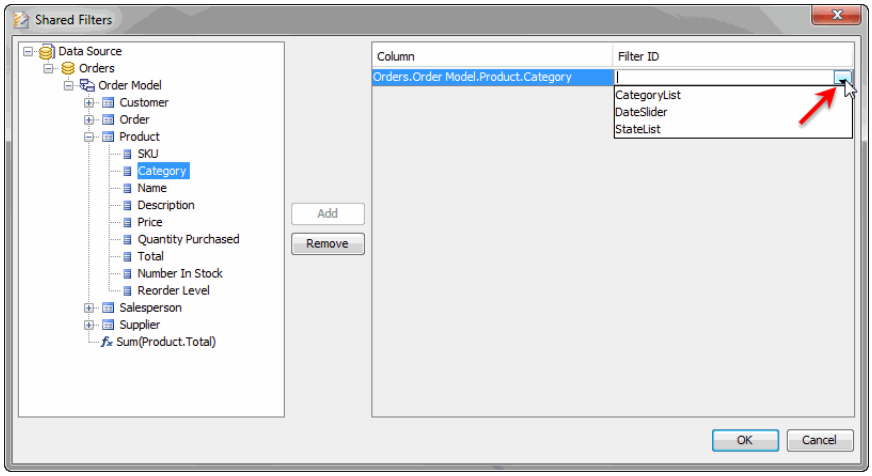
2. Select 'Shared Filters' from the 'Report' menu to open the 'Shared Filters' dialog box.



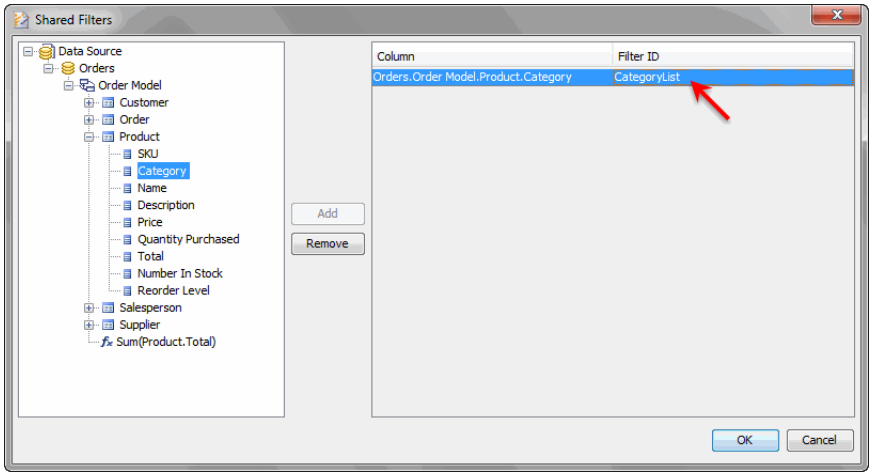
3. From the 'Data Source' tree, select a field in the report that you want to filter based on the Viewsheet settings, and press 'Add'.



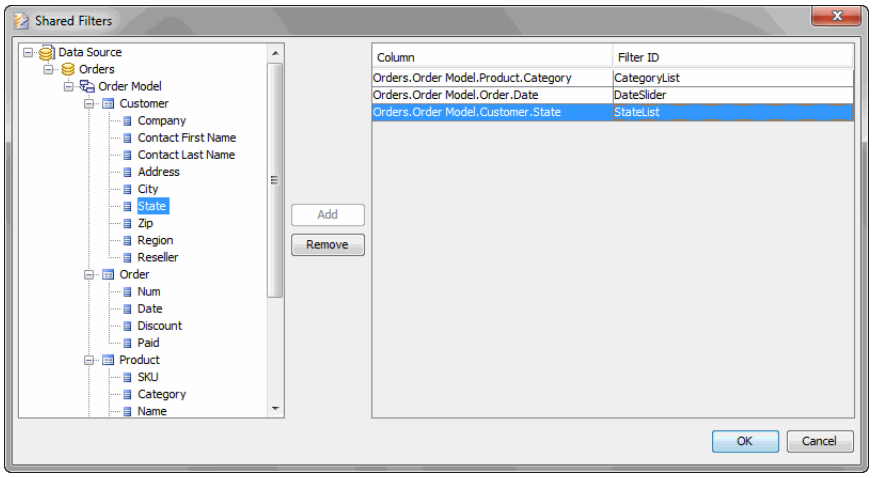
4. Click on the empty field in the 'Filter ID' column. Expand the menu to view all of the available shared filters (from all deployed Viewsheets).



5. Select the ID of the Viewsheet filter that you want to share with the selected report data source.



6. Repeat the above steps to assign all the desired shared filters.



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

Filters are also shared if the user opens the report while the dashboard is open.

When this detail report is accessed via a hyperlink from the dashboard, the data shown in the report table will be filtered to match the current dashboard selections.

7.3 Parameterization

It is often desirable to allow the user to customize certain aspects of report display. For example, the user might want to view only records that satisfy a specific filter condition (e.g., ‘State’ is one of NJ, MD, NY), or they may wish to hide certain report elements and display others.

To provide users with this kind of control, you can define *report parameters*. Parameters are variables whose values are specified when the report is generated. The following sections explain how to define parameters, and how to prompt the user for their values.

Parameter values can also by set by script, drill-down report, Enterprise Manager, scheduled task, or JSP/HTTP request. See the sources below for more information on these topics.

See Also
Hyperlinks, for report drill-down parameters
Filtering Data in Data Binding Wizard, for element data-binding condition parameters
Advanced Toolbar Buttons, in *Advanced Data Modeling*, for query-level parameters
Auto-Drilldown, in *Advanced Data Modeling*, for auto-drill parameters
Hyperlinks, in *Advanced Dashboard Design*, for Viewsheet drill-down parameters
onLoad Handler, in *Advanced Report Scripting*, for report parameters in script
Parameter Tab, in *Advanced Server Administration*, for information about setting parameters in Enterprise Manager.
Creation Parameters, in *Advanced Server Administration*, for information about scheduled task parameters.
Passing a Parameter to a Report, in *Advanced Server Integration*, for HTTP parameters

7.3.1 Parameterizing a Condition

As explained in *Filtering Data in Data Binding Wizard*, you can restrict the data that is displayed by a report element (e.g., table, chart, text) by adding filtering conditions to the **Condition** tab of the 'Data Binding' dialog box. The example in *Walkthrough: Adding Conditions in Data Binding Wizard* illustrates how to add a *fixed* condition, i.e., a condition that is fully specified at design-time and that performs the same filtering operation each time the report is generated. In addition to such fixed conditions, you can also create *adaptive* conditions, i.e., conditions which can be modified at runtime.

To create an adaptive condition on a report element, follow the steps below.

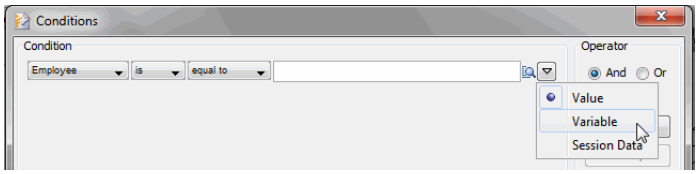
```
To specify a condition on an exiting report element, see Filtering
Data in a Table.
```

1. Select the **Condition** tab of the 'Data Binding' dialog box. (See *Filtering Data in Data Binding Wizard* for information on the 'Data Binding' dialog box.)

2. Use the menus to specify the first part of the condition. For example, for binding to the sample 'All Sales' query, you could create the following condition:

   ```
   [Employee][is][equal to]
   ```

3. To use a variable in the condition, click the “arrow” button at the right side of the condition, and select the ‘Variable’ option. Then follow the steps below:



   a. In the adjacent ‘Name’ field, enter a name for the parameter. (For example, enter “`stateVar`” 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.

   b. Enable ‘Use Selection List’ if you want to prompt the end user with a list of available choices for the parameter. (The choices are drawn from the conditionalized data field, e.g., `Customer.State`.)

See *Defining and Editing Report Parameters* to set the variable data type.

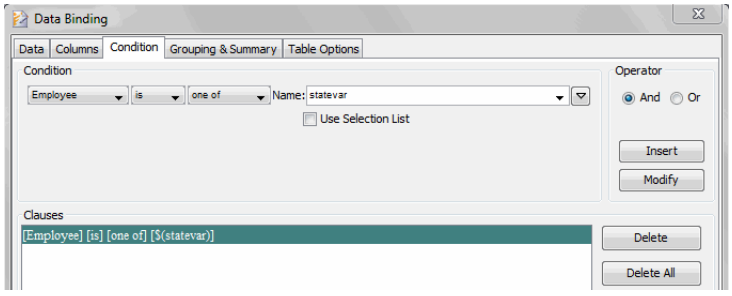
4. To use session data in the condition, click the “arrow” button at the right side of the condition, and select the ‘Session Data’ option.

   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.

Note that the adjacent menu automatically chooses the ‘equal to’ or ‘one of’ operator to match your selection.

5. Click ‘Append’. This adds the new condition to the ‘Clauses’ panel. For the above example, the following clause is added:

```
[Customer.State][is][one of][$(stateVar)]
```



6. Click ‘Finish’ to close the ‘Data Binding’ dialog box, or continue making other data binding modifications as described in *Creating a New Table, Crosstab, or Section using a Wizard*.

By default, users will be prompted during report loading to enter a value for the specified variable. The variable name that you enter in the ‘Name’ field of the condition is used as the label in the user prompt unless you provide an alternate label for this purpose. See *Defining and Editing Report Parameters* to set a variable label and to enable or disable user prompting.

The value of a variable can also be acquired from sources other than user input. See *Acquiring a Parameter Value* for the available alternatives.

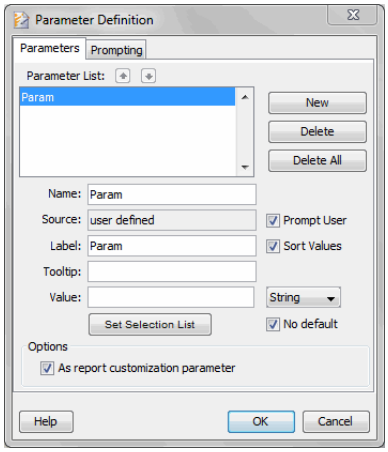
7.3.2 Defining and Editing Report Parameters

In addition to defining parameters within data binding conditions (see *Parameterizing a Condition*), you can also define parameters at the report level within the ‘Parameter Definition’ dialog box. The ‘Parameter Definition’ dialog box also provides a single location for managing all parameters in the report, including those defined within data binding conditions.

To define a report parameter, follow the steps below:

To edit an existing parameter, select the parameter, and then skip to the appropriate step below.

1. Select *Report* > *Parameter* from the Style Studio menu. This opens the ‘Parameter Definition’ dialog box.



See *Creating a New Query* in *Advanced Data Modeling* for information on setting query parameters.

The **Parameters** tab displays all existing report, data-binding, and query parameters. Existing data binding condition parameters and query condition parameters are shown *italicized* in the **Parameters** tab.

2. Click ‘New’. Specify the name of the new parameter in the dialog box, and click ‘OK’. (You can change the name later by editing the text in the ‘Name’ field.)

3. 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.)

4. 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.)

5. To acquire parameter labels/values from a query at runtime, follow these steps:

   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 report parameters if those parameters possess default values.)

6. (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.

7. Click ‘OK’ to close the ‘Selection List’ dialog box.

8. Make the following optional selections in the ‘Parameter 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. (See *Report Toolbar* in *Interacting with Reports*.)

   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.

9. Click ‘OK’ to close the ‘Parameter Definition’ dialog box.

If multiple conditions use parameters with the same name, you can specify whether parameters sharing a name should be treated as the same parameter. To treat them as the same parameter (and prompt the user only once) select *File* > *Configure* from the Style Studio menu, click the **General** tab, and enable the ‘Parameter Name Unique’ option (enabled by default).

7.3.3 Acquiring a Parameter Value

The value of a parameter (variable) is assigned only at runtime, and can be acquired from various sources, as described below:

- **End-user prompting**: By default, the end user is automatically prompted for parameters (variables) used in conditions. To manage user prompting, see *Defining and Editing Report Parameters*. To customize prompting, see *Parameter Sheets*.
- **Drill-down hyperlinks**: Parameter values can be supplied from a drill-down hyperlink in a separate report (see *Hyperlinks*), or by a script-based hyperlink. (See `showReplet` and `sendRequest` in *Advanced Report Scripting* for details.)
- **Report scripts**: Report script placed in the *onLoad Handler* or formula column can set parameter values by accessing the `parameter` array. See *Accessing Report/Request Parameters* in *Advanced Report Scripting*.
- **User session**: A report can obtain an external parameter value from the SRPrincipal object associated with the user session. See *Accessing the User Session with SRPrincipal* in *Advanced Server Integration* for further details.
- **Scheduled task**: A scheduled task can pass values to condition parameters at the time that it automatically executes a report. See *Scheduler Actions* for more details.

7.3.4 Parameter Sheets

If query or report parameters are defined for a report (see *Parameterization* above), then the user is prompted with a default parameter screen when they first open the report. This screen prompts the user for all of the parameters for which ‘Prompt User’ is enabled in the Style Studio ‘Parameter Definition’ dialog box.

You can also create custom parameter prompting by designing an input form ("parameter sheet") with Style Studio's *Form Elements*. This gives you much greater flexibility positioning and styling the input controls. It also allows you to sequence and adaptively populate the input controls so as to smoothly guide the user through the input process, and prevent the report from returning empty result sets.

There are two types of parameter sheets:

- **Reusable parameter sheet**: A reusable parameter sheet is an independent asset, and can be associated with any number of reports. See *Reusable Parameter Sheets* for an example of how to create this type of parameter sheet.
- **Embedded parameter sheet**: An embedded parameter sheet is a basic parameter sheet that is saved with a particular report, and can only be used by that report. See *Embedded Parameter Sheets* for more about this type of parameter sheet.

Reusable Parameter Sheets

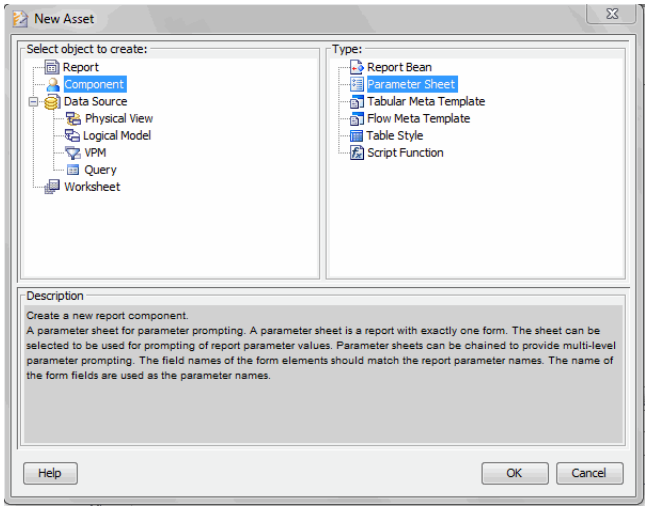
The following walkthrough explains how to create a reusable parameter sheet component, and how to associate the parameter sheet with a particular report.

Walkthrough: Creating a Reusable Parameter Sheet

To create a custom reusable parameter sheet, 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 'Component' node. From the right 'Types' panel, select 'Parameter Sheet'. Click 'OK'.



3. Add a form element to the new report which has a ‘Parameter Name’ identical to the name of the parameter. For this example, follow the steps below to add a Choice element:



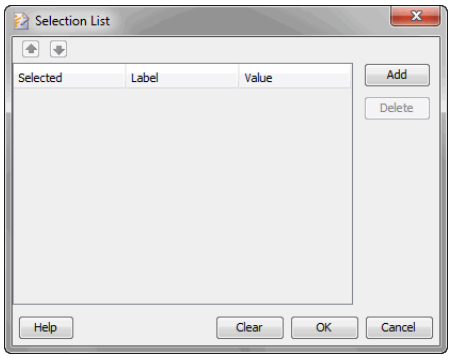
   a. Drag a Choice element to the report.

   b. Right-click the Choice element, and select ‘Properties’ from the context menu. This opens the ‘Choice Properties’ dialog box. Click on the **Choice** tab.

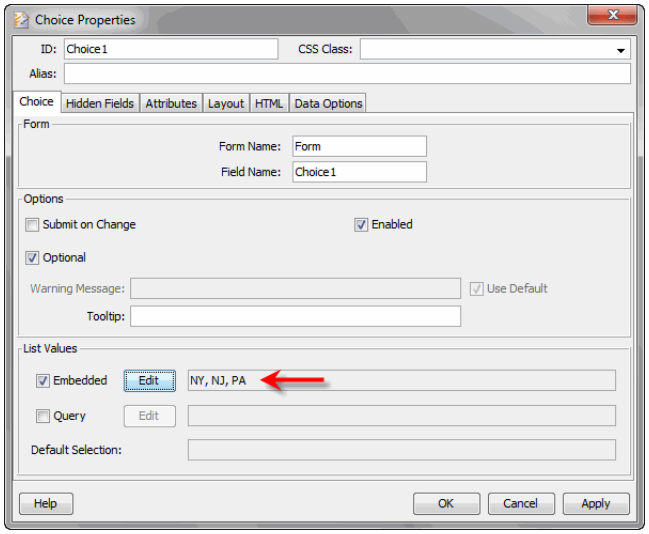
   c. For the ‘Parameter Name’, specify the name of the report parameter into which the value should be passed.

      If you know which report will use the parameter sheet, press the ‘Select Report’ button and choose the desired report. Then select the desired parameter from the ‘Parameter Name’ menu.

   d. In the ‘List Values’ panel, enable the ‘Embedded’ option, and press the ‘Edit’ button. This opens the ‘Selection List’ dialog box.



   e. Press the ‘Add’ button, and enter “NY” for both the ‘Label’ and the ‘Value’. Repeat to add “NJ” and “PA” as additional choices. Press ‘OK’ to close the dialog box.

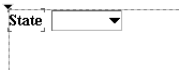


f. Enable the 'Submit on Change' option. This automatically submits the form when the element is used to make a selection, without requiring the user to explicitly click a 'Submit' button.

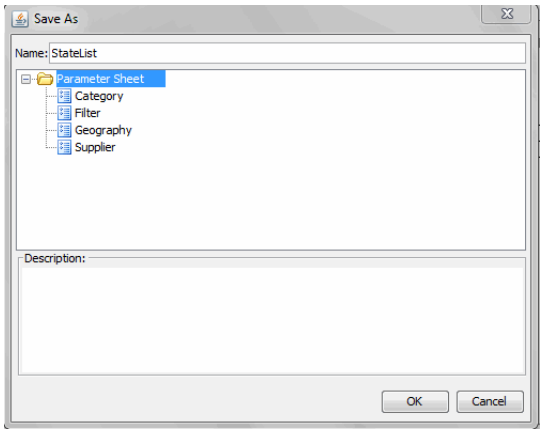
g. In the 'Tooltip' field, enter the text "Select a state." This text will display when the user hovers the mouse over the input control.

h. Click 'OK' to close the 'Choice Properties' dialog box.

4. Add a Text element to complete the form. Set the text contents to "State". (See *Text Element* for more information.)



5. Press the 'Save' button. In the dialog box, enter "StateList" as the name of the parameter sheet, and click 'OK'.



The parameter sheet is saved in *stylereport.srl.d* and is now available for use with any report.

See Also

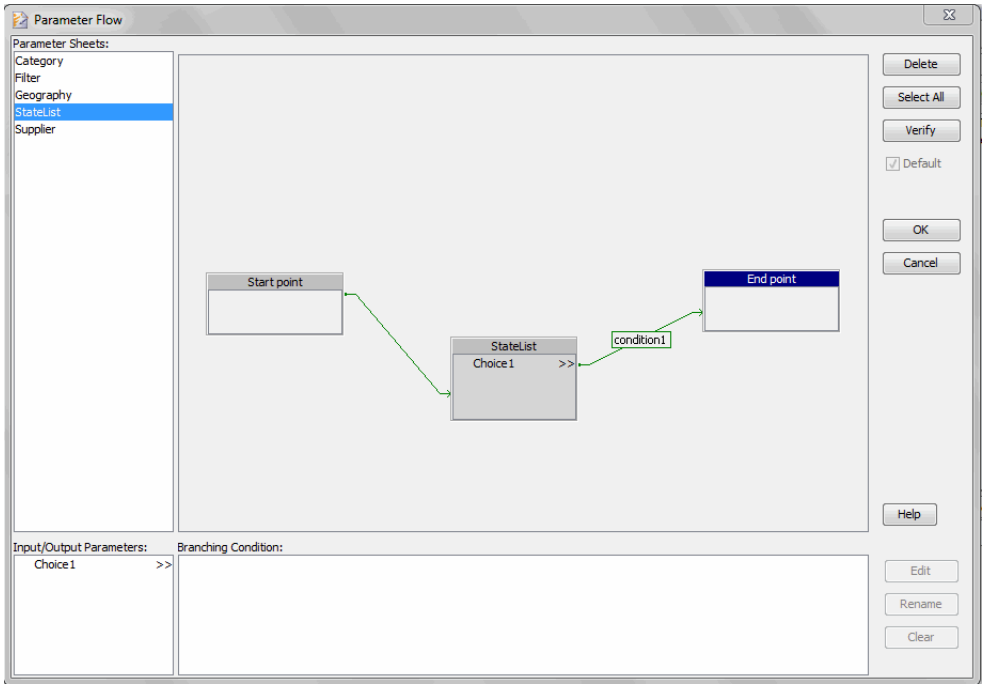
Creating an Embedded Parameter Sheet, for information about how to create an internal parameter sheet.

Walkthrough: Associating a Reusable Parameter Sheet with a Report

To associate a reusable parameter sheet with a particular report so that the parameter sheet is used to acquire user input for the report, follow the steps below:

1. Open the report to which you wish to add the parameter sheet.
2. Select ‘Report’ > ‘Parameter’ from the Style Studio main menu. This opens the ‘Parameter Definition’ dialog box.
3. Click the **Prompting** tab, and select ‘Use Parameter Sheets’.
4. Click the ‘Edit’ button. This opens the ‘Parameter Flow’ dialog box.
5. Select the “StateList” parameter sheet from the left panel, and drag it into the window containing the ‘Start point’ and ‘End point’ boxes.
6. Click on the ‘Start point’ box, and drag the mouse over the ‘StateList’ box. This creates a link between the two boxes.
7. Click on the ‘StateList’ box, and drag the mouse over the ‘End point’ box. This creates a link between the two boxes which is labeled “condition1.

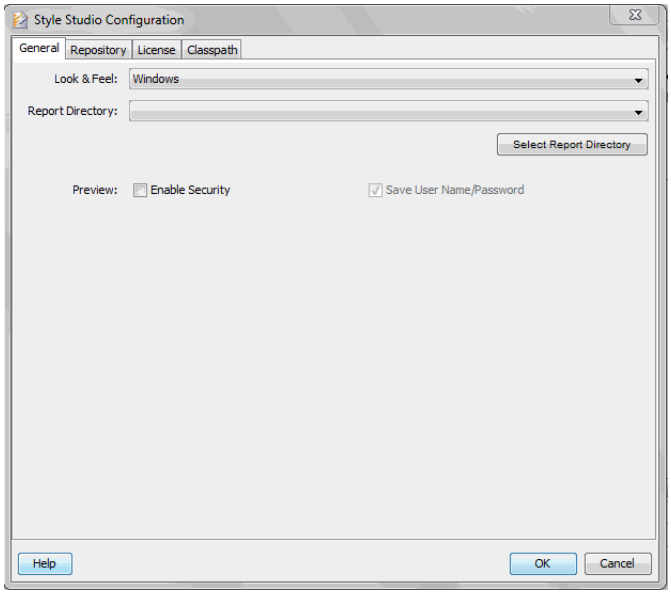
The ‘End point’ box displays in red all report parameters defined in queries, data bindings, and in the ‘Parameters’ dialog box. As you link parameter sheets to the report, the parameters provided by those sheets turn from red to black in the ‘End point’ box. Note that a parameter needs to be provided only on a single path – not on every possible path – in order for its name to appear in black.



For more information about the features of the 'Parameter Flow' dialog box, see *Cascading Parameter Sheets*.

8. Click 'OK' in the 'Parameter Flow' dialog box, and then click 'OK' to exit the 'Parameter Definition' dialog box.

9. Select *File* > *Configure* in the Style Studio menu. Select the **General** tab of the 'Style Studio Configuration' dialog box, and enable 'Preview Server Features'. Click 'OK' to close the dialog box.



10. Preview the report and observe the custom parameter prompting.

See Also

Embedded Parameter Sheets, for information about creating simplified embedded parameter sheets.

Parameter Dialog HTML Template, in *Advanced Server Administration*, for information on customizing the default parameter prompting screen.

Parameter Dialog CSS File, in *Advanced Server Administration*, for information on styling the default parameter prompting screen.

Adding a 'Clear' button to a Parameter Sheet

You can add a 'Clear' button to the parameter sheet to allow the user to clear the current selections. To do this, follow the steps below:

1. Drag a Button element into the parameter sheet in the desired location.

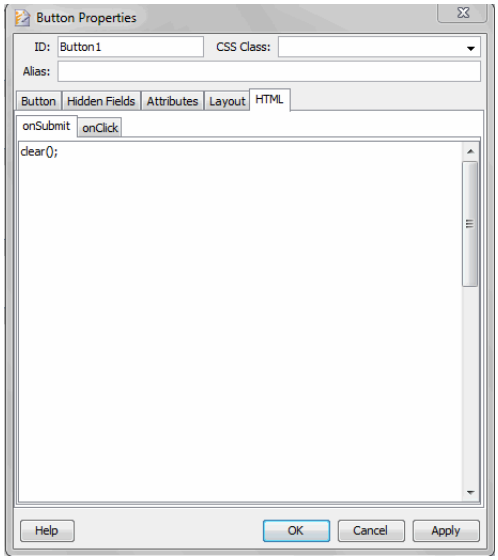
2. Right-click on the Button and select 'Properties'. This opens the 'Button Properties' dialog box.

3. (Optional) Select the **Button** tab in the 'Button Properties' dialog box. Enter the word "Clear" in the 'Label' field.

4. Select the **HTML** tab in the dialog box, then select the **onClick** sub-tab.

5. Enter the following script in the **onClick** sub-tab:

```
clear();
```



6. Press 'OK' to close the dialog box.

```
Note: The onClick script cannot be processed in Style Studio
preview, but will function correctly when the report is viewed in
the Portal.
```

The parameter sheet now contains a ‘Clear’ button. When the user presses the ‘Clear’ button, this clears the currently selected values in all parameter sheet input elements.

Staged Parameters

When you prompt the user for a series of related parameters, it is often desirable to guide the user through these choices in a certain order. This is called *parameter staging*. This technique allows you to constrain the options presented to the user at each stage so that only compatible parameter combinations can be selected.

For example, consider a Parameter Sheet that has two menus, one for ‘State’ and one for ‘City’. Initially, you want only the ‘State’ menu to be enabled, so that the user is forced to select a state first. After the user selects a state, you want to then enable the ‘City’ menu, but restrict its values to those cities that fall within with the selected state.

There are therefore several aspects to parameter staging, as described in the following sections:

- *Creating a Parameter Sheet Layout*
  Add controls, and set ‘Field’ values to map to corresponding parameters.
- *Defining a Staged Parameter Flow*
  Specify the order for parameter control activation.
- *Filtering Parameter Options*
  Populate parameter controls with options that are consistent with previous selections.
- *Linking a Parameter Sheet to a Report*
  Link the parameter sheet to a report that has corresponding parameters.

Creating a Parameter Sheet Layout

Walkthrough

To add controls to a parameter sheet, and associate them with report parameters, 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 ‘Component’ node. From the right ‘Types’ panel, select ‘Parameter Sheet’. Click ‘OK’.

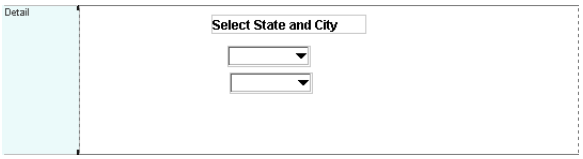
3. Expand the Style Studio **Toolbox** panel.

4. Drag a Section component from the **Toolbox** panel into the report. (Sections allow very precise positioning of elements. See *Section Element* for more details.)

5. Drag a Text element from the **Toolbox** panel into the Section. Enter "Select State and City" as the contents of the Text element, and set the font to 14pt bold. Center-align the Text element. (See *Text Element* for more information.)

Note: The steps below assume that the report to which this Parameter Sheet will be linked contains parameters named 'st' and 'ct'. The 'Field' names of the input controls must exactly match the corresponding report parameter names. (You will create a report with these parameters in *Linking a Parameter Sheet to a Report*.)

6. Drag two Choice boxes from the **Toolbox** panel into the Section.



7. Right-click the first Choice box, and select 'Properties'. This opens the 'Choice Properties' dialog box. Make the following settings:

   a. Set 'ID' to "StateChoice".

   b. Set 'Parameter Name' to "st".

   c. Set 'Enable', 'Submit on Change', and 'Optional'.

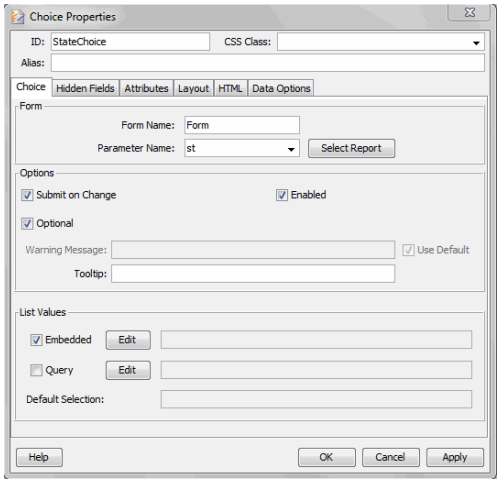
   d. Click 'OK'.

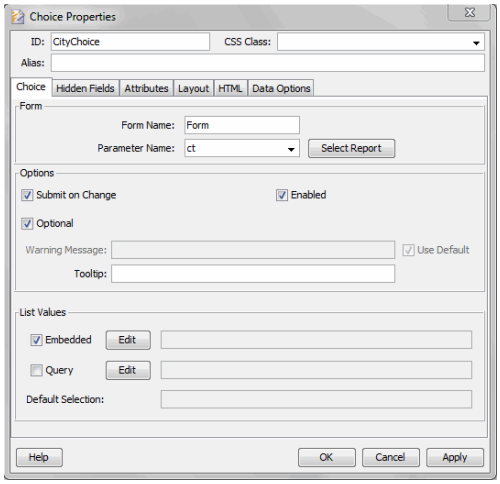
8. Right-click the second Choice box, and select 'Properties'. In the 'Choice Properties' dialog box, make the following settings:

   a. Set 'ID' to "CityChoice".

   b. Set 'Parameter Name' to "ct".

   c. Set 'Enable', 'Submit on Change', and 'Optional'.





d. Click ‘OK’.

The next step is to sequence the controls so that the user must first select a state, and then select a city. You will do this in the next section, *Defining a Staged Parameter Flow*.

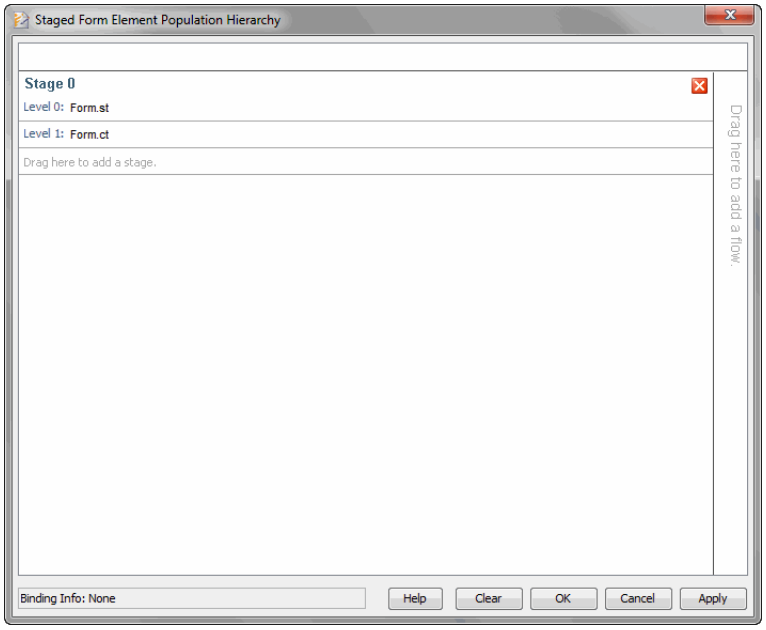
Defining a Staged Parameter Flow

This example continues from *Creating a Parameter Sheet Layout*. To specify the sequence in which the controls are activated, follow the steps below.

1. Select *Report* → *Staged Parameters* from the Style Studio menu. This opens the ‘Staged Form Element Population Hierarchy’ dialog box.

2. In the dialog box, drag the ‘Form.st’ element from the top panel to the area on the right which reads: ‘Drag here to add a flow’. This adds a new parameter flow.

3. Drag the ‘Form.ct’ element to the area which reads ‘Drag here to add a stage’.

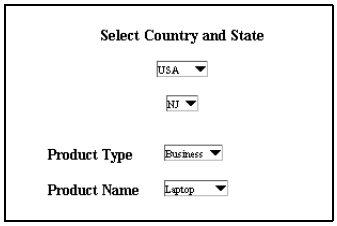


   This places the ‘ct’ parameter under the ‘st’ parameter in the flow hierarchy, which means that the ‘CityChoice’ control will be enabled only after the user submits a selection in the ‘StateChoice’ control. (This sequential control activation only functions when the Parameter Sheet is launched by the linked report, not when the Parameter Sheet is previewed by itself.)

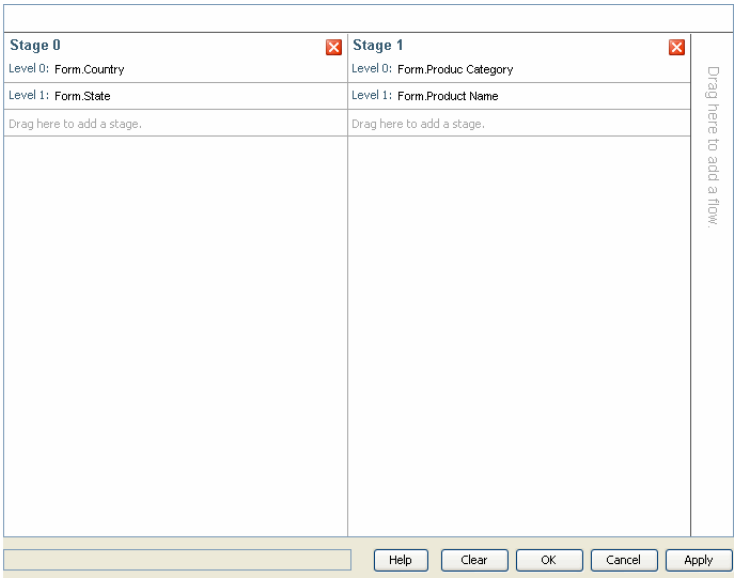
4. Click ‘OK’ to close the dialog box.

The next step is to filter the ‘CityChoice’ control so that it only presents choices that are compatible with the user’s selection in the ‘StateChoice’ field. You will do this in the next section, *Filtering Parameter Options*.

Note: You can create parallel flows of multiple stages in a staged parameter sheet. This is desirable when multiple stages are independent of one another. For example, one stage may possess the levels ‘Country’ and ‘State’, while another stage may possess the levels ‘Product Category’ and ‘Product Name’. These two stages are independent, so it would not be logical to place them into a single flow.



To add a parallel stage, simply drag an element to the region titled ‘Drag here to add a flow’ on the right of the Staged Parameter dialog box. Elements in the ‘Default Level’ are always enabled.



Filtering Parameter Options

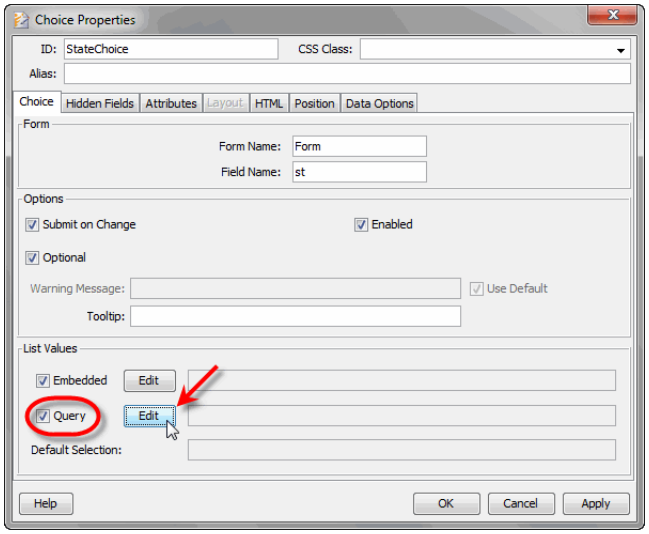
This example continues from *Defining a Staged Parameter Flow*. Because of the parameter flow you defined there, the ‘CityChoice’ control is enabled only after the user selects a ‘StateChoice’ option. In such a case, it is also desirable to limit the options shown in the ‘CityChoice’ control to only those that are consistent with the given ‘StateChoice’ selection.

To populate the ‘CityChoice’ control with values that are consistent with the user’s ‘StateChoice’ selection, follow the steps below.

1. Bind the ‘StateChoice’ control to a data source. Follow these steps:

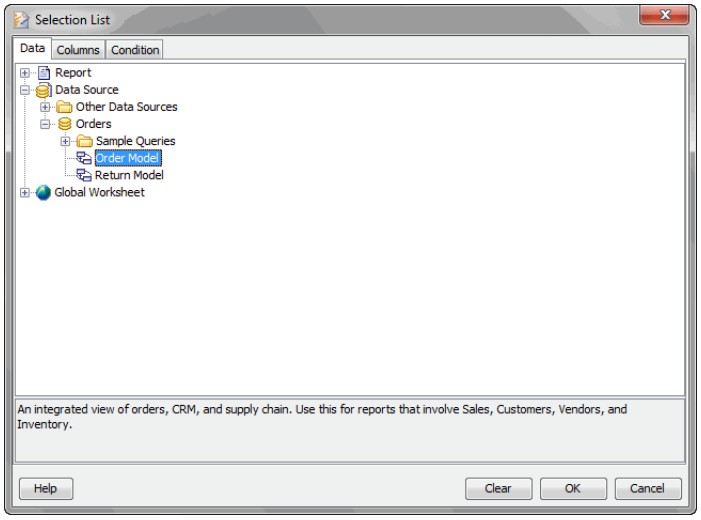
    a. Right-click the ‘StateChoice’ control, and select ‘Properties’ from the context menu. This opens the ‘Choice Properties’ dialog box.

    b. In the ‘List Values’ panel, select ‘Query’ and press the ‘Edit’ button.

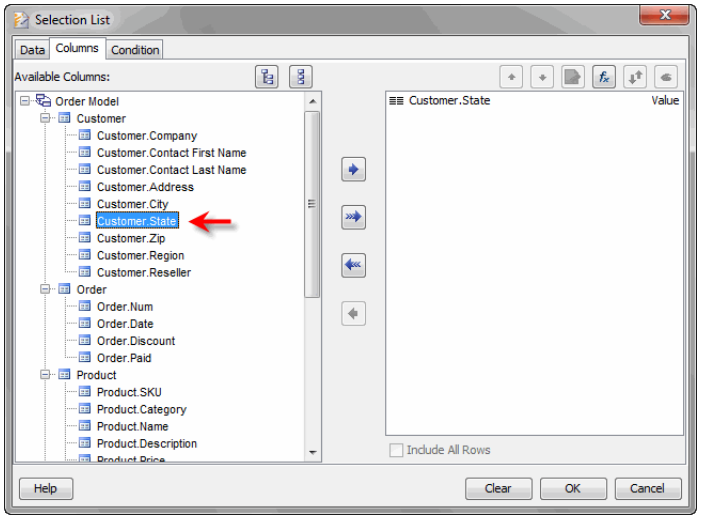


This opens the 'Selection List' dialog box.

c. Under the **Data** tab, expand the 'Orders' node in the tree, and select 'Order Model'.



d. Click the **Columns** tab. Select the 'Customer.State' column in the list, and click the right-arrow. Then click 'OK'.



This will populate the 'StateChoice' control with the data from the 'Customer.State' column. (It is assumed that this is the same 'State' data that will be used in the linked report.)

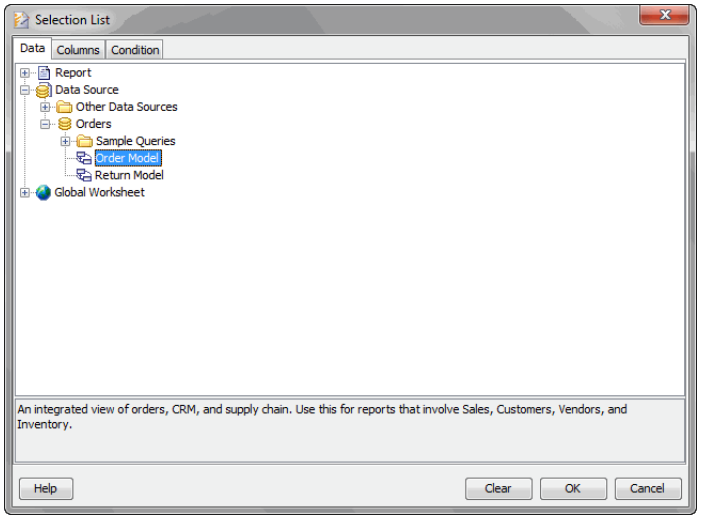
e. Press 'OK' to close the 'Choice Properties' dialog box.

2. Bind the 'CityChoice' control to a data source:

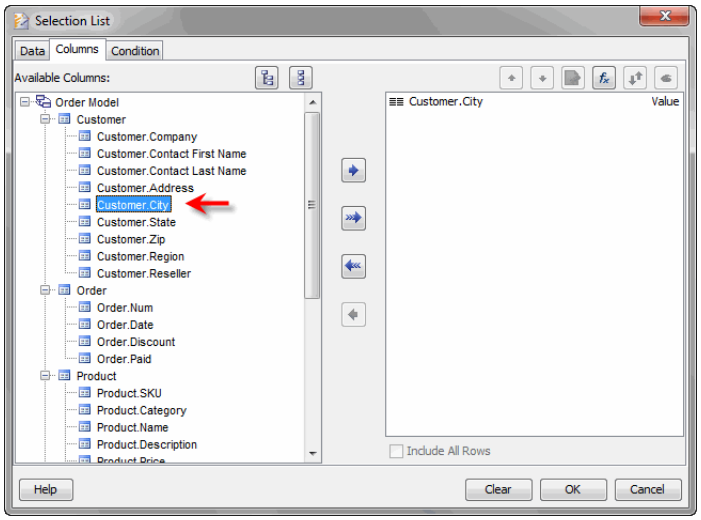
a. Right-click the 'CityChoice' control, and select 'Properties' from the context menu. This opens the 'Choice Properties' dialog box.

b. In the 'List Values' panel, select 'Query' and press the 'Edit' button. This opens the 'Selection List' dialog box.

c. Under the **Data** tab, expand the 'Orders' node in the tree, and select 'Order Model'.



d. Click the **Columns** tab. Select the 'Customer.City' column in the list, and click the right-arrow.

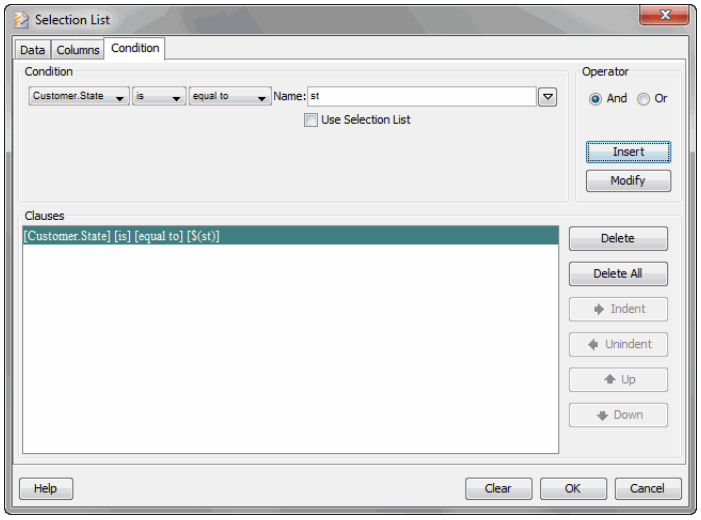


This will populate the 'CityChoice' control with the data from the 'Customer.City' column. (It is assumed that this is the same 'City' data that will be used in the linked report.)

e. Click the **Condition** tab. Using the menus, select the following condition: "`[Customer.State] [is] [equal to]`".

f. Click the "arrow" button at the right side of the condition, and select 'Variable' from the menu.

g. Enter "st" in the 'Label' field, and click 'Append' to apply the condition.



This will populate the 'CityChoice' control with data from the 'Customer.City' field where the 'Customer.State' value is equal to the value of the 'st' parameter set by the user's 'StateChoice' selection. As a result, the 'CityChoice' control displays only cities within the selected state.

h. Press 'OK' to close the 'Selection List' dialog box.

i. Press 'OK' to close the 'Choice Properties' dialog box.

3. Click the Style Studio 'Save' button, and save this Parameter Sheet under the name "StateAndCity".

The next step is to link the Parameter Sheet to the desired report. In the next section, *Linking a Parameter Sheet to a Report*, you will create a new report, and link it to the parameter sheet.

Linking a Parameter Sheet to a Report

This example continues from *Filtering Parameter Options*. To link the 'StateAndCity' parameter sheet to a new report, 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 'Report' node. In the right 'Types' panel, select 'Blank Tabular Report' and click 'OK'.

3. Expand the Style Studio **Toolbox** panel.



4. Click the 'Table' button in the **Toolbox** panel to open the 'Data Binding' dialog box for a new table.

5. Under the **Data** tab, bind the table to 'Order Model' from the 'Orders' data source.

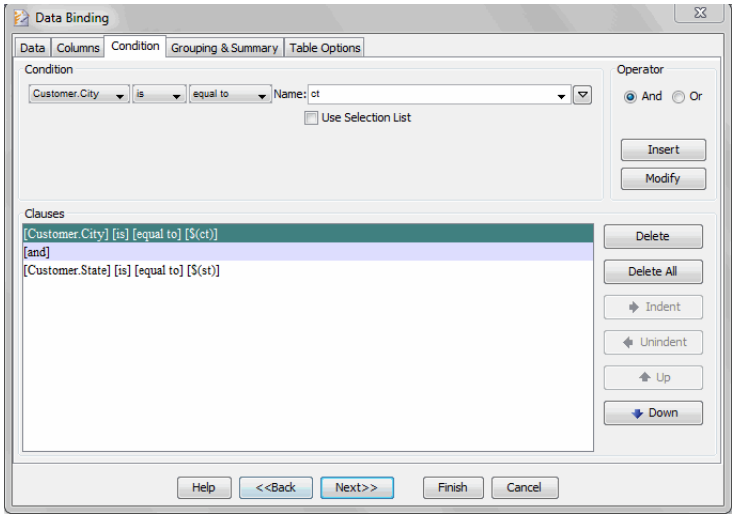
6. In the **Columns** tab, click the right-arrow to select the following columns from the 'Available Columns' pane:

   'Product.Name'
   'Product.Quantity Purchased'
   'Customer.Company'
   'Customer.City'
   'Customer.State'

7. In the **Condition** tab, specify the following compound condition.

```
[Customer.State][is][equal to][$(st)]
  [and]
[Customer.City][is][equal to][$(ct)]
```

   To enter the parameter names 'st' and 'ct' in the statements, select the 'Parameter' option on the right side of the condition, and enter these names into the corresponding text field. Enable 'Use Selection List' for each parameter. (See *Filtering Data in Data Binding Wizard* for more details on specifying conditions.)



8. Click 'Finish' to exit the 'Data Binding' dialog box.

9. Save this report as "*parameterReport.srt*".

In the next steps, you will link this 'parameterReport.srt' report to the 'StateAndCity' parameter sheet developed earlier (see *Creating a Parameter Sheet Layout*). When you do this, the default parameter prompting mechanism is automatically replaced by the staged parameter prompting mechanism you designed.

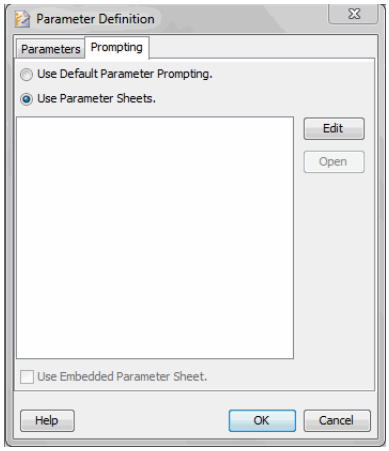
10. Click the ‘Open’ button in the Style Studio toolbar. Locate and open the *parameterReport.srt* report.

11. Select ‘Report’ > ‘Parameter’ from the main menu. This opens the ‘Parameter Definition’ dialog box.

    Note that the **Parameters** tab of the ‘Parameter Definition’ dialog box already lists the ‘st’ and ‘ct’ parameters defined earlier (in the data binding condition).

12. Click the **Prompting** tab, and select ‘Use Parameter Sheets’.

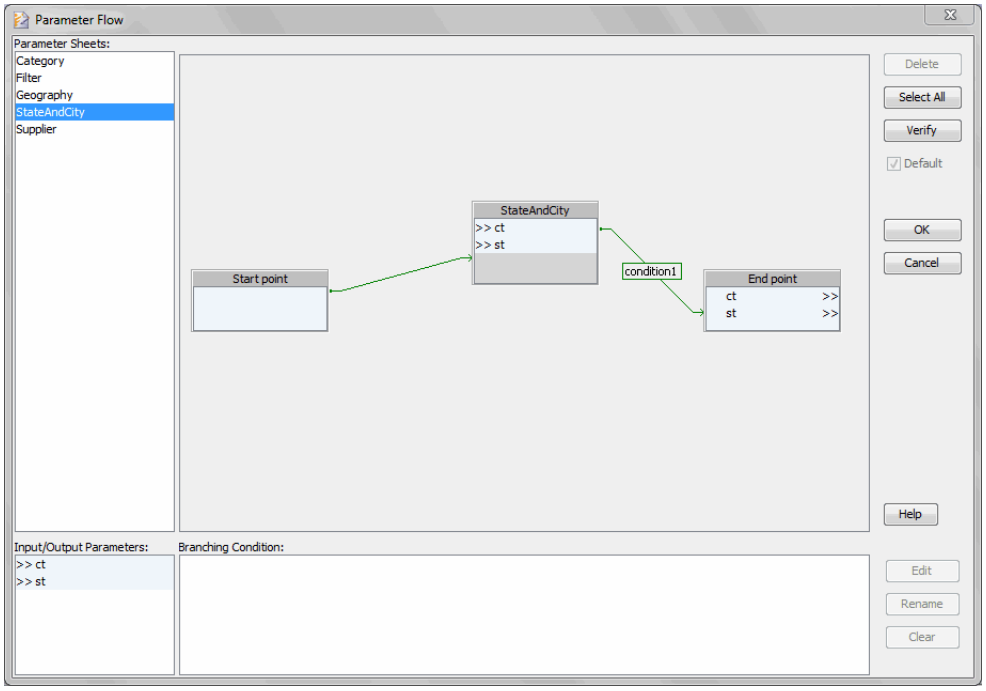
13. Click the ‘Edit’ button. This opens the ‘Parameter Flow’ dialog box.



14. In the ‘Parameter Flow’ dialog box, drag the ‘StateAndCity’ Parameter Sheet into the right panel.

15. Click on the ‘Start point’ box, and drag the mouse over the ‘StateAndCity’ box. This creates a link between the two boxes.

16. Click on the ‘StateAndCity’ box, and drag the mouse over the ‘End point’ box. This creates a link between the two boxes which is labeled “condition1”.



For more information about the features of the 'Parameter Flow' dialog box, see *Cascading Parameter Sheets*.

17. Click 'OK' in the 'Parameter Flow' dialog box, and then click 'OK' to exit the 'Parameter Definition' dialog box.

18. Select 'File' > 'Configure' in the Style Studio menu. Click the **General** tab in the 'Style Studio Configuration' dialog box, and enable 'Preview Server Features'. Click 'OK' to close the dialog box.

19. Preview the report.

The Parameter Sheet is now loaded in place of the default parameter prompting mechanism. The parameters are staged as desired: The user must specify the state before the city, and can only select cities that are within the selected state.

Cascading Parameter Sheets

You can define multiple parameter sheets for a given report, and control the sequence in which these parameter sheets are presented to the user. This approach is called *cascading parameter sheets*. You can specify conditional branching for these parameter sheets, so that user selections on one parameter sheet determine which parameter sheet is next displayed.

It is possible to design a similar kind of adaptive prompting through the use of *Staged Parameters* and scripting. However, by designing multiple parameter sheets in a cascading sequence, you retain the flexibility of reusing each parameter independently.

Walkthrough: Cascading Parameter Sheets

Assume that a company needs a simple report which generates an invoice for the orders placed by its customers. The report displays order information such as product, price, and discount, as well as customer information such as Company Name and Customer ID, which the user will enter as parameters.

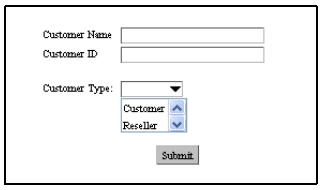
Walkthrough

If some customers are resellers to whom the company provides a special discount scheme, it will be necessary to display different parameter sheets depending on the type of customer. You can easily create the required conditional prompting sequence with cascading parameter sheets.

Create the following three parameter sheets: 'customers', 'reseller discount', and 'customer discount'.

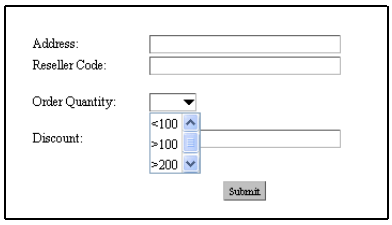
For information on how to create these kinds of parameter sheets, see *Parameter Sheets*.

1. The 'customers' parameter sheet is a simple form that accepts some initial customer information. It has some text labels and fields and a Choice element.

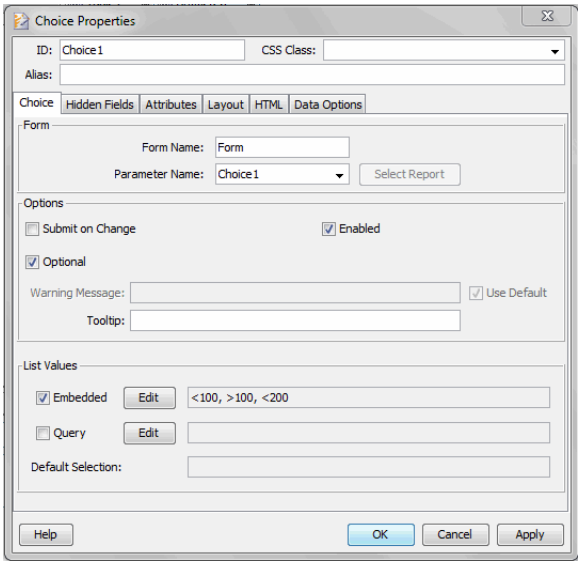


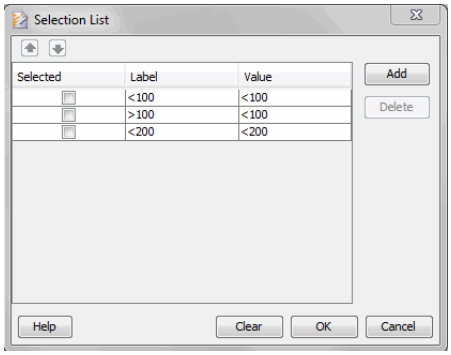
   The Choice element has two choices: 'Customers' and 'Resellers'. The user can choose one and click 'Submit'.

2. The 'reseller discount' parameter sheet accepts some inputs for the reseller details. It has a Choice element which lets the user enter the order quantity.

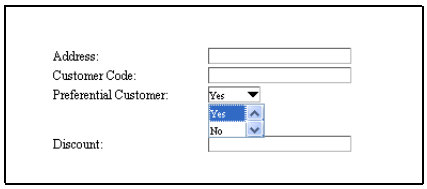


   To assign a label and value to the choices, select the Choice element and right-click to open the 'Choice Properties' dialog box. In the dialog box, on the **Choice** tab, click the 'Advanced' button to open the 'Selection List' dialog box. You can assign labels and values for the list as shown below.





3. The ‘customer discount’ sheet has a different set of inputs.



When you link these cascading sheets to a report (discussed next in *Using Cascaded Parameter Sheets in a Report*), the report is automatically provided with an accumulation of the parameters entered in all of the sheets that are part of the user’s selection flow. Therefore, the report will always get all parameters from either the ‘customer’ and ‘reseller discount’ sheets or the ‘customer’ and ‘customer discount’ sheets, the two possible user selection flows.

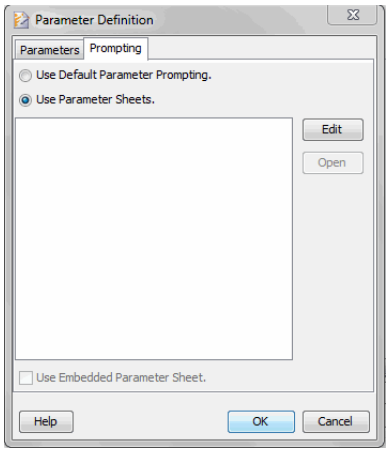
Using Cascaded Parameter Sheets in a Report

This example continues from *Walkthrough: Cascading Parameter Sheets*.

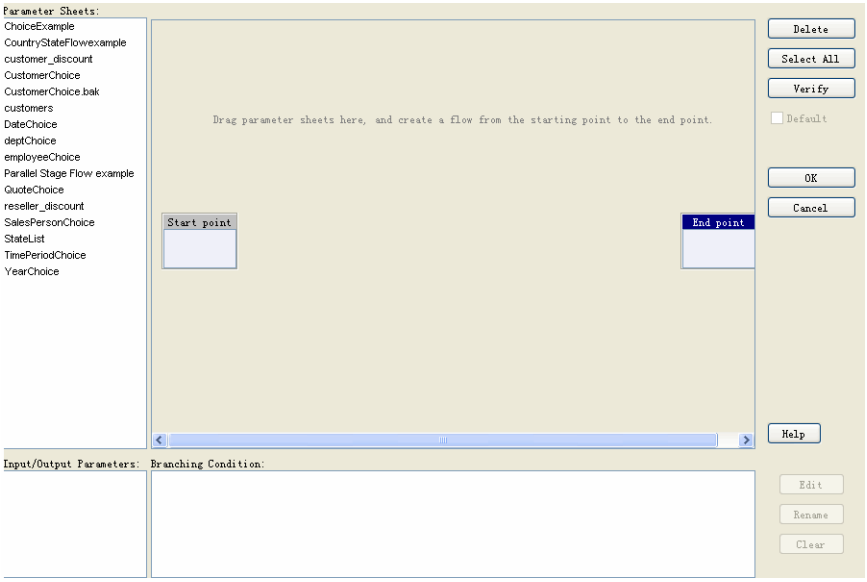
1. Click the ‘New’ button in the Style Studio toolbar. This opens the ‘New Asset’ dialog box.

2. In the left panel, select the ‘Report’ node. In the right ‘Types’ panel, select ‘Blank Tabular Report’ and click ‘OK’.

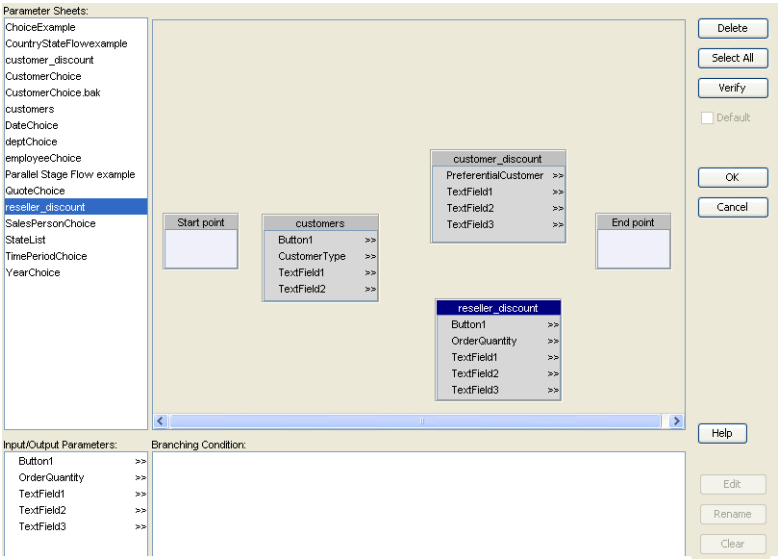
3. Select *Report* → *Parameter*. This opens the ‘Parameter Definition’ dialog box. Select the **Prompting** tab and choose the ‘Use Parameter Sheets’ option.



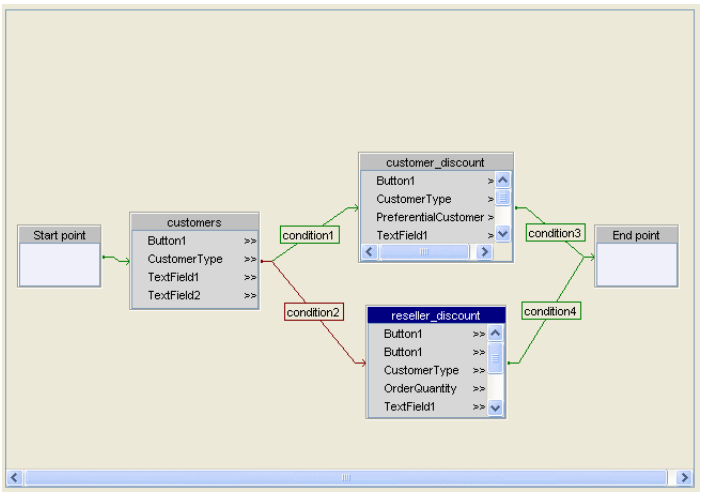
4. Click on the ‘Edit’ button. This opens the ‘Parameter Flow’ dialog box, in which you can set cascaded parameter sheets with conditional branching.



5. Drag and drop the ‘customers’, ‘customer discount’, and ‘reseller discount’ parameter sheets to the center. The ‘Start point’ and ‘End point’ boxes are the two terminal points of the flow.



6. You can now specify different flow sequences for the parameter sheets. Drag-and-drop from one sheet to another to connect them as shown.



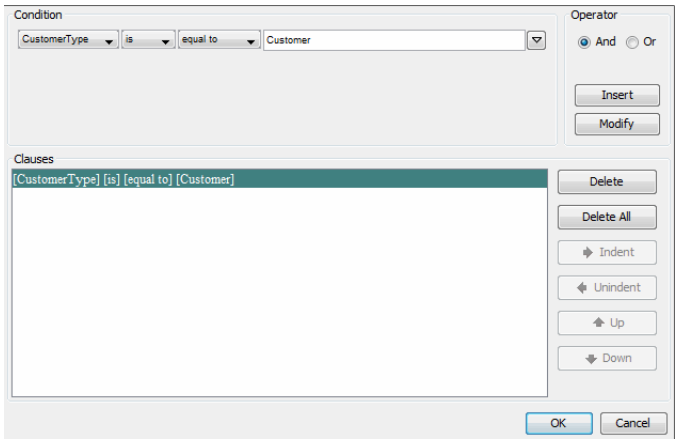
7. Specify conditions at the different branch points:

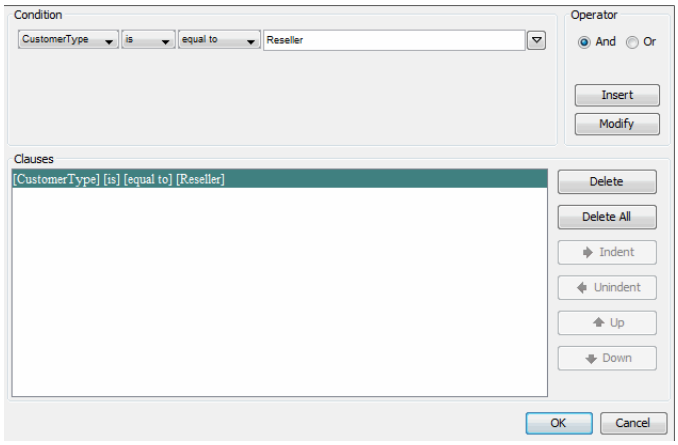
   a. Select ‘`condition1`’ and click on the ‘Edit’ button in the bottom right corner to open the condition dialog box.

   b. Add the following condition: ‘`[CustomerType] [is] [equal to] [Customer]`’. Click ‘OK’.

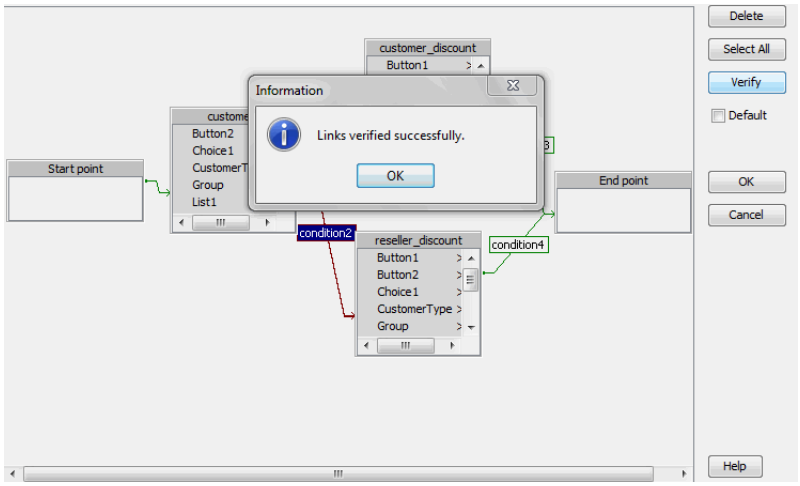
c. Select ‘condition2’ and click on the ‘Edit’ button in the bottom right corner to open the condition dialog box.

d. Add the following condition: ‘`[CustomerType] [is] [equal to] [Reseller]`’. Click ‘OK’.

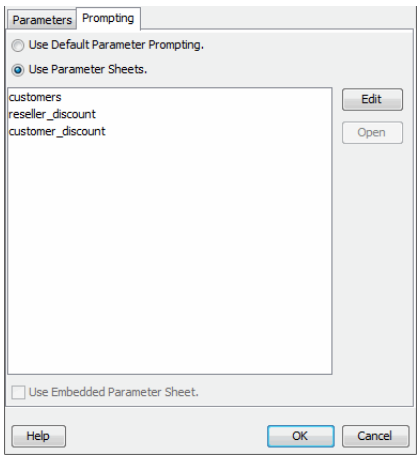




8. Click the ‘Verify’ button. If the links are correct, you will see a ‘Links verified successfully’ message.



9. Click ‘OK’ to close the ‘Parameter Flow’ dialog box. The parameter sheets are now ready to be added to the report.



Note that in addition to proceeding strictly in the forward direction, the cascading parameter sheet flow can also conditionally return to previous parameter sheets in the sequence.

Embedded Parameter Sheets

The *Parameter Sheets* section explains how to create a reusable parameter sheet, a reusable parameter entry form that is stored in *stylereport.srl.d*. This type of parameter entry form is very flexible, because it can be reused in multiple reports and can make use of a wide variety of report components. This is the preferred way to create a custom parameter entry form.

```
A reusable parameter sheet cannot be converted into an embedded
parameter sheet, and an embedded parameter sheet cannot be
converted into a reusable parameter sheet.
```

However, in some cases you may want to create a simpler parameter sheet that provides parameter input for just a single report and is saved together with the report itself (rather than in the global *stylereport.srl.d* directory). This is called an *embedded parameter sheet*. An embedded parameter sheet can only be used by the report in which it is embedded, and provides only a limited set of design components.

The following sections explain how to create an embedded parameter sheet.

See Also

Reusable Parameter Sheets, for information about how to create external (reusable) parameter sheets.

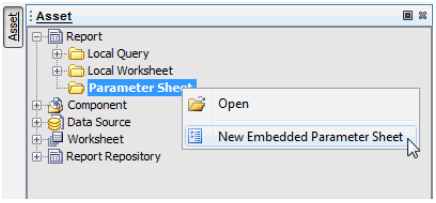
Creating an Embedded Parameter Sheet

To create an embedded parameter sheet, follow the steps below:

1. In Style Studio, open the report to which you want to add an embedded parameter sheet.

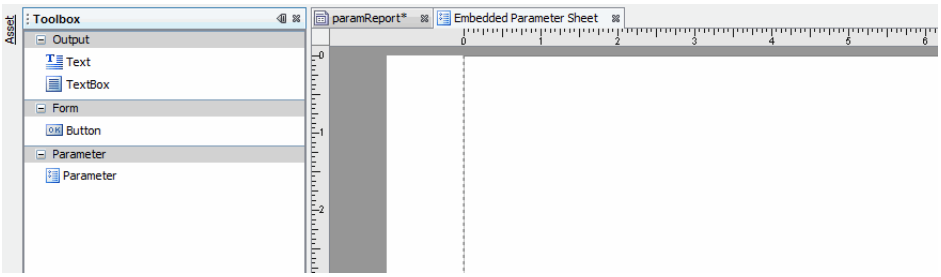
2. Expand the Style Studio **Asset** panel.

3. Expand the 'Report' node on the tree. Right-click on the 'Parameter Sheet' option, and select 'New Embedded Parameter Sheet'.



A report has at most one embedded parameter sheet.

4. This opens the embedded parameter sheet for editing.



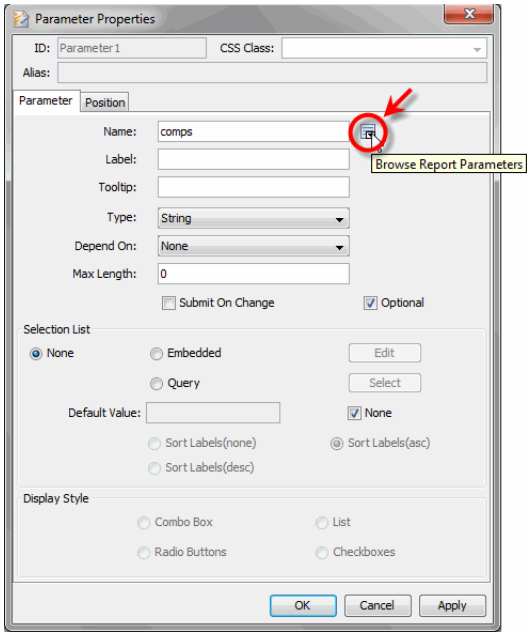
5. Drag Text and TextBox components from the **Toolbox** panel into the parameter sheet to display labels and other static text.

6. Drag a Parameter component into the parameter sheet for each report parameter that you want the user to enter.

7. Right-click each parameter component, and select ‘Properties’ from the context menu. This opens the ‘Properties’ dialog box.

8. Adjust the desired settings in the ‘Properties’ dialog box:

    a. Press the ‘Browse Report Parameters’ button, and select the parameter that the Parameter component should control.



    b. In the ‘Label’ field, enter the label that should appear on the Parameter input control.

    c. In the ‘Tooltip’ field, enter the text of a tooltip to appear when the user hovers the mouse over the Parameter input control.

    d. In the ‘Type’ menu, select the data type of the parameter.

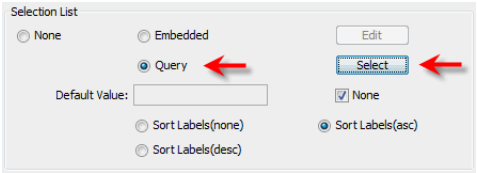
    e. In the ‘Depends On’ menu, select the parameter on which the current parameter depends (if any). When the user runs the report they will need to supply a value for the ‘Depends On’ parameter before they can enter a value for the current parameter.

    f. If you want the form to be submitted immediately when the user chooses a value for the parameter, enable ‘Submit on Change’.

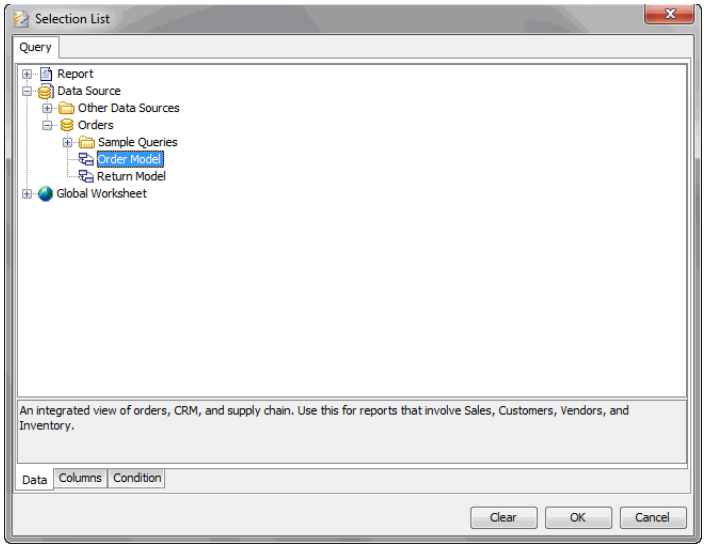
In general, you should also enable ‘Submit on Change’ if another parameter depends on this parameter (i.e., if another parameter has this parameter specified in its ‘Depends On’ field). If you do not enable ‘Submit on Change’ in this case, the user will not be able to enter a value for the dependent parameter.

g. Select ‘Optional’ if the user does not need to supply a value for the parameter.

h. To populate the Parameter options from an existing data source, in the ‘Selection List’ panel select the ‘Query’ option and press ‘Edit’. This opens the ‘Selection List’ dialog box.

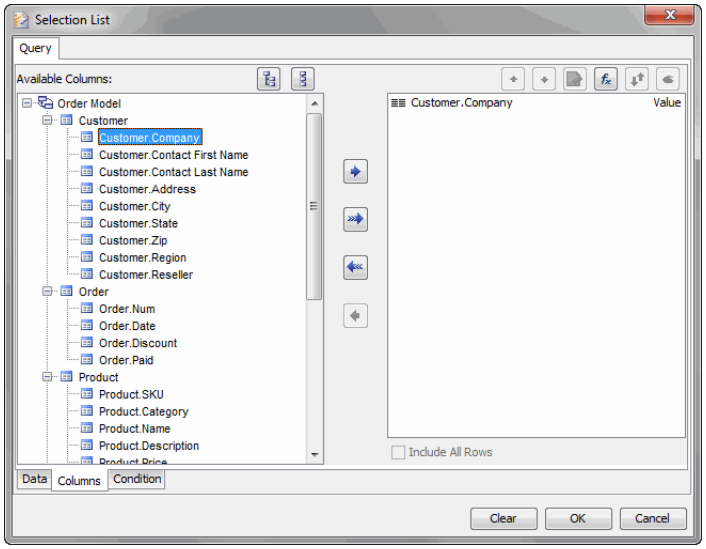


Select the **Data** tab, and choose the data source that will provide the labels and values for the parameter.



If you want the labels and values to be the same, select only a single column.

Select the **Columns** tab. Choose one column to provide the labels for the Parameter, and one column to provide the values for the Parameter.



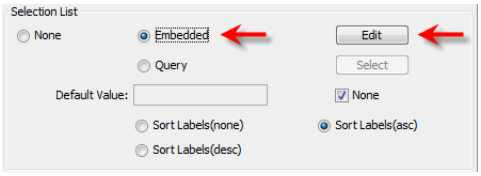
The *labels* determine the choices that are displayed to the user in the Parameter. The *values* determine the corresponding data that is submitted when the user makes a selection. To swap the 'label' field with the 'value' field, click the 'Move up' or 'Move down' arrow above the right panel.

To adapt the Parameter choices based on user input, specify a parameterized condition. See *Parameterizing a Condition*.

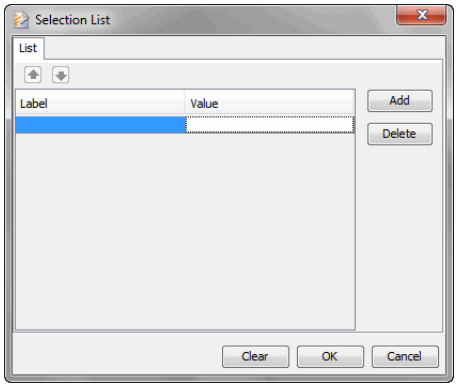
(Optional) Select the **Condition** tab. Specify a condition to filter the options (labels/values) displayed by the Parameter. See *Filtering Data in Data Binding Wizard* for more information on specifying a filter.

Press 'OK' to close the 'Selection List' dialog box.

i. To populate the Parameter options by manually entering the label/value pairs, in the 'Selection List' panel select the 'Embedded' option and press 'Edit'. This opens the 'Selection List' dialog box.



Press the 'Add' button to add a new label/value pair. Double-click on a label or value to edit the text.



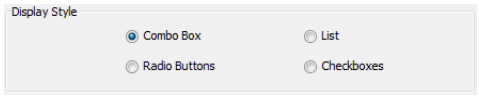
The *labels* determine the choices that are displayed to the user in the Parameter. The *values* determine the corresponding data that is submitted when the user makes a selection. To reorder the label/value pairs, press the arrow buttons above the label/value list.

Repeat to add additional labels. Press 'OK' to close the 'Selection List' dialog box.

j. To supply a default value for the parameter, uncheck the 'None' option, and enter a value in the 'Default Value' field.

k. To sort the labels in ascending order, select the 'Sort Labels(asc)' option. To sort the labels in descending order, select the 'Sort Labels(desc)' option. To restore the original order, select the 'Sort Labels(none)' option.

l. In the 'Display Style' panel, select the type of input control that should be used for the parameter.



m. Press 'OK' to close the 'Parameter Properties' dialog box.

9. Add a 'Button' component to allow the user to submit the form.

To add a 'Clear' button, see *Adding a 'Clear' button to a Parameter Sheet*.

10. Arrange the components on the parameter sheet as desired.

Note that the parameter sheet canvas is the 'Detail' band of a Section component, and provides all of the arrangement features of a section band (align, distribute, etc.) See the following sections for more information about arranging components: *Moving and Resizing Elements in a Section*, *Aligning Elements in a Section*,

Distributing Elements in a Section, *Changing Sizes of Elements in a Section*.

11. Press the 'Save' button in the toolbar to save the embedded parameter sheet.

See Also

Reusable Parameter Sheets, for information about how to create external (reusable) parameter sheets.

Walkthrough: Embedding a Parameter Sheet in a Report

In the example that follows, you will create an embedded parameter sheet that prompts the user for a list of products. First, you will need to create a new report that uses parameter prompting. 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 'Report' node. In the right 'Types' panel, select 'Blank Tabular Report' and click 'OK'. This creates a new report.

3. Expand the Style Studio **Toolbox** panel.



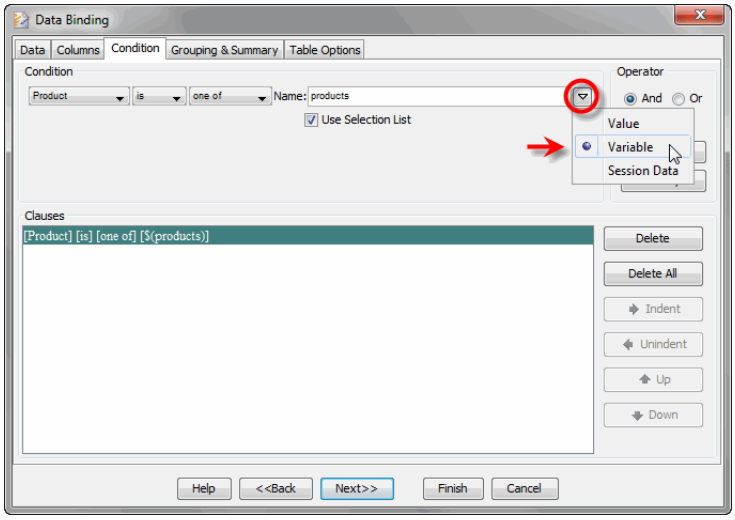
4. Click the 'Table' button in the **Toolbox** panel to open the 'Data Binding' dialog box for a new table.

5. Under the **Data** tab, bind the table to 'Order Details' query in the 'Sample Queries' folder of the 'Orders' data source.

6. In the **Condition** tab, specify the following parameterized condition.

   ```
   [Product][is][one of][$(products)]
   ```

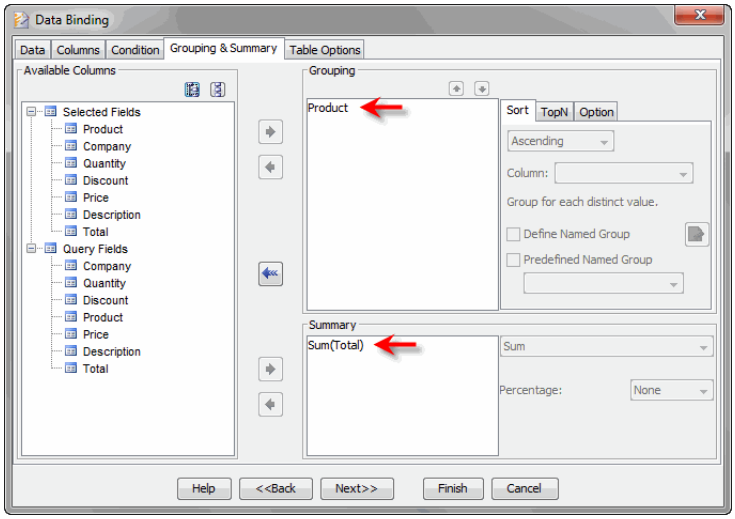
   To enter the parameter name '`products`' on the right side of the statement, select the 'Variable' option from the pop-up menu on the right side of the condition, and type "products" into the 'Name' field. (See *Filtering Data in Data Binding Wizard* for more details on specifying conditions.)



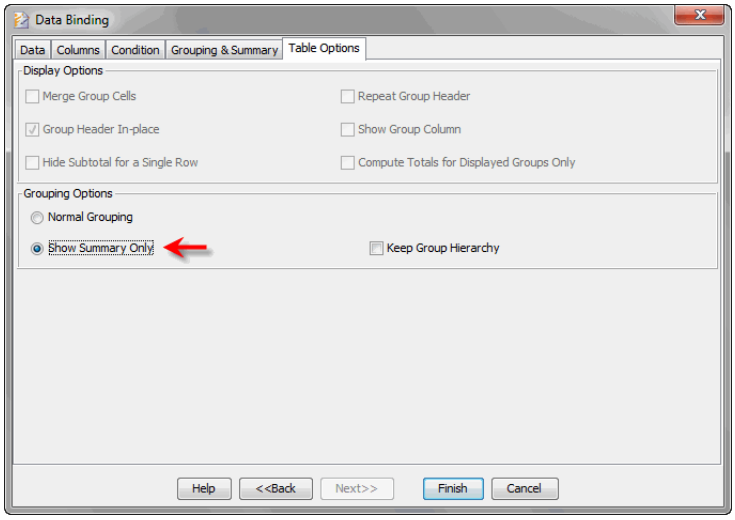
7. Enable the ‘Use Selection List’ option. (This provides a selection list when default prompting is used.)

8. Click ‘Append’ to add the condition to the table.

9. In the **Grouping & Summary** tab, drag the ‘Product’ field to the ‘Grouping’ panel. Drag the ‘Total’ field to the ‘Summary’ panel.



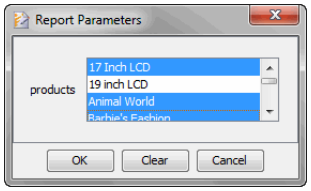
10. In the **Table Options** tab, enable the ‘Show Summary Only’ option.



11. Press 'Finish' to close the Wizard and create the table.

12. Press the 'Save' button on the toolbar and select the **Repository Report** tab. Save the report under the name "Parameter Sheet Report".

13. Press the 'Preview' button on the toolbar to preview the report. Observe that the report prompts the user to enter a value for the 'products' parameter using the default parameter prompting mechanism.



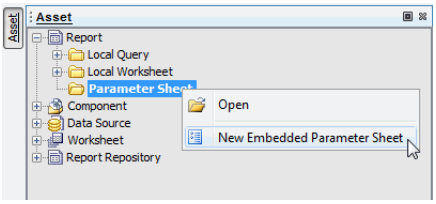
14. Select a set of products, and press 'OK'. The generated report contains a table that lists only the selected products.

Product	Total
17 Inch LCD	373936.5
Animal World	104158.6
Barbie's Fashion	156542.45
Total	634637.55

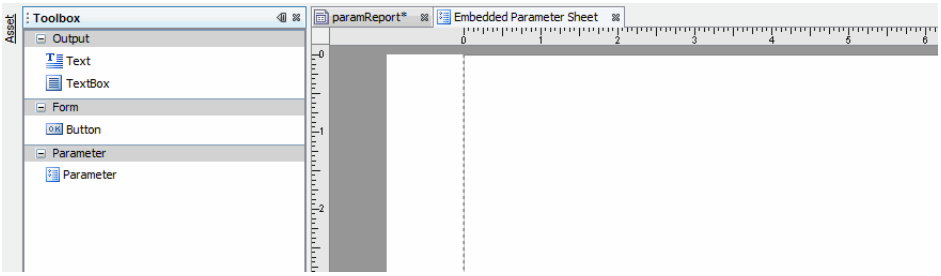
You have created a report that uses the default parameter prompting mechanism to obtain a value for the "`products`" parameter. You will now add a custom parameter sheet to the report. This parameter sheet will replace the default prompting mechanism.

15. Expand the Style Studio **Asset** panel.

16. Expand the ‘Report’ node on the tree. Right-click on the ‘Parameter Sheet’ option, and select ‘New Embedded Parameter Sheet’.



17. This opens a new embedded parameter sheet for editing.

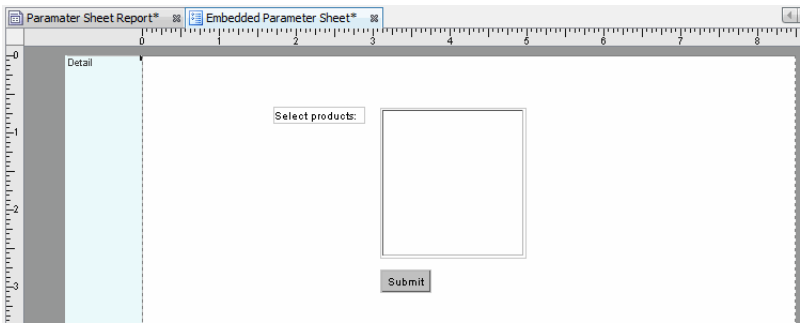


18. Drag a Text component from the **Toolbox** panel to the parameter sheet. Enter the text “Select products:”. This will be the input label.

19. Drag a Parameter component from the **Toolbox** panel into the parameter sheet. This will be product input list.

20. Drag a Button component from the **Toolbox** panel to the parameter sheet. This will be the Submit button.

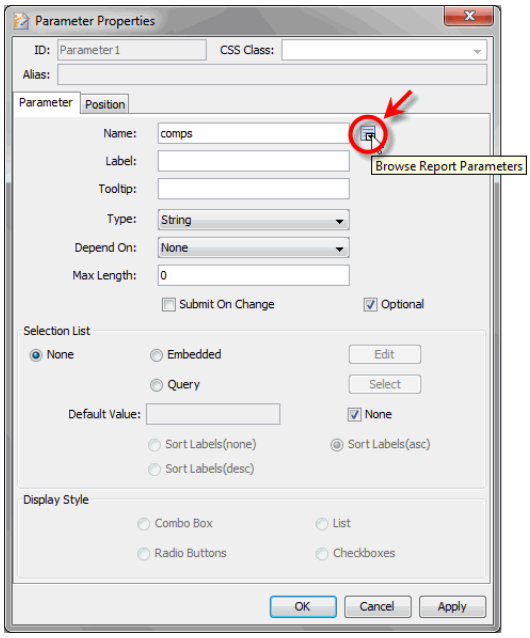
21. Position the elements as shown. Make the Parameter element taller by selecting it and dragging its bottom border downward.



22. To configure the Parameter element to control the “`products`” parameter, follow these steps:

a. Right-click the Parameter component, and select ‘Properties’ from the context menu. This opens the ‘Parameter Properties’ dialog box.

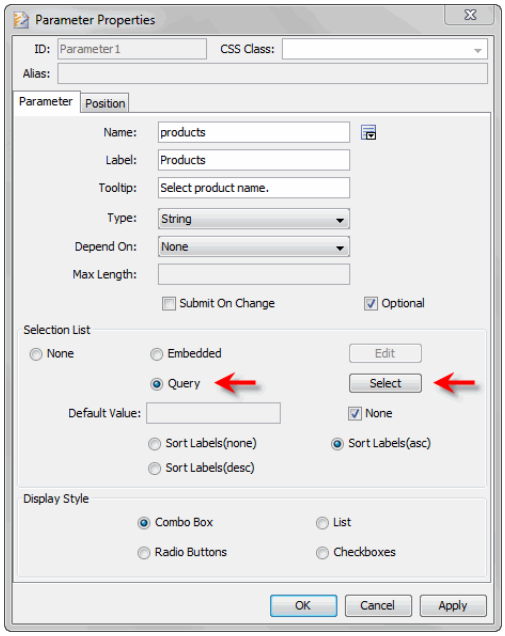
a. Press the ‘Browse Report Parameters’ button, and select the “`products`” parameter. This will allow the Parameter component to set the value of the ‘`products`’ parameter.



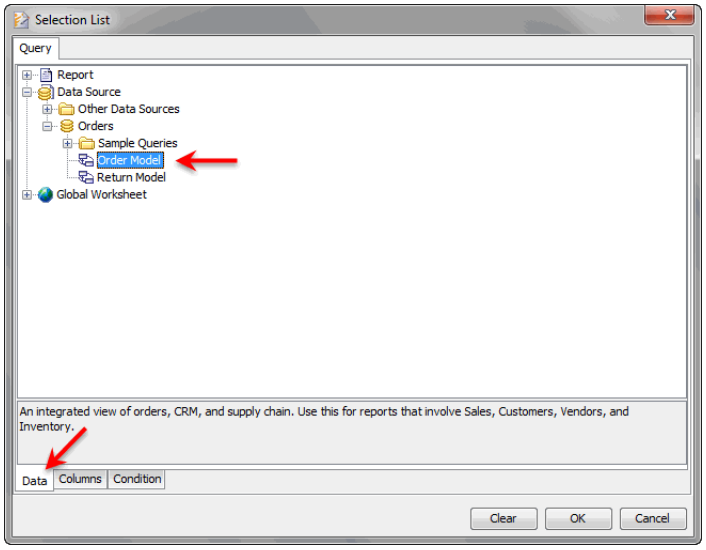
b. In the ‘Label’ field enter “Products”.

c. In the ‘Tooltip’ field enter text to display when the user hovers the mouse over the input control. For example, type “Select product name.”

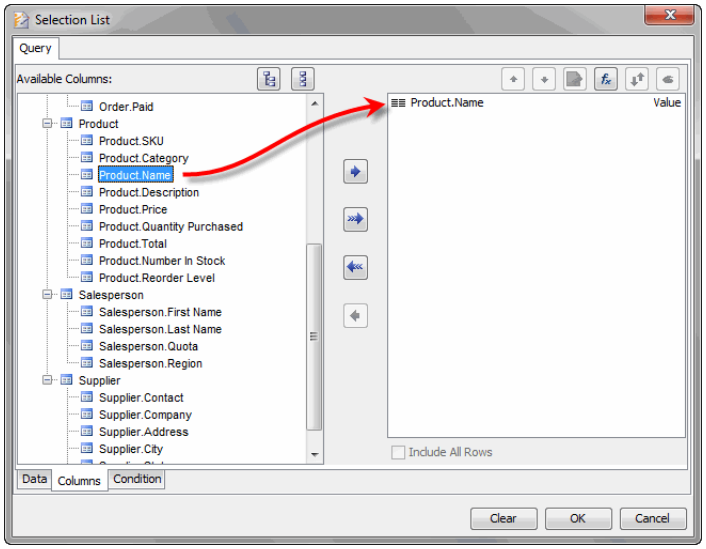
d. In the ‘Selection List’ panel, select ‘Query’ and click the ‘Select’ button. This opens the ‘Selection List’ dialog box.



e. In the bottom **Data** tab, select 'Order Model' from the 'Orders' data source.

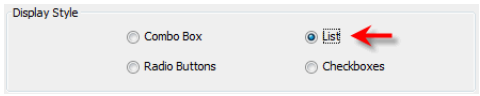


f. In the bottom **Columns** tab, select 'Product.Name'. This field will provide both the labels and values for the Parameter input control.



g. Press ‘OK’ to close the ‘Selection List’ dialog box.

h. In the ‘Display Style’ panel, select the ‘List’ option.



i. Press ‘OK’ to exit the ‘Parameter Properties’ dialog box and return to the parameter sheet.

23. Press the ‘Save’ button in the toolbar to save the embedded parameter sheet. (The parameter sheet is saved together with the report.)

24. Close the parameter sheet.

25. Verify that the parameter sheet is now functional:

    a. Press the ‘Preview’ button in the report toolbar to preview the report. The report should prompt you for the ‘`products`’ parameter using the parameter sheet you just created.

    b. Select the first three products and click ‘Submit’.



The report is now filtered based on your input.

Product	Total
17 Inch LCD	373936.5
19 inch LCD	497007.5
Animal World	104158.6
Total	975102.6

7.4 Hyperlinks

Note: Hyperlinks are not available in Style Report Pro.

You can add hyperlinks to Text, Textbox, Table, Section, and Chart elements. These hyperlinks can direct the user to another report, Viewsheet, or URL. For exported reports, hyperlinks are supported in 'PDF', 'HTML' and 'RTF' formats. If the hyperlink is associated with a Table, Section, or Chart, you can pass the contents of the clicked element as a parameter to the drill-down report or URL.

7.4.1 Defining a Hyperlink

To define a hyperlink, follow the steps below:

1. Select the component on which to place the hyperlink. This can be a Text element, Crosstab cell, Table cell, Chart dimension label, or Chart display element (slice of a pie chart, bar in a bar chart, point in a point chart, etc.).

   Hyperlinks cannot be defined on radar plot dimension labels.

   For a Table or Section element, you can place a hyperlink on any column; the values of all columns are then available as hyperlink parameters. In a crosstab, hyperlinks on the row and column headings can pass the heading text as parameter, while hyperlinks on data cells can pass both header and cell data values. For a Chart element, a hyperlink on the display element can pass X-axis labels, Y-axis values, and subseries (legend) values as parameters. A hyperlink on the dimension label itself can pass only the dimension labels as parameters.

2. Open the Style Studio **Properties** panel. (Select 'Properties Pane' from the 'Windows' menu.)

3. Click the component to which you want to add hyperlinks. Select the **Hyperlink** tab in the **Properties** panel.

4. Select the 'Asset Link' option to target a drill-down report or Viewsheet, or select the 'Web Link' option to target a web page.

5. If you selected 'Asset Link', choose the desired drill-down report or Viewsheet from the 'Link' menu.

To make the report link dynamic, enter *{param_name}* as the 'Link' text, where *param_name* is a hyperlink parameter that contains the deployed report name.

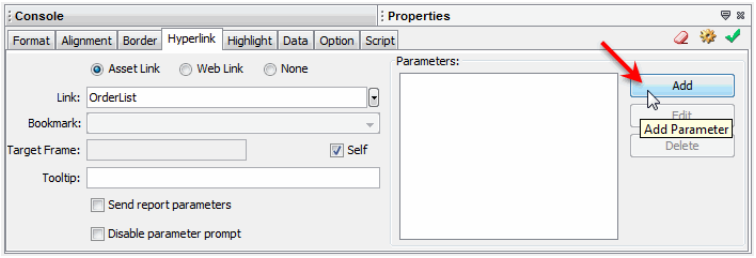
Assets are populated in this 'Link' menu from the *repository.xml* and *asset.dat* file or *asset.dat.d* directory in the repository. The *repository.xml* file lists reports that have been deployed to the server (see *Incremental Deployment of Assets* in *Advanced Server Administration* and *Deploying an Asset* in *Using Style Studio*), and *asset.dat*/*asset.dat.d* lists Viewsheets that have been saved on the server. If there are no deployed assets visible in the 'Link' menu, you can manually enter the name as it appears in the repository. See *Configuring a Repository* for information on specifying the repository location.

6. If you select 'Web Link', enter the target URL in the 'Link' field. (For example, "`http://www.google.com/search`".) You can also specify a table column to provide the URL links by selecting the desired column from the 'Link' menu.

7. (Optional) Enable 'Send report parameters' to pass all existing report parameters to the drill-down report, Viewsheet, or URL. (See *Passing Report Parameters* for details on synchronizing parameters.)

8. Select 'Disable parameter prompt' to suppress the default parameter prompt produced by the drill-down report or Viewsheet. (Prompting is automatically suppressed for parameters explicitly selected in the 'Parameters' panel.)

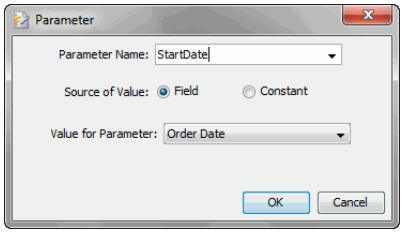
9. In the 'Parameters' panel, press the 'Add' button. This opens the 'Parameter' dialog box.



Enable 'Preview Server Features' on the **General** tab of the 'Style Studio Configuration' dialog box (select 'Configure' from the 'File' menu).

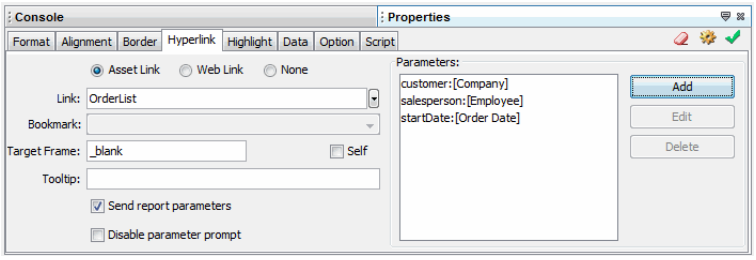
10. In the 'Name' field of the dialog box, select the name of the parameter expected by the receiving report, Viewsheet, or URL.

11. Select 'Field' if the value passed in the hyperlink should be provided by a field value. Select the desired field name from the 'Value' menu. (See *Passing Report Parameters* for details on synchronizing parameters.)



12. Select ‘Constant’ if the value passed in the hyperlink should be a fixed value. Specify the data type of the constant in the ‘Type’ field, and enter the desired value in the ‘Value’ field.

13. Repeat the above steps to add additional parameters to the hyperlink.



14. In the ‘Target Frame’ field, specify a target window for the hyperlink.

    You can enter a name or an HTML target keyword, such as “`_blank`,” to open the target in a new window or tab. If you leave the ‘Self’ option enabled, the target report or web-page will replace the parent report in the browser.

15. If applicable, in the ‘Bookmark’ field, select the Viewsheet bookmark that you want the hyperlink to target. The Viewsheet will automatically open this bookmark when the user clicks the hyperlink.

16. In the ‘Tool Tip’ field, specify the text that should appear when the user hovers the mouse over the hyperlink in the report.

    For Table hyperlinks only, the tool tip can include field values for the current record. To insert the field value into the tooltip, use the placeholder “`{field name}`”. For example, for a Table with fields ‘Order Number’ and ‘Employee’, you can specify the following tooltip: “Invoice for order `{Order Number}` placed by `{Employee}`”.

17. Press the ‘Apply’ button.

To display hyperlink underlining by default, set hyperlink.indicator=true in *sree.properties*.

18. Add any desired visual formatting to the hyperlinks (e.g., underlining).

19. Preview the report to test the hyperlinks.

Passing Report Parameters

You can pass report parameters to a target report, Viewsheet, or web page. The following rules apply:

- **Passing parameters to a report**: When you pass parameters to a drill down report, the parameter names in the sending report must exactly match the parameter names in the receiving report. See *Defining and Editing Report Parameters* for more information.
- **Passing parameters to a Viewsheet**: When you pass parameters to a Viewsheet, the parameter names in the sending report must exactly match the Variable names in the associated Worksheet. (See *Using a Variable* in *Advanced Data Mashup*.) Alternatively, you can access the parameters within Viewsheet script via the `parameter` object (see *parameter* in *Advanced Dashboard Scripting*).
- **Passing parameters to a web page**: When you pass parameters to a web page, the parameter names should exactly match the parameter names expected by the receiving web page.

7.4.2 Hyperlinks to a Target (Anchor)

You can designate any report element as a *target* (similar to an HTML *anchor*), and hyperlink to the target from another element within the same report. You can hyperlink to a fixed target, or dynamically select the target based on user action. Targeted hyperlinks can also be exported to PDF format.

Defining a Target (Anchor)

To define an element (Tables, Charts, Text) as a target, follow the steps below:

1. Right-click on the element, and select ‘Properties’ from the context menu. This opens the ‘Properties’ dialog box.

   For a *Section Element*, you can specify dynamic target names for elements within the bands. This is useful when you wish to link to a specific data record within an element, as opposed to the element itself.

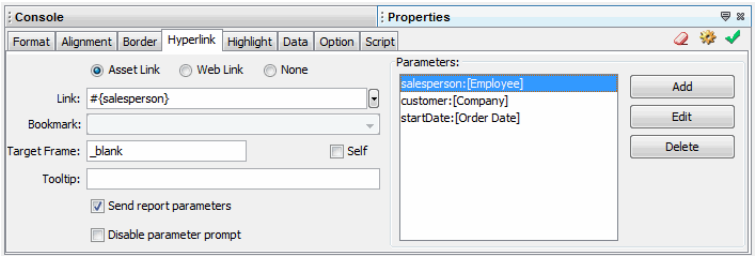
2. Click the **Attributes** tab of the ‘Properties’ dialog box, and enter a valid target name in the ‘Target’ field.

3. Click ‘OK’ to close the dialog box.

Hyperlinking to a Target

To define a hyperlink to a target, follow the steps in *Defining a Hyperlink*. Choose the 'Asset Link' option, and enter one of the following in the 'Link' field:

- `#targetName` to link to fixed target with name "`targetName`" in the same report
- `#{ParameterName}`, to dynamically link to a target in the same report with name "`ParameterName`," where `ParameterName` is one of the parameters names in the 'Parameters' list of the **Hyperlink** tab.



Walkthrough

In the next example you will create a report with a Chart and a Section, one below the other. Both elements will be bound to the 'All Sales' query. You will define hyperlinks for the Chart so that when you click on an employee's bar in the Chart, you will be directed to the page of the report containing the Section group header for the corresponding employee.

To create the report, follow the steps below:

1. Click the 'New' button on the Style Studio menu. This opens the 'New Asset' dialog box.

2. In the left panel select 'Report'. In the right 'Types' panel, select 'Blank Tabular Report' and click 'OK'.

3. From the Style Studio **Toolbox** panel, drag a Chart component into the new report. This opens the **Data Source** and **Binding** panels. Make the following selections:

   a. In the **Data Source** panel, expand the 'Orders' data source, the 'Sample Queries' folder, and the 'All Sales' query.

   b. Drag the 'Employee' field from the 'All Sales' query to the 'X' region in the **Binding** panel.

   c. Drag the 'Total' field from the 'All Sales' query to the 'Y' region in the **Binding** panel

4. Click below the Chart to place the insertion point at that location.

5. In the Style Studio **Toolbox** panel, click the Section button. This opens the 'Data Binding' dialog box for a new Section. Make the following selections:

   a. Select the **Data** tab. Expand the 'Orders' data source, the 'Sample Queries' folder, and the 'All Sales' query.

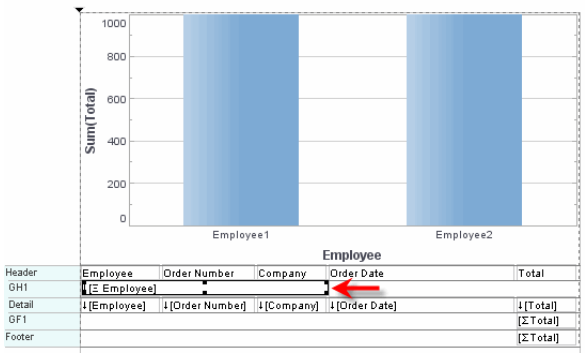
   b. Select the **Grouping & Summary** tab. Drag the 'Employee' field to the 'Grouping' panel, and drag the 'Total' field to the 'Summary' panel.

   c. Click 'Finish' to close the 'Data Binding' dialog box.

6. Define a target for every 'Employee' group header in the Section. Note that the 'Employee' group header is generated once for each employee name. To label each generated header target with the actual employee name, follow the steps below:

   a. Expand the Style Studio **Properties** panel. (Select 'Properties Pane' from the 'Window' menu.)

   b. Select the 'Employee' Text element in the 'GH1' band.



   c. In the **Properties** panel, click the **Script** tab. Enter the following script:

```
target = field.Employee
```

   Click the green 'Apply' button. This script will set the 'Target' property of the header text element in each iteration of the header band to the corresponding employee name. (See *Target Property* in *Advanced Report Scripting* for more information).

   d. Set the font of the header Text element to Bold-12 Point. (Use the buttons in the Style Studio toolbar.)

7. Hyperlink the chart to the Section header targets. Follow the steps below:

   a. Click on one of bars in the Chart.

   b. Select the **Hyperlink** tab in the **Properties** panel.

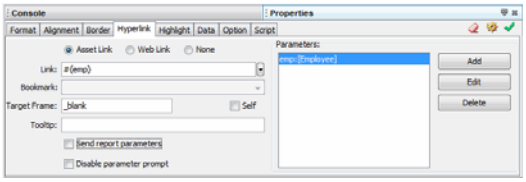
   c. Select the 'Asset Link' option.

   d. In the 'Link' field, type "`#{emp}`".

   e. In the 'Parameters' panel, press the 'Add Parameter' button. This opens the 'Parameter' dialog box.

   f. In the 'Name' field of the dialog box, enter "`emp`" as the parameter name.

   g. Select the 'Field' option and select 'Employee' from the 'Value' menu.



   h. Click the green 'Apply' button.

8. Preview the report. Verify that you can click on any bar in the chart to link to the corresponding Section header.

Enable 'Preview Server Features' on the **General** tab of the 'Style Studio Configuration' dialog box (select 'Configure' from the 'File' menu).

7.5 Highlighting

When you bind a Text, Textbox, Table, or Chart element to data, you can specify a highlight for the element. A *highlight* is a conditional format that applies a visual effect based on the value of the data displayed by the element. A highlight it comprised of a foreground color, a background color, and a font, which are applied to the element when a specific data condition is met.

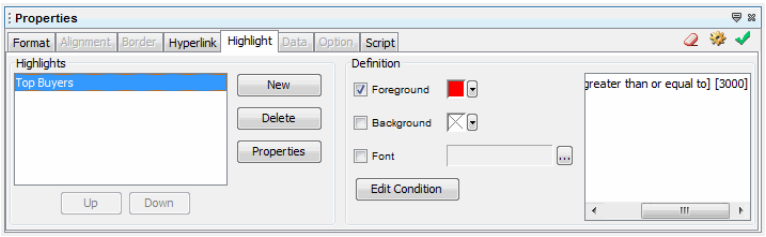
7.5.1 Highlighting a Text or Textbox Element

To add a highlight to a Text or Textbox element, follow these steps:

1. Open the Style Studio **Properties** panel. (Select 'Properties Pane' from the Style Studio 'Window' menu.)

2. Click on the Text element that you wish to highlight.

3. Select the **Highlight** tab in the **Properties** panel.



4. Click the 'New' button. Enter a name for the new highlight.

5. Select a foreground color, background colors, and/or a specific font to apply when the highlight condition is satisfied.

6. Click 'Edit Condition', and specify the condition that determines when the highlight is applied. (See *Filtering Data in Data Binding Wizard* for more information about setting conditions.)

You can create multiple highlights for the same Text element, with each highlight representing a different condition. The Text displays the highlight whose corresponding condition is met by the current data.

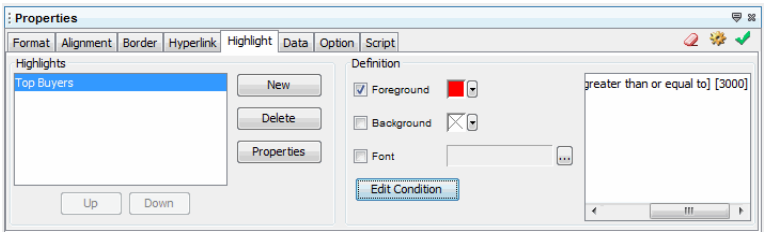
7.5.2 Highlighting Data in a Table

To highlight data in a Table, follow these steps:

1. Open the Style Studio **Properties** panel. (Select 'Properties Pane' from the Style Studio 'Window' menu.)

2. Select the data cell or row of the Table to which you want to apply the highlight. (See *Selecting a Cell, Row, or Column of a Table* for more information.)

3. Click the **Highlight** tab in the **Properties** panel.



4. Specify the highlight formatting and condition. This is the same procedure as for a Text element highlight. See *Highlighting a Text or Textbox Element* for details.

Walkthrough

In this example, you will highlight the cells in the 'Total' column with a a red background when they contain a value under $5000, and highlight the *entire row* with a green background when the total is above $20,000. 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 'Report' node. In the right 'Types' panel, select 'Blank Tabular Report' and click 'OK'.

3. Expand the Style Studio **Toolbox** panel.



4. Click the 'Table' button in the **Toolbox** panel to open the 'Data Binding' dialog box for a new table.

5. Under the **Data** tab, bind the table to the 'All Sales' query from the 'Orders' data source, and press 'Finish'.

Add the red highlights for low totals (under $5000):

6. Open the Style Studio **Properties** panel. (Select 'Properties' from the 'Window' menu.)

7. Select the body cell under the 'Total' header of the Table.

8. Select the **Highlight** tab in the **Properties** panel.

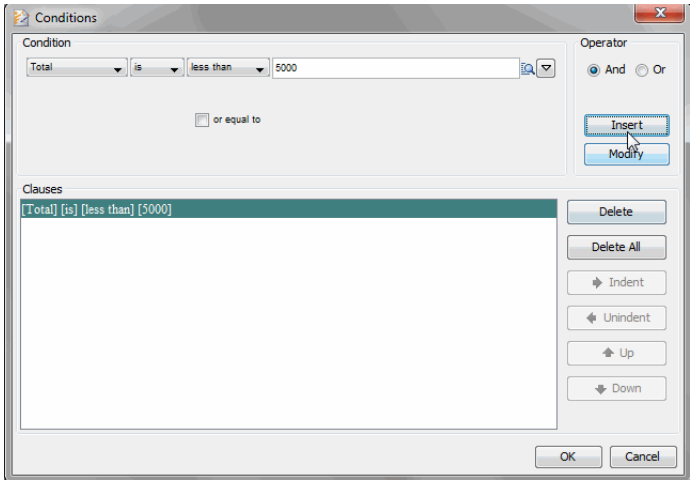
9. Click 'New' to add a new highlight, and name it "Poor Performance".

10. Select a red color for the 'Background' in the 'Properties' pane.

11. Click the 'Edit Condition' button, and append the following condition:

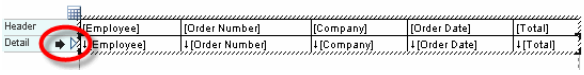
```
[Total] [is] [less than] [5000]
```

12. Click 'OK' to close the dialog box.



Next, add the green row-based highlights for high totals (over $20,000):

13. Select the body row of the table by clicking next to it in the left margin. You will see an arrow indicating that the row is now selected.



14. Select the **Highlight** tab in the **Properties** panel.

15. Click ‘New’ to add a new highlight, and name it ‘Good Performance’.

16. Select a green color for the ‘Background’ in the ‘Properties’ panel.

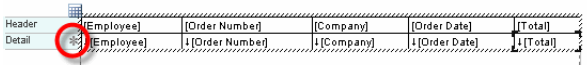
17. Click the ‘Edit Condition’ button, and append the following condition:

    ```
    [Total] [is] [greater than] [20000]
    ```

18. Preview the table and observe the cell and row highlighting.

Employee	Order Number	Company	Order Date	Total
Annie	12002	Nova Tera	2008-03-13 00:00:00	1211
Annie	12003	Folk Mining	2008-03-14 00:00:00	6735
Annie	12005	Quick Stop	2008-03-19 00:00:00	10526
Annie	12007	Specialty Retail	2008-03-20 00:00:00	19895
Annie	12012	Folk Mining	2008-04-01 00:00:00	11400
Annie	12014	Nova Tera	2008-04-09 00:00:00	50882
Annie	12020	Computer Tech	2008-04-14 00:00:00	21119
Annie	12030	West Coast Customs	2008-05-01 00:00:00	25389
Annie	12033	Nova Tera	2008-05-07 00:00:00	94937
Annie	12035	Direct Sales	2008-05-11 00:00:00	5085
Annie	12037	Northwest Outfitters	2008-05-12 00:00:00	5818
Annie	12043	Folk Mining	2008-05-31 00:00:00	49370
Annie	12048	Whittaker & Co, Inc	2008-06-06 00:00:00	12761
Annie	12052	George Services	2008-06-11 00:00:00	20783
Annie	12053	Specialty Retail	2008-06-15 00:00:00	5679
Annie	12060	Tumbleweed Gaming	2008-06-23 00:00:00	354
Annie	12068	Direct Sales	2008-07-05 00:00:00	21056
Annie	12069	FISGA Corp	2008-07-05 00:00:00	88173
Annie	12070	The Big Cat	2008-07-07 00:00:00	35475
Annie	12075	Northwest Outfitters	2008-07-16 00:00:00	15158
Annie	12076	Infinity Software	2008-07-20 00:00:00	5308
Annie	12091	Direct Sales	2008-08-28 00:00:00	6300
Annie	12092	West Coast Customs	2008-08-29 00:00:00	447
Annie	12094	Folk Mining	2008-09-03 00:00:00	74155

The existence of a row-based highlight is indicated by an asterisk in the left margin next to the row, as shown below. To edit a row-based highlight, click the asterisk to select the row, and then proceed as shown above.



7.6 Formatting

A format specifies the algorithm for converting an object to and from a string value. Several common format types can be specified in the **Format** tab of Style Studio's **Properties** panel.

7.6.1 Format Types

The formats available on the visual interface do not include all possible formats. If a format is not on the supported list, it can be specified using the reporting API from the application.

Table 14. Format Types available in Style Studio

FORMAT TYPE	DESCRIPTION
Date Format	Specify conversion for date/time values. See *Date and Time Formats*.
Text Format	Specify formatting of string values. See *Text Formats*.
Number Format	Specify conversion for numeric values. See *Number Formats*.
Currency Format	Specify formatting of numbers as currency (with a currency symbol)
Percent Format	Specify formatting of numbers as percentages

Date and Time Formats

To insert a single-quote in the date format, use a double single-quote (e.g., MMM "yy).

For date and number formats, an optional format pattern string can be specified. 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'.

Date and time formats can be specified at a number of levels:

1. *Object Level*: Date or time format applied to a specific report element (e.g., table column) using the **Format** tab of the **Properties** panel in Style Studio.

2. *Report Level*: Date or time format applied to a specific report using the 'Report Properties' dialog box in Style Studio. (See *Report Properties*.)

3. *Data Model Level*: Date or time format applied to a specific data field. (See *Adding a Format to a Data Field* in *Advanced Data Modeling*.)

4. *Server Level*: Default date or time format applied to all reports using the 'Presentation' page of Enterprise Manager. (See *Presentation* in the *Advanced Server Administration*.)

5. *Worksheet Level*: Date or time format applied to a specific column in an Embedded Table in a Worksheet. (See *Changing the Column Type in an Embedded Table* in *Advanced Data Mashup*.)

The more specific formatting instruction overrides the more general formatting instruction. For example, the format applied to a particular report element (e.g., table) overrides any formatting specified at the report, model, or server levels.

Different geographical regions format dates differently. Instead of explicitly specifying a format for date fields, you can specify a Local Date format (e.g., 'Full', 'Long', 'Medium', 'Short') that automatically adapts itself to the locale of the reporting environment.

When no date or time format is found at any level (element, report, model, or server), an un-localized default format is used.

Text Formats

The 'Text' format allows you to add a prefix or suffix to string data. The special tag '`{0}`' acts as a placeholder for the actual data.

Example 1: Adding a Prefix

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: Annie
```

Example 2: Adding a Prefix and Suffix

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.

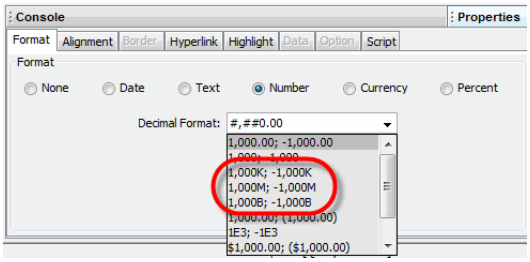
For information on how to map Text IDs to replacement text in the *SreeBundle* localization file, see *Localizing Reports, Viewsheets, Data Models/Queries* in *Advanced Server Administration*.

Number 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*.

7.6.2 Table Formatting

A few formatting options are normally only applicable to tables with query binding. Although the options can be applied to any table, they are dependent on knowing the exact table columns, which is normally only available during design time if a query binding exists.

Cell Alignment

Row attributes override column attributes.

Cell alignment affects the position of the cell contents inside the cell boundaries. Style Studio allows you to control both vertical and horizontal alignments, and to apply the alignment setting to an individual cell, or to an entire column or row. See *Selecting a Cell, Row, or Column of a Table* for more information.

Walkthrough

In this example, you will create a new table and add formatting to one of the columns.

1. Click the 'New' button in the Style Studio toolbar. This opens the 'New Asset' dialog box.

2. In the left panel select the 'Report' node, and in the right 'Types' panel select 'Blank Tabular Report'. Click 'OK'.



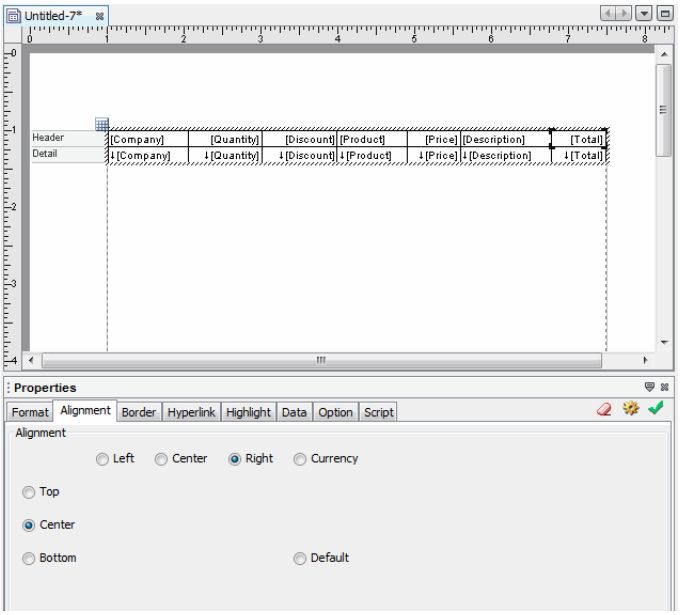
3. Open the Style Studio **Toolbox** panel, and click the 'Table' button. This opens the 'Data Binding' dialog box for a new Table.

4. Under the **Data** tab, bind the table to the ‘Order details’ query in the ‘Sample Queries’ folder of the ‘Orders’ data source, and click ‘Finish’.

5. Select the Table. Click in the margin above the ‘Total’ column to select the entire column. (See *Selecting a Cell, Row, or Column of a Table*.)

6. Expand the Style Studio **Properties** panel. (Select ‘Properties’ from the Style Studio ‘Window’ menu.)

7. Click the **Alignment** tab. Select the ‘Right’ radio button to align the contents of the ‘Total**’** column to the right.



All cells in the ‘Total’ column are now aligned right. Select the ‘Default’ option to clear the alignment setting.

Cell Data Format

Walkthrough

Next, you will add some formats to the table. Both the ‘Price’ and ‘Total’ columns display currency values, so you will apply a currency format to these columns. The ‘Discount’ column contains percentages, so you will apply a percentage format to that column.

1. Expand the Style Studio **Properties** panel. (Select ‘Properties’ from the Style Studio ‘Window’ menu.)

2. Select the **Format** tab in the **Properties** panel.

3. Select the detail cell in the ‘Price’ column of the table. In the **Format** tab of the **Properties** panel, select the ‘Currency’ option.

4. Select the detail cell in the ‘Total’ column of the table. In the **Format** tab of the **Properties** panel, select the ‘Currency’ option.

5. Select the detail cell in the ‘Discount’ column of the table. In the **Format** tab of the **Properties** panel, select the ‘Percent’ option.

6. Preview the report.

Company	Quantity	Discount	Product	Price	Description	Total
Interstate Shop	6	0%	NetStorage	$2,850.00	500 GB network storage	$17,100.00
Interstate Shop	5	0%	Animal World	$59.00	Educational software	$295.00
Interstate Shop	1	0%	Mega Icons	$550.00	Over 10,000 icons	$550.00
Interstate Shop	16	0%	WebCalendar	$248.00	Web enabled calendar/org	$3,968.00
Interstate Shop	18	0%	19 inch LCD	$310.00	19 inch LCD monitor	$5,580.00
Interstate Shop	18	0%	Sync Me	$145.00	Palm power sync	$2,610.00
Interstate Shop	15	0%	Math for Me	$235.00	Math tutor	$3,525.00
Interstate Shop	16	0%	Wireless Mouse	$40.00	Wireless optical USB mouse	$640.00
Interstate Shop	1	0%	NetStorage	$2,850.00	500 GB network storage	$2,850.00
Interstate Shop	13	0%	Mega Icons	$550.00	Over 10,000 icons	$7,150.00
Interstate Shop	2	0%	17 Inch LCD	$210.00	17 inch LCD monitor	$420.00
Interstate Shop	12	0%	Green Planet	$75.00	Personal health monitor	$900.00
Interstate Shop	1	0%	MeToo AppServer	$3,000.00	Application server	$3,000.00
Interstate Shop	7	0%	Barbie's Fashion	$79.00	Girl's game	$553.00
Interstate Shop	11	0%	Fast Mail	$39.00	Lightweight email agent	$429.00
Interstate Shop	3	0%	Easy Chess	$35.00	Chess game	$105.00
Interstate Shop	6	0%	Web Bridge	$105.00	Bridge game	$630.00
Interstate Shop	18	0%	17 Inch LCD	$210.00	17 inch LCD monitor	$3,780.00
Interstate Shop	16	0%	Xconnect Server	$2,250.00	XML connectivity tool	$36,000.00
Interstate Shop	13	0%	Fancy Menus	$49.00	Collection of menus	$637.00
Interstate Shop	14	0%	Easy Chess	$35.00	Chess game	$490.00
Interstate Shop	10	0%	Fast Go Game	$200.00	Chinese Go game	$2,000.00
Interstate Shop	13	0%	Info Folder	$399.00	Document management tool	$5,187.00
Interstate Shop	1	0%	17 Inch LCD	$210.00	17 inch LCD monitor	$210.00
Interstate Shop	18	0%	Sync Me	$145.00	Palm power sync	$2,610.00

Cell Presenter

Walkthrough

A ‘Presenter’ is a graphical representation of a numeric value. This can range from a simple text string to a complex graphical presentation. Follow these steps to select a Table Presenter.

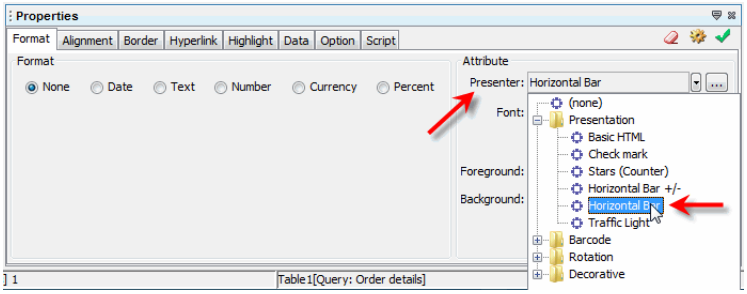
1. Expand the Style Studio **Properties** panel. (Select ‘Properties’ from the Style Studio ‘Window’ menu.)

2. Select the **Format** tab in the **Properties** panel.

3. Select the detail cell in the ‘Quantity’ column.

See *Available Presenters* for additional options.

4. Click the ‘Presenter’ menu in the ‘Attribute’ panel of the **Format** tab in the **Properties** panel, and select the ‘Horizontal Bar’ Presenter (in the ‘Presentation’ folder).



5. Click the button next to the 'Presenter' menu. This opens the 'Presenter' dialog box. Change the 'maximum' to "20", and click 'Close'.

6. Preview the report. Note that the numerical display for the 'Quantity' column has been replaced by the specified graphical display.

Available Presenters

Presenters are grouped under categories of Presentation, Barcode, Rotation, Decorative. They can be assigned to a column or row of a table. The following table presents some commonly used presenters.

Table 15. Presenters

PRESENTER	DESCRIPTION
Basic HTML	Display strings with embedded tags, such as <b> for bold and <u> for underline
Horizontal Bar	Draws a horizontal bar based on the numeric cell value, similar to the bar chart.
Horizontal Bar +/-	Same as Horizontal Bar, but handles both positive and negative numbers.
Barcode	Barcode presenters can be used to support various bar code displays.
2D Code	Two-dimensional QR Code for alphanumeric data.
Checkmark	If a cell contains a true Boolean value, the presenter draws a check mark in the cell. Otherwise, the cell is left blank.
Button	Adds a 3D border around cells.
Stars(Counter)	Draws icons (stars) horizontally. The number of icons is determined by cell values. For example, two icons are drawn for a cell with a value of two. If a cell does not contain a numeric value, the presenter leaves the cell blank.
Shadow	Adds a shadow to cells.
Clockwise Rotation	Rotates cells by 90 degrees in clockwise direction.
Counterclockwise Rotation	Rotates cells by 90 degrees in counter-clockwise direction.

Creating Custom Presenters

The reporting API allows an application to create any presenters by implementing the presenter interface and creating a presenter class. These custom built presenters are automatically displayed in the 'Presenter' menu in the **Format** tab of the **Properties** panel.

To specify a custom presenter in Style Studio, follow these steps:

1. Add the custom presenter class to the classpath:

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

   b. Select the **Classpath** tab. Click 'Add Directory', and select the directory containing the required class.

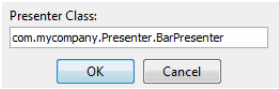
   c. Click 'OK' to close the 'Style Studio Configuration' dialog box.

2. Load the custom presenter into the Style Studio environment:

   a. Select 'Preferences' from the Style Studio 'Report' menu. This opens the 'Preferences' dialog box.

   b. Select the **Components** tab. Click 'Add' and enter the name of the presenter class, e.g., `com.mycompany.presenter.BarPresenter` (the location of which has been previously specified in the classpath).

Click 'Delete' to remove a custom presenter.



   c. Click 'OK'.

   The custom presenter will now be visible in the 'Presenter' menu in the **Format** tab of the **Properties** panel.

7.7 Report Bursting

In a large enterprise reporting deployment, it is important to handle both a large amount of data and a large number of users. Consider a case where a single report such as a monthly statement needs to be delivered to thousands of users. This could present certain inefficiencies, because to ensure data security, the same report will need to be regenerated for each user. This means that the same queries are run multiple times, taxing the database.

Report bursting is a feature that allows a report to run only once, and yet still provide the appropriate output to each consumer of the report. This greatly reduces the load on the database and the reporting server, while still ensuring data security. For example, in the case of a monthly statement, a single query can be executed to return the data for all users. When the report is distributed to users, it is "burst" into parts, and each user receives the partial report containing only the information pertaining to them. Each user thus receives a report personalized with information relevant to their needs.

7.7.1 Implementing Report Bursting

The are two stages to implement 'Report Bursting':

1. Create a report with customized bursting settings. See *Report Partitioning* for more information.

2. Execute the report to generate with the full data set. See *Report Bursting* in *Advanced Server Administration*.

7.7.2 Report Partitioning

Bursting relies on the action of partitioning, or splitting, a report based on a username/data or role/data value pair. A report can only be partitioned on a single data element (Table or Section), and you must perform the partition mapping using Style Studio. Partitions are created by running a query against a live data source and retrieving a list of unique rows, then mapping that data set to another query result set. The users and roles are defined in the Enterprise Manager and cannot be changed in Style Studio. See the *Users Tab* section of the *Advanced Server Administration* for more information.

Walkthrough

You will now implement a simple example report and set its bursting conditions. (This example is available as *bursting1.srt* in the *examples\docExamples\design* directory.)

1. Click the 'New' button in the Style Studio toolbar. This opens the 'New Asset' dialog box.

2. In the left panel, select the 'Report' node. From the right 'Types' panel, select 'Blank Tabular Report' and click 'OK'.

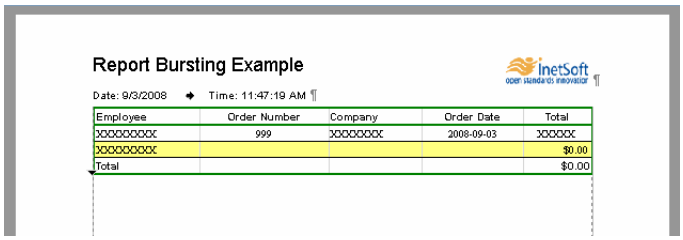
3. Expand the Style Studio **Toolbox** panel. (Select 'Toolbox Pane' from the Style Studio 'Window' menu.)



4. Click the 'Table' button in the **Toolbox** panel. This opens the 'Data Binding' dialog box for a new Table.

5. Under the **Data** tab, bind the table to the 'All Sales' query from the 'Orders' data source (under 'Sample Queries').

6. In the **Grouping & Summarization** tab, specify grouping by 'Employee' and summarization by 'Total'.

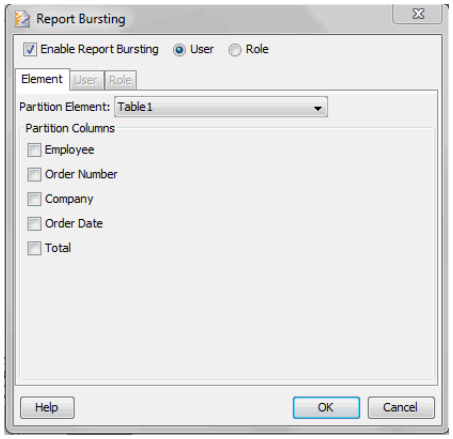


7. Press 'Finish' to close the dialog box.

8. Select 'Bursting' from the Style Studio 'Report' menu. This opens the 'Report Bursting' dialog box.

9. Select 'Enable Report Bursting'.

10. Select 'Table1' as the partition element, and select 'Employee' as the 'Partition Column'.



11. Click on the **User** tab.

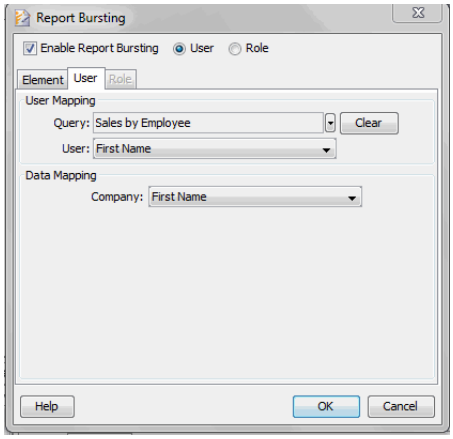
12. In the 'User Mapping' panel, select 'Orders' > 'Sample Queries' 'Sales by Employee' from the 'Query' menu.

    This query returns the names of each user for whom you will filter and burst the data. You can also create this mapping for roles using a query that returns role names.

13. Select 'First Name' as the 'User'. This is the column that returns the user names on which the data will be filtered.

Similar settings can be used in the **Role** tab to perform role mapping. The mapping can include any or all of the user's roles.

14. In the 'Data Mapping' pane select 'First Name' as the 'Employee' value.



15. Click on 'OK' to save the bursting settings, and save the report.

16. Save the report into the current working repository with the name "Bursting Example". (See *Saving a Report into the Working Repository*.)

This completes the first phase of implementing report bursting. To complete the example and then run the report for multiple users, see *Report Bursting* in *Advanced Server Administration*.

7.8 Localization/Internationalization

If deployed reports are viewed in multiple locales, it may be desirable to configure report elements to display text in a language specific to the user locale. You can configure a report element for localization by associating one or more Text ID properties with the element. The specified Text ID for each element is then used to create a mapping between the text string in the element and the replacement text string to be shown to the user.

To specify the Text ID for an element, follow the steps below:

1. Select 'Localization' from the Style Studio 'Report' menu. This opens the 'Localization' dialog box.

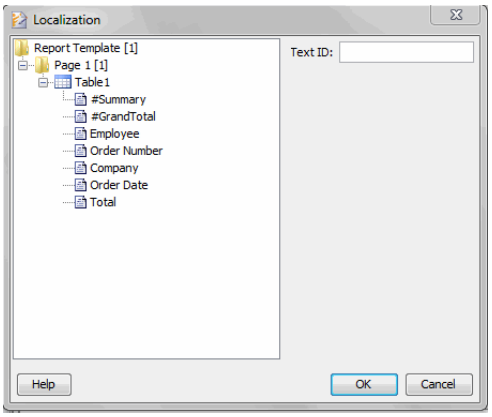
2. Select the element you wish to localize from the element tree.

3. 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. Some elements, such as tables, may have

multiple components which can each be given an independent Text ID.

4. Click 'OK' to exit the 'Localization' 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 report 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.

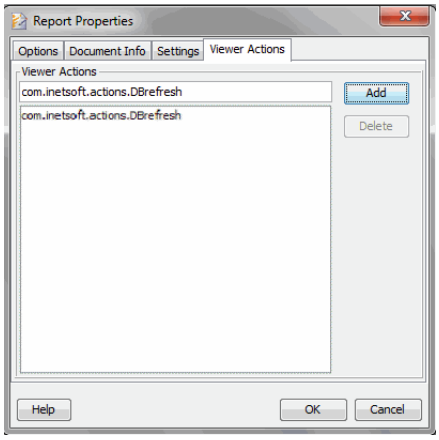
7.9 Viewer Actions

Classes that implement the ViewerAction interface can specify user-defined actions to perform on replets. These may be added in the form of toolbar buttons to every report. Through Style Studio, they may be added to individual reports. Follow the steps below:

1. Select 'Report Properties' from the Style Studio 'File' menu. This opens the 'Properties' dialog box.

2. Select the **Viewer Actions** tab.

3. Enter the name of the Viewer Action class, and click 'Add'.



4. Click 'OK' to close the dialog box.

7.10 HTML Report Features

This section covers various features relating to the HTML content of reports.

7.10.1 Adding HTML to a Report

In addition to using the regular report elements, it is possible to add report content in the form of HTML. The HTML will be added inside a paintable element that controls the location and bounds of the HTML contents.

To add HTML to a report, follow these steps:

1. Expand the Style Studio **Toolbox** panel. (Select 'Toolbox Pane' from the Style Studio 'Window' menu.)

2. Click in the report to place the insertion point at the desired location.

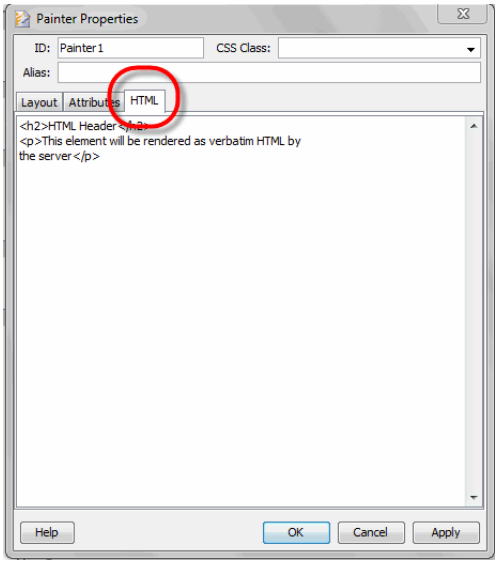


3. Click the 'Painter' button in the **Toolbox** panel. This opens the 'Painter Properties' dialog box.

4. Click on the **HTML** tab.

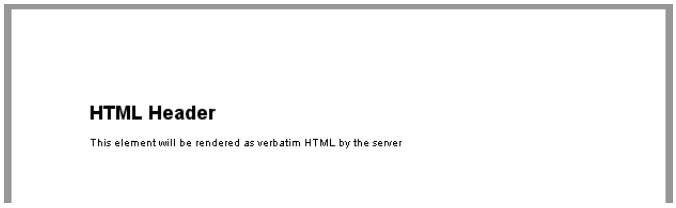
5. Add the desired HTML string, for example:

```
<h2>HTML Header</h2>
<p>This element will be rendered as verbatim HTML by the
server</p>
```



6. Press ‘OK’.

7. Preview the report.



Style Studio’s preview simulates common HTML rendering, but the HTML will be rendered directly by the browser when the user views the report in the Portal.

See Also

Cell Presenter, to display HTML within a table cell.

7.10.2 Styling a Report with CSS

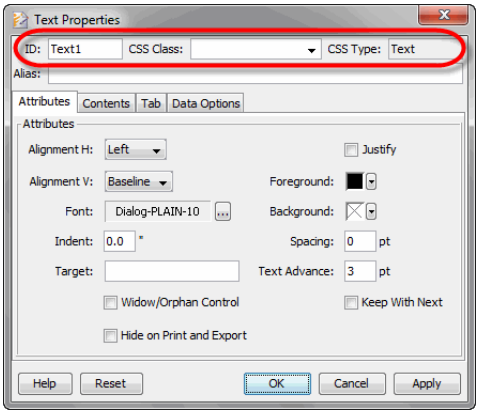
You can use a Cascading Style Sheet (CSS) to define default component styles in a centralized location, which allows you to obtain a consistent appearance across all your reports. The designer of an individual report can override this default CSS formatting as needed.

The following CSS formatting options are available for styling components:

- **CSS Type**: Every report component has a fixed *CSS Type*. You can use the CSS Type to create default styling that is automatically applied to all reports without the need to make modifications to individual reports. See *CSS Type Selectors* for a list of component types.

- **CSS ID**: Each component in a report has a unique fixed CSS ID which is the same as the component ID. The CSS ID allows you to target a particular component for styling. The CSS file should contain a CSS ID definition for each component you want to style by ID.
- **CSS Class**: You can assign a CSS Class to any component in a report. This allows you to style a group of components (all having the same CSS Class) with a single set of instructions. The CSS file should contain a CSS Class definition for each CSS Class you assign in the report.

The CSS Type, CSS ID, and CSS Class are shown in the 'Properties' dialog box for a component.



You can combine these selectors as needed. For example, you can define a default "Text" CSS Type to apply to most Text components, and also define special Text CSS Classes (e.g., 'Header1', 'Header2', 'Quoted', 'Hyperlink', etc.) to use on individual Text components. See *Creating a Component CSS Definition File* in *Advanced Server Administration* for details on how to create these CSS definitions.

Before you can apply CSS styles to report components, you must specify a CSS file for the report. See *Specifying the Report CSS File* for more information.

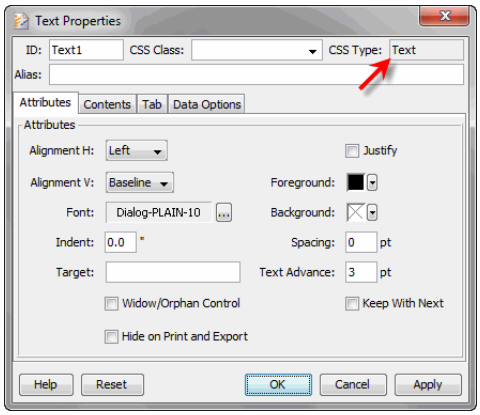
Assigning CSS Style based on Class and ID

To set a component's style by using CSS Class and ID, follow the steps below:

1. Verify that a CSS file has been specified for the report, or that a default CSS file is available. See *Specifying the Report CSS File* for more information.

2. Right-click on the component you want to style, and select 'Properties' from the context menu. This opens the 'Properties' dialog box.

The text next to the ‘CSS Type’ label indicates the CSS Type selector that should be used in the CSS file to define the style for the selected component region.




```
Note: CSS styles are overridden by styles that you set manually in
Style Studio. The ‘Reset’ button in the dialog box removes the
manually-set styles, restoring any CSS defaults. Manually-set
transparency will affect the appearance of CSS colors.
```

3. Enter an ‘ID’ and/or select a ‘CSS Class’ to define the component’s style based on ID and Class. Multiple components can share the same CSS Class but the ID is unique to a given component.

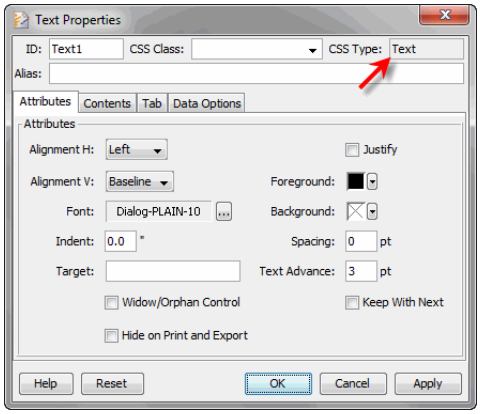
   For table components, you can select only a TableStyle ‘Class’. Note that a table style specified in the ‘Style’ panel of the ‘Properties’ dialog box overrides any CSS TableStyle formatting. You must therefore clear table styles from the ‘Style’ panel in order to apply CSS styles.

   If you do not see the desired CSS ID or CSS Class listed in the ‘ID’ or ‘CSS Class’ menus, you can manually type a new Class or ID name. However, this will not apply any styles until a corresponding definition is created in the CSS definition file.

CSS properties that have not been assigned explicit values at the component level will be set by the global default (`*`) if the CSS file specifies one. See *CSS Component Properties* in *Advanced Server Administration* for more information.

CSS Type Selectors

Every report component has a CSS type. The CSS type of a component is displayed in the top-right corner of the ‘Properties’ dialog box.



The table below lists the available component CSS types. For information on detailed formatting of Tables, Crosstabs and Formula Tables, see *CSS TableStyle Properties* in *Advanced Report Design*.

COMPONENT	REGION	CSS TYPE SELECTOR
Table, Formula Table	Table component	Table
Crosstab	Crosstab component	Crosstab
Chart	Chart component	Chart
	Chart axis titles	ChartAxisTitle[axis=x] ChartAxisTitle[axis=y] ChartAxisTitle[axis=x2] ChartAxisTitle[axis=y2]
	Chart axis labels	ChartAxisLabels[axis=x] ChartAxisLabels[axis=y] ChartAxisLabels[axis=x2] ChartAxisLabels[axis=y2]
	Chart labels generated by 'Show values' and 'Text' visual binding	ChartPlotLabels
	Chart legend title	ChartLegendTitle
	Chart legend content	ChartLegendContent
	Chart target line labels	ChartTargetLabels
Text	Text component	Text
TextBox	TextBox component	TextBox
Image	Image component	Image
Painter	Painter component	Painter
TextField	TextField component	TextField
TextArea	TextArea component	TextArea
Button	Button component	Button
Image Button	Image Button component	ImageButton
CheckBox	CheckBox component	CheckBox
Radio Button	Radio Button component	RadioButton
Choice	Choice component	Choice

COMPONENT	REGION	CSS TYPE SELECTOR
List	List component	List
Date Combo	Date Combo component	DateCombo
Separator	Separator component	Separator
Bullet	Bullet component	Bullet
TOC	TOC component	TOC
Heading	Heading component	Heading

Specifying the Report CSS File

To specify the CSS file for a report, follow the steps below:

1. Using a text editor, create a CSS file containing the desired class, ID, and type definitions. See *Creating a Component CSS Definition File* in *Advanced Server Administration* for information on how to specify these properties in the CSS file.

2. Save the CSS file (e.g., *myStyle.css*).

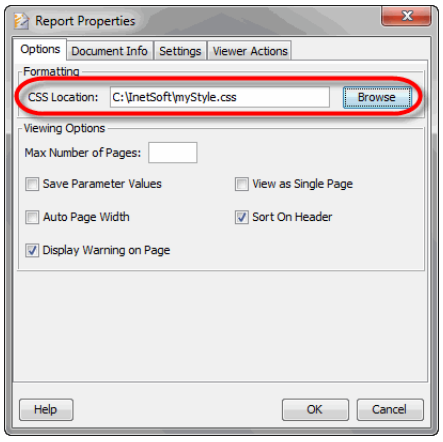
See *Report Properties* for more information.

3. Select 'Report Properties' from the Style Studio 'File' menu. This opens the 'Properties' dialog box.

4. Select the **Options** tab.

See *Configuring the Style Studio Classpath* for information about classpath directories.

5. In the 'CSS Location' field, enter a resource or absolute path for the CSS file. For example, if the file '*myStyle.css*' resides in a directory on the classpath, simply enter '`/myStyle.css`' as the 'CSS Location'.



6. Press 'OK' to close the dialog box.

Any CSS formatting defined by type (e.g., `Chart`, `RadioButton`, etc.) is now applied to the report, and no further action is required. To apply CSS formatting defined by ID class, see *Assigning CSS Style based on Class*

and ID. Note that CSS formats are overridden by any formats that you apply manually from within Style Studio.

If a CSS file is not specified for a report, the report will default to the CSS file specified by the '`css.location`' property in the `.stylereport` file. Note that **report beans** use the CSS file specified in the 'CSS Location' property for the bean definition. However, if none is specified, the bean uses the report CSS file.

Example: Applying CSS Styles

In this example, you will apply CSS styles to a report.

1. Using a text editor, create a CSS file. For this example, copy the CSS definitions from *Example: Creating a CSS Definition File* in *Advanced Server Administration*, and paste the definitions into a text editor.

2. Save the CSS file as *myStyle.css*.

See *Report Properties* for more information.

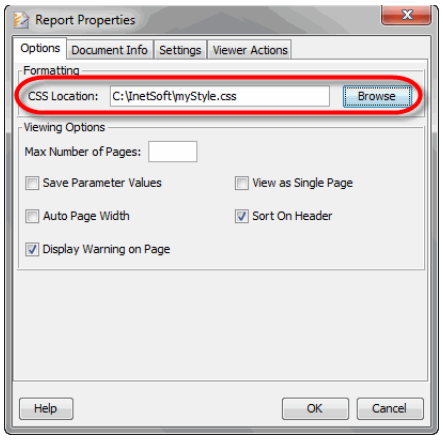
3. Create a new blank report.

4. Select 'Report Properties' from the Style Studio 'File' menu. This opens the 'Properties' dialog box.

5. Select the **Options** tab.

See *Configuring the Style Studio Classpath* for information about classpath directories.

6. Press the 'Browse' button next to the 'CSS Location' field, and select the *myStyle.css* file.



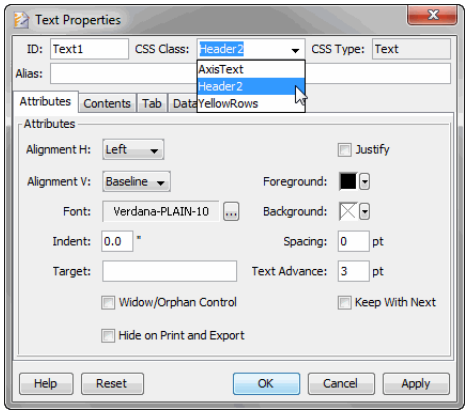
7. Press 'OK' to close the dialog box.

8. Add a Text element to the report. Enter the text 'Total Sales'.

Note that the text you enter uses the font “Verdana”. This font is used as the default for all text in the report because it is set as the global default in the CSS file. See *Example: Creating a CSS Definition File* in *Advanced Server Administration* for more information about this CSS file.

9. Right-click the Text component, and select ‘Properties’ from the context menu to open the ‘Text Properties’ dialog box.

10. From the ‘CSS Class’ menu, select ‘Header2’, and press ‘OK’.



This applies the Header2 style to the text element.



11. Add a new Chart component. In the Chart Editor, expand the following nodes: ‘Global Worksheet’ > ‘Tutorial’ > ‘Product Sales’. Drag the ‘Month(Date)’ field to the ‘X’ region, and drag the ‘Total’ field to the ‘Y’ region.

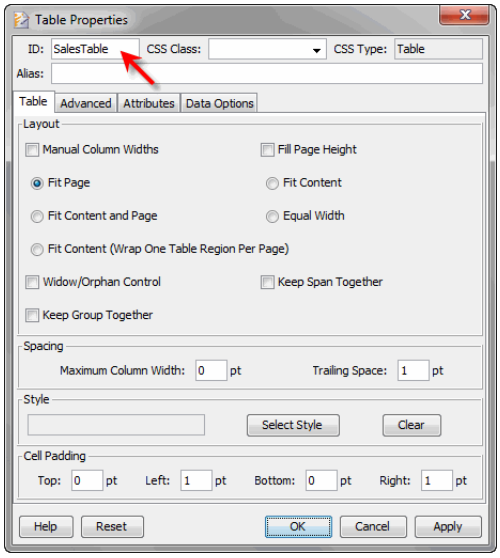
    Note that the chart background has color “azure” because the CSS file specifies this color for all components of type “Chart”. The Y-axis title has background color “yellow” because the CSS file specifies this color for all components of type “ChartAxisTitle”. The X-axis title has background color “dark khaki” because the CSS file specifies this color for all components of type “ChartAxisTitle[axis=x]”. The font used for all axis text is “Verdana” because the CSS file specifies this font as the global default. See *Example: Creating a CSS Definition File* in *Advanced Server Administration* for more information about this CSS file.

12. Press the ‘Select Chart Style’ button in the **Data** panel of the Chart Editor, and select the bar-chart option. This changes the display to a bar chart.

13. From the **Data Source** panel, drag the entire ‘SalesByDate’ data block onto the report page below the chart. This creates a new table.

14. Right-click on the table and select ‘Properties’ from the context menu to open the ‘Table Properties’ dialog box.

15. In the ‘ID’ menu, enter ‘SalesTable’, and press ‘OK’.

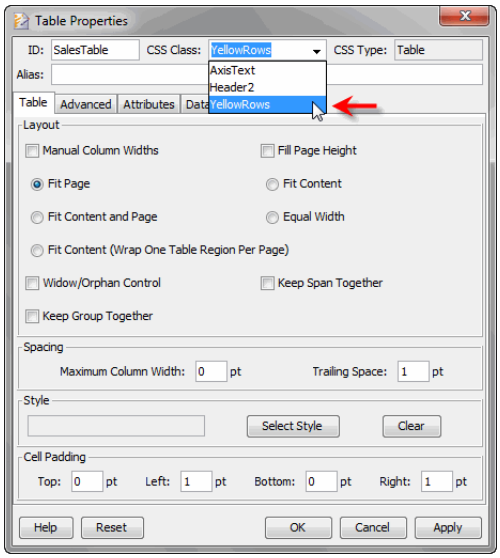


This applies the `SalesTable` style to the table as a whole, which applies background color #FFEEBB. Preview the report to see the table styling.

Company	Month(Date)	Quantity Purchased	State	Total
Folk Mining	2012-03-01 00:00:00	9	AZ	$6,735
Quick Stop	2012-03-01 00:00:00	48	CO	$10,526
Loctite Corp.	2012-03-01 00:00:00	46	CT	$16,410
Specialty Retail	2012-03-01 00:00:00	73	IL	$19,895
XO Resellers	2012-03-01 00:00:00	83	MA	$38,840
FMG Consulting	2012-03-01 00:00:00	65	MD	$30,628
Software Specialists	2012-03-01 00:00:00	95	NJ	$33,431
Old World Insurance	2012-03-01 00:00:00	108	NY	$59,155
Big Ed's BBQ	2012-03-01 00:00:00	14	TX	$4,694
Nova Tera	2012-03-01 00:00:00	17	TX	$1,211
Northwest Outfitters	2012-03-01 00:00:00	59	WA	$10,842
Folk Mining	2012-04-01	96	AZ	$44,777

16. Again right-click on the table and select ‘Properties’ from the context menu to open the ‘Table Properties’ dialog box.

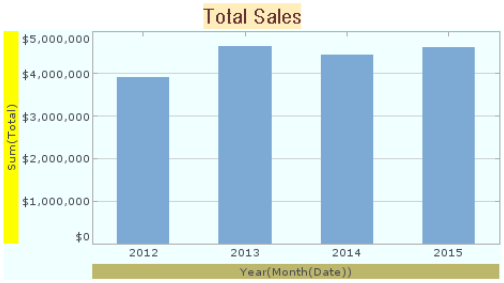
17. From the ‘CSS Class’ menu, select ‘YellowRows’ and press ‘OK’.



This applies the `YellowRows` class definition. Preview the report to view the table styling.

Company	Month(Date)	Quantity Purchased	State	Total
Folk Mining	2012-03-01 00:00:00	9	AZ	$6,735
Quick Stop	2012-03-01 00:00:00	48	CO	$10,526
Loctite Corp.	2012-03-01 00:00:00	46	CT	$16,410
Specialty Retail	2012-03-01 00:00:00	73	IL	$19,895
XO Resellers	2012-03-01 00:00:00	83	MA	$38,840
FMG Consulting	2012-03-01 00:00:00	65	MD	$30,628
Software Specialists	2012-03-01 00:00:00	95	NJ	$33,431
Old World Insurance	2012-03-01 00:00:00	108	NY	$59,155
Big Ed's BBQ	2012-03-01 00:00:00	14	TX	$4,694
Nova Tera	2012-03-01 00:00:00	17	TX	$1,211
Northwest Outfitters	2012-03-01 00:00:00	59	WA	$10,842
Folk Mining	2012-04-01	96	AZ	$44,777

You can manually make additional modifications to the report components, as desired. Format settings that you adjust manually will override format settings controlled by CSS.



Company	Month(Date)	Quantity Purchased	State	Total
Folk Mining	2012-03-01 00:00:00	9	AZ	$6,735
Quick Stop	2012-03-01 00:00:00	48	CO	$10,526
Loctite Corp.	2012-03-01 00:00:00	46	CT	$16,410
Specialty Retail	2012-03-01 00:00:00	73	IL	$19,895
XO Resellers	2012-03-01 00:00:00	83	MA	$38,840
FMG Consulting	2012-03-01 00:00:00	65	MD	$30,628
Software Specialists	2012-03-01 00:00:00	95	NJ	$33,431
Old World Insurance	2012-03-01 00:00:00	108	NY	$59,155
Big Ed's BBQ	2012-03-01 00:00:00	14	TX	$4,694
Nova Tera	2012-03-01 00:00:00	17	TX	$1,211
Northwest Outfitters	2012-03-01 00:00:00	59	WA	$10,842
Folk Mining	2012-04-01	96	AZ	$44,777

¢

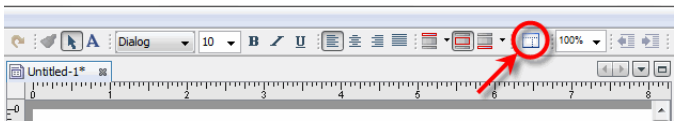
7.11 Page Areas

Page Areas are used in Flow Layout templates. (See *Flow Report Layout* for more information.) The following sections explain how to add and configure page areas.

7.11.1 Adding a New Page Area

To add a new page area to the flow report layout, follow the steps below:

1. Press the 'Layout View' button in the Style Studio toolbar.



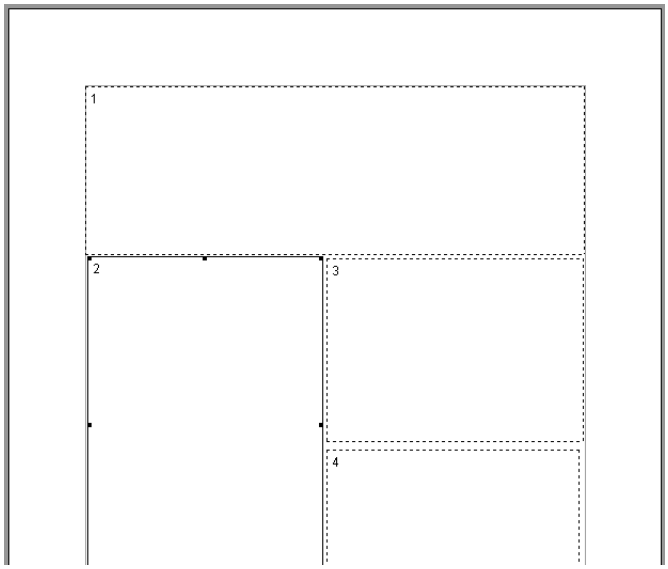
   This switches the report to layout mode.



2. Click the 'Draw Area' button. This switches the report to drawing mode.

3. Drag with the mouse to draw a page area with the desired position and size.

4. Repeat the previous step to add additional areas.



Each area displays a sequence number at the top-left corner. This determines the order in which contents will flow from one region into another.



5. To change the area flow order, click the ‘Order Area’ button. This changes the cursor is to a “hand” pointer.

6. Click all the areas, one after the other, in the order that you wish them to be used. After you have clicked on every area, Style Studio returns to its normal editing mode.

If report elements do not need to appear in a particular page area, you do not need to use area breaks. However, if you require that certain elements remain in a particular area, you must use area breaks to make sure that the element stays in its intended area.

See Also
Creating a New Blank Report, for information on creating a new flow report.

7.11.2 Non-Flow Page Areas

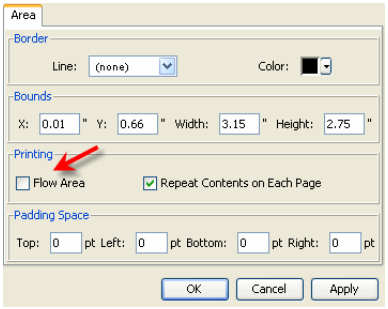
Another special type of page area is the non-flow page area. Unlike regular page areas, non-flow page areas do not participate in the normal flow of the element. The elements in the main flow are never placed in a non-flow page area. The simplest usage for a non-flow area is to create a rectangular box on a page. If no element is attached to the non-flow area, the area appears as a rectangle with the selected border.



A non-flow area can be created in the *Layout View* like a regular page area. Use the ‘Draw Area’ button to create the area and after the area is created,

right-click on it to open the ‘Area Properties’ dialog box where it can be marked as a non-flow area by deselecting the ‘Flow Area’ attribute.

Figure 22. Area Properties Dialog Box



Fixed Position Elements

A more interesting application of the non-flow page area is the ability to place elements in these areas. Elements added to a non-flow page area do not belong to the main document flow. They are only printed in the non-flow area to which they belong. Another important difference is that all elements in a non-flow area have fixed positions and sizes.

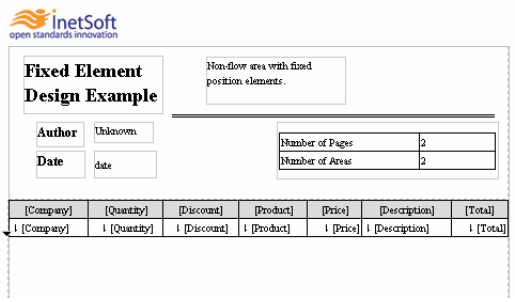
Elements in non-flow areas do not flow like regular elements. Instead, the position and size of the elements are fixed at the time they are created. The position and size of the elements do not change regardless of the size of other elements. If the contents of an element are larger than the size assigned to it, the element is only partially printed.

Adding elements to a non-flow area is similar to adding elements to the main document. The same element buttons are used. To add elements to a non-flow area, first select the area on the page design view. When a non-flow area is selected, a thick border is placed around the area.

Elements can be added to the non-flow area by clicking on one of the element buttons. When an element button is clicked, Style Studio enters element insertion mode, where an element can be positioned in the area by clicking the mouse at the starting position and dragging the mouse to set the size.

This procedure is the same for all elements. If an element does not take up the entire space assigned to it, it is formatted using the same algorithm as when it is the only element on a report with the specified printable area. For example, a text element is wrapped inside the area and flows down until it reaches the end of the area assigned to it.

Figure 23. Report with Elements in a Non-flow area (layout4.srt)



Advantages and Disadvantages of Fixed Position Elements

Fixed position elements are useful in form design and in other situations where the positions of the elements are fixed, such as mailing labels, pre-printed forms, etc. While the ability to easily position an element at any location on a page seems very attractive, there are some disadvantages of using fixed elements. First, if an element outgrows the space assigned to it, its size will not change dynamically, and this could cause truncation. Second, since the positions of the elements are fixed at design time, they cannot be adjusted if the size of the elements expands. If care is not taken, fixed position elements could potentially overlap other elements that flow on the same area.

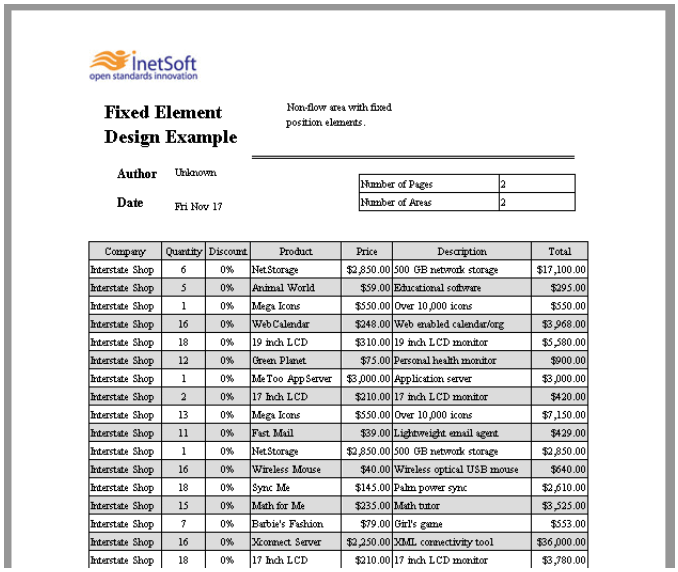
Walkthrough: Fixed Position Elements

Walkthrough

In the next example, you will create a template with a non-flow area on the top. It is used to hold the title and fixed information about the report. The second half of the report is assigned to a flow area, where the main table is placed. A page layout (page areas) is also associated with the table, so when the table flows to the next page, it starts at the top of the page instead of from the same position as the first page.

The generated report contains a fixed portion on the top of second page, and the rest of the document flows as usual.

Figure 24. Report using Non-flow area and Fixed Elements (layout4.srt)



InetSoft
open standards innovation

Fixed Element Design Example

Non-flow area with fixed position elements.

Author Unknown

Date Fri Nov 17

Number of Pages	2
Number of Areas	2

Company	Quantity	Discount	Product	Price	Description	Total
Interstate Shop	6	0%	NetStorage	$2,850.00	500 GB network storage	$17,100.00
Interstate Shop	5	0%	Animal World	$59.00	Educational software	$295.00
Interstate Shop	1	0%	Mega Icons	$550.00	Over 10,000 icons	$550.00
Interstate Shop	16	0%	WebCalendar	$248.00	Web enabled calendar/org	$3,968.00
Interstate Shop	18	0%	19 inch LCD	$310.00	19 inch LCD monitor	$5,580.00
Interstate Shop	12	0%	Green Planet	$75.00	Personal health monitor	$900.00
Interstate Shop	1	0%	MeToo AppServer	$3,000.00	Application server	$3,000.00
Interstate Shop	2	0%	17 Inch LCD	$210.00	17 inch LCD monitor	$420.00
Interstate Shop	13	0%	Mega Icons	$550.00	Over 10,000 icons	$7,150.00
Interstate Shop	11	0%	Fast Mail	$39.00	Lightweight email agent	$429.00
Interstate Shop	1	0%	NetStorage	$2,850.00	500 GB network storage	$2,850.00
Interstate Shop	16	0%	Wireless Mouse	$40.00	Wireless optical USB mouse	$640.00
Interstate Shop	18	0%	Sync Me	$145.00	Palm power sync	$2,610.00
Interstate Shop	15	0%	Math for Me	$235.00	Math tutor	$3,525.00
Interstate Shop	7	0%	Barbie's Fashion	$79.00	Girl's game	$553.00
Interstate Shop	16	0%	Xconnect Server	$2,250.00	XML connectivity tool	$36,000.00
Interstate Shop	18	0%	17 Inch LCD	$210.00	17 inch LCD monitor	$3,780.00

Parallel Report Flows

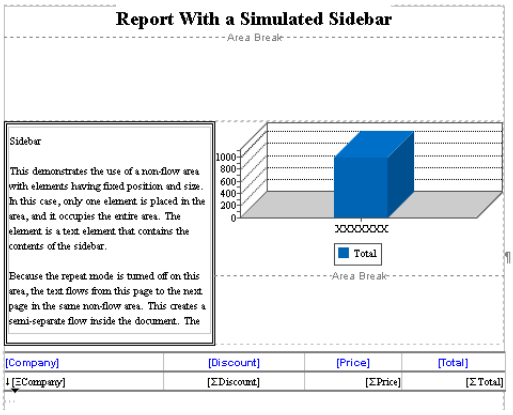
When the contents of a fixed element do not fit in the space assigned to it, there are two options. By default, the element is partially printed in the space, and any leftover contents are discarded. Alternatively, the element can continue onto the next page, in the same non-flow area, if the non-flow area is repeated. This behavior is controlled by the 'Repeat Contents on Each Page' attribute of the non-flow area. This is enabled by default.

The non-repeat mode of a non-flow area has an interesting property. The element by itself, 'flows' across multiple pages, in the same fixed position. While this is not a true flow since only one element is involved, it does allow contents of one element to continue across pages. This is a useful feature for creating a sidebar on a document, where the sidebar flows inside the sidebar from one page to the next, apart from the main report elements.

Walkthrough: Parallel Report Flows

In this example, you created a sidebar on the report page. Three regular page areas cover the leftover space.

Figure 25. Report With a Simulated Sidebar (flow1.srt)



A border is added to the non-flow area by using the 'Area Properties' dialog box in the *Layout View*. Mark the area as a non-flow and non-repeat area by deselecting 'Flow Area' and 'Repeat Contents on Each Page' so the text inside the area is continued onto the next page.

Figure 26. Area Properties

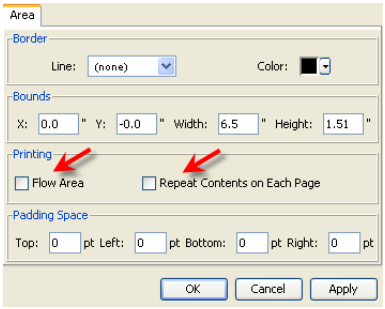
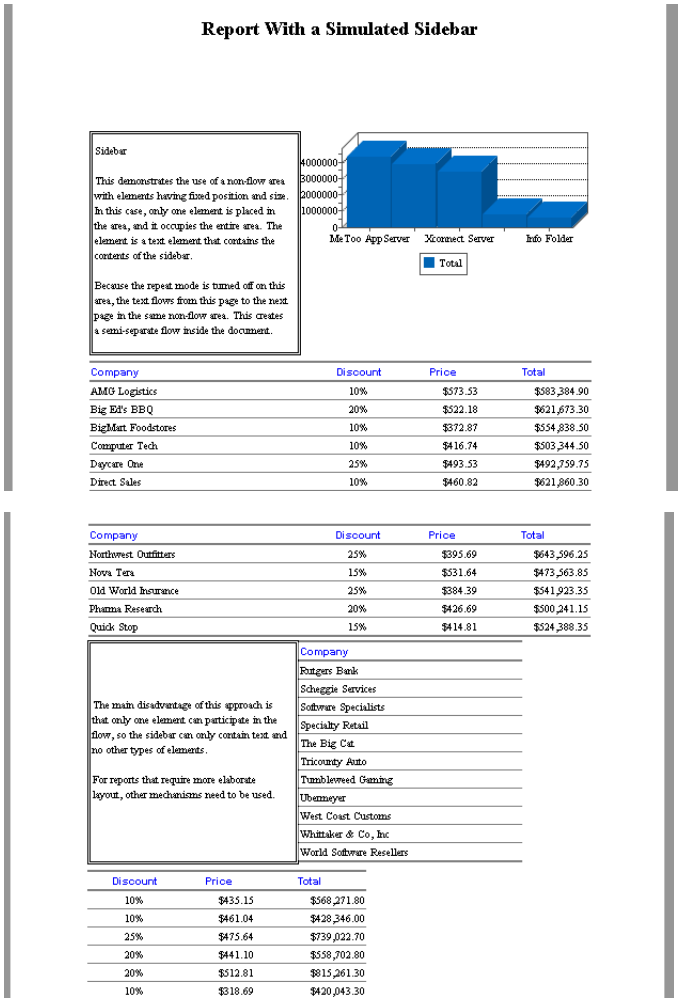


Figure 27. Report with a Simulated Sidebar



7.12 Formula Tables

Previous sections have presented a variety of techniques for displaying data within a Table element, including the following:

- *Creating a Table Using the Data Binding Wizard*
- *Creating a Crosstab Using the Data Binding Wizard*
- *Creating a New Table Using Freehand Operations*
- *Creating a Crosstab Using Freehand Operations*

However, these methods may not be flexible enough to create very specific layouts, display hybrid data from multiple queries, or address other advanced display needs. For such cases, you can gain greater flexibility by using a *Formula Table*.

The easiest way to use a Formula Table is to follow the steps below. See *Walkthrough: Creating a Formula Table* for an example.

1. Create a regular Table or Crosstab.

2. Modify the Table or Crosstab until it is as close as possible to the final design you desire.

3. Convert the Table or Crosstab to a Formula Table. (Right click the table, and select 'Table' > 'Convert to Formula Table' from the context menu.

4. Add additional rows, columns, and fields and set expansions, references, and formulas, as described in the following sections, to further enhance the table.

Note: To use advanced Formula Tables effectively, you should possess a working knowledge of JavaScript. See *General JavaScript Functions* for an introduction.

5. (Optional) If required, use advanced formula table script to create the final table. Advanced features are covered in the following sections: See *Creating a Formula Table Using Script (Advanced)* for more details.

Formula tables bridge the gap between spreadsheet applications (like Excel) and traditional reporting tools. In a spreadsheet, you usually start with columns of data and then define formulas that reference other cells. To copy these formulas with appropriate references you can 'fill-down' in a range of cells.

A formula table is similar. You can extract parts of a query result set and then dynamically fill the rows and columns of the table. You can then reference these cells to perform statistical calculations in formulas. The 'fill-down' operation happens automatically when the table expands to show actual data.

You can therefore use formula tables to create real-time, spreadsheet-like reports with very specific and complex layouts, grouping of data, and aggregation which are commonly required in accounting and financial applications.

See Also

Binding Data to a Table Element, to display tabular data from a single result set.

Creating a Crosstab Using the Data Binding Wizard, to display summarized data in compact form.

Creating a Table Using Freehand Operations, to flexibly position data from a single result set.

7.12.1 Walkthrough: Creating a Formula Table

The following walkthrough illustrates the ability of a formula table to satisfy complex data layout needs.

Walkthrough

In this example, you will create a crosstab that breaks down sales totals both according to individual 'State' and 'Category' combinations and simultaneously according to 'Order Date' and 'Category' combinations.

If you were to use a regular Crosstab for this purpose, the data will be represented with a nested header structure, such as that below. Note that the aggregate ('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	$63,250	$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, you will begin by creating a regular Crosstab that contains just the 'State' and 'Category' combinations. You will then convert this table to a Formula Table, and use basic Formula Table techniques to add the 'Date' headers.

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 'Report' node. In the right 'Types' panel, select 'Blank Tabular Report' and click 'OK' This creates a new report.

3. Select 'Page Setup' in the Style Studio 'File' menu. Make the following changes:

a. Choose the ‘Landscape’ page option in the ‘Orientation’ panel

b. In the ‘Margin’ panel, set the ‘Right’ and ‘Left’ margins to 0.5 inches.

c. Press ‘OK’.

4. Expand the Style Studio **Toolbox** panel. (Select ‘Toolbox Pane’ from the Style Studio ‘Window’ menu.)



5. Click the ‘Crosstab’ component in the **Toolbox** panel. This opens the Data Binding Wizard for a Crosstab.

6. Select the **Data** tab. Expand the ‘Orders’ data source, and select the ‘Order Model’ data model.

7. Select the **Crosstab** tab. Bind the headers and summary fields in the following way:

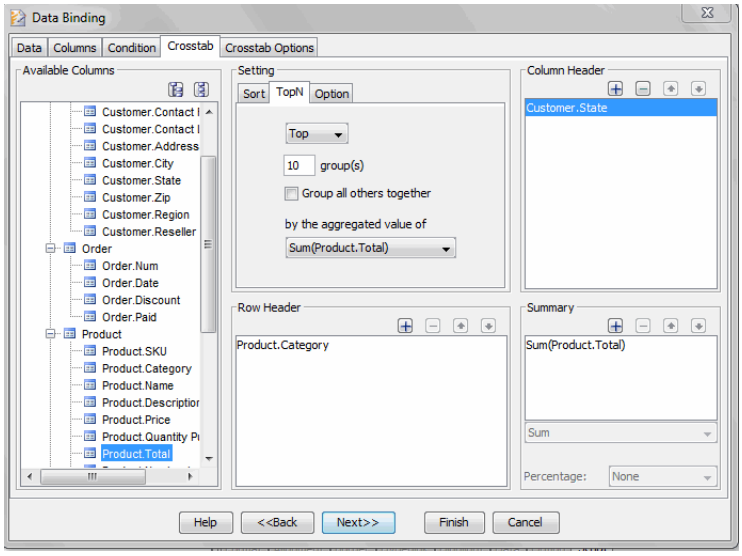
a. Drag the ‘Product.Category’ field from the ‘Available Columns’ panel to the ‘Row Header’ panel.

b. Drag the ‘Customer.State’ field from the ‘Available Columns’ panel to the ‘Column Header’ panel.

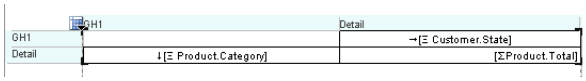
c. Drag the ‘Product.Total’ field from the ‘Available Columns’ panel to the ‘Summary’ panel

d. Select the Customer.State’ field in the ‘Column Header’ panel, and click the **TopN** tab in the ‘Setting’ panel.

e. Select ‘Top’ in the menu, and enter ‘10’ as the number of groups. This will return the top 10 groups according to the summed total.

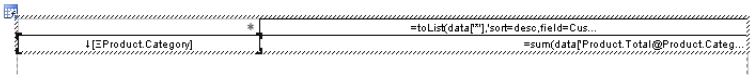


f. Click ‘Finish’ to close the Data Binding Wizard. This creates a new Crosstab in the report.



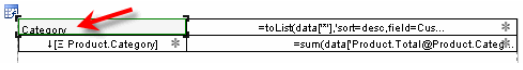
8. Expand the Style Studio **Properties** panel. (Select ‘Properties Pane’ from the Style Studio ‘Window’ menu.)

9. Right-click the Table, and select ‘Table’ > ‘Convert to Formula Table’ from the context menu.



Observe that some of the cells have now changed to display a formula. To edit the formula for a cell, select the cell and view the **Data** tab of the **Properties** panel. You will make a simple formula modification later in this example. For more complete information on editing formulas, see *Formula Tables*.

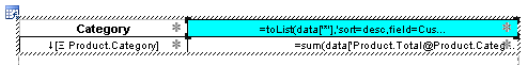
10. Click the top-left corner cell of the table to select it. Type the text “Category” into the cell.



(Optional) Use the Style Studio toolbar buttons to make the text large and bold.

11. Click the top-right corner cell of the table to select it. Select the **Format** tab in the **Properties** panel.

12. In the 'Attribute' panel, select a blue color for the 'Background' property.



13. Right-click the Table, and select 'Properties' from the context menu. This opens the 'Table Properties' dialog box.

14. In the 'Table Properties' dialog box, select the **Table** tab. In the 'Layout' panel, select 'Fit Content' and click 'OK'.

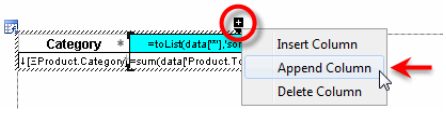
15. Click the 'Preview' button in the Style Studio toolbar, and view the resulting table.

Category	NJ	TX	NY	MA	CA	MD	WA	CO	IL	AZ
Business	1462625	1054125	1149625	957625	739125	590875	473625	318125	301750	272125
Educational	126055	77128	88560	71215	66231	50187	33104	41581	17119	16698
Games	98186	77087	104723	96547	92760	59051	38110	32052	26482	39463
Graphics	178675	89081	162487	110378	74444	62644	25087	48443	46559	24875
Hardware	1055030	891830	678730	610960	430800	314450	225140	319550	82320	82870
Office Tools	212760	153958	144355	120292	139349	133333	55071	37316	26767	16228
Personal	180642	128738	139506	168501	143939	95168	76426	46731	58504	15839

16. Close the preview and return to the design view.

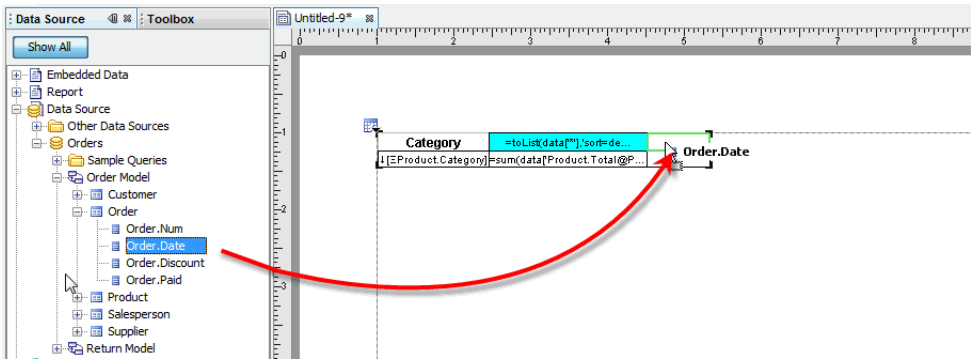
17. You will now use formula table operations to add the 'Date' headers to the table. Follow the steps below:

    a. Click above the top-right corner cell and press the '+' button. Select 'Append Column' from the context menu to add a new column.

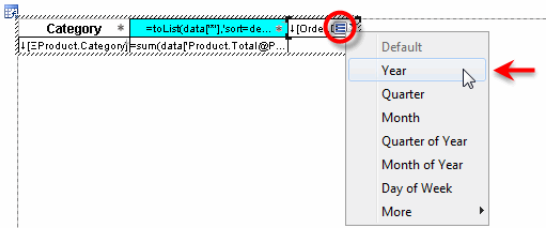


    b. In the Style Studio **Data Source** panel, expand the 'Orders' data source and the 'Order Model' data model.

    c. Expand the 'Orders' node, and drag the 'Date' attribute to the new top-right corner cell of the table. This adds the 'Order.Date' attribute as a column header.



d. Press the ‘Group’ button in this new ‘Order.Date’ cell, and select ‘Year’ from the context menu. This will group the values in the ‘Date’ field by unique year, resulting in a set of year-based headers.



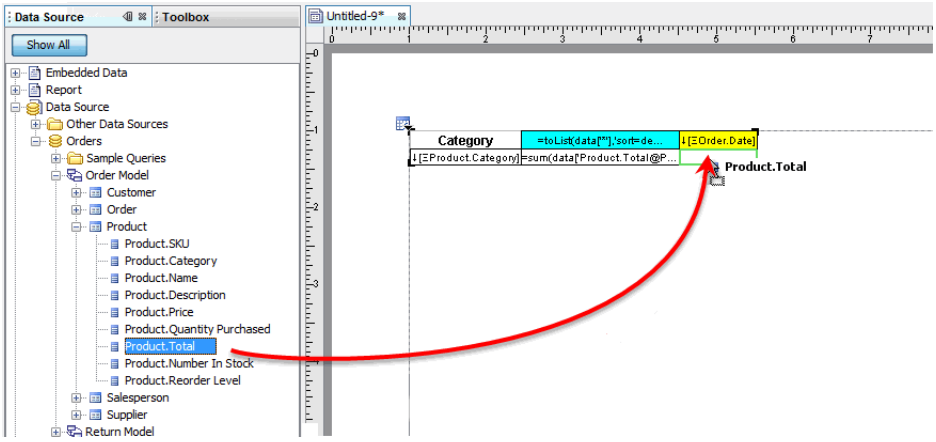
e. Select the **Data** tab in the Style Studio **Properties** panel. In the ‘Expansion’ panel, select the ‘Horizontal’ option. This sets the year groups in the ‘Order.Date’ field to display horizontally, as multiple columns headers.

f. Select the **Format** tab in the Style Studio **Properties** panel. In the ‘Attribute’ panel, set a yellow ‘Background’ color for the ‘Order.Date’.

18. You will now use formula table operations to add an aggregate ‘Total’ measure corresponding to the year groups. Follow the steps below:

a. In the Style Studio **Data Source** panel, expand the ‘Orders’ data source and the ‘Order Model’ data model.

b. Expand the ‘Product’ node, and drag the ‘Total’ attribute to the bottom-right corner cell of the table. This adds the ‘Product.Total’ attribute as a measure.



c. Press the ‘Aggregate’ button on this new ‘Product.Total’ cell, and select ‘Sum’ from the context menu. This will aggregate the totals for each combination of ‘Category’ and ‘Date’.



19. Press the ‘Preview’ button in the Style Studio toolbar to preview the table.

Category	NJ	TX	NY	MA	CA	MD	WA	CO	IL	AZ	2008	2009	2010	2011
Business	1462625	1054125	1149625	957625	739125	690875	473625	318125	301750	272125	1961500	2071250	1894375	2166375
Educational	126055	77128	88560	71215	66231	50187	33104	41581	17119	16698	134998	183837	131248	175417
Games	98186	77087	104723	96547	92760	59051	38110	32052	26482	39463	172330	185884	189400	201599
Graphics	178675	89081	162487	110378	74444	62644	25087	48443	46559	24875	195529	316686	199193	200287
Hardware	1055030	891830	678730	610960	430800	314450	225140	319550	82320	82870	995040	1289310	1456350	1231610
Office Tools	212760	153958	144355	120292	139349	133333	55071	37316	26767	16228	220541	320372	251236	334689
Personal	180642	128738	139506	168501	143939	95168	76426	46731	58504	15839	247885	281887	330693	317333

You can make further modifications to the table by editing the underlying formulas. In the design view, simply select the cell that you want to edit, and open the **Data** tab of the Style Studio **Properties** panel. In the ‘Binding’ panel, edit the existing cell formula as desired. For more details on formula syntax see *Creating a Formula Table Using Script (Advanced)* and *Formula Tables* in *Advanced Report Scripting*.

For example, observe that the ‘Category’ row headers are currently sorted alphabetically (e.g., Business, Educational, Games, etc.). If you wish to instead sort them in *reverse* alphabetical order, you can make a small change to the existing formula. Follow the steps below.

20. Click the bottom-left cell of the table to select it.

21. Select the **Data** tab in the Style Studio **Properties** panel

22. In the 'Binding' panel, change the 'Formula' for the cell from

    ```
    toList(data['Product.Category'],'sort=asc')
    ```

    to

    ```
    toList(data['Product.Category'],'sort=desc')
    ```

23. Preview the report again, and note the change in the order.

If you wish to instead sort the categories by the value of the corresponding aggregate total, you can do this by making another change to the formula. First, observe that the 'State' row headers are ordered by the aggregate value. You can therefore modify and reuse the 'State' header formula to achieve the same effect for the 'Category' headers.

Follow the steps below.

24. Click on the top-middle ('State' header) cell, and select the **Data** tab in the Style Studio **Properties** panel.

25. In the 'Binding' panel, copy the text from the 'Formula' field. The formula should be as follows:

    ```
    toList(data['*'],'sort=desc,field=Customer.State,sorton=sum
    (Product.Total),maxrows=10')
    ```

26. Click on the bottom-left ('Category' header) cell, and again select the **Data** tab in the Style Studio **Properties** panel.

27. In the 'Binding' panel, select the 'Formula' option, and paste the copied formula into the text field.

28. Modify the formula as follows:

    ```
    toList(data['*'],'field=Product.Category,sort=desc,sorton=s
    um(Product.Total)')
    ```

    This orders the categories according to the summed total, from greatest to least (descending order).

29. Click the green 'Apply' button to save the formula.

30. Preview the report again. Observe that the categories are now ordered according to the totals (e.g., Business, Hardware, Personal, etc.).

Category	NJ	TX	NY	MA	CA	MD	WA	CO	IL	AZ	2008	2009	2010	2011
Business	1462625	1054125	1149625	957625	739125	690875	473625	318125	301750	272125	1961500	2071250	1894375	2166375
Hardware	1055030	891830	678730	610960	430800	314450	225140	219550	82320	82870	995040	1289310	1456350	1231610
Personal	180642	128738	139506	168501	143939	95168	76426	46731	58504	15839	247885	281887	330693	317333
Office Tools	212760	153958	144355	120292	139349	133333	55071	37316	26767	16228	220541	320372	251236	334689
Graphics	178675	89081	162487	110378	74444	62644	25087	48443	46559	24875	195529	316686	199193	200287
Games	98186	77087	104723	96547	92760	59051	38110	32052	26482	39463	172330	185884	189400	201599
Educational	126055	77128	88560	71215	66231	50187	33104	41581	17119	16698	134998	183837	131248	175417

7.12.2 Combining Multiple Data Sources in a Formula Table

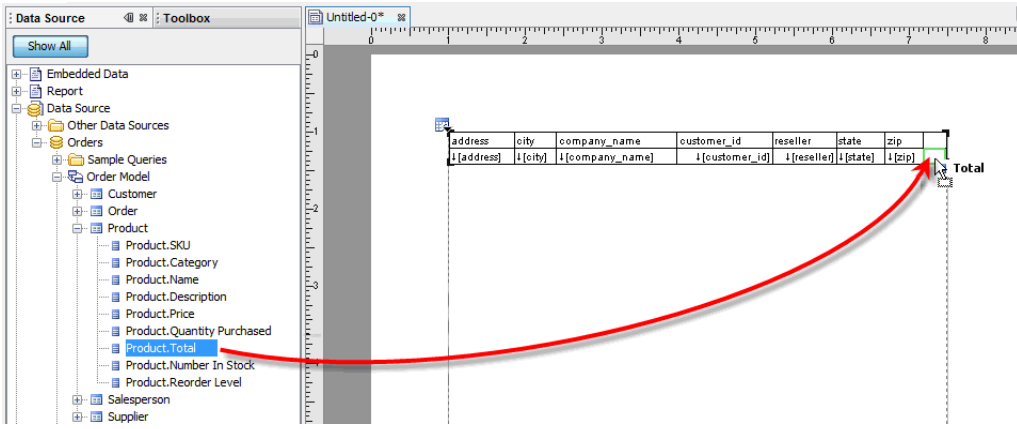
You can use a formula table to represent data from more than one data source. To bind more that one data source in a formula table, follow the steps below:

1. Create a formula table with a binding to a single data source, as you usually would. (The easiest way to do this is to convert a regular table. See *Walkthrough: Creating a Formula Table* for an example.)

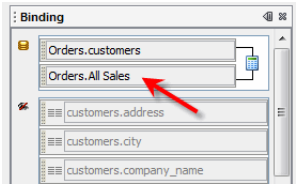
2. Click on the formula table to select it. In the **Data Source** panel, press the 'Show All' button. This displays all available data sources.

3. Bind the second data source to the formula table. You can do this in one of two ways:

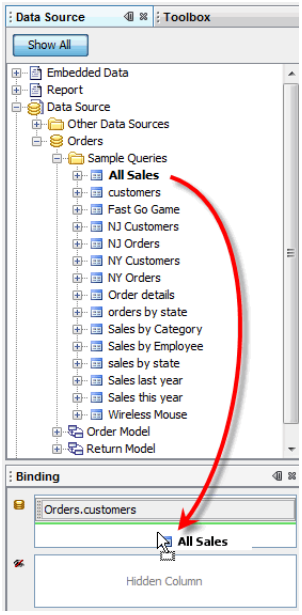
   a. Insert a new column or columns into the table (see *Inserting Rows and Columns into a Table*), and then drag a field from the desired second data source in the **Data Source** panel directly into the formula table.



   This adds the second data source to the **Binding** panel under the original data source, and makes all of the remaining fields in the second data source available for binding within the table.

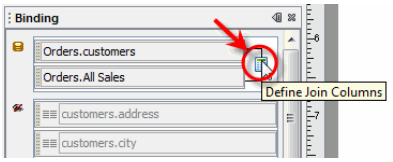


b. Alternatively, drag the desired second data source from the **Data Source** panel into the **Binding** panel below the first data source. This will make the fields from the second data source available for binding within the formula table.



4. Repeat the above steps to bind additional data sources to the Table.

5. Press the 'Define Join Columns' button in the **Binding** panel.

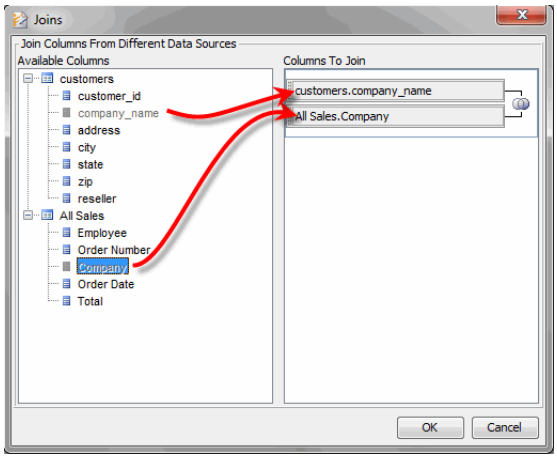


This opens the 'Joins' dialog box. The 'Joins' dialog box allows you to specify the relationships between the multiple data sources so that corresponding rows from each source can be appropriately aligned in the table. To do this, in the following steps you will select one or more pairs of columns ("join columns") that will be used to align rows from the different data sources.

6. In the ‘Joins’ dialog box, drag a desired join column from the first data source in the ‘Available Columns’ tree to the ‘Columns To Join’ region.

Look for the green highlight immediately below the existing column.

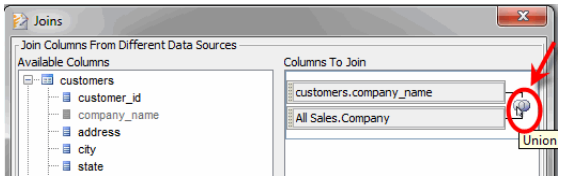
7. Drag a corresponding join column from the second data source in the ‘Available Columns’ tree to the ‘Columns To Join’ region. Drop the column immediately below the first join column.



8. (Optional) Repeat the above steps to add join columns from any additional available data sources to this join association.

   By joining different data sources on a particular set of “join columns,” you specify that the records displayed in the formula table will be matched based on the values in these columns. For example, if the table contains columns ‘company_name’, ‘address’, and ‘city’ from query A, and contains columns ‘Company’, ‘Employee’, and ‘Total’ from query B, by joining the ‘company_name’ field to the ‘Company’ field, you ensure that records corresponding to the same company (from the two queries) are printed on the same table row.

9. Press the ‘Union’/’Intersection’ button to select the desired type of join.





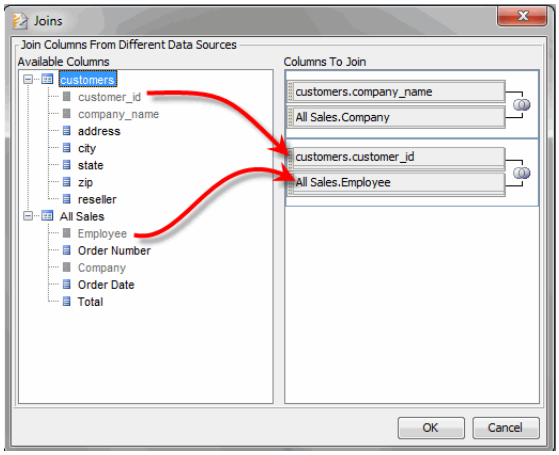
The ‘Union’ option provides the equivalent of a *full outer join*. When this option is selected, *all* values that appear in a join column will be represented in the table, even if those values have no match

in the other join columns. For example, if the 'Company' field from query B contains the value "InetSoft" but the 'company_name' field from query A does not contain this value, the 'Union' option will still display the "InetSoft" rows from query B in the table. This means that the corresponding rows from query A will appear empty in the table, because query A contains no data corresponding to the value "InetSoft".



The 'Intersect' option provides the equivalent of an *inner join*. When this option is selected, *only* values that have a match in all join columns will be represented in the Table. For example, if the 'Company' field from query B contains the value "InetSoft" but the 'company_name' field from query A does not contain this value, the 'Intersect' option will *not* display the "InetSoft" rows from query B in the table. The 'Intersect' option therefore produces a table with no empty records.

10. (Optional) To specify another set of join columns, drag the first column of the new join set to an empty region within the 'Columns To Join' panel. Then proceed to add the additional columns and set the join types as described above.



By specifying multiple join sets, you introduce additional constraints on the records that the Table displays. For example, in the above illustration, the only complete rows displayed are those for which the value of 'company_name' matches the value of 'Company', and simultaneously the value of 'customer_id' matches the value of 'EmployeeID'. However, as mentioned above, the 'Union' option will generate partially empty rows where matching values do not exist in all join columns.

```
Join columns in the table must be grouped.
```

11. Assign grouping to the join column in the table. See *Grouping Data in a Table* for more details.

The formula table will now display data from the multiple data sources you specified in the **Binding** panel, and records from these various data sources will be coordinated according to the join relationships you specified in the 'Joins' dialog box. The example below provides additional guidance about formatting and display considerations.

Example: Multiple Data Sources

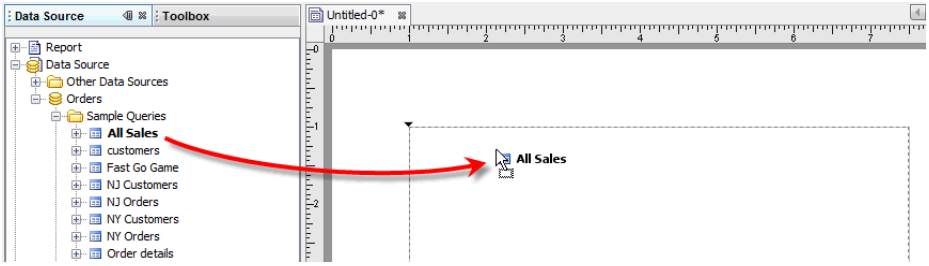
In this example, you will create a formula table that displays data drawn from two different queries, the 'All Sales' query and the 'customers' query. The 'customers' query will provide company addresses, and the 'All Sales' query will provide the sales details.

Customer:	AMG Logistics	283 Handley Rd.	Springfield	MA
	Employee	*Order Date*	*Order Number*	*Total*
	Eric	07/04/2009	12067	27098
	Eric	08/14/2009	12085	54117
	Eric	06/08/2010	12217	56614
	Eric	12/19/2010	12320	10602
	Eric	05/15/2011	12400	48287
	Eric	12/21/2011	12511	55785
	Eric	01/16/2012	12521	50310
	Eric	01/21/2012	12524	1984
	Eric	09/13/2012	12647	66991
	Eric	11/15/2012	12681	5267
	Eric	11/16/2012	12682	62631
	Eric	12/13/2012	12691	2733
	Sue	06/29/2009	12064	24139
	Sue	01/08/2010	12149	3160
	Sue	06/12/2010	12219	16934
	Sue	09/23/2010	12273	22390
	Sue	12/29/2010	12329	240
	Sue	08/06/2011	12440	47264
	Sue	06/21/2012	12605	29352
Customer:	Big Ed's BBQ	23 Tasaga Ave.	Houston	TX
	Employee	*Order Date*	*Order Number*	*Total*
	Annie	05/04/2010	12201	10915
	Annie	06/16/2010	12220	73409
	Annie	04/12/2011	12378	51000
	Annie	09/17/2011	12458	56573
	Annie	10/21/2011	12476	15793

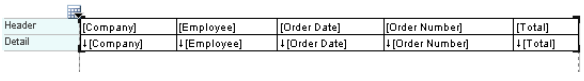
To build this formula table, follow the steps below:

1. Create a new blank report.

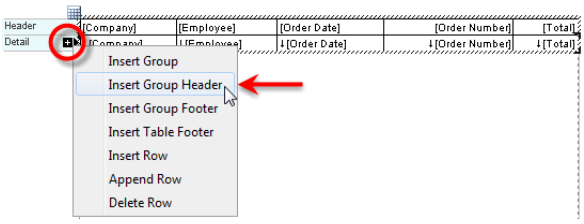
2. From the Style Studio **Data Source** panel (under 'Orders' > 'Sample Queries') drag the entire 'All Sales' query into the report.



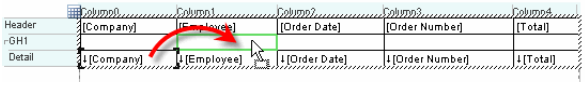
This creates a new table based on the 'All Sales' query.



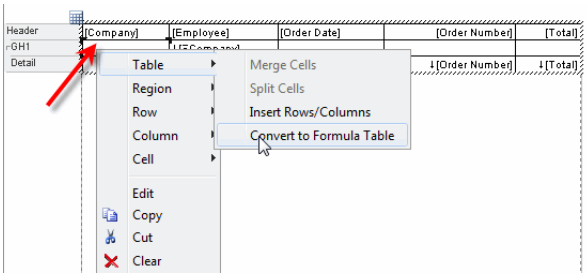
3. Click in the left margin next to the detail row, and then press the '+' icon. Select 'Insert Group Header' from the context menu. This creates a 'GH1' header row.



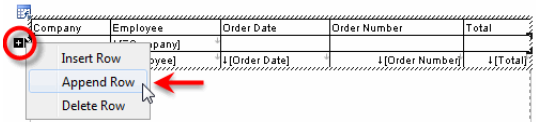
4. Drag the 'Company' field from the 'Detail' row to the cell in the 'Employee' column of the new 'GH1' row. This creates a new grouping based on 'Company'.



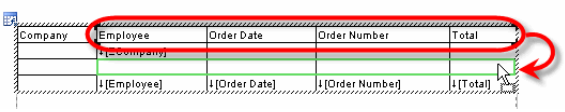
5. Click any cell in the table to select it. Right-click, and select 'Table' > 'Convert to Formula Table' from the context menu. This changes the table to a formula table, which will allow you to perform advanced operations such as binding multiple data sources.



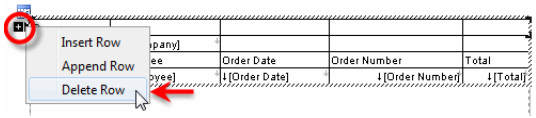
6. Click in the left margin next to the middle row, and press the '+' icon. Select 'Append Row' from the context menu. This adds a new row above the final row.



7. Shift-click to select all cells in the top row except for 'Company'. Drag these cells together into the new row that you created in the previous step.



8. Click in the left margin next to the top row, and press the '+' icon. Select 'Delete Row' from the context menu. This deletes the top row of the table.

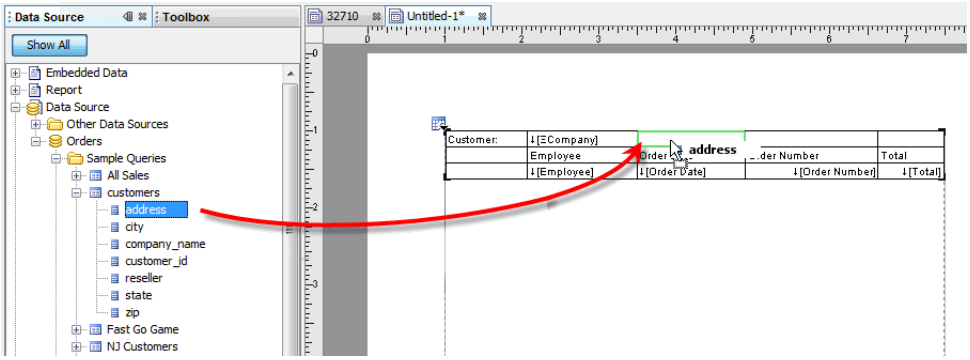


9. In the top-left corner cell, type the text "Customer:".

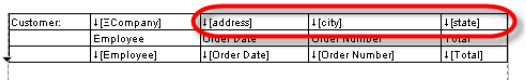


10. In the Style Studio **Data Source** panel, press the 'Show All' button to display all available data sources. Expand the 'customers' query (under 'Orders' > 'Sample Queries').

11. From the 'customers' query, drag the 'address' field into the table next to the 'Company' field.



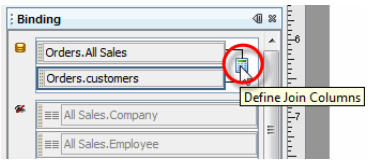
12. Repeat the above step to bind the 'city' and 'state' fields from the 'customers' query.



The table now contains fields from the two different queries. However, because the queries have not yet been related to one another ("joined"), the table cannot yet represent the data correctly.

In the next steps, you will specify the association between the two queries based on the company names.

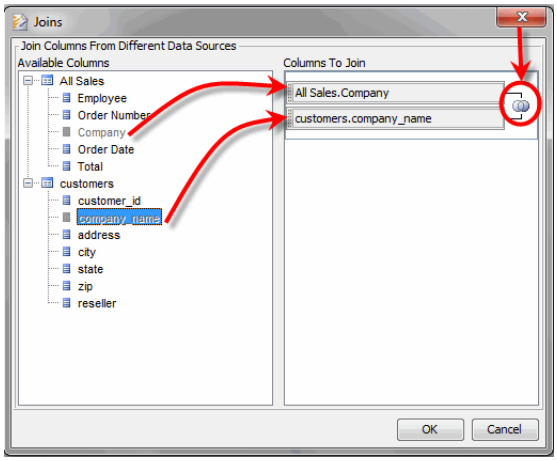
13. In the Style Studio **Binding** panel, press the 'Define Join Columns' button. This opens the 'Joins' dialog box.



14. In the 'Joins' dialog box, drag the 'Company' field from the 'Available Columns' list into the 'Columns To Join' list.

```
Join fields that are displayed in the table must have grouping
applied. (The 'Company' field was already specified as a grouping
field in an earlier step.)
```

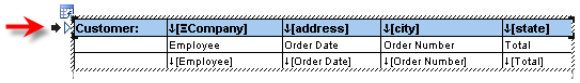
15. Drag the 'company_name' field from the 'Available Columns' list into the 'Columns To Join' list. Drop it immediately below the 'Company' field when you see the green highlight.



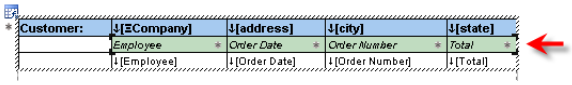
16. Press the 'Union/Intersect' button once to select the 'Intersect' option (see above). This ensures that only those companies that appear in *both* queries will be represented in the table.

17. Press 'OK' to exit the 'Joins' dialog box.

18. Click in the left margin next to the top row of the table. This selects the top row of the table. Use the Style Studio toolbar or **Properties** panel to apply any desired visual formatting to the top row, such as fonts or colors.



19. Select the non-empty cells in the middle row, and apply any desired additional desired formatting, such as background color or a data format (for example, a Date format for the 'Order Date' field).



20. Preview the report.

Customer:	AMG Logistics	283 Handley Rd.	Springfield	MA
	Employee	*Order Date*	*Order Number*	*Total*
	Eric	07/04/2009	12067	27098
	Eric	08/14/2009	12085	54117
	Eric	06/08/2010	12217	56614
	Eric	12/19/2010	12320	10602
	Eric	05/15/2011	12400	48287
	Eric	12/21/2011	12511	55785
	Eric	01/16/2012	12521	50310
	Eric	01/21/2012	12524	1984
	Eric	09/13/2012	12647	66991
	Eric	11/15/2012	12681	5267
	Eric	11/16/2012	12682	62631
	Eric	12/13/2012	12691	2733
	Sue	06/29/2009	12064	24139
	Sue	01/08/2010	12149	3160
	Sue	06/12/2010	12219	16934
	Sue	09/23/2010	12273	22390
	Sue	12/29/2010	12329	240
	Sue	08/06/2011	12440	47264
	Sue	06/21/2012	12605	29352
Customer:	Big Ed's BBQ	23 Tasaga Ave.	Houston	TX
	Employee	*Order Date*	*Order Number*	*Total*
	Annie	05/04/2010	12201	10915
	Annie	06/16/2010	12220	73409
	Annie	04/12/2011	12378	51000
	Annie	09/17/2011	12458	56573
	Annie	10/21/2011	12476	15793

The table now displays data drawn from the two different queries. Records from the two queries are correctly synchronized by the join which you defined on the respective company name fields.

¢

See Also

Advanced Formula Table Walkthrough, for example of binding multiple data sources using script (runQuery).

7.12.3 Creating a Formula Table Using Script (Advanced)

The fundamental approach to creating formula tables in script is as follows:

1. Fill out row and/or column headers. (*Cell Expansion*)
2. If there are multiple level row/column headers, set up an expansion hierarchy. (*Expansion Hierarchies*)

3. Define aggregation formulas by referencing the header cells or other summarized cells. (*Defining Formulas by Referencing Cells*).

Cell Expansion

If you assign a formula to a cell of a table, and this formula returns an array of values, you can specify that the cell should expand in the horizontal/ vertical direction to 'fill' the table with the values of the array.

Walkthrough

In this example, you will create a formula table, and 'fill' it with dynamically-generated row and column headers extracted from a query.

1. Click the 'New' button in the Style Studio toolbar. This opens the 'New Asset' dialog box.

2. In the left panel, select the 'Report' node. In the right 'Types' panel, select 'Blank Tabular Report' and click 'OK'

3. Expand the Style Studio **Toolbox** panel. (Select 'Toolbox Pane' from the Style Studio 'Window' menu.)

4. Drag the 'Table' component from the **Toolbox** panel to the report. This creates a new Table without a binding.



5. Right click the table and select 'Table' > 'Convert to Formula Table' from the context menu.

6. Right-click on a blank area of the report, and select 'Script' from the context menu. This opens the Script Editor.

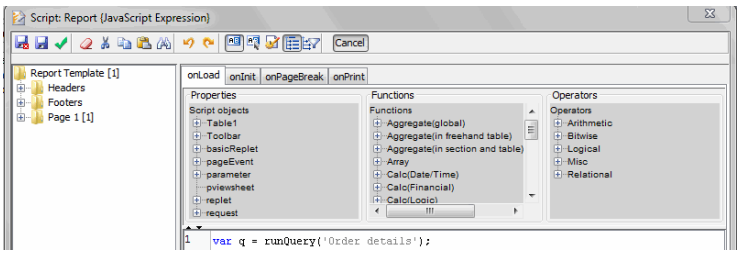
7. Add the following script in the **onLoad** tab.

```
var q = runQuery('Order details');
```

This script runs the 'Order details' query and stores the result set in a variable '`q`'.



8. Click the 'Save and Close' button.



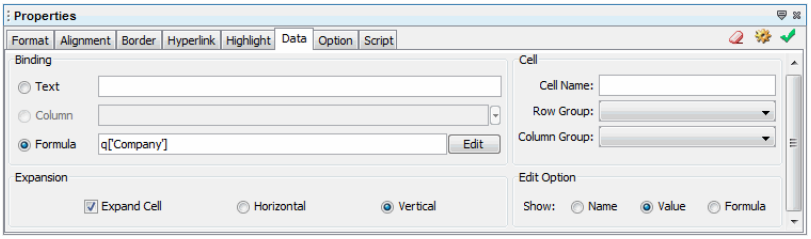
9. Expand the Style Studio **Properties** panel. (Select 'Properties Pane' from the Style Studio 'Window' menu.)

10. Select `cell[1,0]` (second row, first column) and select the **Data** tab in the **Properties** panel.

11. In the 'Binding' region of the **Data** tab, choose the 'Formula' option, and enter the following formula:

    ```
    q['Company']
    ```

12. Check the 'Expand Cell' box and select the 'Vertical' radio button. This sets the cell to expand (fill) vertically.



13. Click the 'Apply' button.

14. Preview the report and notice how the cell fills out all the values of the 'Company' column from the query result set. Notice how each Company name repeats multiple times.

Note: `q['Company']` will return an array of records under the 'Company' column of the query result set. For more information on how to extract and manipulate data from a query result set, see *Formula Tables* in *Advanced Report Scripting*.

See *toList* in *Advanced Report Scripting* for more information.

15. To retrieve a *distinct* set of companies, use the `toList()` function, which returns an array of unique values from a given input array. Change the previous formula to the following:

    ```
    toList(q['Company'])
    ```

16. Preview the report and notice the unique listing of companies.

17. Select `cell[0,1]` (first row, second column).

18. In the 'Binding' region of the **Data** tab, choose the 'Formula' option. Add the following formula:

    ```
    toList(q['Product'])
    ```

19. Check the 'Expand Cell' box and select the 'Horizontal' radio button. This sets this cell to expand (fill) horizontally.

20. Click the 'Apply' button.

21. Double-click to open the ‘Table Properties’ dialog box. Click the **Table** tab, and select the ‘Fit Content’ radio button. Click ‘OK’.

22. Preview the report, and notice the horizontal and vertical expanding rows and columns.

	17 Inch LCD	19 inch LCD	Animal World	Barbie's Fashion	Combat Hero	Easy Chess	Fancy Menus
AMG Logistics							
Big Ed's BBQ							
BigMart Foodstores							
Computer Tech							
Daycare One							
Direct Sales							
Eastern Data							
Ernst Handel							
Fast Transportation							
FISGA Corp							
FMG Consulting							
Folk Mining							
George Services							
High Society Investments							

Expansion Hierarchies

It is common to have multiple levels of row/column headers nested in a hierarchy, e.g., State and City. Formula tables accommodate this structure in the following ways:

- Two or more vertically expanding cells defined on the same row will automatically expand in hierarchy. The order of the hierarchy moves from left to right.
- Two or more horizontally expanding cells defined under the same column will automatically expand in hierarchy. The order of the hierarchy moves from top to bottom.

Walkthrough

In this example, you will create a formula table which uses data from the ‘customers’ query. The query is executed in the onLoad script and is stored in a variable called ‘q’:

1. Click the ‘New’ button in the Style Studio toolbar. This opens the ‘New Asset’ dialog box.

2. In the left panel, select the ‘Report’ node. In the right ‘Types’ panel, select ‘Blank Tabular Report’ and click ‘OK’

3. Expand the Style Studio **Toolbox** panel. (Select ‘Toolbox Pane’ from the Style Studio ‘Window’ menu.)

4. Drag the ‘Table’ component from the **Toolbox** panel to the report. This creates a new Table without a binding.



5. Right-click the table and select ‘Convert to Formula Table’ from the context menu.

6. Right-click on a blank area of the report, and select ‘Script’ from the context menu. This opens the Script Editor.

7. Add the following script in the **onLoad** tab.

   ```
   var q = runQuery('customers');
   ```

   This script runs the ‘customers’ query and stores the result set in a variable ‘`q`’.

8. Expand the Style Studio **Properties** panel. (Select ‘Properties Pane’ from the Style Studio ‘Window’ menu.)

9. Select `cell[1,0]` (second row, first column) and select the **Data** tab in the **Properties** panel.

10. In the ‘Binding’ region of the **Data** tab, choose the ‘Formula’ option, and enter the following formula:

    ```
    toList(q['state'])
    ```

11. Check the ‘Expand Cell’ box and select the ‘Vertical’ radio button. This sets the cell to expand (fill) vertically.

12. Set the ‘Cell Name’ field in the ‘Cell’ panel to be “`st`”. Click the ‘Apply’ button.

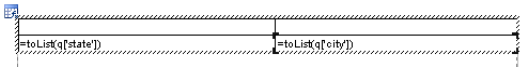
13. Select `cell[1,1]` (second row, second column).

14. In the ‘Binding’ region of the **Data** tab, choose the ‘Formula’ option, and enter the following formula

    ```
    toList(q['city'])
    ```

15. Check the ‘Expand Cell’ box and select the ‘Vertical’ radio button. This sets the cell to expand (fill) vertically.

16. Set the ‘Cell Name’ field in the ‘Cell’ panel to be “`ct`”. Click the ‘Apply’ button.



17. Preview the report.

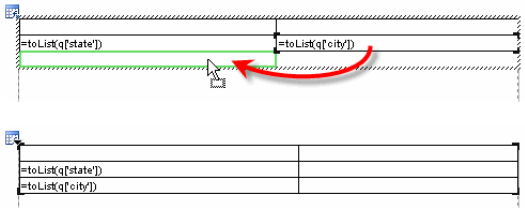
AZ	Albany
AZ	Austin
AZ	Baltimore
AZ	Boston
AZ	Boulder
AZ	Cambridge
AZ	Camden
AZ	Chicago
AZ	Dallas
AZ	Denver
AZ	Hartford
AZ	Houston
AZ	Las Vegas
AZ	Los Angeles
AZ	Middletown
AZ	New Brunswick
AZ	New York
AZ	Newark
AZ	Olympia
AZ	Orlando
AZ	Plainfield
AZ	Rockville
AZ	San Francisco
AZ	Seattle
AZ	Springfield
AZ	Tucson
AZ	Wayne
AZ	York
CA	Albany
CA	Austin
CA	Baltimore
CA	Boston
CA	Boulder
CA	Cambridge
CA	Camden

In the preview, *all* cities are shown for each state, not just the cities actually belonging to the given state. See *Formula Tables* in *Advanced Report Scripting* and the *Sales Revenue Ledger (formula_table1.srt)* example for an illustration of how to filter the field so that only cities within the given state are listed.

Note that the two cells expand in a hierarchy (i.e., for each state in the 'State' field, the table displays every city from the 'City' field). The cell to the left is the parent (State) and the cell to the right is the child (City). These state and city cells expand in hierarchy automatically because they share the same row. By default, two vertically expanding cells on *different* rows will expand independently of each other. To make them expand in a hierarchy you need to explicitly group them together. (The same applies to two horizontally expanding cells on different columns).

18. Right-click `cell[1,0]` (second row, first column), and select 'Row' > 'Append Row' from the context menu. This adds a new row beneath the two existing rows.

19. Drag the exiting formula from `cell[1,1]` to `cell[2,0]` (third row first column).

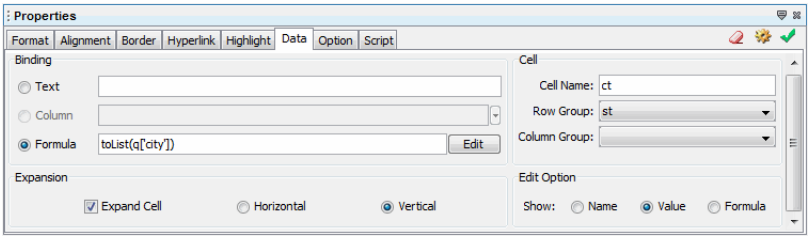


20. Preview the report.

AZ		
CA		
CO		
CT		
FL		
IL		
MA		
MD		
NJ		
NV		
NY		
PA		
TX		
WA		
Albany		
Austin		
Baltimore		
Boston		
Boulder		
Cambridge		
Camden		
Chicago		
Dallas		
Denver		
Hartford		
Houston		
Las Vegas		
Los Angeles		

Notice how the two cells expand independently. To force them to expand in a hierarchy, you must explicitly group them together.

21. Select `cell[2,0]`. From the 'Row Group' menu of the 'Cell' panel, select '`st`'. This specifies that the 'City' field should expand in hierarchy with the 'State' field.



22. Preview the table and notice how the cells expand in a hierarchy.

AZ	
Albany	
Austin	
Baltimore	
Boston	
Boulder	
Cambridge	
Camden	
Chicago	
Dallas	
Denver	
Hartford	
Houston	
Las Vegas	
Los Angeles	
Middletown	
New Brunswick	
New York	
Newark	
Olympia	
Orlando	
Plainfield	
Rockville	
San Francisco	
Seattle	
Springfield	
Tucson	
Wayne	
York	
CA	
Albany	
Austin	
Baltimore	
Boston	
Boulder	
Cambridge	
Camden	

Defining Formulas by Referencing Cells

Once you have defined row/column headers, you can define aggregation formulas. These formulas summarize (sum, count, average, etc.) data returned by the data source for a given row header element and/or column header element. For example, you can count the number of customers within each city and each state.

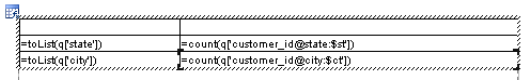
Walkthrough

Consider the table you designed in the *Expansion Hierarchies* section, which contains a listing of states and cities. You will now add formulas to count the number of customers within each city and each state. To do this, you will use cell referencing to aggregate data for a specific city and state.

```
Note: Do NOT set the cells to expand. The formula will
automatically be calculated for each cell of the expanding “st” and
“ct” cells.
```

1. Add the following formulas to `cell[1,1]` (second row, second column) and `cell[2,1]` (third row, second column), respectively

```
cell[1,1]: count(q['customer_id@state:$st'])
cell[2,1]: count(q['customer_id@city:$ct'])
```



For more details on extracting data from a query result set, see *Formula Tables* in *Advanced Report Scripting*.

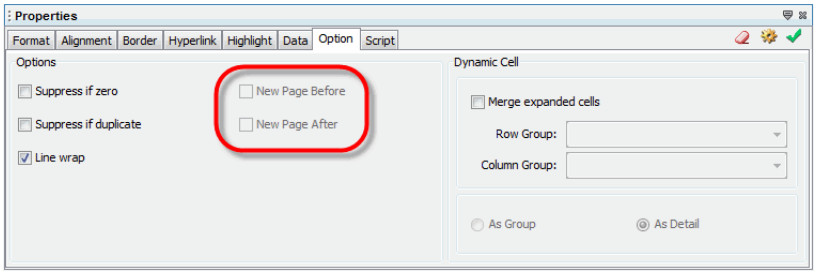
The formula `q['customer_id@state:$st']` returns an array of all the ‘customer_id’ records where the corresponding ‘state’ record is equal to ‘$st’ (‘$st’ refers to the current value of the expanding cell ‘st’). The `count()` formula counts the number of elements in this array.

2. Preview the report and notice the counts. You can verify them by inspecting the output of the base query.

AZ	1	
Albany	1	
Austin	1	
Baltimore	2	
Boston	2	
Boulder	1	
Cambridge	1	
Camden	1	
Chicago	1	
Dallas	2	
Denver	1	
Hartford	1	
Houston	1	
Las Vegas	1	
Los Angeles	2	
Middletown	1	
New Brunswick	1	
New York	4	
Newark	1	
Olympia	1	
Orlando	1	
Plainfield	1	
Rockville	1	
San Francisco	1	
Seattle	1	
Springfield	1	
Tucson	1	
Wayne	1	
York	1	
CA	4	
Albany	1	
Austin	1	
Baltimore	2	

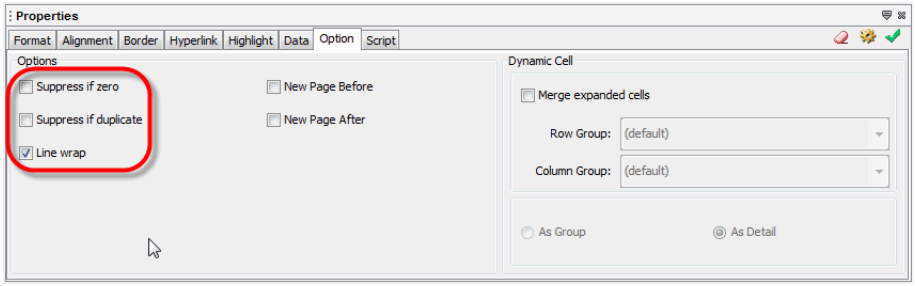
Page Break on Group in Formula Tables

When you define multiple row headers in a hierarchy (e.g., State and City), you can add a page break and print each member of a (parent) group on a new page (e.g., print each state on a new page). To do this, select the 'New Page After' option under the **Option** tab of the **Properties** panel.



Suppressing Duplicates and Zeros in Formula Tables

To suppress the display of a zero value (and instead display an empty cell), select the 'Suppress if zero' option in the **Option** tab of the **Properties** panel. To suppress the display of duplicate rows (and instead display just a single representative row), select the 'Suppress if duplicate' option.



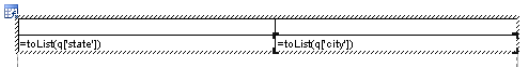
Spanning Cells

Walkthrough

When you create multiple levels of grouping, you can set a 'parent' to merge across each instance of its 'child'. For example, you can merge a 'State' cell over the 'City' cells. You can also make a static cell span an expanding cell, e.g., a title for a column header span across the horizontal cells.

The following example illustrates simple cell spanning. Follow the steps below:

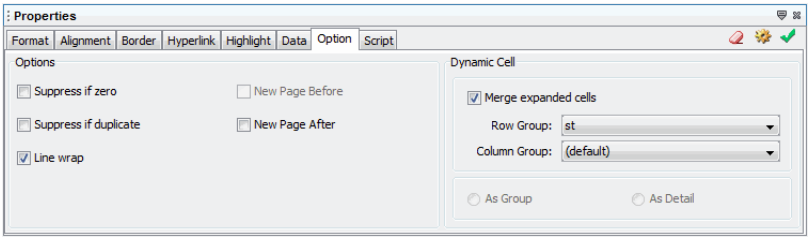
1. Start with the formula table discussed in the *Expansion Hierarchies* example.



2. Expand the Style Studio **Properties** panel. (Select 'Properties Pane' from the Style Studio 'Window' menu.)

3. Select `cell[1,0]` (second row, first column) and select the **Option** tab in the **Properties** panel.

4. In the 'Dynamic Cell' region, enable 'Merge expanded cells'. From the 'Row Group' menu select '`st`'.



5. Preview the report to observe how the cells of the parent row group are merged together.

AZ	Dallas	
	Denver	
	Hartford	
	Houston	
	Las Vegas	
	Los Angeles	
	Middletown	
	New Brunswick	
	New York	
	Newark	
	Olympia	
	Orlando	
	Plainfield	
	Rockville	
	San Francisco	
	Seattle	
	Springfield	
	Tucson	
	Wayne	
	York	
CA	Albany	
	Austin	
	Baltimore	
	Boston	
	Boulder	
	Cambridge	
	Camden	
	Chicago	
	Dallas	
	Denver	
	Hartford	
	Houston	
	Las Vegas	
	Los Angeles	
	Middletown	
	New Brunswick	
	New York	
	Newark	

Advanced Formula Table Walkthrough

This section walks you through the creation of two advanced formula table reports:

- *Sales Revenue Ledger (formula_table1.srt)*
- *Employee Performance (formula_table2.srt)*

Note: You may wish to review *Formula Tables* in *Advanced Report Scripting*, which explains the scripting elements that are used in the current section.

The completed reports (*formula_table1.srt* and *formula_table2.srt*) are included in the *examples/docExamples/script* folder of your installation.

To create more basic formula tables using drag-and-drop techniques, see *Walkthrough: Creating a Formula Table*.

Sales Revenue Ledger (formula_table1.srt)

The sales revenue ledger is a commonly used report layout. An example is shown below.

	Total Quantity Average Price	Gross Revenue Discounted Revenue	Net Revenue
AZ			$447,444.52
Folk Mining	1,107.00	$456,333.85	$447,444.52
	$476.65	$8,889.33	
CA			$1,647,424.23
Computer Tech	1,301.00	$503,344.50	$500,026.98
	$416.74	$3,317.52	
Direct Sales	1,420.00	$621,860.30	$616,392.17
	$460.82	$5,468.13	
West Coast Customs	1,131.00	$344,364.30	$340,112.07
	$324.42	$4,252.23	
Whittaker & Co, Inc	794.00	$196,310.95	$190,893.00
	$256.63	$5,417.95	
CO			$833,632.62
Ernst Handel	664.00	$313,914.85	$310,247.82
	$469.16	$3,667.03	
Quick Stop	1,175.00	$524,388.35	$523,384.80
	$414.81	$1,003.55	
CT			$161,218.88
Loctite Corp.	676.00	$161,689.50	$161,218.88
	$316.02	$470.62	
FL			$366,092.64
Fast Transportation	1,102.00	$377,534.80	$366,092.64
	$347.94	$11,442.16	
IL			$558,064.24
Specialty Retail	1,456.00	$558,702.80	$558,064.24
	$441.10	$638.56	

Walkthrough

The sales revenue ledger report contains a vertical listing of grouped entities (e.g., State and Customers), and a horizontal multi-layered listing of headers. The headers might correspond directly to a column of data, or to calculated fields. In this example, the table is populated with data from two separate queries ‘customers’ and ‘Order details’.

Follow the steps below to create this formula table:

1. Click the ‘New’ button in the Style Studio toolbar. This opens the ‘New Asset’ dialog box.

2. In the left panel, select the ‘Report’ node. In the right ‘Types’ panel, select ‘Blank Tabular Report’, and click ‘OK’.

3. Right-click on an empty region of the report and select ‘Script’ from the context menu. This opens the Script Editor.

4. Select the **onLoad** tab of the Script Editor, and enter the following lines:

```
var q1 = runQuery('customers');
var q2 = runQuery('Order details');
```

This script executes the 'customers' and 'Order details' queries and stores the resulting tabular data in variables `q1` and `q2`, respectively.

5. Expand the Style Studio **Toolbox** panel. (Select 'Toolbox Pane' from the Style Studio 'Window' menu.)

6. Drag the 'Table' component from the **Toolbox** panel to the report. This creates a new Table without a binding.



7. Right-click the Table and select 'Convert to Formula Table' from the context menu.

8. Change the Table dimensions to 6 rows and 4 columns. See *Inserting Rows and Columns into a Table* for details.

In the following steps, you will fill the table with the vertical listing of 'companies' grouped by 'state', starting with the top level of grouping (state), followed by the sub-group (company within each state).

9. Select `cell[2,0]` (third row, first column) and select the **Data** tab in the **Properties** panel.

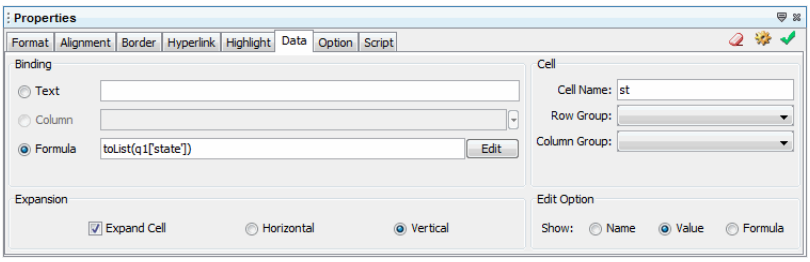
10. In the 'Binding' region of the **Data** tab, choose the 'Formula' option, and enter the following formula:

```
toList(q1['state'])
```

The "`q1['state']`" construct extracts all the values from the 'state' column of the query result set '`q1`'. The `toList` function extracts only the unique values from that result set. (See *toList* in *Advanced Report Scripting* for more information.)

11. Check the 'Expand Cell' box and select the 'Vertical' radio button. This sets the cell to expand (fill) vertically.

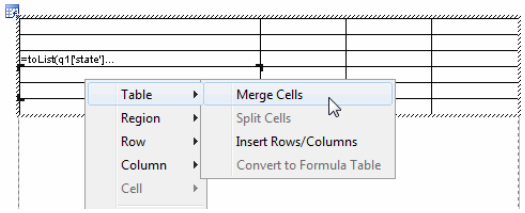
12. Set the 'Cell Name' field in the 'Cell' panel to be "`st`". Click the 'Apply' button.



Note: To merge two cells: left click on the first cell and drag your mouse over the second cell to select both, then right-click > 'Table' > 'Merge Cells'.

13. Shift-click on both `cell[3,0]` (fourth row, first column) and `cell[4,0]` (fifth row, first column) to select both cells.

14. Right-click on the selected region, and choose 'Table' > 'Merge Cells' from the context menu.



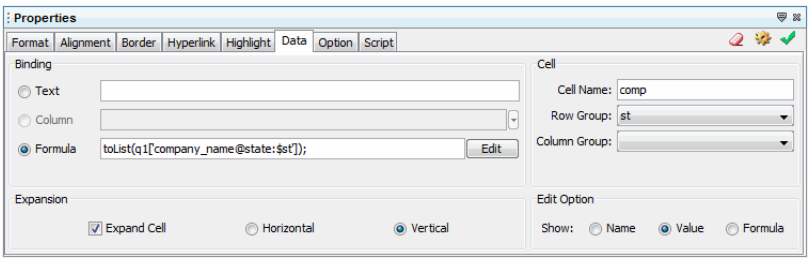
15. In the 'Binding' region of the **Data** tab, choose the 'Formula' option, and enter the following formula for the merged cell:

    ```
    toList(q1['company_name@state:$st']);
    ```

    This extracts a unique list of companies for a given state '`st`'. The '`@state:$st`' clause ensures that you obtain only companies within the particular state '`st`'. If this clause were omitted, you would obtain a list of all the companies in the result set.

16. Check the 'Expand Cell' box and select the 'Vertical' radio button.

17. Set the 'Cell Name' to 'comp', and select '`st`' from the 'Row Group' menu. This creates the grouping hierarchy between 'State' and 'Company'.



18. Merge `cell[0,3]` (first row, fourth column) and `cell[1,3]` (second row, fourth column) as described earlier.

See *Changing a Table Cell Data Binding* for information on setting static text.

19. Add the following headers using static text:

    ```
    Cell[0,1] = 'Total Quantity'
    Cell[0,2] = 'Gross Revenue'
    Cell[1,1] = 'Average Price'
    Cell[1,2] = 'Discounted Revenue'
    Cell[5,0] = 'Total Revenue'
    Merged Cell[0,3] = 'Net Revenue'
    ```



20. Add the summary formulas. These return summaries of numeric data fields for a given entity in the group/sub-group, e.g., sum of the quantity purchased for a given company. You will add formulas to the lowest level of grouping, and then work your way up.

    a. Add the following summary formulas to the following cells:

    ```
    Cell[3,1]: sum(q2['Quantity@Company:$comp']);
    Cell[4,1]: average(q2['Price@Company:$comp']);
    Cell[3,2]: sum(q2['Total@Company:$comp']);
    Cell[4,2]: sum(q2['=Total*Discount@Company:$comp']);
    ```

    b. For `cell[3,2]`, set the 'Cell Name' to '`grossRevenue`'.

    c. For `cell[4,2]`, set the 'Cell Name' to '`discountRevenue`'.

21. Calculate the 'Net Revenue' by subtracting the 'grossRevenue' from the 'discountRevenue'.

    a. Merge `cell[3,3]` (fourth row, fourth column) and `cell[4,3]` (fifth row, fourth column) as described earlier.

    b. Add the following formula to the merged cell:

    ```
    $grossRevenue-$discountRevenue
    ```

    c. For the merged cell, set the 'Cell Name' to 'netRevenue'. This is the net revenue for every company within a state.

22. Add the following formula to `cell[2,3]`:

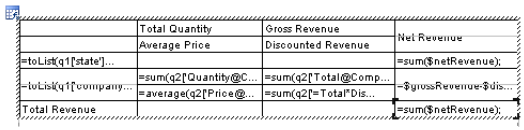
    ```
    sum($netRevenue);
    ```

    This is the total net revenue for each state.

23. Add the following formula to `cell[5,3]`:

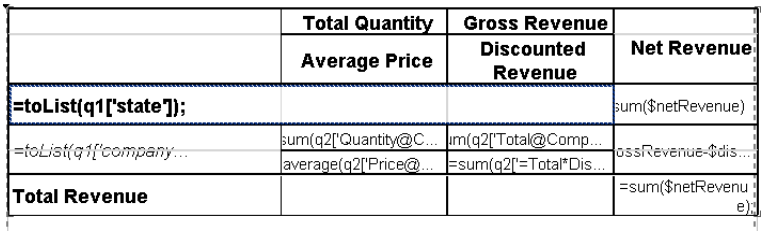
    ```
    sum($netRevenue);
    ```

    This is the grand total of the net revenue.



```
Note: Define the formulas at the lowest grouping level, and then
use the cell reference to perform summaries at the higher levels of
grouping. The same formula, 'sum($netRevenue)', yields different
results based on the context of the cell to which it is added.
```

24. (Optional) To match the example, add appropriate number and currency formats to the formula and static text cells, merge adjacent blank cells, and set the desired cell alignments. Bolding, italics, and larger font size help make the row and column headers stand out.



	Total Quantity	Gross Revenue	
	Average Price	Discounted Revenue	Net Revenue
=toList(q1['state']);			sum($netRevenue)
=toList(q1['company'...	sum(q2['Quantity@C...	im(q2['Total@Comp...	ossRevenue-$dis...
	average(q2['Price@...	=sum(q2['=Total*Dis...	
Total Revenue			=sum($netRevenue);"

25. Preview the report to view the output, which should match the figure at the beginning of this section, *Sales Revenue Ledger (formula_table1.srt)*.

Employee Performance (formula_table2.srt)

This example is an Employee Performance report which displays the sales made by sales employees to clients in different states, and breaks down performance by fiscal year and product category.

	Year				Category						
	2002	2003	2004	2005	Business	Educational	Games	Graphics	Hardware	Office Tools	Personal
Annie	2407	2161	2675	2898	1250	1298	2073	666	2119	1231	1504
AZ	261	173	83	236	122	101	265	36	125	73	31
CA	572	583	896	474	240	256	527	192	531	340	439
CO	206	171	149	111	46	143	117	48	103	89	92
CT	0	0	0	0	0	0	0	0	0	0	0
FL	0	0	0	0	0	0	0	0	0	0	0
IL	173	54	187	225	65	67	152	17	150	59	129
MA	0	0	0	0	0	0	0	0	0	0	0
MD	122	271	92	201	87	115	99	28	132	140	85
NJ	0	0	0	0	0	0	0	0	0	0	0
NV	14	55	162	42	69	30	27	19	71	46	11
NY	124	298	109	85	65	101	54	129	103	68	96
PA	0	0	0	0	0	0	0	0	0	0	0
TX	600	418	704	979	376	307	518	180	613	341	366
WA	335	138	293	545	181	178	314	17	291	75	255
Eric	3628	3111	2800	2666	1216	1438	2229	1195	2521	1533	2073
AZ	0	0	0	0	0	0	0	0	0	0	0
CA	0	0	0	0	0	0	0	0	0	0	0
CO	7	41	144	129	22	46	64	22	91	26	50
CT	292	91	77	27	18	34	122	18	107	37	151
FL	329	100	0	33	64	48	122	59	40	31	98
IL	0	0	0	0	0	0	0	0	0	0	0
MA	931	696	941	820	331	424	597	294	729	414	599
MD	437	252	242	41	119	91	176	94	171	149	172
NJ	945	1010	1028	908	358	504	622	429	885	569	524
NV	0	0	0	0	0	0	0	0	0	0	0
NY	451	666	239	689	239	256	392	223	338	240	357
PA	236	255	129	19	65	35	134	56	160	67	122
TX	0	0	0	0	0	0	0	0	0	0	0
WA	0	0	0	0	0	0	0	0	0	0	0

Walkthrough

This report uses dynamic listing in both the horizontal and vertical directions. It also demonstrates how to manipulate date fields in formula tables and how to implement hybrid column or row headers.

For this example, you will use a query called 'Employee Sales'. This query is embedded (i.e., stored locally) in the sample '*formula_table2.srt*'

template. Before you can use this query, you must make it accessible within the global scope. To import the query into the shared query registry, follow the steps below:

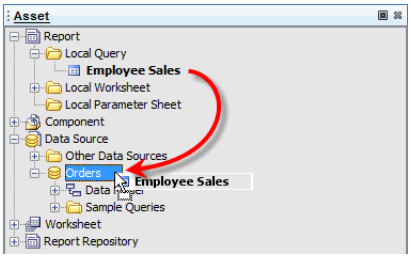
1. Open the '*formula_table2.srt*' report in Style Studio. You can find this report template in the *StyleIntelligence\examples\docExamples\script* directory.

2. Expand the Style Studio **Asset** panel.

3. Expand the 'Report' > 'Local Query' node on the tree.

4. Expand the 'Data Source' node on the tree.

5. Drag the 'Employee Sales' query from the 'Local Query' node to the 'Orders' node.



   This exports the query to the global query registry, where it will be accessible to other reports. (See *Creating a Local Query* in *Advanced Data Modeling* for more details.)

6. Close the '*formula_table2.srt*' report. (Do not save it.)

Now that the 'Employee Sales' query is available in the shared registry, follow with the steps below to create the desired formula table:

1. Click the 'New' button in the Style Studio toolbar. This opens the 'New Asset' dialog box.

2. In the left panel, select the 'Report' node. In the right 'Types' panel, select 'Blank Tabular Report', and click 'OK'.

3. Right-click on an empty region of the report and select 'Script' from the context menu. This opens the Script Editor.

4. Select the **onLoad** tab of the Script Editor, and enter the following line:

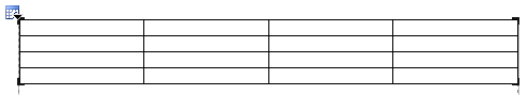
```
var q = runQuery('Employee Sales');
```

5. Expand the Style Studio **Toolbox** panel. (Select 'Toolbox Pane' from the Style Studio 'Window' menu.)

6. From the **Toolbox** panel, drag a Table component into the report.

7. Right-click the table, and select 'Table' > 'Convert to Formula Table' from the context menu.

8. Resize the Table to have 4 rows and 4 columns. (See *Inserting Rows and Columns into a Table* for details.)



9. Fill the table with the vertical listing of sales employees, grouped by state. To do this, start with the top level of grouping (Employee) followed by the sub-group (State). Follow the steps below:

   a. Expand the Style Studio **Properties** panel. (Select 'Properties Pane' from the Style Studio 'Window' menu.)

   b. Select `cell[2,0]` (third row, first column) and select the **Data** tab in the **Properties** panel.

   c. In the 'Binding' region of the **Data** tab, choose the 'Formula' option, and enter the following formula:

   ```
   toList(q['Employee']);
   ```

   d. Check the 'Expand Cell' box and select the 'Vertical' radio button.

   e. Set the 'Cell Name' to 'emp'.

   f. Select `cell[3,0]` (fourth row, first column), and add the following formula:

   ```
   toList(q['State']);
   ```

   The formula `toList(q['State'])` will list *all* states, regardless of employee. If you prefer to list only those states within which the given employee has sales, change the formula to the following:

   ```
   toList(q['State@Employee:$emp']);
   ```

   g. Check the 'Expand Cell' box and select the 'Vertical' radio button.

h. Set the ‘Cell Name’ to ‘st’, and select ‘emp’ as the ‘Row Group’. This creates the grouping hierarchy between ‘emp’ and ‘State’.

10. Fill the table with the horizontal listing of years, a horizontal listing of product categories, and an empty cell between them. To do this, follow the steps below:

    a. Select `cell[1,1]` (second row, second column), and add the following formula:

    ```
    toList(q['OrderDate'], 'date=year');
    ```

    b. Check the ‘Expand Cell’ box and select the ‘Horizontal’ radio button.

    c. Set the ‘Cell Name’ to be “yr”.

    d. Select `cell[1,3]` (second row, fourth column), and add the following formula:

    ```
    toList(q['Category']);
    ```

    e. Check the ‘Expand Cell’ box and select the ‘Horizontal’ radio button.

    f. Set the ‘Cell Name’ to be “cat”.

You have now configured the headers, and you can add the summary formulas. These formulas return summaries of numeric data fields for a given entity in the vertical and horizontal group/sub-group, e.g., sum of the quantity purchased for a given employee, in a given year, or sum of the quantity purchased in a given state, for a given employee, in a given year. etc.

First add formulas to the lowest level of grouping, and then work your way up.

11. Select `cell[3,1]` (fourth row, second column), and add the following formula:

    ```
    sum(q['Quantity@=year(OrderDate):$yr;Employee:$emp;State:$st']);
    ```

12. Set the ‘Cell Name’ to be “salesEmpYearState”.

Note the difference in the syntax of the two formulas. The formula for cell[3,1] uses a function “year(OrderDate)” to extract the year of a given order date. (See the *Formula Tables* section of the *Advanced Report Scripting* for more details).

13. Select `cell[3,3]` (fourth row, fourth column), and add the following formula:

```
sum(q['Quantity@Employee:$emp;State:$st;Category:$cat']);
```

14. Set the 'Cell Name' to be "salesEmpCatState".

`You should define the formulas at the lowest grouping level, and use the cell reference to perform summaries at the higher levels of grouping.`

15. Add the summary formulas at the Employee level (the higher level of grouping). Follow these steps:

    a. Select `cell[2,1]` (third row, second column), and add the following formula:

    ```
    sum($salesEmpYearState);
    ```

    b. Select `cell[2,3]` (third row, fourth column), and add the following formula:

    ```
    sum($salesEmpCatState);
    ```

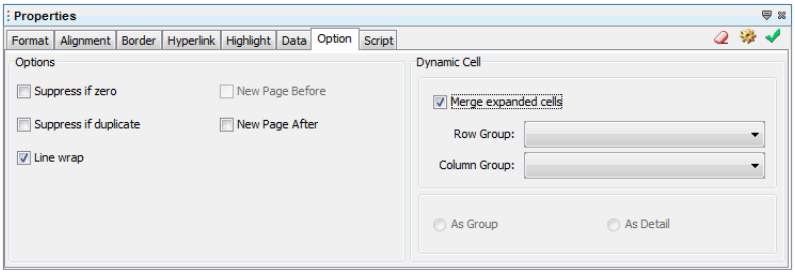
16. Now add column title cells which span their entire datasets:

    a. Click to select `cell[0,1]` (first row, second column), and type the text "Year".

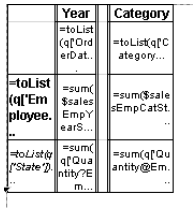
    b. Select the **Option** tab in the **Properties** panel, and enable 'Merge Expanded Cells'.

    c. Click to select `cell[0,3]` (first row, fourth column), and type the text "Category".

    d. Select the **Option** tab in the **Properties** panel, and enable 'Merge Expanded Cells'.



17. Resize the columns, modify the fonts, assign borders, etc., so that the report displays nicely.



18. Preview the report to view the output. The final result can be seen in the report named '*formula_table2.srt*'.

8 Viewing a Report

Report elements do not display data when Style Studio is in its default *design view*. The suppression of data allows you to focus on overall report design and layout, and avoids delays associated with query execution.

However, during the design process you may want to view the report with actual data so that you can evaluate how well the report page is being used when the data is rendered on the elements. The following sections explain various methods for viewing a report together with its data.

8.1 Displaying a Report in Live Edit View



To display actual data in the Style Studio design view, click the 'Live Edit' button in the Style Studio toolbar. This populates the report elements with data retrieved from the database, and permits you to continue making design manipulations.

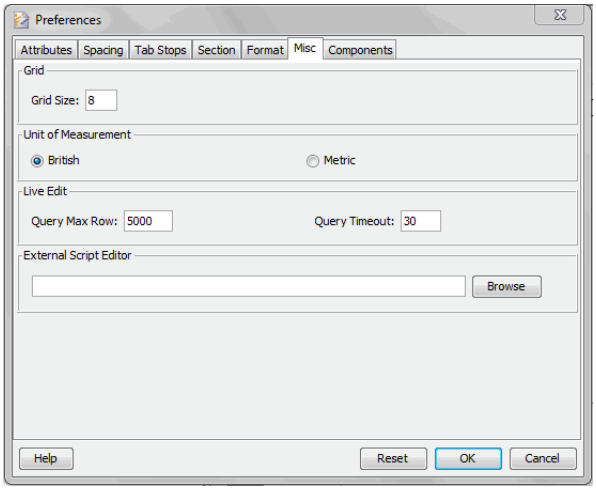
Because displaying a large dataset may slow down editing, the retrieved result set is limited to 500 rows by default. To change the data size limit, follow the steps below:

1. Select 'Preferences' from the Style Studio 'Report' menu. This opens the 'Preferences' dialog box.

2. Select the **Misc** tab.

3. In the 'Query Max Row' field of the 'Live Edit' panel, enter the maximum number of query rows that the database should return in Live Edit mode.

4. In the 'Query Timeout' field, enter the maximum duration (in seconds) that a query should be permitted to execute in Live Edit mode before timing out.



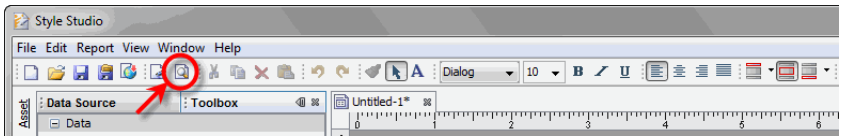
5. Press ‘OK’ to close the ‘Preferences’ dialog box.

To disable Live Edit mode, click the ‘Live Edit’ button again.

8.2 Displaying a Report in the Preview Tab

To close the preview, click the ‘X’ in the ‘Preview’ tab.

To display a rendering of the report as it will appear to users (including all data and scripting), press the ‘Preview’ button in the Style Studio toolbar.



This opens a new tab to display the rendered report.

The row immediately below the ‘Preview’ tab contains the report paging controls and the following additional options:



- **Save Parameters**: The ‘Save Parameters’ option caches the parameter values that you enter when first previewing the report. Subsequent report previews automatically reuse these same cached parameters, bypassing the parameter prompting, until you disable the ‘Save Parameters’ option. This feature is useful in cases when you need to repeatedly preview a report (for example, when making layout or formatting adjustments), and do not wish to repeatedly re-enter parameter values.
- **Enter Parameters**: The ‘Enter Parameters’ button reopens the parameter prompt dialog box, allowing you to enter a new set of report parameters.

```
Note: For reports containing scripted behavior, you may need to
press the ‘Preview’ button in order to correctly re-execute the
scripts.
```

- **Refresh on Change**: The ‘Refresh on Change’ option regenerates the preview whenever you click the existing ‘Preview’ tab. When this option is enabled, you do not need to press the ‘Preview’ button in the Style Studio toolbar to regenerate a preview; instead, simply click the existing ‘Preview’ tab to regenerate the preview. This prevents the proliferation of tabs, and reduces memory consumption.

You can set other preview options under the **General** tab of the ‘Style Studio Configuration’ dialog box. (Select ‘Configure’ from the Style Studio ‘File’ menu) See *Configuring General Style Studio Properties* for more information about the available options.

8.3 Displaying a Report in the User Portal

To display a report in the User Portal, you must first save or deploy the report into a repository that is accessible to the Style Intelligence server. See *Saving a Report into the Working Repository* or *Deploying an Asset* in *Using Style Studio*.

For information on how to open the report within the User Portal, see *Interacting with Reports*.

9 Saving a Report

Style Studio allows you to save a report to the local file system or into a repository, where is can be made accessible to a server. You can also export a report to one of a variety of formats or a portable JAR archive. The following sections explain these options in detail.

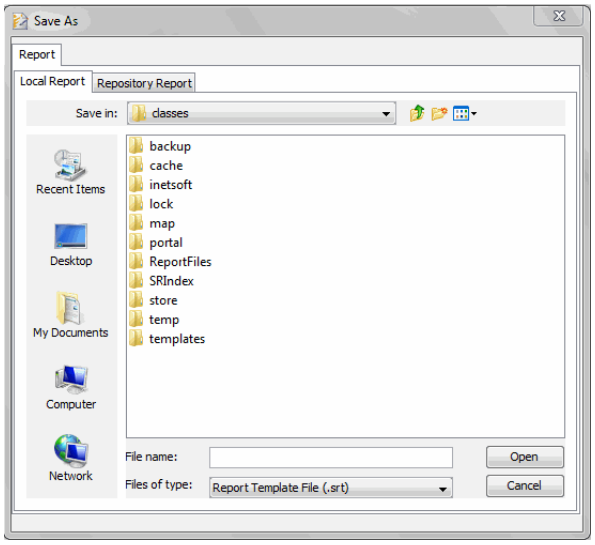
9.1 Saving a Report into the File System

To save a report to the local file system, follow the steps below:

1. From the Style Studio ‘File’ menu, select ‘Save’ or ‘Save As’. This opens the ‘Save As’ dialog box.

   If the report has been saved previously, the ‘Save’ command simply saves the current version under the same name and location, whereas the ‘Save As’ command allows you to specify a new name and location. If the report has *not* previously been saved, the ‘Save’ and ‘Save As’ commands produce the same results.

2. In the ‘Save As’ dialog box, select the **Report** tab.



3. Navigate to the location in the file system where you want to save the report, and enter the desired report name in the ‘File name’ field.

4. Press ‘Save’ to save the report in the specified location.

Report names end with the extension *.srt*, which designates “style report template.” A saved report is referred to as a “template” because it does not contain the *data* that will ultimately be displayed. The saved report

contains only formatting and data-binding information; the data itself is retrieved "live" from the database at the time that the report is generated.

To save the report template together with its data, follow these steps:

1. Preview the report. (See *Displaying a Report in the Preview Tab*.)

2. Save the preview as an *.sro* file.

The *.sro* format is chiefly useful for debugging purposes. To create a distributable copy of the report containing data, see *Exporting a Report*.

9.2 Saving a Report into the Working Repository

To view a report within the User Portal, you must deploy the report to a repository that can be accessed by a Style Intelligence server.

Note: To deploy the report to an alternate repository, see *Deploying an Asset* in *Using Style Studio*.

To save (deploy) a report to the working repository, follow the steps below.

1. In Style Studio, open the report you wish to save if it is not already open.

2. From the Style Studio 'File' menu, select the 'Save As' command. This opens the 'Save As' dialog box.

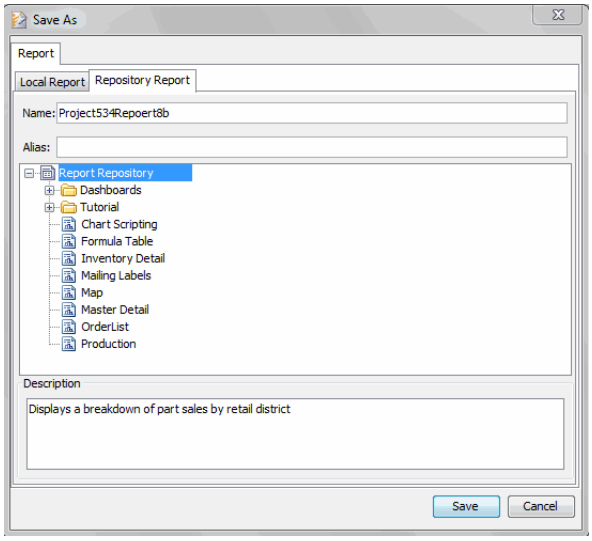
3. In the 'Save As' dialog box, click the **Repository** tab.

   The **Repository** tab provides access to the *working repository*, the location from which Style Studio accesses reports and other assets. The physical storage of reports and assets depends on the type of repository that you have configured in Style Studio. (See *Configuring a Repository* in *Using Style Studio* for further information.)

4. In the 'Name' field, enter a name for the deployed report.

   This name that you enter is used as the name of the physical report file (a "*.srt*" extension is added automatically). The name is also displayed as the report's label in the 'Repository' listing in the User Portal, unless you specify an alternate 'Alias' for this purpose (below).

5. (Optional) In the 'Alias' field, enter the label under which the report should be displayed in the 'Repository' listing of the User Portal. If you do not supply an 'Alias', the 'Name' field is used to provide this label.

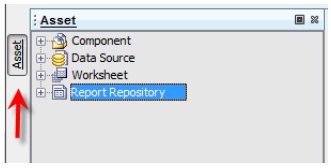


6. (Optional) Enter a 'Description' for the report. The description is displayed to users when they hover the mouse over the report name in the 'Repository' listing of the User Portal.

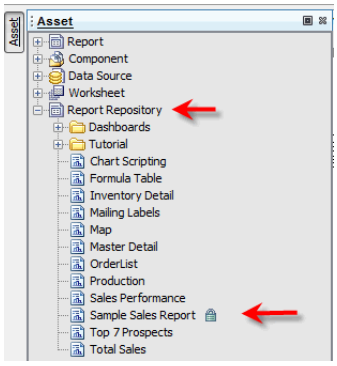
7. Press 'Save' to deploy the report into the working repository.

To verify that the report has been successfully deployed into the working repository, follow the steps below:

1. Expand the Style Studio **Asset** panel.



2. Expand the 'Report Repository' node on the tree.



3. Verify that the deployed report is displayed in the 'Report Repository' listing. Note that in the Style Studio **Asset** panel the report is displayed using the label supplied by the 'Name' field rather than the label supplied by the 'Alias' field.

The deployed report will be available in the User Portal of a server that shares the same working repository. If the repository is of type 'Local', the report template file (*.srt*) is copied to a folder called 'templates' in that location.

See *Interacting with Reports* for information on how users can access this report in the User Portal.

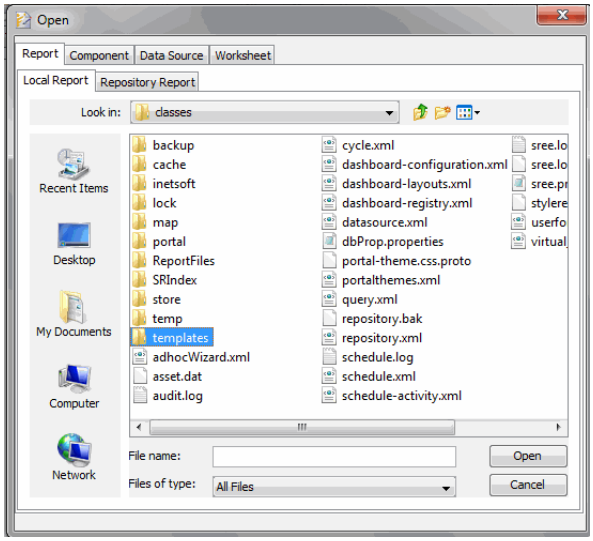
9.3 Opening a Report in Style Studio

To open an existing report, follow the steps below:

1. From the Style Studio 'File' menu, select 'Open'.

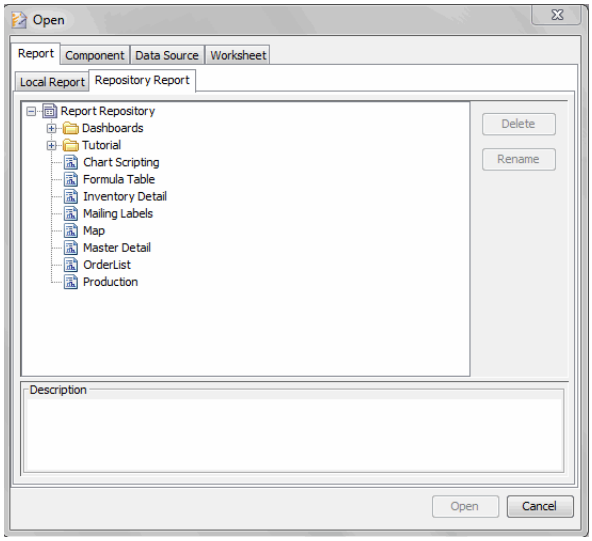
See *Saving a Report into the File System* for more information.

2. If the report was saved in the local file system, select the **Report** tab.



See *Saving a Report into the Working Repository* for more information.

3. If the report was saved in the working repository, select the **Repository** tab.

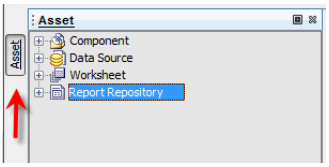


4. Navigate to the location in the file system or repository listing that contains the report you wish to open.

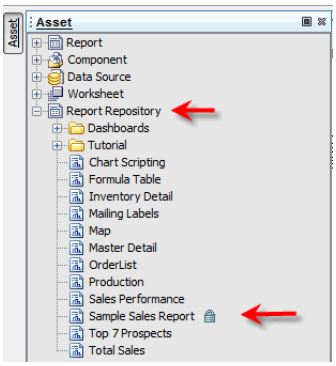
5. Select the desired report, and press 'Open'. This opens the selected report for editing in Style Studio.

You can also open a repository report from within the Style Studio **Asset** panel. Follow the steps below:

1. Expand the Style Studio **Asset** panel.



2. Expand the 'Report Repository' node on the tree.



You can also double-click on the report name.

3. Right-click the report you wish to open, and select ‘Open’ from the context menu.

9.4 Deploying a Report

You can easily deploy a report 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* in *Using Style Studio*.

9.5 Exporting a Report

To export a report from Style Studio, open the report and select ‘Export’ from the Style Studio ‘File’ menu. The following export formats are available:

Table 16. Export Format Options

EXPORT OPTION	DESCRIPTION
PDF	Exports the report as a .pdf file.
Excel (match exact layout)	Exports the report as an Excel (.xls) file which includes the headers and footers of all the report pages. (Report headers/footers are exported to Excel headers/footers.)
Excel (no pagination)	Exports the report as a .xls file with only one header and footer, i.e., without pagination.
Excel (best data editing)	Exports the report as a .xls file with two sheets. The version of the report on the first sheet is the same as the ‘Excel (match exact layout)’ version, while the version of the report on the second sheet attempts to preserve the report layout without using any cell spanning.
RTF (editable document)	Exports the report as a .rtf file.
RTF (match exact layout)	Exports the report as a .rtf file without pagination.
HTML (match exact layout)	Exports the report as a .html file.
HTML (no pagination)	Exports the report as a single page HTML file, i.e., the html file consists of only one header and one footer.
CSV	Exports the report as a comma separated file.
SVG	Exports the report as a .svg file.
Text	Exports the report as a pure text file. The text file approximates the report layout as much as possible and is suitable for sending reports to line printers or e-mailing to people with low bandwidth.

PowerPoint	Exports the report as a .ppt file. Each page will be a separate slide in the presentation.
XML	Exports the report as a well-formed XML file, intended for data analysis, including data from Text elements, Tables, Charts, and Sections. Crosstab and Chart data are represented in flattened form. Section data appears without group or band delimitation.

See Also

Deploying an Asset, in *Using Style Studio*, to bundle a report together with its required assets.

Displaying a Report in the Preview Tab, to view a report in Style Studio or generate an debugging output (*.sro*) file.

Page Break, for information on breaking a report across multiple Excel sheets.

10 Best Practices

The following sections present techniques and strategies to make report development easier and more efficient.

10.1 Planning a Group of Reports

Style Intelligence provides a number of features to improve reusability. Most reports are not completely unique, but are created among a set of related reports. These reports may share just a few elements, an entire layout, or may be entirely identical except for the data bound to the elements. Try to decide, in advance, how much overlap there will be among the reports. Then follow an appropriate strategy for reusability:

- If the reports share a common layout and some elements (such as headers, footers, or a title page) use a meta-template to serve as a basis for the reports. See *Meta-Templates* for further details.
- If the reports will share just a few common elements, such as a header, (while the layout may vary) create report beans to capture the common features. See *Report Bean* for further details.
- If the reports are identical except for the data source bound to the elements, you do not need to create multiple template files. Instead, create a single template file and bind the data dynamically using script.

10.2 Planning Report Design

When you design a report, some initial planning can help streamline the design process.

10.2.1 Deciding on Elements and Queries

First, examine what data needs to be presented and decide which report elements will best capture the information. Make sure that the existing queries will provide the data in a usable form, subject to the modifications available during data binding.

In some situations, queries are required for a particular report, but will not be used in other reports. These queries should be created as *local queries*, which are embedded and saved within the report template (*.srt*) file, as opposed to global queries which are stored in *asset.dat.d*. This avoids unnecessary clutter within the query registry. See *Creating a Local Query* in *Advanced Data Modeling* for more information.

10.2.2 Determining the Appropriate Layout

Next, determine the best layout. The *Flow Report Layout* is ideal for newsletters and similar reports where specific areas on the page must be delineated. A flow report *without* page areas is identical to a *Tabular*

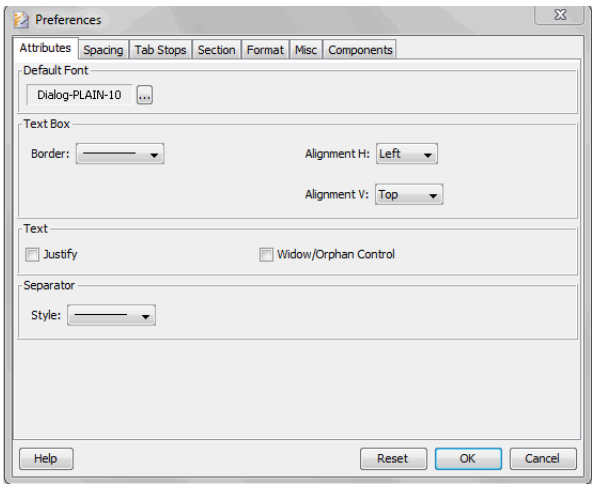
Report Layout with only one cell, but the tabular layout type has many advantages. For example, two grid cells forming two columns enable side-by-side tables or table and chart combination.

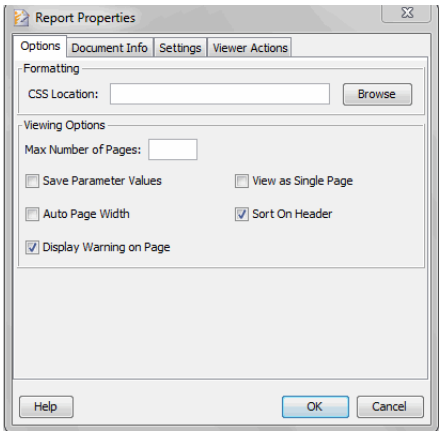
For small reports in particular, the tabular layout provides great control over the placement of elements. When there are many fixed-size elements, such as charts and text, you can designate the element location precisely. With dynamically-sized elements, the grid cells grow to accommodate the contents, so the element placement remains the same relative to the contents of other grid cells.

10.2.3 Specifying Preferences

Set your preferences before creating your report. Select ‘Preferences’ from the Style Studio ‘Report’ menu to open the ‘Preferences’ dialog box. This allows you to specify defaults for font, textbox and separator styles, margins, section properties, etc. (See the following sections for more information: *Using Grids and Rulers*, *Setting Global Section Options*, *Tab*.)

To set other report-wide settings, select ‘Report Properties’ from the Style Studio ‘File’ menu to open the ‘Report Properties’ dialog box.





10.2.4 Tables vs. Sections

Tables and Sections can both present tabular data, but there are major differences in the corresponding presentation options:

- Table elements always use a tabular grid to display data. Sections allow completely free data positioning.
- Tables can be assigned a predefined or custom table style that globally defines aesthetic attributes such as colors, fonts, and borders. Sections do not offer global style specification.

- The Table element can automatically size itself to the data that it contains. A Section presents data within individual Text elements. Because the elements do not automatically resize to fit the data, some contents may be truncated

10.2.5 Design for Accessibility

To design reports for compliance with Section 508 of the federal Rehabilitation Act of 1973, see *Section 508 Accessibility Compliance* in *Advanced Server Administration*.

10.3 Performance Tuning

Text and Textbox elements are both commonly used to add text to a report. The Textbox element differs from the Text element in that it can have borders and be anchored to a particular position. However, if you do not need these features of Textbox, it more efficient to use the Text element. This also applies to data displayed inside a Section. (See *Section Options for a Freehand Section* for information on selecting Text or Textbox representation for data in a Section.)

It is generally more efficient to specify *sorting* at the query level, rather than at the data-binding level. See *Creating a New Query* in *Advanced Data Modeling* for more information.

10.4 Precautions and Safeguards

When you develop reports for an enterprise deployment, you should take precautions to restrict user actions that might negatively affect server performance. The following are some examples of potential problems:

- Most reports contain certain parameters that can be specified by the user at runtime. However, a user might unwittingly set a combination of parameters that retrieves a very large number of records, which could impact database performance.
- Users can create and modify reports using Ad Hoc reporting tools, which include the capability to modify filtering conditions. Again, a user might unwittingly change or remove a filter in such a way that the report retrieves an excessive number of records.
- Many reports make heavy use of scripting, and user modification of such reports (using the Ad Hoc tools) can have undesirable results.

To prevent these kinds of potential problems, consider taking the following precautions when you design your reports.

Disabling Ad Hoc Analysis

Deselect the ‘Enable Adhoc Analysis’ option in the **Data Options** tab of a component ‘Properties’ dialog box to prevent a user from modifying that

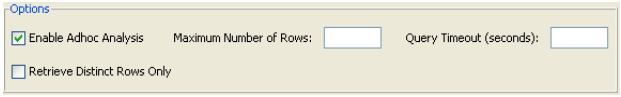
report component using the Ad Hoc tools. This may be necessary, for example, when Ad Hoc modifications could conflict with a script that is attached to the element.

Limiting the Maximum Number of Rows

Set the 'Maximum Number of Rows' option in the **Data Options** tab of a component 'Properties' dialog box to limit the number of rows returned by the raw query. This limit overrides all other maximum settings defined in Style Studio or Enterprise Manager.

Setting a Query Timeout

Set the 'Query Timeout' option in the **Data Options** tab of a component 'Properties' dialog box to specify the amount of time (in seconds) allowed for query execution. If no response is received from the database within this time, the reporting engine will attempt to cancel the execution of the query by the database, and will then proceed to complete report execution. (It may not be possible to cancel the query in all cases.)



Limiting the Maximum Number of Pages

```
It is especially important to limit the maximum number of pages for
prototype reports that will be used by the Ad Hoc Wizards.
```

Specify a value for the 'Max Number of Pages' option in the 'Report Properties' dialog box to restrict the number of pages generated by the report engine. (See *Report Properties* for more details.) Although you can also limit the raw number of records returned by a query (as discussed above), in some cases an Ad Hoc report user can make layout selections that nevertheless would generate a very large number of pages. Therefore, it is still valuable to set a maximum page limit.

See Also

Limiting Query Size and Execution Time, in *Advanced Server Administration*, for global safeguards.

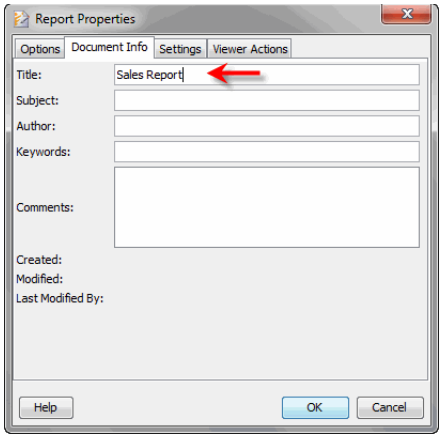
Limiting the Number of Rows in a Table, in *Advanced Data Mashup*, for Worksheet-level safeguards.

10.5 Designing Reports for Section 508 Compliance

Consider the following points when designing reports for compliance with Section 508 of the federal Rehabilitation Act of 1973.

- Set the 'Tooltip' property for all *Image* elements.
- Set the 'Tooltip' property for all *Form Elements*.

- Do not convey information solely by using color. Augment color cues with other visual cues or text.
- Avoid complex table structures such as those created with *Formula Tables*. Complex tables may not be representable to users in an accessible manner.
- Avoid using components that cause the screen to flicker. For example, do not use animated images or background color changes with frequencies between 4Hz and 59Hz.
- Use *Heading* numbering in reports to comply with PDF export accessibility requirements.
- Set the 'Title' property for the report (under the **Document Info** tab of the '*Report Properties*' dialog box).



See Also

Section 508 Accessibility Compliance, in *Advanced Server Administration*, for further information about accessibility.

Advanced Report Scripting

The *Report Scripting* provides comprehensive coverage of the JavaScript-based scripting language in Style Intelligence. Scripts can be associated with a report or report element to provide interactive behavior and implement data processing logic.

1 Contents

This Guide covers the following major topics.

- *Introduction*

  Introduction to scripting in Style Intelligence.

- *Element Script*

  Elements have properties which can be modified to determine their presentation characteristics.

- *Report Handlers*

  Scripts can be associated with particular events to support user clicks and special issues related to page breaks.

- *Server-Side Features*

  Scripts can handle many of the interactive features of server reports.

2 Introduction

Many reporting environments require dynamic report customization and implementation of specialized business logic. These needs can range from simply changing text colors based on field values to creating hyperlinks and drilldowns between reports. Style Intelligence provides a complete scripting environment to support such dynamic report behavior.

The Style Intelligence scripting language is based on the ECMA-262 (JavaScript) standard. The scripting environment provides full access to all report elements, and can be used to perform actions on report initialization and loading, as well as handling ongoing user interactions.

The following sections introduce you to the basics of JavaScript programing, and explain how to apply JavaScript in the reporting environment. Since there are many publicly available JavaScript resources, this Guide concentrates mostly on the Style Intelligence host environment, and provides extensive examples of report scripting.

2.1 Introduction to JavaScript Programming

The purpose of this section is to provide a broad overview of JavaScript programming, and to explain how JavaScript works within reports.

See Also

Appendix JS:*General JavaScript Functions*, for a simple JavaScript reference.

http://www.w3schools.com/js, for one of the publicly-available tutorials.

2.1.1 The JavaScript Language

Netscape originally introduced JavaScript as the scripting language for its Navigator web browser. This 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.

2.1.2 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 data-storage locations associated with an object. You can 'get' or 'set' these properties to observe or alter the corresponding attributes of the object. For example, every report element has a visibility property:

  ```
  Table1.visible = false;
  ```

- *Methods*: Methods are predefined functions associated with an object. (In general, these functions operate on the object itself.) For example, the CALC object provides a method for obtaining today's date:

  ```
  Text1.text = CALC.today();
  ```

- *Events*: Events are predefined actions that are recognized by an object, such as mouse movement or clicking. For example, the onPageBreak handler executes every time a page-break event occurs.

2.1.3 JavaScript Syntax 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);
```

The following sections cover 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 keywords.

All symbols in JavaScript are case-sensitive.

Declaration and Assignment

JavaScript is a weakly-typed language. This means that variables are not assigned a type when they are declared. A variable can be assigned any value, and its type is determined by the value currently assigned. Consequently, a local variable does not need to be declared before it is used.

```
var variable_name = "Hello"; // -- recommended
message1 = "Hello"; // -- without “var” also works
var count = 100;
```

If a variable is used as a report-level variable, it must be declared using the ‘var’ keyword.

```
// Place declaration in the onLoad 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 (OOP) language, properties and methods associated with each object generally 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.

```
Text1.text = "Hello"; // in report script
text = "Hello";       // in Text1 script
```

JavaScript also has a global scope which provides common methods. Since every script runs within global scope, these 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 the number object and give a decimal point of zero:

```
text = total.toFixed(0); // generates the string “5”
```

A number constant is in decimal format by default, 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
```

Numerical computations are performed with the usual operators, +, *, /, -. The increment and decrement operators (++, -—) are also available.

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 whether a value is defined by using the value in an if/else condition. For example:

```
if(parameter['start_time']) {
   // action if 'start_time' is defined
   ...
else {
   // action if 'start_time' is not defined
   ...
}
```

String Type

You can designate String constants using single or double quotes:

```
var mystring = "InetSoft Technology Corp."
mystring = 'InetSoft Technology Corp.'
```

Concatenate strings with 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, for example, 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 without specifying any parameters gives the current time:

```
var now = new Date();
```

Convert a Date to a string using the global function formatDate().

```
str = formatDate(now, 'yyyy-MM-dd'); // 2002-02-21
```

The Date format uses the same syntax as the java.text.SimpleDateFormat class.

Arrays

An array is a list contained in a single variable. Each item in the list is known as an “element,” and the entire list is enclosed in square brackets (`[]`). When you create an array, it is initialized with the specified values as its elements. The following example creates the `coffeelist` array with three elements and a length of three:

```
var coffeelist = ["French Roast", "Columbian", "Kona"];
```

Multidimensional arrays are represented as an array of arrays. A two-dimensional array (rows and columns) may be created as follows:

```
var monthly_rain = [['Jan', 'Feb', 'Mar'],
                    [ 100,    10,   30  ],
                    [ 30,     10,   300 ],
                    [ 10,     10,   10  ]];
```

Conditionals

The if/else statement evaluates a Boolean expression to decide between two alternative actions.

```
if (x > 0) {
  reg = event.region;
}
else {
  reg = event.firstRegion;
}
```

The “else if” construct allows you to test additional cases, similar to the *Switch Statement*.

```
var day = CALC.weekdayname(CALC.today())

if (day == 'Thursday') {
  Text1.text = 'Note: ' + day + ' hours are 10am-4pm.';
}
else if (day == 'Friday') {
  Text1.text = 'Note: ' + day + ' hours are 10am-12pm.';
}
else if (day == 'Sunday') {
  Text1.text = 'Note: ' + day + ' office closed.';
}
else {
  Text1.text = 'Note: ' + day + ' hours are 9am-5pm.';
}
```

For Loop

A for loop instructs the server 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 (`j = 1`) initializes the loop variable. The second expression (`j < 10`) is a condition used to check when to terminate the loop (i.e., when the condition evaluates false). The third expression (`j++`) is the variable increment that is evaluated at the end of every iteration. The “`j++`” notation is the same as “`j=j+1`,” i.e., increment by one.

While Loop

The while loop iterates the contents of the loop until the loop 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;
```

Use the "`break`" command inside any loop to exit the loop, or use "`continue`" to skip to the next iteration.

Switch Statement

The switch statement is similar to the if/else if/else structure. However, rather than choosing a branch to execute based on a binary condition, the Switch chooses a branch (i.e., a case) based on a value.

```
var day = CALC.weekdayname(CALC.today())

switch(day) {
case 'Sunday':
  Text1.text = 'Note: ' + day + ' office closed.';
  break;
case 'Thursday':
case 'Friday':
case 'Saturday':
  Text1.text = 'Note: ' + day + ' hours are 10am-12pm.';
  break;
default:
  Text1.text = 'Note: ' + day + ' hours are 9am-5pm.'
  break;
}
```

Note that one or more values can be listed on each case (e.g., Thursday, Friday, Saturday). A 'break' statement should be included at the end of each case to terminate the switch statement.

The "`default`" label at the end serves as the catch-all case. If the switch value does not match any of the explicit case values, then the "default" block is executed.

Functions

JavaScript allows you to package a block of code into a *function*, which is a subprogram that performs a particular task. To execute the block of code, you simply "call" the function that contains the code. By packaging code into a function, you can easily this common code in multiple elements and reports, which makes script maintenance much easier.

JavaScript functions behave a little differently than functions in other languages:

- Input arguments do not require reference vs. value specification. (Primitive types are passed by value, whereas objects are passed by reference.)
- Input arguments do not require data type specification
- Return values are optional and do not require data type specification

```
function max(a,b) // -- Two input arguments
{
   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*.

2.2 Editing Script

This section explains how to create and edit scripts using the Script Editor.

Adding Element-Level Script

To attach a script to a report element, follow the steps below:

1. Right-click the desired report element and select 'Script' from the context menu. This opens the Script Editor.

2. Click the **Script** or **onClick** tab, and enter the desired element-level script.

Alternatively, you can follow the steps below:

The **Script** tab in the **Properties** panel provides quick access for making minor edits.

1. Expand the Style Studio **Properties** panel. (Select 'Properties Pane' from the Style Studio *Window* menu.)

2. Select the **Script** tab in the **Properties** panel. Click the 'Script' or 'onClick' radio button, and enter the desired element-level script.

Adding Report-Level Script

To create a report level handler, follow the steps below:

Hint: Check the status bar at the bottom of Style Studio to make sure no element is selected.

1. Click in a blank area of the report to deselect all elements.

2. Right-click in a blank area of the report and select 'Script' from the context menu.

   This opens the Script Editor, which has four tabs corresponding to the four report-level handlers: **onLoad**, **onInit**, **onPageBreak**, and **onPrint**.

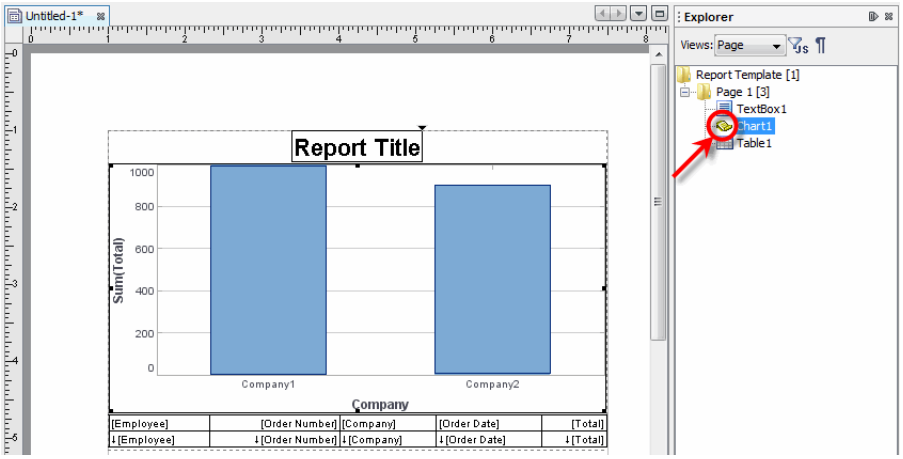
3. Click the desired tab, and enter the report-level script.

See Also

Element Script, for information about scripting elements.
Report Handlers, for information about onLoad, onInit, onPageBreak, onPrint.
Using the Default Script Editor, for information about entering the script.

2.2.1 Locating Scripted Elements

To find all scripts in a report, open the Report Explorer. Elements with script attached are displayed in the Report Explorer with a special “script” icon. To edit an element’s script, select a scripted element node in the Explorer, right click and select ‘Script’.



See Also

Report Explorer View, in *Advanced Report Design*, for more information.

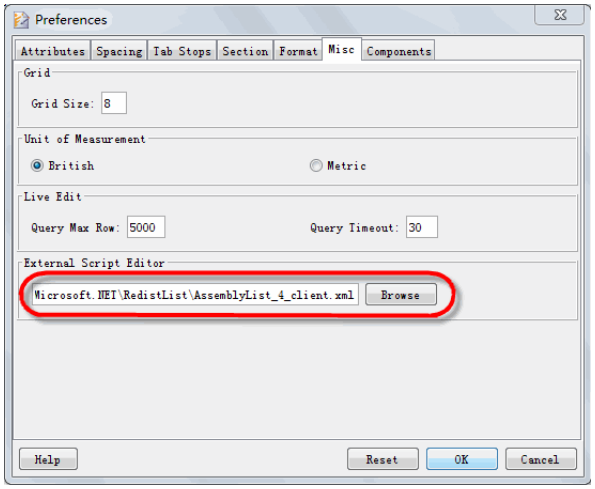
2.2.2 Using an External Script Editor

To specify an external script editor, follow the steps below:

1. Select ‘Preferences’ from the Style Studio *Report* menu. This opens the ‘Preferences’ dialog box.

2. Select the **Misc** tab.

3. In the ‘External Script Editor’ panel, click the ‘Browse’ button. Locate the desired text editor, and click OK.



4. Click ‘OK’ to close the ‘Preferences’ dialog box.

When you select the ‘Script’ option for a report or report element, the script now opens with the specified external editor. Element script and report script are stored in different “*.js*” files, with bracketed headings demarcating the script scopes.

```
[[[ Report onInit ]]]

[[[ End of script ]]]

[[[ Report onLoad ]]]

[[[ End of script ]]]

[[[ Report onPageBreak ]]]

[[[ End of script ]]]

[[[ Report onPrint ]]]

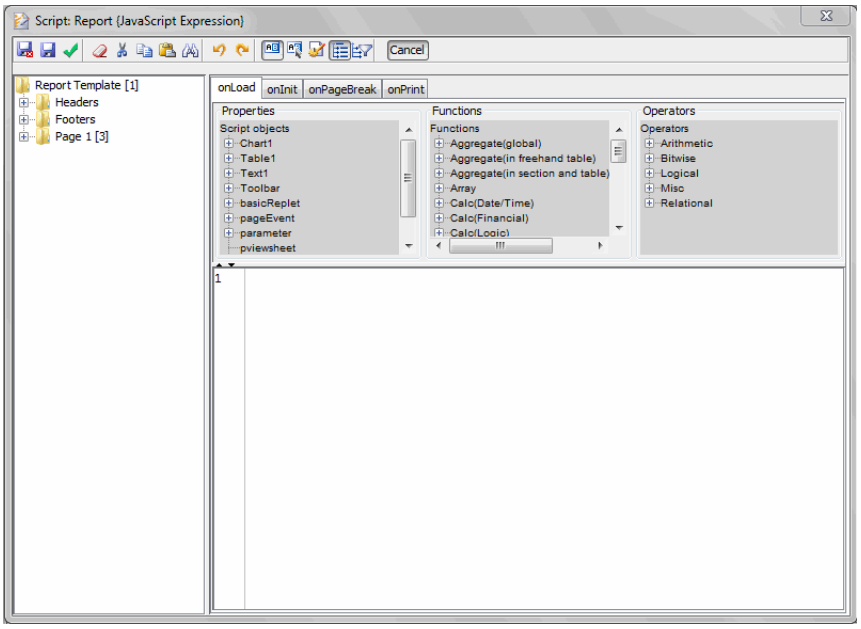
[[[ End of script ]]]

```

2.2.3 Using the Default Script Editor

The Script Editor provides a large text area where you can enter the desired script. The three panels above the text area show the ‘Properties’, ‘Functions’, and ‘Operators’ available for each type of element and object.

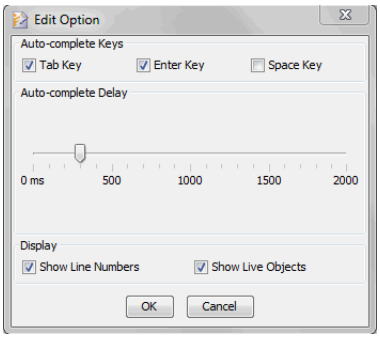
Figure 1. Script Editor



Script Editor Options



Click the ‘Edit Option’ button in the toolbar to configure the Script Editor. This opens the ‘Edit Option’ dialog box.

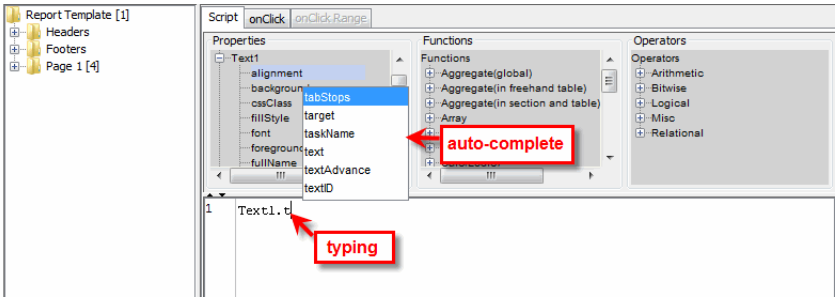


Note: If 'High Contrast' is enabled in the Windows-platform Accessibility Options, text that you type in the Editor may appear black on a black background. To make the text white, add 'high.contrast=true' to the *.stylereport* file (located in the user’s home directory). To change the text color, set 'high.contrast.color' to a hex value, e.g., 0000EE.

Enable ‘Show Live Objects’ to allow the ‘Properties’ tree to display properties for current (“live”) elements contained in the report, rather than all available element and property types. Enable ‘Show Line Numbers’ to display line numbers beside the script.

Syntax Auto-Completion

The Editor's auto-completion feature dynamically suggests valid syntax completions for the object and property names you type. This can help you avoid many errors caused by misspellings.



To use one of the suggestions in the auto-completion list, do one of the following:

- Continue typing until the correct option is highlighted. Then press 'Tab' or 'Enter' on the keyboard to make the completion.
- Press the cursor keys on the keyboard to highlight the desired option. Then press 'Tab' or 'Enter' on the keyboard to make the completion.
- Click the desired option with the mouse to make the completion.

Auto-completion is available in the following cases:

- Completing object names (e.g., "`Table1`," "`CALC`").
- Completing global function names (e.g., "`dateAdd()`").
- Completing object property names (e.g., "`Table1.visible`").
- Completing report parameter names (e.g., "`parameter.companyName`").
- Completing the right-hand side of an assignment (e.g., "`foreground = ...`"). Note that when you type the assignment operator (=), the initial auto-complete list contains an 'Edit Value' option as an aid to specifying fixed values. See *Fixed Value Selection* for more information.

The Editor shows a complete list of properties when you type the dot (.) after the object name.

Note that the auto-complete feature cannot provide auto-completions for all possible cases. For example, variables declared within the current script are not auto-completed.

See Also

Common Element Properties for information on graphically selecting property values.
Editing Script, for information on how to access the Script Editor.
Basic Editing Functions, for information on the Script Editor toolbar.

Script Debugging, for tips on preventing and trapping script errors.

Fixed Value Selection

When you use the Script Editor to assign a value to a property, the Editor can provide you with a menu of available values. This helps you to avoid syntax errors and choose an acceptable values.

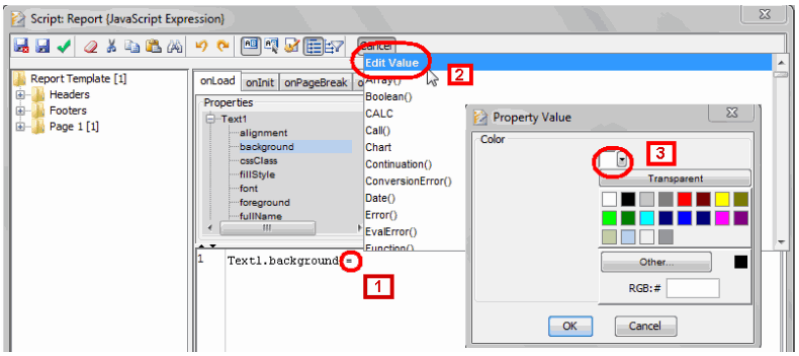
To use the Editor's menu of values, follow the steps below:

1. In the Script Editor, type a property name followed by an "equals" sign. For example:

   ```
   Text1.font =
   ```

   When you enter the "equals" sign, the Editor prompts you with a menu of available options.

2. Select the 'Edit Value' option in the menu. This opens the 'Property Value' dialog box.



3. Choose the desired value in the dialog box, and click 'OK'.

This adds the appropriate value to your script (e.g., `[255,0,0]` for the color red).

See Also
Syntax Auto-Completion, for information on property name suggestions.

Syntax Checking

To manually check the syntax, click the 'Check' button in the toolbar.

When you save a script, the Script Editor automatically checks the script's syntax. If the Editor finds a syntax problem, it provides a description and location for the error. Because the source of the error may not lie on the flagged line, you should always check the lines adjacent to the flagged location. Mismatched parentheses and missing delimiters/operators on the preceding lines are a very common cause of errors.

Because JavaScript is a weakly-typed language, it is not possible for the Editor to check for correct variable usage. For example, the syntax checker cannot detect whether a script references a non-existent variable. However, this kind of error will be reported in the Console window when you preview the report in Style Studio. See *Script Debugging* for more information about using the Console window.

See Also

Editing Script, for information on how to access the Script Editor.
Basic Editing Functions, for information on the Script Editor toolbar.
Script Debugging, for tips on preventing and trapping script errors.

2.2.4 Basic Editing Functions

The Script Editor toolbar provides basic editing and syntax checking functions listed below.

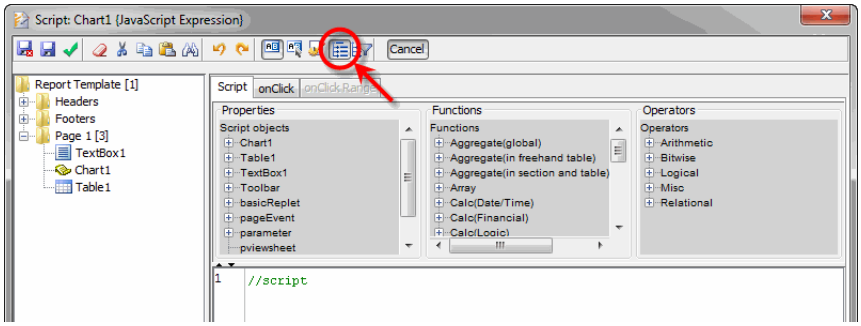
Table 1. Script Editing Toolbar Icons

	Save script and close the window. The syntax of the script is checked before saving, and error information is displayed. See *Syntax Auto-Completion*.
	Save the script. This is similar to the Save and Close button except that it does not close the script window after saving the changes.
	Check the script syntax. See *Syntax Auto-Completion*.
	Clear the current script.
	Cut the selected text into clipboard.
	Copy the selected text into clipboard.
	Paste clipboard contents into editing area.
	Enable text Search and Replace in the Script Editor.
	Undo the last edit action.
	Enable/disable auto-completion.
	Enable/disable auto-completion on mouse click.
	Open the script explorer to edit other element and report scripts. See *Editing Multiple Element Scripts*.
	Toggle the script explorer between a full element tree and a script-only element tree, showing only elements that have scripts attached.
	Edit options for Script Editor.
Close	Close the window and discard changes.

2.2.5 Editing Multiple Element Scripts

When the Script Editor is open, you can access scripts for multiple report elements, as well as report-level scripts. Follow the steps below:

1. In the Script Editor, click the 'Script Explorer' button. This displays a report element tree.



2. Select an element from the tree to view and/or edit the element's script.

See Also

Locating Scripted Elements, to determine which report elements are scripted.

Editing Script, for information on how to open the Script Editor.

2.2.6 Using the Script Library

You can save commonly-used scripts as functions in the *Script Library*. This allows you to easily reuse these scripts in multiple report elements and multiple reports. By saving common script tasks to the script library, you can keep your main scrips short, simple, and easy to maintain.

To understand the advantages of the Script Library, consider the script below, which computes the total of a table column in a report:

```
var total = 0;

for(var r = 1; r < table.length; r++) {
  total += table[r][col];
}
```

This is a common task, which you might need to perform on different table columns, on different tables, and in different reports. Rather than writing the same script again and again for each case, you should reuse the original script. For example, you can save this simple script into the Script Library as a function called "`tableTotal()`".

```
function tableTotal() {
  var total = 0;
```

```
  for(var r = 1; r < table.length; r++) {
    total += table[r][col];
  }
  return total;
}
```

You can then execute the script by calling the "`tableTotal()`" function from within any element or report script. The following sections explain how to create and use JavaScript functions and the Script Library.

See Also

Functions, in *Advanced Report Scripting*, for details of the JavaScript function syntax.

Functions, in *Advanced Dashboard Scripting*, for details of the JavaScript function syntax.

Client-Side JavaScript, in *Advanced Report Scripting*, for information on storing client-side JavaScript functions.

Adding a Function to the Script Library, for instructions on adding a function to the Script Library.

Adding a Function to the Script Library

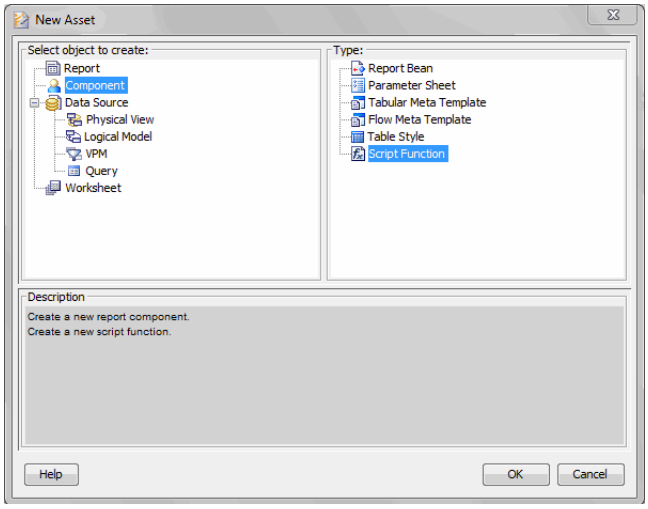
The Script Library is a collection of functions, which can be called from any report or Viewsheet with access to the Library. To add a new function to the library, follow the steps below.

```
Note: The Script Library is independent of any report or Viewsheet.
You do not need to open a report or Viewsheet in order to access
the Script Library.
```

1. Click the 'New' button in the Style Studio toolbar. This opens the 'New Asset' dialog box.

2. In the 'New Asset' dialog box, select the 'Component' node in the right panel.

3. In the left 'Types' panel, select 'Script Function', and click 'OK'.



4. This opens the new script for editing, and displays the Properties and Functions panels as an aid to composing the script.

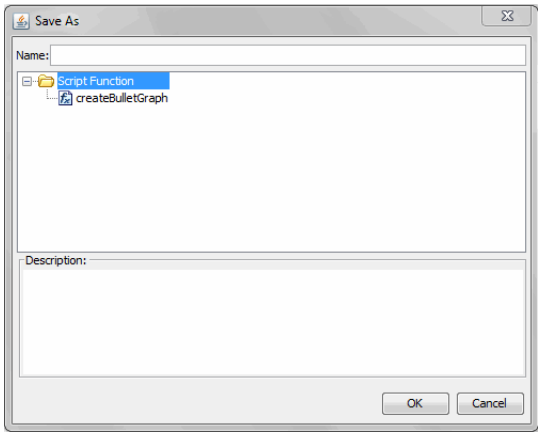
5. Write the desired script, beginning with "`function`" keyword.



Since JavaScript does not support type declaration, parameters are not qualified with types.

Any parameters (e.g., "`col`") should be placed in a comma-separated list inside the parentheses that follow the function name. These parameters can then be used inside the function body.

6. Click the 'Save' button in the Style Studio toolbar.



A function name should begin with an uppercase or lowercase letter, followed by letters, digits, or '_'. No spaces or other special characters should be used.

7. When prompted, enter a valid name for the function, and click 'OK'. This saves the script into the Script Library.

Note in the above function that the property "`table`" is not qualified by an element ID (e.g., "`Table1.table`"). Functions are scoped dynamically (see *Function Scope*); therefore, the property "`table`" refers to whichever Table element called the `tableTotal()` function. You can therefore call this `tableTotal()` function from within any table element to operate on that particular table element.

See Also

Editing a Function in the Script Library, to edit an existing function.

Functions, in *Advanced Report Scripting*, for details of the JavaScript function syntax.

Functions, in *Advanced Dashboard Scripting*, for details of the JavaScript function syntax.

Editing a Function in the Script Library

The Script Library is a collection of functions, which can be called from any report with access to the Library. To edit a function in the library, follow the steps below.

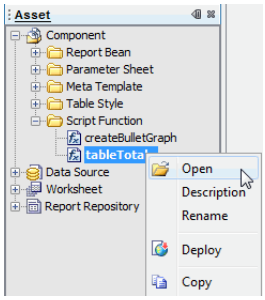
Note: The Script Library is independent of any report. You do not need to open a report in order to access the Script Library.

1. Expand the Style Studio **Asset** panel.

2. Expand the 'Component' > 'Script Function' node.

You can also double-click the desired script.

3. Right-click on the script that you wish to edit, and select 'Open' from the context menu.



This opens the Script Editor to display the chosen script.

4. Make the desired edits.

5. Click ‘Save’ to save the script into the Library.

See Also
Adding a Function to the Script Library, to create a new function.
Functions, in *Advanced Report Scripting*, for details of the JavaScript function syntax.
Functions, in *Advanced Dashboard Scripting*, for details of the JavaScript function syntax.

Function Scope

Function scoping is dynamic. This means that the scope in which the function executes is the scope of the function *caller*. For example, if a function is called from the scope of a script on the element with ID `Table1`, then all of the properties of element `Table1` are available to the function. Properties that are *not* qualified with an element ID (e.g., “`table`,” “`visible`,” as opposed to “`TableX.table`,” “`TableX.visible`”) refer to the caller, `Table1`.

This scope rule means that if a variable is not declared within a function, the script engine will try to find it in the enclosing scopes. Exactly which object will be located depends on where the function is called. To avoid ambiguity, when you use a variable in a function, you should either declare it in the function, or pass it in as a parameter.

2.3 Script Debugging

Even though the Script Editor makes scripting fast and convenient, the JavaScript language poses its own special challenges. One of the biggest challenges is debugging. Unlike strongly typed languages, JavaScript does not have fixed type for variables, which means that no compile type error checking can be done.

This chapter offers a few tips and hints on how to deal with debugging in JavaScript.

2.3.1 Keep the Script Small

The smaller the code, the easier it is to debug if anything goes wrong. Since scripts are normally used to perform relatively simple calculations, there is little reason to introduce complicated structures or organizations. Use functions and the Script Library to help keep your code concise. (See *Using the Script Library*.)

2.3.2 Associate Script with Elements

Since a script can access any element in a report, in many cases it is possible to write a single large script to perform all tasks. This poses a few problems, however:

- A large script tends to grow and become more complicated over time. This makes debugging the script much more difficult.
- If a single script affects many elements, whenever you rename or delete one of those elements, the script will generate an “object not defined” error. This kind of dependency makes report maintenance difficult, because it is not clear to developers which report elements might be under the influence of a remote script.
- When a script on one element affects the properties of other element, the efficacy of the script depends on the order in which the elements are evaluated. Because the rules for element evaluation (see *Script Evaluation*) are varied, it can be difficult to predict the final result of the script.

To avoid all of these problems, the script that you place on an element should affect only the properties of that element. For example, if you need a script that modifies the properties of element `Text1`, then add this script to element `Text1`. The majority of scripts are element-specific and should therefore each be attached to their associated element.

However, certain scripts need to access multiple elements in order to perform calculations, and it would be difficult to break these up into smaller pieces. In general, if a script works on multiple elements, it should be placed in the *onLoad Handler*. This eliminates the ambiguity of evaluation order: Since the onLoad handler is called before any element-script is evaluated, all onLoad actions take effect in the current thread.

See Also
Editing Script for information on adding element-level script.
Script Evaluation, for information about the order of element-script evaluation.
onLoad Handler, for information on placing script for pre-query execution.

2.3.3 Use log() to View Diagnostic Messages

Log() is a global function that can be called to send a message to the log. In Style Studio, log messages are displayed in the Console window. On the report server, log messages are added to the *sree.log* file and printed to stderr. Log messages can be used to verify whether a script is invoked or to check the order of script execution.

```
log("Text changed to: " + text);
```

See Also
Log Service Properties, in *Advanced Server Administration*, for instructions on how to set the “stderr” option.

2.3.4 Protect Reports from Script Errors

Because scripts are executed when the report is generated on the server, a script error can cause report generation to fail. To prevent this from happening, you should wrap failure-prone code inside a “try-catch” block, which allows you to trap errors before they affect report execution. See JS.5.4, *The ‘try-catch’ Statement*, for more information.

2.4 Script Evaluation

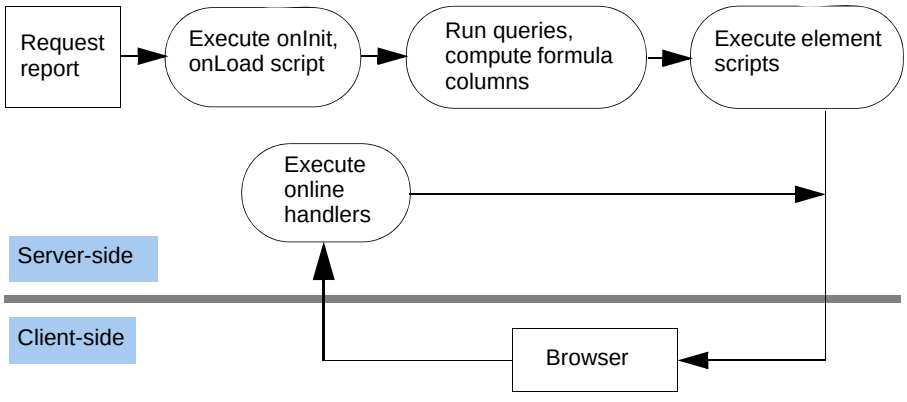
Unlike conventional JavaScript on web pages, JavaScript that you embed in a Style Intelligence report is executed as part of the report generation process on the server, *not* the client browser. However, scripts can control certain client-side interactions through event handlers and hyperlinks.

The overall sequence of report script evaluation is as follows.

1. Execute the *onInit Handler*.
2. Prompt the user for any parameters, and processes element data bindings. (Queries are not yet executed.)
3. Execute the *onLoad Handler*.
4. Execute all element queries.
5. Evaluate all formula column scripts.
6. Execute all element-level scripts.

Tip: Because of its sequence position, the onLoad handler has access to all parameters and data binding information (metadata), but not to actual data returned by queries.

Figure 2. Evaluation Sequence



The last stage in the script evaluation sequence above (element-level scripts), is governed by the following rules:

- Elements in the body of each page are evaluated before elements in the page header and footer.

- Header elements on each page are evaluated before footer elements.
- In a flow report, elements are evaluated in flow order: Scripts attached to elements at the front of the flow are evaluated before scripts attached to elements at the rear of the flow.
- In a tabular report, elements within a grid cell are evaluated in flow order: Scripts attached to elements at the front of the flow within a given cell are evaluated before scripts attached to elements at the rear of the flow in the same cell.
- In a tabular report, elements within the grid are evaluated in row order: Scripts attached to elements in an earlier grid row are evaluated before scripts attached to elements in later grid rows.
- In a tabular report, elements within different grid cells of the same row do not have a pre-determined evaluation sequence.

```
If a script modifies only the properties of the same element to
which it is attached, the evaluation order does not matter.
```

In practice, if an element is visually below another element in the report, it is normally safe to assume that the lower element is evaluated after the upper element. The primary exception is elements residing in different tabular report cells.

See Also

Script Debugging, for evaluation order-issues, and scripting best practices.
Bean Handlers, for information about execution of scripts within beans.
Client-Side JavaScript, for information about attaching client-side JavaScript.

2.5 Host Environment

The JavaScript specification defines the language's syntax and semantics. It also defines a few built-in object types together with their properties and functions. The *host environment* defines other objects that can be accessed and controlled by scripts.

```
All packages starting with java, inetsoft, com, and org are
automatically available to the scripting environment. Other
packages must be registered either by using the registerPackage()
function or by setting the javascript.java.packages property for
the application using the script.
```

The host environment of Style Intelligence is similar to the Document Object Model (DOM) used in browser environments. Scripts can access report elements as JavaScript objects to read or modify their properties.

See Also

Appendix SI:*Style Intelligence Object Reference*, for the objects and functions that are available in the Style Intelligence host environment.

Appendix JS:*General JavaScript Functions*, for core JavaScript functions.

2.6 Accessing Element Properties

You can refer to an element in a report by its element ID. For example, to set the “text” property of an element with ID ‘Text1’, preface the property name with the element ID:

```
Text1.text = "testing";
```

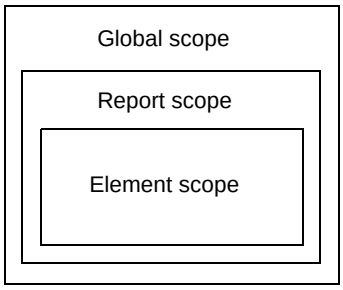
The ID in this case needs to conform to JavaScript naming requirements (see *Comments and Names*). If the ID contains special characters or white space characters that do not conform to JavaScript naming, you can reference the element as a property of the top-level “report” object.

```
report["Text1"].text = "testing";
```

This is equivalent to the previous form of reference.

The script environment is organized into three scope levels. A script attached to an element runs within element scope. When a symbol (variable, property) is encountered in the script, the server checks for the reference in the following sequence:

1. Search element scope.
2. Search report scope.
3. Search global scope.



In practice, this means that in an element script, you can reference the element’s own properties without the qualifying element ID. This is recommended.

```
background = java.awt.Color.red;       // recommended
Text1.background = java.awt.Color.red; // this also works
```

In contrast, report-level script (e.g., *onLoad Handler*) can only reference element properties by qualifying with the element ID:

```
Text1.background = java.awt.Color.blue;
```

An element script can use the fully-qualified syntax to reference properties of other elements as well. However, see the precautions noted in *Script Debugging*.

See Also

Bean Scripting Scopes, for information on scoping issues related to report beans.

Report Handlers, for information on adding script at the report level.

Element Script, for information on adding script to elements.

3 Element Script

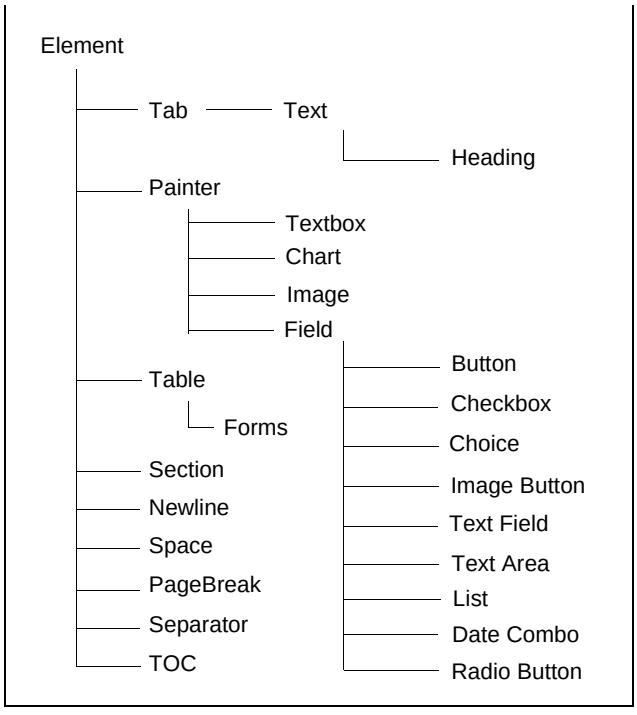
A script attached to a report element is evaluated before the element is processed. The script can change the properties of the element, including visibility. Because element script executes after query execution, the script also has access to the element's data.

In fact, an element script can access any element in the report. However, it is not good practice to modify the properties of one element from within the script of another element. See *Script Debugging* for alternative approaches.

See Also
Editing Script, for instructions on accessing and using the Script Editor.
Script Evaluation, for the order of script evaluation.

3.1 Element Inheritance Hierarchy

Report elements are organized in the object inheritance structure shown below. Properties defined in base classes are inherited by their subclasses.



3.2 Common Element 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.

3.2.1 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 = 0xFF00FF; // RRGGBB
foreground = [255, 255, 0];
foreground = {r:255,g:255,b:0};
```

Alternatively, you can create a color object by calling the constructor with the 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.

3.2.2 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 2. 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

SI.41, *StyleReport Object*, for commonly-used constant values.

3.2.3 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 *onLoad Handler*. Otherwise, the location of the element in the report may influence the outcome.

See Also

Script Debugging, for information on making scripts self-contained.
Script Evaluation, for information about order of evaluation and its effects.

3.2.4 Alignment Property

To align text within Text elements, use the Style Studio 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
SI.41, *StyleReport Object*, for commonly-used constant values.
Text and TextBox, for more information about Text properties.

3.2.5 fullName Property

Each element has a unique name, contained in the `fullName` property. For regular elements, this is the same as the element ID. For elements within a bean, this is given by the bean ID appended with the element ID. For example, an element with ID "`TitleText`" that is contained within a bean with ID "`HeaderBean`" has a `fullName` property of "`HeaderBean.TitleText`".

3.2.6 Target Property

The *target* name for an element can be used to make the element the destination of a hyperlink (similar to an HTML anchor).

```
target = 'SalesTitle' // e.g., static target text
target = text         // e.g., dynamic target text
```

As shown above, you can set the target property dynamically, for example, to the `text` property of a Text element. See *Hyperlinks to a Target (Anchor)*, in *Advanced Report Design*, for information about targeting hyperlinks to report elements.

See Also
target, in SI.2.1, *General Properties*, for reference information.
SI.4, *Text Object*, for information about Text hyperlinks.
SI.8, *Painter Object* , for information about Painter hyperlinks.
addLink, in SI.37, *Replet Object*, information on general hyperlinks.

3.3 Common Constants

Many properties use predefined constant values. Some of these constants are defined in individual classes, such as the StyleFont styles. But most of the constants are defined in the global *StyleReport Object*. This includes all line styles, chart types, table layout options, wrapping styles, page orientation, alignments, point shapes, and textbox shapes.

See Also

Appendix SI:*Style Intelligence Object Reference*, for a complete reference to all constants.

3.4 Adding Tooltips to Elements

To add a tooltip to a report element, use the function *addStatus(eid, item, message)*. The 'message' string is shown as a tooltip when the user moves the mouse over the report element with ID specified by 'eid'. If the user's browser permits status bar scripting, the 'message' string is also displayed in the status bar. (Status bar scripting is disabled by default in modern browsers.)

Note: Tooltips are not displayed in Style Studio preview. In the Style Studio preview, tooltip text is displayed in the status bar.

As an example, consider the following script, which you could place either in element-level or report-level script:

```
replet.addStatus("Title",null,"Example report");
```

The element with the ID "Title" will now display the tooltip text "Example report" when viewed in the User Portal.

See Also

addStatus(eid, item, message), in Appendix SI:*Style Intelligence Object Reference*.
toolTip, for a simple way to construct custom Chart tooltips.
Adding Tooltips to a Table or Chart, for information about the 'item' parameter.
Report Elements, in *Advanced Report Design*, for information on element IDs.
Element Script, for information on element-level scripting.
Report Handlers, for information on report-level scripting.

3.4.1 Adding Tooltips to a Table or Chart

To specify tooltips for individual table cells or chart data, use the following syntax:

```
replet.addStatus("ElementID", item, "Tooltip");
```

The '`item`' parameter specifies the particular table cell or chart data that should display the tooltip. The '`item`' parameter is an EventPoint object, [*column*, *row*].

For example, consider the following element-level script for a chart with ID ‘Chart1’. Note the index order in the ‘item’ parameter.

```
for(var i = 1; i < data.length; i++) {
  replet.addStatus("Chart1", [1,i-1], data[i][1]);
}
```

The loop iterates through every value in the first chart dataset (index ‘1’), given by “`data[i][1]`,” and assigns each value to the tooltip of the corresponding chart graphical element. Use “`data[i][0]`” to obtain the X-labels.

To perform a similar assignment of tooltips to individual cells in a table (ID ‘Table1’), you can use the following script. Again, note the index order.

```
for(var i = 1; i < table.length; i++) {
   for(var j = 0; j < table.size; j++) {
      replet.addStatus("Table1",[j,i],table[i][j]);
   }
}
```

To add a single tooltip to an entire table or chart element, leave the ‘item’ parameter as ‘undefined’ or ‘null’:

```
replet.addStatus("Table1",null,"Example table");
replet.addStatus("Chart1",null,"Example chart");
```

See Also

toolTip, for a more straightforward approach to Chart tooltips.

3.4.2 Specifying a Data Format for Tooltip Text

To specify a data format for the tooltip text, format the ‘message’ parameter input using the ‘formatNumber()’ function. In the example below, the tooltips on ‘Chart1’ are given a particular numeric format:

```
for(var i = 1; i < data.length; i++) {
  var fdata = formatNumber(data[i][1],"#,###");
  replet.addStatus("Chart1", [1,i-1], fdata);
}
```

3.5 Text and TextBox

The Text and TextBox elements are the most commonly used elements in a report. They can display static text such as titles, headers, footers, and descriptions, or data retrieved from a query (for example, in a Section element).

You can modify the properties of a Text and TextBox from within script, including font, color, and text contents.

See Also

Text Element, in *Advanced Report Design*, for basic properties of Text and Textbox.

Common Element Properties, for properties that are shared by various elements.

3.5.1 Text Property

You can access the contents of a Text or TextBox element through the '`text`' property. To change the contents, simply assign a new value.

```
Text1.text = CALC.today(); // display current date/time
```

A TextBox element has several properties in addition to those of the Text element. The '`alignment`' property controls the alignment of the TextBox within the document flow. The '`textAlignment`' property controls the alignment of the text within the TextBox boundaries.

```
// right align on the page
alignment = StyleConstant.H_RIGHT;

// center align text inside textbox
textAlignment = StyleConstant.H_CENTER;
```

See Also

textAlignment, for more information on the property.

Alignment Property, for more information on setting alignments.

3.5.2 Useful Text/String Functions

Two common string operations are changing case and searching for substrings.

Changing a String to Upper/Lower Case

To change a string to uppercase or lowercase, use the 'toUpperCase()' and 'toLowerCase()' functions, respectively. For example:

```
var s = 'John Lennon';
Text1.text = s.toLowerCase();
```

For example, to change the header cells of a table to uppercase, add the following lines to the table script:

```
for(var col = 0; col < table.size; col++) {
  table[0][col] =
    table[0][col].toUpperCase();
}
```

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, the function 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';
}
```

3.5.3 Useful Date Functions

This section discusses several basic date functions: computing the difference between dates, computing a date in the past or future, formatting a date, and extracting date components.

Finding the Difference Between Dates

Use the 'dateDiff()' function to find the difference between two dates in terms of days, months, or years.

```
dateDiff('d', fromDate, toDate);    // days
dateDiff('m', fromDate, toDate);    // months
dateDiff('yyyy', fromDate, toDate); // years
```

For example, if a table displays the column 'Birth Date', you can create a formula column to calculate the current age of an individual by subtracting the 'Birth Date' from today's date:

```
dateDiff('yyyy', field['Birth Date'], CALC.today())
```

See Also

Creating a Formula Column in Data Binding Wizard, in *Advanced Report Design*, for more on formula columns.

Calculating a Past or Future Date

Use the 'dateAdd()' function to find a date which is *n* units (days, months, or years) into the past or future.

```
// 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, consider a query 'Order Bookings' that accepts two date parameters, 'StartDate' and 'EndDate'. Assume you want the query to return data from the date range of 15 days before today's date until 15 days

after today's date. Use the `dateAdd()` function to obtain these dates as follows:

```
var sd =   dateAdd('d', -15, CALC.today());
var ed =   dateAdd('d', 15, CALC.today());
var q = runQuery('Order Bookings',
        [['StartDate', sd], ['EndDate', ed]]);
```

Extracting a Date Component

The CALC library provides a number of functions to extract different components of a date object: year, month, day, quarter, etc.

```
// extract the current date and time
var todDate = CALC.today(); //e.g., Feb-21-2007

// extract the year
var y = CALC.year(todDate); // 2007

// extract the quarter
var q = CALC.quarter(todDate); // 1

// extract the day of the week
var dow = CALC.weekdayname(todDate); // Wednesday

// extract the date
var d = CALC.day(todDate); // 21

// extract the month
var m = CALC.month(todDate); // 2
```

Formatting a Date

Use the 'formatDate()' function to format a Date field. For example:

```
// Get today's date, e.g., Thur Oct 01 16:54:22 EDT 2009
var d = CALC.today();

// Format as "Today Is: 10/01/09"
Text1.text = 'Today Is: ' + formatDate(d, 'MM/dd/yy');
```

3.6 Data Tables

Tables offer a number of special properties that allow you to access the data values from script, and to add visual style to rows, columns, and individual cells. The following sections explain how to use these properties.

See Also
SI.6, *Table Object* , for functions pertaining to Tables.
SI.2.3, *Fixed Tables*, for additional functions pertaining to Tables.

3.6.1 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];
   }
}
```

The `data` attribute can also be accessed using formula table syntax, for example:

```
data['Total@Company:$Company?inGroups(["Company",Company])'
]
```

See *Formula Tables* for more information on this syntax.

3.6.2 Setting Cell and Row Visual Properties

There are variety of properties for setting aspects of row, column, and cell visual presentation. Some common properties are listed below:

- *cellForeground* – Text color of the cell specified by row and column indices.

  ```
  cellForeground[1][3] = [123,0,123]
  ```

- *cellBackground* – Fill color of the cell specified by row and column indices.

  ```
  cellForeground[1][3] = [123,0,123]
  ```

- *colFont* – Font settings for column specified by integer index or name.

  ```
  colFont['Total']='Verdana-Bold-14'
  ```

- *colWidth* – Width in pixels for column specified by integer index or name.

```
colWidth['Total Sales'] = 0
```

- *rowHeight* – Width in pixels for row specified by integer index.

```
rowHeight[3] = 0
```

- *rowBackground* – Fill color of the row specified by the integer index

```
rowBackground[5] = [255,0,0]
```

- *setHyperlink(row, col, hyperlink)* – Hyperlink for cell specified by row and column indices.

```
setHyperlink(1, 1, "Tutorial/Ad Hoc");
```

3.6.3 Displaying Images in Table Cells

A script can load an image file from a URL or as a resource, as long as the file is located on the *classpath* (i.e., within a classpath directory, or within a JAR file on the classpath). To load the image into a report, use the global getImage() method.

```
Note: The 'checkon.gif' and 'checkoff.gif' images are packaged in
the JAR files inside the webapp lib folder (e.g., sree.jar,
bisuite.jar, etc.)

      var onIcon =
        getImage('/inetsoft/report/painter/images/checkon.gif');

      var offIcon =
        getImage('/inetsoft/report/painter/images/checkoff.gif');
```

Note: A painter/image element has an 'image' property that contains the current image. It can also be assigned an image object from a script.

Once you have loaded an image, you can assign it to a table cell as a regular data object. For example, the script below replaces `true` and `false` with the "checkmark" images loaded earlier.

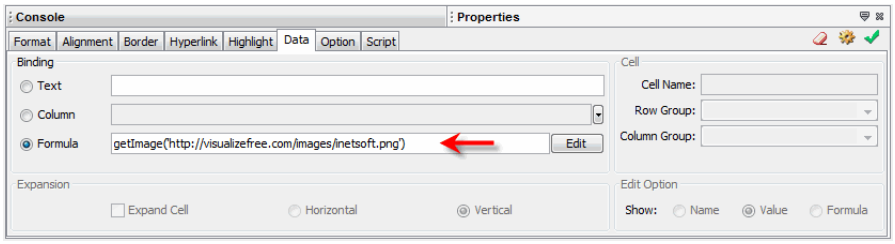
```
      for(var r = 1; r < table.length; r++) {
        if(table[r][1]) {
          table[r][1] = onIcon;
        }
        else {
          table[r][1] = offIcon;
        }
      }
```

You can load an image from an ASCII-encoded string, such as an XML file. (ASCII Hex and ASCII85 are supported.) If the argument to `getImage()` does not point to a valid resource, then the parameter is treated as an encoded image.

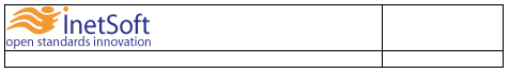
```
      // picture column is ascii85-encoded gif image
      for(var r = 1; r < table.length; r++) {
        var img = getImage(table[r]['picture']);
        table[r]['picture'] = img;
      }
```

You can also specify the image to display in a table cell by calling `getImage()` from within the cell 'Formula' in the **Data** tab of the **Properties** panel. For example, select a cell of a table (regular table or formula table), and enter the following text in the **Data** tab of the **Properties** panel:

```
getImage('http://visualizefree.com/images/inetsoft.png')
```



Press 'Apply' and preview the report. The image is displayed in the selected table cell.



3.7 Formula Tables

Formula tables are used to create real-time, spreadsheet-like reports with highly specific or complex layouts. These tables can be used to implement the kind of data grouping and aggregation which is commonly required in accounting and financial applications.

Formula tables bridge the gap between spreadsheet applications (like Excel) and traditional reporting tools. In a spreadsheet, you usually start with columns of data and then define formulas that reference other cells. To copy these formulas with appropriate references you can 'fill-down' in a range of cells.

A formula table is similar. You extract parts of a query result set and then dynamically fill the rows and columns of the table. You can then reference these cells to perform statistical calculations in formulas. The 'fill-down' operation happens automatically when the table expands to show actual data.

Most of the common ways of using a formula table can be accomplished through the point-and-click GUI of Style Studio. Sometimes the GUI is not enough. The next sections discuss the fundamentals of using scripting to reference data from query results and table cells.

See Also
Formula Tables, in *Advanced Report Design*.

Accessing Table Data, for information about the *table* and *data* attributes.

3.7.1 Extracting Data from a Query

To populate a formula table with data from a query result set, execute the query using the runQuery() method and save the result in a variable.

You should generally call the runQuery() method within the *onLoad Handler*.

```
var q = runQuery('customers');
```

You can then selectively extract parts of that query result set (see *Referencing Query Data*) and fill the cells of your formula table.

3.7.2 Extracting Data from a Report Element

To extract data from another report element (Table/Section) use the element's *table* or *data* attributes:

```
var q = Table1.table['column_name']
```

See Also
Data Tables, for information about accessing tabular data.

3.7.3 Referencing Query Data

Once you have executed the query (see *Extracting Data from a Query*), you can access specific ranges of the query result set. The data-referencing syntax allows you to also group and filter the results, and to create expressions.

Referencing a Query Column

You can reference all cells of a field (column) in the result set by using the field name:

```
q['state'];
```

The following example illustrates this approach.

Walkthrough

Consider the sample 'customers' query. In this example, you will extract all the values under the 'state' column and use them to populate a formula table. 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 'Report'. In the right 'Types' panel, select 'Blank Tabular Report' and click 'OK'.

3. Expand the Style Studio **Toolbox** panel, and drag a Table element into the report. This creates a new blank Table.

4. Right-click the table, and select ‘Table’ > ‘Convert to Formula Table’ from the context menu.

5. Right-click the table, and select ‘Table’ > ‘Insert Rows/Columns’ from the context menu.

6. Insert two additional columns so that the table has two rows and four columns.

7. Click away from the Table to deselect it.

8. Expand the Style Studio **Properties** panel. (Select ‘Properties Pane’ from the Style Studio *Window* menu.) Click the **Script** tab in the **Properties** panel.

9. Select the ‘onLoad’ button in the **Script** tab, and enter the following script:

   ```
   var q = runQuery('customers')
   ```

   This executes the ‘customers’ query in the onLoad script. (See *Extracting Data from a Query* for more details.)

10. Click `cell[1,0]` in the Table (second row, first column) to select it.

11. Click the **Data** tab in the **Properties** panel. Select the ‘Formula’ button in the ‘Binding’ section, and enter the following formula for `cell[1,0]`:

    ```
    q['state']
    ```

    This extracts the entire ‘state’ column from the ‘customers’ query to populate the table.

12. In the ‘Expansion’ region of the **Data** tab, select ‘Expand Cell’ and choose ‘Vertical’. (This causes the extracted data to fill down vertically.)

13. Preview the report and see how the table is populated with all of the records from the ‘state’ column.

NJ			
NJ			
NY			
CO			
MD			
CT			
NY			
CO			
CA			
IL			
TX			
FL			
CA			
MD			
NJ			
CA			
NY			
MA			
NJ			
AZ			
CA			
TX			
MA			
NY			
NJ			

Deriving a Result Set from Query Columns

In some cases you may need to calculate the data you want to display from existing query columns. To do this, modify the formula by placing '=' in front of the expression string.

For example, consider the formula table described previously (*Referencing a Query Column*). To merge the 'state' and the 'zip' fields into a single cell, separated by a comma (e.g., NJ, 08901), use the following formula:

```
q['=state + ", " + zip'];
```

NJ, 18322		
NJ, 09882		
NY, 03887		
CO, 38923		
MD, 82938		
CT, 39283		
NY, 08854		
CO, 98839		
CA, 08854		
IL, 88938		
TX, 32343		
FL, 12343		
CA, 11223		
MD, 83923		
NJ, 12334		
CA, 83923		
NY, 28122		
MA, 29183		

Referencing a Query Column With Field Filtering

You can filter out records of a field (column) based on the values of other fields in the result set. To do this, use '@' as the delimiter between the column name and the filtering expression and ':' to introduce the values to filter.

For example, consider the formula table described previously (*Referencing a Query Column*). To extract all the companies within a certain state (NJ), you can adapt the formula as follows:

```
q['company_name@state:NJ'];
```

Interstate Shop		
Daycare One		
Scheggie Services		
Software Specialists		
Rutgers Bank		
Pharma Research		

Note: To retrieve a sub-table rather than an array, add an initial asterisk: q['*@state:NJ'];

To filter based on multiple fields, use ‘;’ as the delimiter between the filtering expressions. For example, to find all the companies within a certain city (New Brunswick) and state (NJ), adapt the formula as follows:

```
q['company_name@state:NJ;city:New Brunswick'];
```

If the filtering expression is based on a derived field, place ‘=’ in front of the expression. For example, to find all the companies within a certain ‘state, zip’ pair (see *Deriving a Result Set from Query Columns*), adapt the formula as follows:

```
q['company_name@=state + ", " + zip:NJ, 08854'];
```

If the column name contains a space, use the `rowValue` operator:

```
q['company_name@=rowValue["US State"] + ", " + zip:NJ,
08854'];
```

Referencing a Column With Expression Filtering

You can filter out the records of a column based on a conditional expression. Use “`?`” as the delimiter between the filtering expression and the column name.

```
Note: Although Expression Filtering can achieve the same result as
Field Filtering, use Field Filtering wherever possible, since it is
better optimized.
```

For example, consider the table described previously (*Referencing a Query Column*). To extract all the customers whose customerIDs are between 20 and 30, adapt the formula as follows:

```
q['company_name?customer_id > 20 && customer_id < 30'];
```

See Also

Referencing a Query Column With Field Filtering, for optimized filtering of the result set.

Referencing a Column with Positional (Index) Filtering

You can filter out records of a column based on a range of row indices. For example, consider the table described previously (*Referencing a Query Column*). To extract the first five records from the ‘`company_name`’ column, adapt the formula as follows:

```
q['[0,company_name]:[4, company_name]'];
```

An asterisk '*' in place of the row index represents the last row in the result set.

3.7.4 Cell Referencing

You can assign a name to any cell within a formula table; expanding cell, summary cell, or a static cell with a hard-coded value. You can then use the cell's name to reference the value of the cell from within another formula.

To reference a named cell from another formula, prefix the name with a '$'. The following sections provide some examples.

Referencing a Cell for Column Filtering

The *Referencing Query Data* section explained how to extract and filter records from a specified column of a query result set. All of the examples in that section used hard-coded values as the filtering parameters.

To perform dynamic filtering, you can use cell references as the filtering parameters. This is particularly useful when the table has multiple levels of row/column headers, and you wish to filter the sub-level based on the parent level.

Walkthrough

In this example, you will create a formula table (based on the 'customers' query) with a two-level row header consisting of 'State' and 'Cities within the State'.

1. Click the 'New' button in the Style Studio toolbar. This opens the 'New Asset' dialog box.
2. In the left panel, select 'Report'. In the right 'Types' panel, select 'Blank Tabular Report' and click 'OK'.
3. Expand the Style Studio **Toolbox** panel, and drag a Table element into the report. This creates a new blank Table.
4. Right-click the table, and select 'Table' > 'Convert to Formula Table' from the context menu.
5. Right-click the table, and select 'Table' > 'Insert Rows/Columns' from the context menu.
6. Insert an additional column so that the table has three columns.
7. Repeat the above step, and add two additional rows so that the table has four rows. The table should now have four rows and three columns
8. Click away from the Table to deselect it.

9. Expand the Style Studio **Properties** panel. (Select 'Properties Pane' from the Style Studio *Window* menu.) Click the **Script** tab in the **Properties** panel.

10. Select the 'onLoad' button in the **Script** tab, and enter the following script:

    ```
    var q = runQuery('customers')
    ```

    This executes the 'customers' query in the onLoad script. (See *Extracting Data from a Query* for more details.)

11. Click `cell[1,0]` in the Table (second row, first column) to select it.

12. Select the **Data** tab in the **Properties** panel. Follow the steps below:

    a. In the 'Binding' panel of the **Data** tab, select the 'Formula' option. Enter the formula '`q['state']`' for `cell[1,0]`.

    b. In the 'Expansion' panel of the **Data** tab, select 'Expand Cell' and choose 'Vertical'.

    c. In the 'Cell' panel of the **Data** tab, set the 'Cell Name' to '`st`'.

    d. In the 'Row Group' menu, select 'default'. In the 'Column Group' menu, select 'default'.

The **Data** tab of the 'Format' panel should still be open.

13. Select `cell[1,1]` in the table (second row, second column).

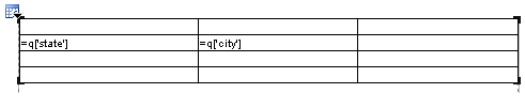
    a. In the 'Binding' panel of the **Data** tab, select the 'Formula' option. Enter the formula `q['city']` for cell[1,1].

    b. In the 'Expansion' panel of the **Data** tab, select 'Expand Cell' and choose 'Vertical'.

    c. In the 'Cell' panel of the **Data** tab, set the 'Cell Name' to '`ct`'.

    d. In the 'Row Group' menu, select 'default'. In the 'Column Group' menu, select 'default'.

    The table should appear as below.



14. Preview the report.

NJ	Wayne	
NJ	Middletown	
NJ	New York	
NJ	Boulder	
NJ	Baltimore	
NJ	Hartford	
NJ	New York	
NJ	Denver	
NJ	San Francisco	
NJ	Chicago	
NJ	Austin	
NJ	Orlando	
NJ	Los Angeles	
NJ	Rockville	
NJ	Camden	
NJ	Albany	
NJ	New York	
NJ	Boston	
NJ	Newark	
NJ	Tucson	
NJ	Los Angeles	
NJ	Houston	
NJ	Boston	
NJ	New York	
NJ	New Brunswick	
NJ	Dallas	
NJ	Seattle	
NJ	York	
NJ	Olympia	
NJ	Springfield	
NJ	Dallas	
NJ	Las Vegas	
NJ	Baltimore	
NJ	Cambridge	
NJ	Plainfield	

Notice that *all* of the cities are listed for each state, not just the cities within the corresponding state. In most cases, it is desirable to see only those cities within the corresponding state. To filter out the cities based on the state, include a field-filtering condition with a reference to the cell ‘`st`’.

Note: When making comparisons with 'null', use the syntax “`$st['.']`” to reference the cell value.

15. Change the formula in `cell[1,1]` to ‘`q['city@state:$st']`’.

16. Preview the report.

NY	New York	
NY	New York	
NY	New York	
NY	New York	
CO	Boulder	
CO	Denver	
MD	Baltimore	
MD	Rockville	
MD	Baltimore	
CT	Hartford	

Notice that the table now lists only those that correspond to the given state. (This example continues in the next section, *Referencing Cells in Summary Formulas*.)

Referencing Cells in Summary Formulas

Walkthrough

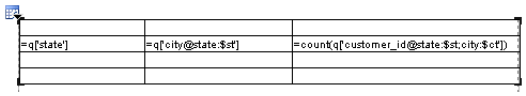
Consider the formula table from the previous example in *Referencing a Cell for Column Filtering*. It consists of a two-level row header listing ‘States’ and ‘Cities within a State’.

In the following example, you will add a formula to count the number of customers within each city.

1. Add the following formula to `cell[1,2]` (second row, third column):

```
count(q['customer_id@state:$st;city:$ct'])
```

This counts all the customers within the given city and state. The table should now appear as shown below.



2. Preview the report and note the output.

Group Numbering

To obtain the sequence number of an element within an expanding cell, use the '`#`' qualifier. (This is useful when you wish to add numbering to row/column headers.)

Walkthrough

For example, consider the formula table from the example in *Referencing Cells in Summary Formulas*. You will now add numbering for the expanding row header cell '`st`'.

1. Expand the Style Studio **Properties** panel.
2. Right-click any cell in the first column and select 'Column' > 'Insert Column' from the context menu. This adds a new column to the left.
3. Select `cell[1,0]` (second row) in the newly inserted column.
4. Select the **Data** tab in the **Properties** panel. In the 'Binding' panel, select the 'Formula' option, and add the formula '`$st['#'] + 1`'.

Do not specify any cell expansion. Note that numbering starts with 0, hence the '+1'.

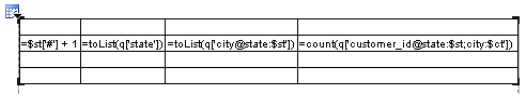
5. Select the **Option** tab. Enable 'Merge expanded cells', and select 'st' from the 'Row Group' menu.

```
Note: The toList() function extracts a unique list of values from a
given array. It can also extract a unique set of years, months,
etc., from a set of date fields.
```

6. Select `cell[1,1]` (second row, second column) in 'State' column. Under the **Data** tab, change the formula to '`toList(q['state'])`', which will show only unique values.
7. Under the **Option** tab, select 'Merge expanded cells', and select 'st' from the 'Row Group' menu.
8. Select `cell[1,2]` (second row, third column) in the 'City' column. Under the **Data** tab, change the formula to '`toList(q['city@state:$st'])`', which will show only unique values.

9. Select `cell[1,3]` (second row, fourth column) in the 'Count' column. Under the **Option** tab, select 'Merge expanded cells', and select 'ct' from the 'Row Group' menu.

   The table should appear as shown below:



10. Preview the report and notice the numbering of the groups.

1	AZ	Tucson	1
2	CA	Albany	1
		Los Angeles	2
		San Francisco	1
3	CO	Boulder	1
		Denver	1
4	CT	Hartford	1
5	FL	Orlando	1
6	IL	Chicago	1
7	MA	Boston	2
		Cambridge	1
		Springfield	1
8	MD	Baltimore	2
		Rockville	1
9	NJ	Camden	1
		Middletown	1
		New Brunswick	1
		Newark	1
		Plainfield	1
		Wayne	1

Referencing a Cell with Relative Parent Group Reference

Relative cell referencing is used primarily when comparing different summary cells with respect to their header cell. For example, you might want find the difference between the total sales for the current year and the previous year.

The syntax for relative cell referencing is as follows:

```
$cellName['grpName:+/-relative_index']

e.g., $sales['state:-1']
      $sales['yr:+1']
```

Here, `$cellName` is the name of the cell/column containing the value(s) to be compared, and `grpName` is name of the cell/column that indexes the different values.

Walkthrough

Consider a formula table based on the 'All Sales' query. In this example, you will find the difference in sales between successive fiscal years. 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 'Report'. In the right 'Types' panel, select 'Blank Tabular Report' and click 'OK'.

3. Expand the Style Studio **Toolbox** panel, and drag a Table element into the report. This creates a new blank Table.

4. Right-click the table, and select 'Table' > 'Convert to Formula Table' from the context menu.

5. Right-click the table, and select 'Table' > 'Insert Rows/Columns' from the context menu.

6. Insert an additional column so that the table has three columns.

7. Repeat the above step, and add an additional row so that the table has three rows. The table should now have three rows and three columns

8. Click away from the Table to deselect it.

9. Expand the Style Studio **Properties** panel. (Select 'Properties Pane' from the Style Studio *Window* menu.) Click the **Script** tab in the **Properties** panel.

10. Select the 'onLoad' button in the **Script** tab, and enter the following script:

    ```
    var q = runQuery('All Sales');
    ```

    This executes the 'All Sales' query in the onLoad script. (See *Extracting Data from a Query* for more details.)

11. Click `cell[1,0]` in the table (second row, first column) to select it.

12. Select the **Data** tab in the **Properties** panel. Follow the steps below:

    The second *toList* argument groups the returned dates by year.

    a. In the 'Binding' panel, select the 'Formula' option, and enter '`toList(q['Order Date'],'date=year')`' as the formula.

    b. In the 'Expansion' panel of the **Data** tab, select 'Expand Cell' to set `cell[1,0]` to expand 'Vertical'.

    c. In the 'Cell' panel of the **Data** tab, enter 'yr' for the 'Cell Name' of `cell[1,0]`.

    The **Data** tab of the 'Format' panel should still be open.

13. Select `cell[1,1]` in the table (second row, second column).

    a. Select the **Data** tab. In the 'Binding' panel, select the 'Formula' option, and enter the following formula:

    ```
    sum(q['Total@=year(field["Order Date"]):$yr'])
    ```

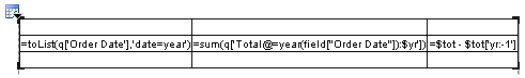
In words, this says: “For each year in column ‘yr’, find the ‘Order Dates’ falling within that year, and sum the ‘Totals’ for those order dates.” Effectively, this calculates the total revenue generated for a given fiscal year.

b. In the ‘Cell’ panel of the **Data** tab, enter ‘tot’ for the ‘Cell Name’ of `cell[1,1]`.

14. Select `cell[1,2]` (second row, third column). In the ‘Binding’ panel of the **Data** tab, select the ‘Formula’ option, and enter the following formula.

```
$tot - $tot['yr:-1']
```

This formula uses relative cell referencing to calculate the difference between the total revenue (computed in the column named ‘tot’) of the current year and the previous year. The table should appear as shown below:



15. Preview the table:

2009	3927823	3927823
2010	4649226	721403
2011	4452495	-196731
2012	4627310	174815

Referencing a Cell with Absolute Parent Group Reference

You can also use the value of the parent group to compare summary cells. To refer to a summary cell in another header group, use the absolute value of the header group, as shown below:

```
$cellName['grpName:absolute_value']

e.g., $sales['state:NJ']
      $sales['yr:"2008"'] (specify numeric values in quotes)
```

Walkthrough

Consider a formula table based on the ‘All Sales’ query. In this example, you will find the relative sales for each year compared to the fixed year 2008. 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 ‘Report’. In the right ‘Types’ panel, select ‘Blank Tabular Report’ and click ‘OK’.

3. Expand the Style Studio **Toolbox** panel, and drag a Table element into the report. This creates a new blank Table.

4. Right-click the table, and select 'Table' > 'Convert to Formula Table' from the context menu.

5. Right-click the table, and select 'Table' > 'Insert Rows/Columns' from the context menu.

6. Insert an additional column so that the table has three columns.

7. Repeat the above step, and add an additional row so that the table has three rows. The table should now have three rows and three columns

8. Click away from the Table to deselect it.

9. Expand the Style Studio **Properties** panel. (Select 'Properties Pane' from the Style Studio *Window* menu.) Click the **Script** tab in the **Properties** panel.

10. Select the 'onLoad' button in the **Script** tab, and enter the following script:

    ```
    var q = runQuery('All Sales');
    ```

    This executes the 'All Sales' query in the onLoad script. (See *Extracting Data from a Query* for more details.)

11. Click `cell[1,0]` in the table (second row, first column) to select it.

12. Select the **Data** tab in the **Properties** panel. Follow the steps below:

    The second *toList* argument groups the returned dates by year.

    a. In the 'Binding' panel, select the 'Formula' option, and enter '`toList(q['Order Date'],'date=year')`' as the formula.

    b. In the 'Expansion' panel of the **Data** tab, select 'Expand Cell' to set `cell[1,0]` to expand 'Vertical'.

    c. In the 'Cell' panel of the **Data** tab, enter 'yr' for the 'Cell Name' of `cell[1,0]`.

    The **Data** tab of the 'Format' panel should still be open.

13. Select `cell[1,1]` (second row, second column) in the table.

    a. Select the **Data** tab. In the 'Binding' panel, select the 'Formula' option, and enter the following formula:

    ```
    sum(q['Total@=year(field["Order Date"]):$yr'])
    ```

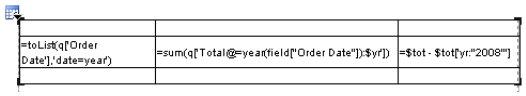
In words, this says: “For each year in column ‘yr’, find the ‘Order Dates’ falling within that year, and sum the ‘Totals’ for those order dates.” Effectively, this calculates the total revenue generated for a given fiscal year.

b. In the ‘Cell’ panel of the **Data** tab, enter ‘tot’ for the ‘Cell Name’ of `cell[1,1]`.

14. Select `cell[1,2]` (second row, third column). In the ‘Binding’ panel of the **Data** tab, select the ‘Formula’ option, and enter the following formula.

```
$tot - $tot['yr:"2008"']
```

This formula uses absolute cell referencing to calculate the difference between the total revenue (computed in the column named ‘tot’) of the current year and the year 2008.The table should appear as shown below:



15. Preview the table:

2011	3927823	3927823
2012	4649226	4649226
2013	4452495	4452495
2014	4627310	4627310

See Also

Referencing a Cell with Relative Parent Group Reference, to compare based on relative location.

Referencing a Cell with Parent Group Reference as an Expression

You can also specify the referenced group with a JavaScript expression. The JavaScript expression should be prefixed to the expression string with an ‘=’, as shown below:

```
$company['state:=iif($type == "T", $state, $province)']
```

The syntax of the `iif` function used above is *CALC.iif(condition, value_if_true, value_if_false)*.

See Also

Referencing a Column With Expression Filtering, for another use of expressions.

CALC.iif(condition, value_if_true, value_if_false), for more information about the iif syntax.

3.7.5 Special Functions

The functions listed below are commonly used to populate cells in a formula table.

inArray

The `inArray()` function determines whether a value is part of an array.

```
inArray(array, value);
e.g., inArray([1,2,3], $num);
```

This function is useful as a condition check together with the iif operator.

toArray

The `toArray()` function converts a *tableLens* object or a delimited string to a JavaScript Array.

```
toArray(tableLens);
toArray(delimString);
e.g., var a = toArray(Table1.tableLens);
      var a = toArray('1,2,3');
```

You can use the returned JavaScript Array to access table data in the usual fashion:

```
e.g., a[1][2] is the data in 2nd row, 3rd column
```

rowList

The `rowList()` function generates a list of values from a result set with column grouping, conditional filtering, and other sorting options.

```
rowList(tableLens,'conditionalspec','options string');
e.g., rowList(q, 'quantity ? discount > 0','sort=asc');
```

Here, *conditionalspec* is the column name with grouping or conditional specifications, and the *options string* values are as follows (multiple options are separated by a comma delimiter):

- `sort=asc/desc/false`: Sort the values in ascending or descending order. Default is false.
- `distinct=true/false`: Retrieve only distinct values. Default is false.
- `maxrows=num`: Limit the number of elements returned.
- `sortcolumn=column_name`: Sort data according to a specific query column which is not returned in the list.

The variable '`q`' contains the results of a query. It is usually generated in *onLoad Handler* script by the runQuery function, e.g.,

```
var q = runQuery('All Sales').
```

The `rowList()` function also adds the fields of the result set to the formula scope of the table cells, so that they can be accessed as `field['colName']`. Therefore, you can use `rowList()` to extract a portion of a query result set, and then fill out the table with other fields of the result set which correspond to the records of the extracted row:

Example: The rowList Function

Consider a formula table based on the 'All Sales' query. Assume that you want to extract all of the 'Order Numbers' where the total sales are greater than 6000, and then fill the next cell with the employee associated with the order.

rowList(q,'Order Number? Total >6000');	field['Employee']

The output is shown below.

12003	Annie
12005	Annie
12007	Annie
12012	Annie
12014	Annie
12020	Annie
12030	Annie
12033	Annie
12043	Annie
12048	Annie
12052	Annie
12068	Annie

¢

See Also

field, for more information about accessing columns of a result set.

mapList

The `mapList()` function groups a list of values according to a specified mapping. The mapping array consists of hybrid alternating name-value pairs or [name array]-value pairs.

```
mapList(list, mapping_array, 'options string')
e.g.,
mapList(q['State'],[['AZ','CA'],'West',['NY','NJ'],'East'])
```

The *options string* values are as follows (multiple options are separated by a comma delimiter):

- `others=groupOthers/leaveOthers`: Group all unmapped values in a generic group called 'Others' (`groupOthers`), or display each unmapped value as its own group (`leaveOthers`). Default is '`groupOthers`'.
- `sort=asc/desc/false`: Sort the values in ascending or descending order. Default is `false`.
- `remainder`: Specifies the label for the 'Others' group when Top/Bottom-N filtering is in effect. If not specified, the 'Others' group is not displayed.

- `distinct=true/false`: Retrieve only distinct values. Default is `false`.

toList

The *toList(list [,options])* function is the most commonly used formula table function. It generates a unique, sorted, and grouped list from the values in a JavaScript Array or *tableLens*.

In most cases, you will use `toList()` to obtain distinct values from a query array when you create header rows/columns. For example, "`toList(q['state'])`" obtains a distinct list of states in ascending order from query variable "`q`".

The full syntax of the function is as follows,

```
toList(list, 'options string');
```

where *options string* values are as follows (multiple options are separated by a comma delimiter):

- `sort = asc/desc/false`: Sort the values in ascending or descending order. Default is `asc`.
- `sorton`: Sort the values according to a specified measure, e.g., `sorton=sum(Total)`. Sorting by `sorton` is applied before Top/Bottom-N filtering implemented by `maxrows`.
- `sorton2`: Sort the values according to a specified measure, e.g., `sorton2=sum(Total)`. Sorting by `sorton2` is applied after Top/Bottom-N filtering implemented by `maxrows`.
- `remainder`: Specifies the label for the 'Others' group when Top/Bottom-N filtering is in effect. If not specified, the 'Others' group is not displayed.
- `maxrows = num`: Limits the number of rows returned.
- `distinct = true / false`: Retrieve only distinct values. Default is `true`.
- `date = year / quarter / month / week / day / hour / minute / second / weekday / monthname / weekdayname`: Group date values according to specified period, and return the period designation.
- `rounddate = year / quarter / month / week / day / hour / minute / second / weekday / monthname / weekdayname`: Group date values according to specified period, and return the rounded date value.

The `rounddate` option is useful when you want to group by month and year, (e.g., Jan 2005 and Jan 2006, etc.). For example, if the 'Order Date' field in a query has the following values,

```
[Jan-2-2002, Feb-21-2004, Feb-25-2004, Nov-25-2005]
```

then the toList function with a `rounddate` grouping of month,

```
toList(q['Order Date'], 'rounddate=month');
```

returns a unique list of dates containing the first day of the month of the given year:

```
[Jan-1-2002, Feb-1-2004, Nov-1-2005]
```

See Also

Referencing a Cell with Relative Parent Group Reference, for a complete example.

Extracting Data from a Query, to extract query data into a JavaScript Array.

3.7.6 Visual Properties

You can set the visual properties (font, alignment, background, etc.) of expanding cells, rows, and columns in the formula table script. There are three general steps:

1. Call functions to modify the visual properties of the "pre-expansion" table.
2. Call the "`expandCalcTable()`" function to expand formula table.
3. Call functions to modify the visual properties of the "post-expansion" table.

Any script that precedes the "`expandCalcTable()`" command applies to the table prior to expansion. Any script that follows the "`expandCalcTable()`" command applies to the expanded table.

See Also

SI.6, *Table Object* , for formula table functions.

Walkthrough: Formula Table Expansion

Walkthrough

Construct a formula table based on the 'customers' query, with a multilevel header comprising 'State' and 'City'.

1. Click the 'New' button in the Style Studio toolbar. This opens the 'New Asset' dialog box.
2. In the left panel, select 'Report'. In the right 'Types' panel, select 'Blank Tabular Report' and click 'OK'.
3. Expand the Style Studio **Toolbox** panel, and drag a Table element into the report. This creates a new blank Table.

4. Right-click the table, and select ‘Table’ > ‘Convert to Formula Table’ from the context menu.

5. Right-click the table, and select ‘Table’ > ‘Insert Rows/Columns’ from the context menu.

6. Add two additional rows so that the table has four rows. The table should now have four rows and two columns

7. Click away from the Table to deselect it.

8. Expand the Style Studio **Properties** panel. (Select ‘Properties Pane’ from the Style Studio *Window* menu.) Click the **Script** tab in the **Properties** panel.

9. Select the ‘onLoad’ button in the **Script** tab, and enter the following script:

   ```
   var q = runQuery('customers');
   ```

   This executes the ‘customers’ query in the onLoad script. (See *Extracting Data from a Query* for more details.)

10. Select `cell[1,0]` (second row, first column). Click the **Data** tab in the **Properties** panel.

    a. In the ‘Binding’ panel of the **Data** tab, select the ‘Formula’ option. Enter the formula ‘`toList(q['state'])`’.

    b. In the ‘Expansion’ panel of the **Data** tab, set ‘Expand Cell’ to expand ‘Vertical’.

    c. In the ‘Cell’ panel of the **Data** tab, set the ‘Cell Name’ to be “`st`”.

The **Data** tab of the ‘Format’ panel should still be open.

11. Select `cell[2,1]` (third row, second column).

    a. In the ‘Binding’ panel of the **Data** tab, select the ‘Formula’ option. Enter the formula ‘`toList(q['city@state:$st'])`’.

    b. In the ‘Expansion’ panel of the **Data** tab, set ‘Expand Cell’ to expand ‘Vertical’.

    c. In the ‘Cell’ panel of the **Data** tab, set the ‘Cell Name’ to be “`ct`” and set the ‘Row Group’ to ‘st’.

    The table should appear as shown below.



12. Preview the table.

AZ	
	Tucson
CA	
	Albany
	Los Angeles
	San Francisco
CO	
	Boulder
	Denver
CT	
	Hartford
FL	
	Orlando
IL	
	Chicago
MA	
	Boston
	Cambridge
	Springfield
MD	
	Baltimore
	Rockville

This example continues in the next section, *Pre-Expansion Script*, where you will add visual formatting to the table prior to expansion.

See Also
Creating a Formula Table Using Script (Advanced), in *Advanced Report Design*, for basic information on cell expansion.

Pre-Expansion Script

Walkthrough

This section continues the example from the previous section (*Walkthrough: Formula Table Expansion*). In this example, you will format the 'State' rows with a red background color, and format the 'City' cells with a yellow background color.

It is easiest to color these cells using pre-expansion script, because the formatting that you add to pre-expansion rows and cells is *repeated* for all corresponding rows and cells in the post-expansion table. Thus, the automatic table expansion takes the place of the *Iteration Statements* that you would otherwise need to write.

To add the pre-expansion script, follow the steps below:

1. Right-click anywhere on the table and select 'Script' from the context menu. This opens the Script Editor to edit the element script.

2. Add the following script to the table element.

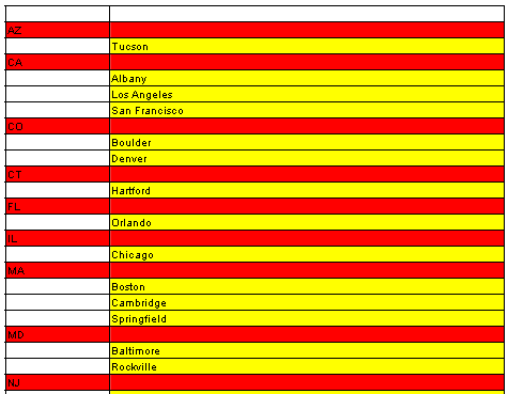
```
rowBackground[1] = [255,0,0];
cellBackground[2][1] = [255,255,0];
```

```
expandCalcTable();
```

The first line of the script specifies a red background for the second row of the pre-expansion table (the row containing ‘State’). The next line of the script specifies a yellow background for the cell in the third row, second column of the pre-expansion table (the row containing ‘City’). The third line of the script initiates table expansion, which propagates the specified formatting to all corresponding cells in the expanded table.

3. Click ‘Save and Close’ to exit the Editor.

4. Preview the result.



This example continues in *Post-Expansion Script*, where you will add post-expansion script to modify some further table properties.

Post-Expansion Script

This section continues the example from the previous section (*Pre-Expansion Script*).

In this example, you will assign every fifth row of the table a thick border. This formatting should be applied to the post-expansion table, so you will place the code for this after the `expandCalcTable()` function.

1. Right-click anywhere on the table, and select ‘Script’ from the context menu. This opens the Script Editor to edit the element script.

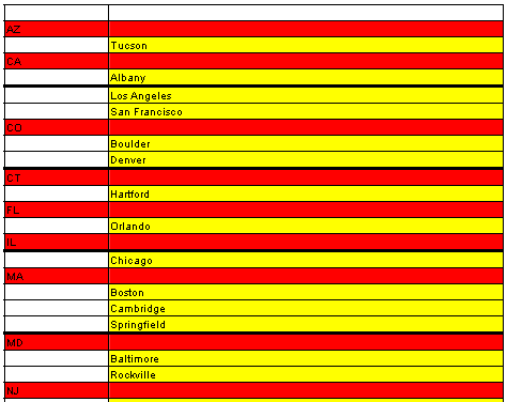
2. Add a “*for*” loop after the `expandCalcTable()` function. The full script should appear as follows:

```
rowBackground[1] = [255,0,0];
cellBackground[2][1] = [255,255,0];
expandCalcTable();

for (i=4; i<table.length; i+=5) {
```

```
    rowBorder[i] = StyleConstant.THICK_LINE;
}
```

3. Preview the result.



3.8 Charts

Charts display grouped and aggregated data in a graphical manner. The 'Properties' and 'Data Binding' dialog boxes for Charts provide many options for customization, including style, formatting, axis information, ranking, sorting, and so on. All of these various options are also accessible through script, as well as many others.

There are two levels at which you can target a Chart script:

- **Chart Data Binding**: Modify properties of Chart's data binding by accessing the `bindingInfo` attribute. See *Modifying Chart Properties* and *Modifying a Chart Data Binding* for more information.
- **Chart Structure**: Modify the primitive objects which define a Chart (Scales, Forms, VisualFrames, GraphElements, etc.) by using low-level Chart API functions. See *Modifying a Chart Element using API Functions* and *Creating a Chart Using API Functions* for more information.

Both methods enable you to modify an existing Chart or build a completely new Chart from scratch. The Chart API provides the highest level of control over chart appearance, but requires a more complex syntax. See *Chart Script Tutorial* for examples.

See Also

Chart Properties, for details on Chart properties.

Binding Data to a Chart Element, in *Advanced Report Design*, for binding details.

3.9 Sections

A Section is a container of other elements. In most cases, you add script to the individual elements *within* the Section, rather than to the Section element itself. However, Section scripts also have some special behaviors, which are discussed below.

See Also
Tables and Sections, for element functions pertaining to Sections.
SI.22, *Section Object*, for functions pertaining to Sections.
Section Element, in *Advanced Report Design*, for general information on Sections.

3.9.1 Accessing Data

When you bind a Section to a data source (i.e., to generate a pseudo-table), each band of the Section is repeated multiple times, corresponding to the number of rows returned by the query. For example, the Content band of the Section is repeated once for each detail record returned by the query.

If you place a script on a component *within* a repeating band, this script is re-executed for every band repetition. This has the effect of wrapping the script in an implicit "for" loop. Within the script, you can reference the values of elements in the same band:

- `value` – the value (in current band iteration) of the element to which the script is attached.
- `field['column_name']` or `field.column_name` – the value (in current band iteration) of the element in the specified field.

Use auto-complete to enter the correct syntax.

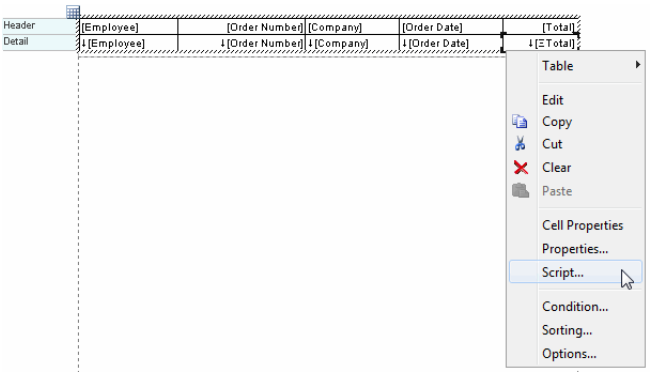
You can also refer to an element in the band by its Element ID (e.g., "`Text1.text`") to obtain the current value. To obtain the row number corresponding to the current band iteration, use the `sectionRow` property.

Example: Referencing Section Data

In the following example, you will read the value of an element in a Section, and use that value to conditionally hide or show another element. Follow the steps below:

1. Create a new report, and add a Section element. Bind the Section to the 'All Sales' query.

2. Right-click the Text element in the 'Total' column content band (detail row), and select 'Script'. This opens the Script Editor to edit the Text element.



3. Enter the following script for the Text element.

```
if (field.Employee != 'Annie') {
  visible=false
}
else {
  visible=true
}
```

The if/else statement checks the value of the 'Employee' field, and hides the current 'Total' element if the employee is other than Annie.

In this example, you may note that the "else" block of the "if/else" statement is not strictly required. However, when you use the above method to set visual properties of the Section, you *will* need to include the "else" block to obtain the desired results. See *Setting Visual Properties* for an example.

¢

3.9.2 Setting Visual Properties

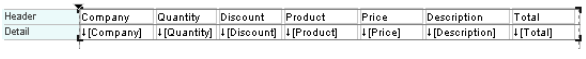
If you place a script on a component *within* a repeating band, this script is re-executed for every band repetition. This has the effect of wrapping the script in an implicit "for" loop. You can take advantage of this repetition to compactly and effectively add visual style to the entire Section.

To access the visual properties of current band, use the '*band*' object.

Walkthrough

In this example, you will modify the Section content band when the query fields meet certain conditions. Follow the steps below:

1. Create a new report, and add a Section element. Bind the Section to the 'Order details' query in the 'Sample Queries' folder of the 'Orders' data source.



2. Select any Text element in the Content band of the Section. Right-click the element, and select ‘Script’ from the context menu. This opens the Script Editor for the Text element.

3. Add a script that hides the current band if the value of the ‘Price’ field is less than 100.

```
if(field['Price'] < 100) {
  band.visible = false;
} else {
  band.visible = true;
}
```

4. Add an additional script that colors the band red if the value of the ‘Quantity’ field is less than or equal to 2.

```
if(field['Quantity'] <= 2) {
  band.background = [255,0,0];
} else {
  band.background = [255,255,255];
}
```

Company	Quantity	Discount	Product	Price	Description	Total
Interstate Shop	6	0	NetStorage	2850	500 GB network storage	17100
Interstate Shop	1	0	Mega Icons	550	Over 10,000 icons	550
Interstate Shop	16	0	WebCalendar	248	Web enabled calendar/org	3968
Interstate Shop	18	0	19 inch LCD	310	19 inch LCD monitor	5580
Interstate Shop	18	0	Sync Me	145	Palm power sync	2610
Interstate Shop	15	0	Math for Me	235	Math tutor	3525
Interstate Shop	1	0	NetStorage	2850	500 GB network storage	2850
Interstate Shop	13	0	Mega Icons	550	Over 10,000 icons	7150
Interstate Shop	2	0	17 Inch LCD	210	17 inch LCD monitor	420
Interstate Shop	1	0	MeToo AppServer	3000	Application server	3000
Interstate Shop	6	0	Web Bridge	105	Bridge game	630

See Also

band, for more information about the band object, and its properties.

3.10 Report Beans

A report bean is a reusable set of report components that can be managed as a single unit. It is a very effective tool for building component-based reports, improving reusability and reducing report development time.

You can configure report beans (without scripting) to expose certain sets of properties to the report developer, and this approach is sufficient for most reports. However, you can also implement reusable report logic by embedding scripts within beans. The following sections discuss issues related to placing script inside report beans.

See Also

Using the Script Library, for another method of reusing report logic.
Report Bean, in *Advanced Report Design*, for examples of bean usage.

3.10.1 Bean Handlers

A report bean is treated as a mini-report. The bean has its own *onLoad Handler* and *onInit Handler*, just like a regular report. However, a bean does *not* have an *onPageBreak Handler* or *onPrint Handler*. All page break events are handled by the parent report.

The bean's *onInit* handler is invoked when the report loads, before the report parameters are prompted. Like the report *onInit* handler, the bean's *onInit* handler is invoked exactly once per instance. If a report bean is used in multiple instances in one report, the *onInit* handler for each instance is executed once.

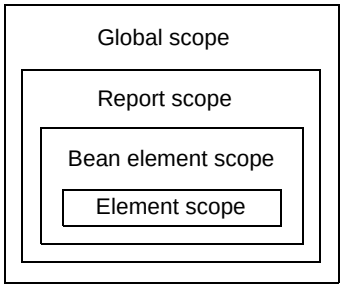
Script that you attach to a bean runs within bean scope. This means that you can have multiple instances of a bean within the same report, and any script reference to an element within the bean is resolved to the instances inside the self-same bean.

See Also
Script Evaluation, for the general order of execution of report scripts.

3.10.2 Bean Scripting Scopes

A bean has its own scripting scope, similar to a report's scope, and all elements within the bean fall within the bean's scope. Therefore, if you place a script on an element within a bean, when the script encounters a symbol (variable, property), the server checks for the reference in the following sequence:

1. Search element scope (i.e., properties defined by the element to which the script is attached).
2. Search the bean "report" scope (i.e., properties defined by the *onInit Handler* and *onLoad Handler* of the bean containing the element).
3. Search the bean "element" scope (i.e., properties defined by element-level script attached to the bean element within the parent report).
4. Search parent report scope.
5. Search global scope.



Accessing Report Elements of Same Name

Because the server searches bean scope first, variables and elements in the bean will *hide* objects of the same name in the containing report. To access such objects in the containing report, use the *ParentReport Object*. For example, if the bean contains a text element with Element ID of 'Title', and the parent report also contains a text element with Element ID of 'Title', you can access the parent report instance as follows:

```
parentReport.Title
   or
parentReport['Title']
```

Dynamically Modifying the Parent Report

Scripts in a bean can dynamically access elements and variables in the parent report scope. For example, you could develop a "toolbar bean" that contains different chart type "buttons". When the user clicks a button, this changes a chart in the parent report to the selected type. To do this, you could add an *onClick Handler* script to each button, such as the following:

```
// script for line chart button
Chart1.chartStyle['Count(Total)'] = Chart.CHART_LINE
```

This bean can then be embedded in any report that has a chart called 'Chart1'. The bean itself contains no chart called 'Chart1'; the server locates the parent report's 'Chart1' at runtime by looking up the name in the scope hierarchy.

Accessing Bean Properties from Parent Report

From within the parent report scope, you can modify the properties of elements located in a bean by using the '*elements*' property of the bean.

For example, consider a report that contains a bean called "Bean1," which in turn contains a text element called "Text1" and a table element called "Table1." To access the properties of these bean elements from with parent report script, using the following syntax.

```
Bean1.elements.Text1.text= 'hello';
Bean1.elements.Table1.table[0][0] = 'Order ID';
```

See Also

Server-Side Features, for an example of reusing bean script across reports.
Accessing Element Properties, for the general scope hierarchy within a report.

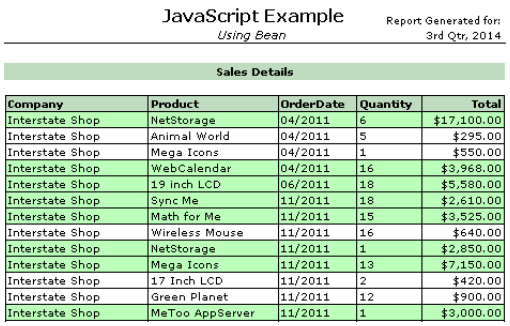
3.10.3 Report Bean Example

This example (`bean.srt` in the `examples/docExamples/script` directory) is based on the table example. It changes two parts of the report into report beans:

```
Note: Set your Repository location to examples/docExamples/script.
See Configuring a Repository in Using Style Studio for
instructions.
```

- The report header section is converted into a bean, and the script for calculating the date range is contained in the bean. This allows the bean to be reused across multiple reports without re-implementing the script.
- The table label and field are changed to a bean. The bean is used to create a total over 1,000 and a total under 1,000.

Figure 3. Report Bean Example (bean.srt)



JavaScript Example
Using Bean
Report Generated for:
3rd Qtr, 2014

Sales Details

Company	Product	OrderDate	Quantity	Total
Interstate Shop	NetStorage	04/2011	6	$17,100.00
Interstate Shop	Animal World	04/2011	5	$295.00
Interstate Shop	Mega Icons	04/2011	1	$550.00
Interstate Shop	WebCalendar	04/2011	16	$3,968.00
Interstate Shop	19 inch LCD	06/2011	18	$5,580.00
Interstate Shop	Sync Me	11/2011	18	$2,610.00
Interstate Shop	Math for Me	11/2011	15	$3,525.00
Interstate Shop	Wireless Mouse	11/2011	16	$640.00
Interstate Shop	NetStorage	11/2011	1	$2,850.00
Interstate Shop	Mega Icons	11/2011	13	$7,150.00
Interstate Shop	17 Inch LCD	11/2011	2	$420.00
Interstate Shop	Green Planet	11/2011	12	$900.00
Interstate Shop	MeToo AppServer	11/2011	1	$3,000.00

Report Header Bean

Walkthrough

No additional design work is required to create a report bean from the existing report header. Simply copy and paste the appropriate elements into a report bean and define the appropriate properties.

1. Copy and paste the title elements into a report bean.



JavaScript Example
Using Bean
Report Generated for:
1st Qtr, 2007

Sales Details

2. Select ‘Bean Definition’ from the Style Studio *Report* menu. This opens the ‘Bean Definition’ dialog box.

3. Add the ‘Text’ property of the subtitle element to the right frame, and enter ‘subtitle’ for the property name.

4. Add the ‘Text’ property of the section title element to the right frame, and enter ‘section title’ for the property name.

The script on the date field remains the same:

```
var now = new Date();

switch(now.getMonth()) {
case 0: case 1: case 2:
   text = "1sr Qtr";
   break;
case 3: case 4: case 5:
   text = "2nd Qtr";
   break;
case 6: case 7: case 8:
   text = "3rd Qtr";
   break;
case 9: case 10: case 11:
   text = "4th Qtr";
   break;
}

text = text + ", " + formatDate(now, 'yyyy');
```

Since the script uses the property of the text element to which it is attached, references are automatically resolved to the current element. To use the bean, simple replace the elements in the report with the bean.

Total Field Bean

Walkthrough

The elements for displaying the two totals are almost identical. You will now replace these elements with a report bean following a similar procedure:

1. Copy and paste the total label and textbox fields into a report bean.

2. Select ‘Bean Definition’ from the Style Studio *Report* menu. This opens the ‘Bean Definition’ dialog box.

3. Add the label and total field ‘Text’ properties to the bean properties in the ‘Bean Definition’ dialog box.

4. Replace the total fields with two ‘total field’ beans.

5. Change the table script to assign the calculation results to the bean

```
var total1 = 0, total2 = 0;

for(var i = 1; i < table.length; i++) {
   var price = table[i]['Total'];

   if(price > 1000) {
      for(var c = 0; c < table[i].length; c++) {
         cellBackground[i][c] = 0xBBFFBB;
      }
```

```
      total1 += price;
   }
   else {
      total2 += price;
   }
}

totalOver.value = formatNumber(total1, '$#,###.00');
totalBelow.value = formatNumber(total2, '$#,###.00');
```

A report bean can include presentation elements, properties, and dynamic behaviors to create a self-contained module.

4 Queries

The following sections explain how to execute a query in script and save the results to an array, and also how to modify an element's data binding to utilize a different query.

4.1 Running a Query

Normally, you will bind a query to a report element using the 'Data Binding' dialog box (or the equivalent property, e.g., *query*). The query that you bind will execute automatically when the report is generated, and the data returned by the query will populate the associated element.

In some situations, however, a report element must contain results from more than one query. For example, a summary table may display summary data generated from different queries. One way to do this is to construct a "master query" that provides all the desired aggregates, and then bind this single query to the element. However, this kind of query might prove very complex and difficult to write.

An easier way to bind multiple queries is to run the individual queries using script. You can then bind results from each distinct query to the individual table cells that you want to display those results. The following sections examine how to do this.

See Also
Formula Tables, for the most common application of multiple query binding.
Data Binding, in *Advanced Report Design*, to bind an element to a query.

4.1.1 Running a Query from Script

If possible, use Style Studio to bind queries to elements. See *Query Performance Considerations*. Note: runQuery does not retrieve data from Data Models.

To run a query or a Data Worksheet, use the `runQuery()` command.

```
// Run a query:
var q = runQuery('total sales');

// Run the query for primary Data Block in a Worksheet:
var q = runQuery('ws:global:path/worksheetName');

// Run the query for primary Data Block in a user-scope
Worksheet:
var q = runQuery('ws:user_name:path/worksheetName');

// Run the query for primary Data Block in a local report-
scope Worksheet:
var q = runQuery('ws::path/worksheetName');
```

To run the query for a *non-primary* Data Block in a Worksheet, simply append the desired table name to the specified string. For example:

```
// Run the query for non-primary Data Block in a Worksheet:
```

```
var q = runQuery('ws:global:path/worksheetName:tableName');

// Run the query for non-primary Data Block in a local
report-scope Worksheet:
var q = runQuery('ws::path/worksheetName:tableName');
```

In all cases, 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, and assign them to a table cell or text element:

```
// Assign first column of first data row to a table cell:
Table1.table[1][1] = q[1][0];
```

Query parameters (if any) can be passed as part of the runQuery() call. For example, to pass the report '`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

Independent Query, in *Advanced Data Modeling*, to design a new query.
Advanced Data Mashup, for information on creating Worksheet assets.

4.1.2 Query Performance Considerations

For performance reasons, you should avoid using too many large queries in one report. In addition, you should only run a query from script in cases where you cannot directly bind the query. In general, it is more efficient to use the 'Data Binding' dialog box to bind the query to an element, which allows the query to automatically run as part of report generation.

There are two complimentary methods for controlling and improving the performance of a report.

- **Size limit**: You can limit the number of rows that the query returns.
- **Time limit**: You can limit the amount of time a query may execute.

For example, if you only use the first few rows of a query, you should set a tight limit on the query size. You can make these settings at the query level or data binding level. See *Advanced Toolbar Buttons* in the *Advanced Data Modeling* to limit at query level, and *Precautions and Safeguards* in the *Advanced Report Design* to limit at binding level.

See Also

Performance Options and Safeguards, in *Advanced Server Administration*, for other limits.

4.1.3 Query Example

Assume that you need to create a table containing sales summary information: Total sales volume, product with highest volume, and customer with the highest volume. There are several possible ways that you can design this table:

- Write a stored procedure to return the three unrelated aggregates as the result of a single query, and bind the table to this query. However, designing such a stored procedure is awkward and requires advanced knowledge of database programming. Therefore, this approach is not recommended.
- Use a Data Worksheet to compute the three desired aggregates, and merge these into a single Data Block. Then bind the table to this Data Block. (See the *Advanced Data Mashup* for details).
- Design three simple queries to return the three desired aggregates. Use script to independently run the three queries, and then explicitly assign the results to corresponding table cells.

Walkthrough

The following example illustrates the third approach. Assume that you have defined the following three simple queries in Style Studio:

- `'total sales'` – returns total sales volume
- `'top product'` – returns product with highest volume
- `'top customer'` – returns customer with the highest volume

Follow the steps below to create the table that displays these values:

1. Create a new table with three rows and two columns.
2. Select each of the three cells in the left column, and enter the following strings into the cells:
   a. “Total Sales”
   b. “Top selling product”
   c. “Highest Volume Customer”
3. Right-click the table, and select ‘Script’ from the context menu. This opens the Script Editor for the table.
4. Enter the following script. This runs each query in turn, and assigns the result to the corresponding table cell.

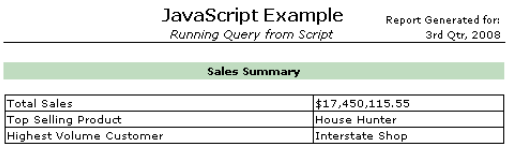
```
var sales = runQuery('total sales');
var product = runQuery('top product');
var customer = runQuery('top customer');
```

```
table[0][1] = formatNumber(sales[1][0], "$#,###.00");
table[1][1] = product[1][0];
table[2][1] = customer[1][0];
```

Note: Set your Repository location to *examples/docExamples/script*. See *Configuring a Repository* in *Using Style Studio* for instructions.

See the `summary.srt` report in the `examples/docExamples/script` directory for a similar example.

Figure 4. Query Example (summary.srt)



See Also

Data Tables, for further information about scripting tables.

Table Element, in *Advanced Report Design*, for information on creating a table.

Formula Tables, for more advanced use of query data in tables.

4.2 Binding Queries

You can dynamically change the data binding of an element from within script by setting the "`query`" property.

```
if(parameter['type'] == 'Details') {
   Table1.query = 'sales details';
}
else {
   Table1.query = 'sales summary';
}
```

You should only change the "query" property from within the *onInit Handler* or *onLoad Handler*.

Query Compatibility

The data binding of an element specifies information other than just the query name, such as grouping and summary fields, and formula columns. If you change a query from within script, you must make sure that the new query is compatible with these other specifications.

For Table and Chart elements, a query is considered compatible if it contains the same grouping and summary columns specified in data binding. Since the table can adjust itself to accommodate any tabular data, no change in table layout is necessary.

Because Section elements have fixed layout, the compatibility rules are stricter. If you use script to change the query binding for a Section, the new query must contain exactly the same columns as the query that it replaces. Otherwise, the Section will not display the new columns and will use default values for all columns that are missing in the new query.

In practice, dynamic query binding is most useful when your reports have tables with the exact same columns, but with different conditions or data sources. Keep in mind however, that *Report Bean* are the preferred way to achieve component-based reporting and report reuse.

See Also

Binding Data to a Table Element, in *Advanced Report Design*.
Binding Data to a Chart Element, in *Advanced Report Design*.
Binding Data to a Section Element, in *Advanced Report Design*.
Running a Query, for information on executing a query without binding.

5 Report Handlers

Previous chapters explained how you can add scripts to various report elements (Tables, Charts, etc.) to modify those elements' properties. These element-level script are executed when the element is processed in the course of report generation.

The current chapter explains how you can use various *report handlers* to implement advanced business logic. This includes adding actions to report elements, and modifying report generation based on user input or the runtime environment.

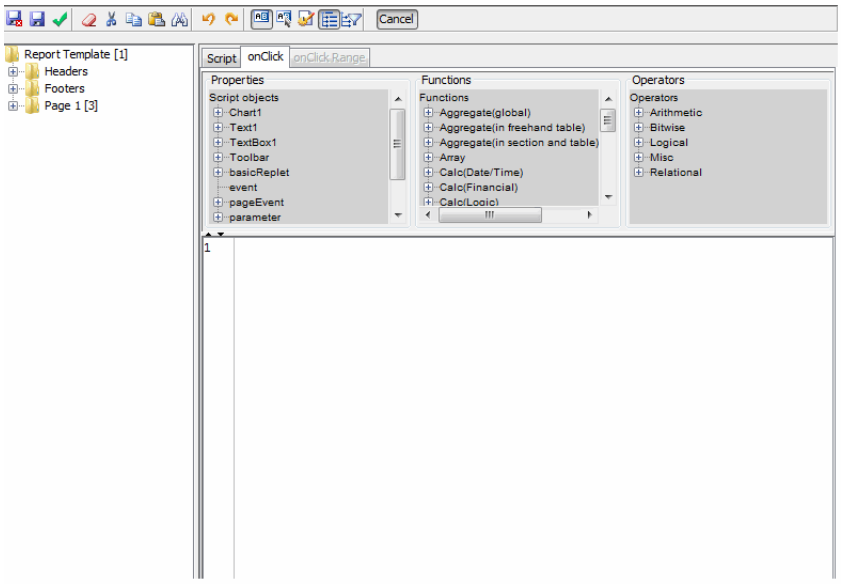
See Also
Server-Side Features, for information on interactivity (drilldown, parameters, etc.)

5.1 onClick Handler

An element's *onClick* handler is executed when a "selection" event (i.e., mouse click) takes place on the element. You can use the *onClick* handler to implement interactions such as hyperlinks and drilldowns, or to initiate any other business logic in response to user selections.

To access the *onClick* handler for an element, right-click the element, and select 'Script' from the context menu. Then select the **onClick** tab at the top of the Script Editor.

Figure 5. The onClick tab of the Script Editor



5.1.1 Using the onClick Handler

For simple hyperlinks, use the 'Hyperlink' dialog box: See *Hyperlinks* in the *Advanced Report Design*.

The *onClick* handler is executed on the report server, as are all scripts. Because the handler does not execute on the client browser, it cannot perform client-side actions directly. Instead, the handler controls the behavior of the report by returning one of the pre-defined viewer actions. Actions related to hyperlinks are `showReplet()` and `showURL()`. Other viewer actions are discussed in a later chapter.

Note: The viewer action functions (`showReplet`, `sendRequest`, etc.) should be the final statement of the onClick script.

The *onClick* handler is primarily used for Tables and Text elements, in situations where you cannot completely define the hyperlinks at design time. For example, if a hyperlink is required to pass parameters that might change as a result of user interactions, you can compute the parameter values in the *onClick* handler, and then create the hyperlink by using the `showReplet()` action.

For example, consider the following *onClick* script attached to an element. When the user clicks the element, the script tests a condition, and then loads one of two possible reports.

```
if(condition) {
   showReplet("order info",[["start_date",CALC.today()]]);
}
else {
   showReplet("customer info",[["state", "NJ"]]);
}
```

See Also

Style Intelligence Global Object, for a list of available viewer functions.

5.1.2 Setting the onClick Range

For a Table, the *onClick range* specifies the range of cells for which the *onClick* script is active. To set the *onClick range* for an element, right-click the element, and select 'Script' from the context menu. In the Script Editor, select the **onClick Range** tab. The options for the *onClick range* are as follows:

- All rows
- All columns
- Specific column
- Header row
- Trailer row
- Header column
- Trailer column

It is very common to pass the value in the clicked cell as a parameter in the hyperlink. For example, the user clicks a state name in the 'State' column, and you want to pass this clicked value to the drill-down report. To obtain the clicked value, first find the row and column indices of the cell by using the `event.getRow()` and `event.getColumn()` functions.

```
var rowIx = event.getRow();
var colIx = event.getColumn();
```

Then use these indices with the Table's *table* property to obtain the data value.

```
showReplet("customers",[["state",table[rowIx][colIx]]]);
```

5.2 onInit Handler

A report's *onInit* script is executed only once, before the report is processed. Because all parameter prompting and automatic query execution occurs after the *onInit* script executes, *onInit* script is ideal for one-time initialization tasks. The typical usages are as follows:

- **Defining report parameters**: Because all other scripts execute *after* parameter prompting, parameter definitions can only be changed from the *onInit* script.
- **Defining report parameter default values**: You can manually run a query to supply the default values. (See *Running a Query* for details.)

```
Note: A variable declared in onInit script will have global scope.
To declare a local variable with the same name elsewhere, use the
keyword 'var' in the declaration to remove ambiguity.
```

- Creating session-level variables.
  For example, you can use the *onInit* script to perform all one-time calculations, then store the results in global variables so that you can access them from other scripts.

See Also

onLoad Handler, to declare report-level variables, or validate user parameters.

5.3 onLoad Handler

The *onLoad* handler is similar to the *onInit* handler, and is also executed at the beginning of report generation. It differs from *onInit* in two important ways:

- *onLoad* script is executed every time a report is processed.
- *onLoad* script is executed *after* report parameter prompting.

The typical usages of the *onLoad* handler are the following:

- Declaring report-level variables.
  For example, to keep a subtotal on each page, declare the 'subtotal' variable in the *onLoad* script and then update it using *onPageBreak Handler* script.
- Initializing the report based on user input parameters.
  For example, *onLoad* script can set chart styles, report headers, element visibility, etc. The *onLoad* handler has access to the '`parameter`' array that contains all report parameter values. For example, to hide a chart if a parameter is false:

```
if(!parameter['showChart']) {
   Chart1.visible = false;
}
```

- Dynamically running queries.
  An element's 'query' property can only be set in the *onLoad* handler, not in element-level script. See *Binding Queries* for details.
- Modifying binding characteristics (column visibility, grouping and summarization, etc.) using the element's `bindingInfo` attributes.
- Modifying multiple elements from a central location.

See Also

The Table bindingInfo Property, for information on data binding for Tables.

Chart bindingInfo Property, for information on data binding for Charts.

The Section bindingInfo Property, for information on data binding for Sections.

onInit Handler, to declare session-level variables or report parameters.

5.4 onPageBreak Handler

The *onPageBreak* handler is executed at the end of every page, after all elements on the page have been processed (except for headers and footers). Because the *onPageBreak* handler follows body-element processing, it should not modify the body contents on the current page. It can, however, modify contents in the header and footer of the current page.

The main use of *onPageBreak* is to calculate page-level summarization. It is also useful in cases where you wish to display page numbering in the body of the report. You can use the onPageBreak handler to increment a "page number" variable, and then reference that variable in the desired elements.

See Also

Headers and Footers, in *Advanced Report Design*, to implement page numbering without scripting.

The OnPageBreak Event Handler

A special variable, ‘event’, is available in the *onPageBreak* handler. This object contains information about what is on the current page. It has the following properties:

Table 3. OnPageBreak Event Handler Properties

PROPERTY	DESCRIPTION
elementID	The ID of the last element on the current page.
region	A rectangle containing the last table region on the page: *region.x* is the left-most column number, *region.y* is the top row number, *region.width* is the number of columns, *region.height* is the number of rows. This property is only defined if the last element on the page is a table element.
firstElementID	The ID of the first element on the current page.
firstRegion	A rectangle containing the first table region on the page. This property is only defined if the first element is a table element.
lastPage	True if this is the last page of the report.
rewound	True if the last band on the page has been rewound.
pageIndex	The current page index (first page = 0)

Example: onPageBreak Handler

The following *onPageBreak* script displays a Text element (in the header) that says “continued…” if the current page is a continuation of a table from the previous page. The Text element has ID of “`continuteLabel`”.

```
if(event.elementID == 'Table1' && event.region.y > 1) {
  continueLabel.visible = true;
}
else {
  continueLabel.visible = false;
}
```

¢

See Also
SI.30, *PageBreak Event Object*, for more detail on handler properties.

The rewound property

The “`rewound`” property has special significance in Section scripting. When a Section is printed on a page, the bands are repeated for each row of the table. If a band goes outside of the boundary of the page, the band is “rewound” and printed on the next page.

Normally this does not cause a problem, because when the band is rewound, it does not re-execute any associated scripts. The only time this is an issue is when the calculation of elements in a band is used in the page header and footer. For example, if a script is attached to an element in the band for calculating a running total on the page, the total would contain an extra value from the next page.

To correct this problem, you need to handle rewinding for scripts in the *onPageBreak* handler. First, check if the `rewound` property is set. If it is, the script undoes the effect of the previous script and pushes the value to the next page.

```
// assume the band script saves the last value added to the
total
// in a report variable 'lastValue'
if(event.rewound) {
   // assume total is the report variable used to hold the
page total
   total -= lastValue;
}

// Text1 is the text element on footer for displaying the
page total
Text1.text = total;

// reset page total for next page
if(event.rewound) {
   total = lastValue;
}
else {
   total = 0;
}
```

5.5 onPrint Handler

The *onPrint* handler is executed before a report is printed or exported. The typical usages of onPrint handlers are as follows:

Note: The best way to do this is to set 'Hide on Print and Export'. See *Property Dialog Boxes* in the *Advanced Report Design* for details.

- Hiding non-printable elements.
- Including a title page in a report
- Performing logic specific to printing and exporting.

The *onPrint* event handler offers a special property, '`report.format`', through the 'docInfo' object.

```
docInfo['report.format']
```

When the user exports a report to a specific format, this property holds the file extension of the specified format (e.g., `xls`, `html`, `rtf`, `txt`, etc.) This allows you to implement different logic based on the selected export format.

5.6 Report Handler Example

In this example (`handlers.srt` in the `examples/docExamples/script` directory) you will use report-level scripts to calculate the per-page total of sales over and below $1,000. It uses handlers to perform the following tasks:

Note: Set your Repository location to *examples/docExamples/script*. See *Configuring a Repository* in *Using Style Studio* for instructions.

- In the page footer, display the sales totals for the current page over and below $1,000.
- Highlight the rows with sales over $1,000.

Figure 6. Report Handler Example (handlers.srt)



AMG Logistics	Dec 19, 2010	7	$1,470.00
AMG Logistics	Sep 23, 2010	17	$3,570.00
AMG Logistics	Aug 14, 2009	15	$3,150.00
AMG Logistics	Jun 29, 2009	10	$2,099.90
Northwest Outfitters	May 23, 2012	4	$840.00
Northwest Outfitters	Apr 15, 2012	3	$630.00
Northwest Outfitters	Feb 08, 2012	3	$629.75
Northwest Outfitters	Jan 09, 2012	11	$2,310.00
Northwest Outfitters	Oct 27, 2009	18	$3,779.95
Northwest Outfitters	Jun 27, 2009	7	$1,470.00
Tricounty Auto	Feb 01, 2012	10	$2,099.90
Tricounty Auto	Feb 01, 2011	15	$3,150.00
Tricounty Auto	Dec 04, 2010	19	$3,990.00
Tricounty Auto	Nov 08, 2010	12	$2,520.00
Tricounty Auto	Oct 21, 2010	19	$3,990.00
	Total Sales from Orders Over $1000:		$73,289.65
	Total Sales from Orders Below $1000:		$4,619.75

5.6.1 Calculate Page Total

Calculating the page total itself is very simple. First declare two report-level variables in the *onLoad Handler*, `total1` and `total2`. These will be used to store the totals over 1000 and below 1000 on the current page.

```
var total1 = 0; // for total over 1000
var total2 = 0; // for total under 1000
```

Next, add a script to the total field (Text) in the Content band to add the total value to the two total variables, and to display the appropriate highlighting.

```
// calculate page total and highlight
if(value > 1000) {
   total1 += value;
   band.background = [0,255,0]; // green
}
else {
   total2 += value;
   band.background = [255,255,255]; // white
}
```

Finally, add an *onPageBreak Handler* script to display the two totals in the footer, and to reset the total variables.

```
// assign to footer text element
totalOver.text = formatNumber(total1, "$#,##0.00");
totalBelow.text = formatNumber(total2, "$#,##0.00");

// reset totals
total1 = 0;
total2 = 0;
```

5.6.2 Band Rewinding

One complication of page-level calculation is the fact that a band can be rewound to a new page during printing. Therefore, if you leave the report as it currently is, the page total may contain an extra value from the next page. To address this, you will add logic to handle the rewinding of a band.

To “undo” the effect of rewinding, you first need to determine what the last band did before it was rewound. In the *onPageBreak Handler* handler, you can check if the last band was rewound, and undo the last action. For this example, declare a report-level variable in the *onLoad Handler* to hold the last value added to the total variables.

```
var lastValue = 0;
```

In the script for the total field, simply store the last number added to the total in the variable.

```
lastValue = value;
```

Finally, add code to the *onPageBreak* script to determine if rewinding occurred, and to undo its effect.

```
// if rewound, undo the last value total
if(event.rewound) {
   if(lastValue > 1000) {
      total1 -= lastValue;
   }
   else {
      total2 -= lastValue;
   }
}
```

Since you don’t want to lose the last value for the next page, add another block at the end of the *onPageBreak* to push the `lastValue` to the next page’s totals.

```
// if rewound, push last value to next page
if(event.rewound) {
   if(lastValue > 1000) {
      total1 = lastValue;
   }
   else {
      total2 = lastValue;
   }
}
```

6 Server-Side Features

Note: Features covered in this chapter are not available in Style Report Pro.

Previous chapters discussed scripts for controlling report contents, performing calculations, and other presentation-oriented tasks. Another class of scripts deals with user interactions. By adding scripts that respond to user actions, you can turn a report from a static presentation into an interactive experince. All user interactions are controlled by the report server.

6.1 Accessing Report/Request Parameters

You can access all parameters defined in a report by using the *parameter* object. Two forms of syntax are available:

1. `parameter.{parameter name}`

   ```
   e.g.,
   parameter.start_date = CALC.today();
   Table1.visible = parameter.table_visible;
   ```

2. `parameter['{parameter name}']`

   ```
   e.g.,
   parameter['StartDate'] = CALC.today();
   Table1.visible = parameter['table_visible'];
   ```

6.2 HTTP Request, Session, and Principal

When you use the servlet-based report server (Repository Servlet) you can access the http request and session objects via script. Use the *parameter* array to access the *HttpServiceRequest/HttpServiceResponse* object, which are wrappers, respectively, for the *HttpServletRequest*/*HttpServletResponse* objects.

```
Note: Although you can set HttpSession properties directly, the it
is best to set them on the SRPrincipal. SRPrincipal can be accessed
universally within the script of a report or a VPM trigger.
```

For example, the script below handles the http session attributes and request parameters via the HttpServiceRequest object.

```
//accessing the service request object
var serReq = parameter['__service_request__'];

// getting session attributes
var sessAttr = serReq.getAttribute('attribute_name');

// setting session attributes
serReq.setAttribute('attribute_name', value_object);

// setting HTTP request parameter
serReq.setParameter('param_name', 'param_value');

// getting the SRPrincipal object
```

```
var p = parameter['__principal__'];

// getting a standard SRPrincipal property, user locale:
var locale = p.getLocale();

// getting a custom principal property
var prop = p.getProperty("myprop");
```

To get a direct handle on the HttpServletRequest object, use the following syntax:

```
var httpReq = serReq.getRequest();
```

Use the following simplified syntax to acquire the current username:

```
var userName = parameter['_USER_'];
```

See Also

Changing the Server Type, in *Advanced Server Administration*, to configure servlet.

Accessing SRPrincipal via Login Listener, in *Advanced Server Integration*, for the recommended approach.

RepletParameters Object, for a complete list of special parameters.

6.3 Interactive Forms

Style Intelligence supports data entry and report customization through interactive forms. A form is a collection of interactive controls, such as Text fields, Radio Buttons, and Combo Boxes. Interactive forms are used primarily in two roles: Embedded in reports to allow report customization, and embedded in parameter sheets to allow parameter entry.

See the *Parameterization* section of *Advanced Report Design* for the basic methods of obtaining user input (without scripting). The following sections here expand on some of the more advanced features available for user interaction. There are two basic aspects:

- *Creating a Form*: Client-side presentation and event scripting
- *Processing a Form*: Server side scripting

See Also

Parameterization, in *Advanced Report Design*, for automatic user prompting.

Parameter Sheets, in *Advanced Report Design*, for common form deployment.

6.3.1 Creating a Form

There are two steps in creating a form: Adding form controls, and adding client-side script.

Adding Form Controls

The first step in creating an interactive form is to add form controls to a report or parameter sheet. (These form elements are represented as basic HTML form elements when the report is rendered in the User Portal.) Form control elements all share two properties:

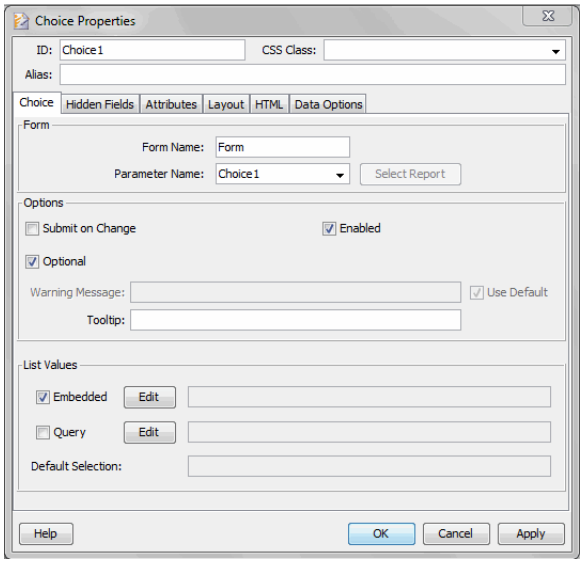
- 'Form Name'
- 'Field Name'

Controls with the same form name are grouped into a single form. To add form elements, follow the steps below:

1. Open the Style Studio **Toolbox** panel.
2. Drag a form element from the **Toolbox** panel into the report.
3. Right-click the new report element and select 'Properties' from the context menu.
4. Enter a 'Form Name' for the element. Use the same form name for all elements belonging to the same form.

```
The specified 'Field Name' is assigned as the name of the HTML
control in the Portal, and is used by client-side and server-side
script to access the control's value.
```

5. Enter a unique 'Parameter Name'.

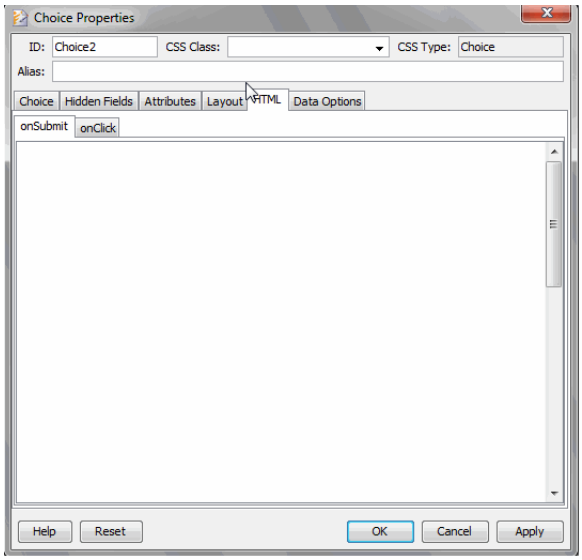


For Radio Buttons, use the same 'Parameter Name' for buttons in the same Radio Button group. The group is then treated as a single

field, with the value of the group being the value of the selected Radio Button.

6. (Optional) Click the **HTML** tab to open the client-side JavaScript editor.

7. In the **onSubmit** and **onClick** tabs, enter JavaScript commands to be executed on the client side when an 'onSubmit' or 'onClick' event occurs in the browser. See *Client-Side JavaScript* below.



8. Repeat the above steps to add the desired controls, including at least one button or image button to use for submitting the form.

See Also

Parameter Sheets, in *Advanced Report Design*, for example of form usage.
Form Design, in *Advanced Report Design*, for information about the various form controls.

Client-Side JavaScript

Script that you enter in the **onSubmit** and **onClick** tabs (see *Adding Form Controls*) is standard JavaScript that is executed by the client browser when the corresponding event occurs.

Within the context of these scripts, you can use the '`Form`' variable to reference all elements on the form. For example, if the form contains a Choice element with field name '`Choice1`' and a TextField element with field name '`TextField1`', you can add an **onSubmit** script on the `Choice1` element to set the value of `TextField1`:

```
Form.TextField1.value = Form.Choice1.value;
return true;
```

This script executes whenever a "submit" action occurs in the form (triggered by any form element). Although you can attach the 'onSubmit' script to any element, a given form should contain only a single 'onSubmit' script. To assign a submit action to a form element (other than a Button), enable the 'Submit on Change' option in the element's 'Properties' dialog box.

If the 'onSubmit' script returns "`true`," the form data is sent on to the server; otherwise, it is not sent. This allows you to use client-side script to verify input values before the form is sent to the server. For example, the 'onSubmit' script below verifies the value entered in `TextField1`:

```
if (Form.TextField1.value < 10) {
  return true;   //send to server
}
else {
  alert('Maximum value exceeded.');
  return false;   //do not send to server
}
```

Storing JavaScript in an External File

You can store client-side JavaScript function definitions in an external file. This JavaScript file must reside in the SREE Home directory (`WEB-INF/classes`, by default) or in one of its subdirectories, and must be registered by setting the '`replet.custom.js`' property in the `sree.properties` properties. For example, if the function definition file is named '`JSFunctions.js`' and resides in the '`WEB-INF/classes/JS`' directory, register the file in `sree.properties` as follows:

```
replet.custom.js = JS/JSFunctions.js
```

The JavaScript file is imported into the report when the report loads so that functions defined in the file are accessible to client-side scripts (i.e., scripts in the **HTML** tab of Form elements).

6.3.2 Processing a Form

A form is submitted to the server whenever a "submit" action occurs on the form (e.g., 'Submit' button is pressed or an 'onSubmit' script returns "true"). The submission contains a single *repletRequest* object that includes all form values.

Scripting the onClick Handler

To process the form on the server end, add script to the *onClick Handler* handler of the Form elements. To do this, open the Form in Style Studio, and follow the steps below:

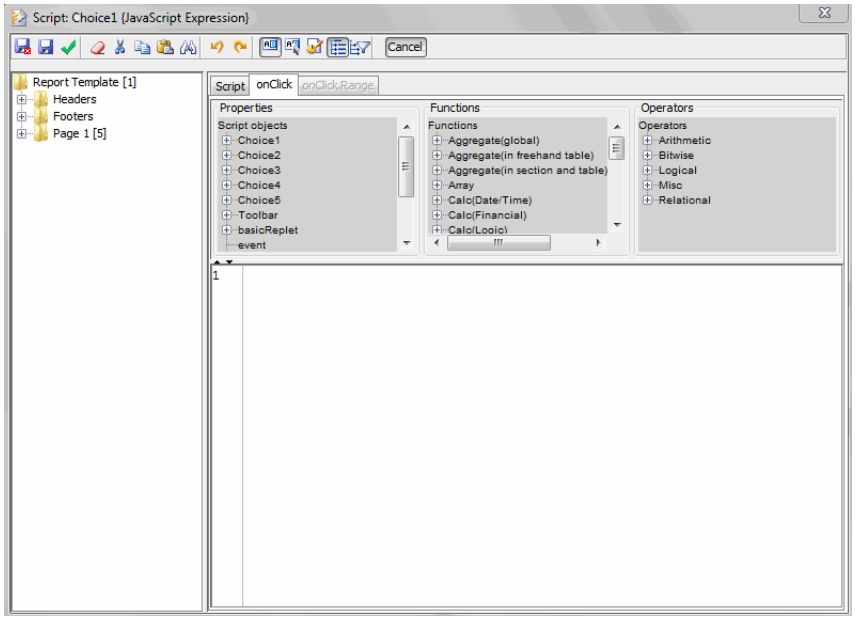
1. Select a form element to which you want to add an 'onClick' script.

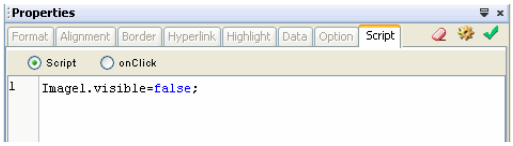
Note: This is the *server-side* 'onClick' handler, not the client-side 'onClick' handler described in *Creating a Form*. You can also edit the server-side 'onClick' handler from the Style Studio **Properties** panel.

You can add 'onClick' script to any Form element. (If you add 'onClick' scripts to multiple elements, all of these scripts are executed when the form is submitted, and all have access to the same set of form values.)

2. Right-click the Form element, and select 'Script' from the context menu. This opens the Script Editor.

3. Select the **onClick** tab at the top of the Editor.





4. Enter the commands to be executed by the report server when the form is submitted. See *Server-Side JavaScript* below for more details.

Server-Side JavaScript

The server-side 'onClick' script has access to a *RepletRequest Object* containing all form element values. You can access a particular field value by using the element 'Field Name' as index:

```
request['FieldName']   or   request.FieldName
```

Field names must be unique within a form (with the exception of Radio Buttons).

If there are multiple "submitting" form elements, use 'request.__eventSource__' to obtain the field name of the element that triggered the submit action. For example:

```
if(request.__eventSource__ == 'Choice1') {
   ...
}
else if(request.__eventSource__ == 'Button1') {
   ...
}
else {
   ...
}
```

Note: If the 'onClick' script of a form element on a Parameter Sheet calls the sendRequest() function, the submission from sendRequest() is sent in place of the normal form submission request.

If your script modifies element values and needs to refresh the screen, it should call the reprint() or refresh() function. The 'reprint' action causes the report pages to be regenerated without refreshing the data binding. The 'refresh' action forces the report data to be refreshed before the report is reprinted. If the *onClick Handler* does not change the data or the way the data is bound to report elements, the reprint action is sufficient.

Example: Server -Side JavaScript

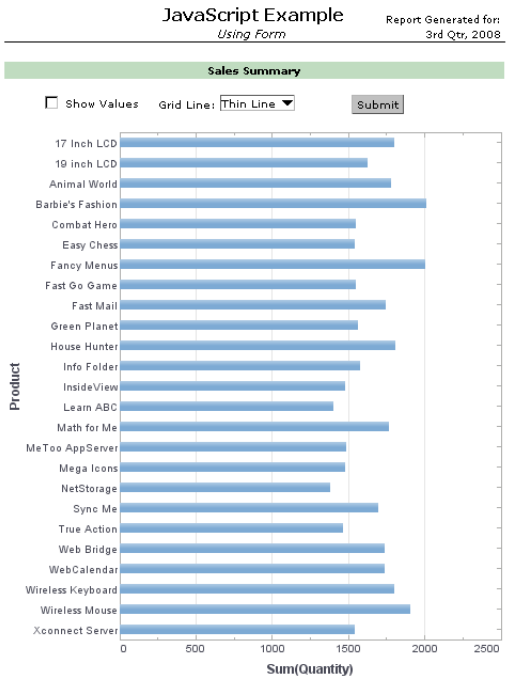
The following sample code is taken from the *form.srt* report in the *examples/docExamples/script* directory. The script checks various form field values, and sets report characteristics appropriately.

Note: Set your Repository location to *examples/docExamples/script*. See *Configuring a Repository* in *Using Style Studio* for instructions.

```
if(request['values']) {
  Chart1['value.visible'] = true;
}
else {
  Chart1['value.visible'] = false;
}

switch(request['hGrid']) {
  case '(none)':
    Chart1.yGridStyle = Chart.NONE;
    break;
  case 'Thin Line':
    Chart1.yGridStyle = Chart.THIN_LINE;
    break;
  case 'Dot Line':
    Chart1.yGridStyle = Chart.DOT_LINE;
    break;
}
```

Figure 7. Form Example (form.srt)



¢

APPENDIX SI: Style Intelligence Object Reference

This appendix describes all objects in the Style Intelligence host reporting environment.

See Also

Common Reference, for general JavaScript and charting functions.
Advanced Dashboard Scripting, for functions specific to the Viewsheet environment.

SI.1 Global Functions

Style Intelligence provides a few global functions in addition to those defined by the standard JavaScript runtime (see Appendix JS:*General JavaScript Functions*). This section presents the additional functions.

ADDPARAMETER(NAME, DEFAULT, TYPE, ALIAS, HIDDEN)

Adds a parameter to the report. This function is available only within the *onInit Handler*, and is equivalent to defining a report parameter using the 'Parameter Definition' dialog box in Style Studio.

Parameters

name	name of the parameter (String)
default	parameter default value
type	an *XType Data Types* constant
alias	label to use when prompting (String)
hidden	Boolean: false (default): prompt user true: do not prompt user

Example

```
addParameter('start_time', new Date(), XType.DATE,
   'Start date', false);
```

The '`hidden`' flag allows you to suppress user prompting for the parameter, equivalent to deselecting the 'Prompt User' option in the 'Parameter Definition' dialog box.

See Also

Defining and Editing Report Parameters, in *Advanced Report Design*, for the parameter interface.

NEWINSTANCE(NAME)

Creates a new Java object.

Parameters

name	name of the class to instantiate (String)

Example

```
var presenter =
newInstance('inetsoft.report.painter.IconCounterPresenter')
;
```

You can usually accomplish the same thing using the 'new' operator, but this sometimes does not handle Java packages correctly.

GETIMAGE(STRING)

Loads an image file from a Java resource or string. The `getImage()` method 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).

Parameters

string	image file resource path or URL or ascii hex/ascii85 encoded gif/jpeg image data

Example

```
var img = getImage('/com/mypackage/icon.gif');
img = getImage('http://visualizefree.com/images/
inetsoft.png');
```

Example

```
// picture column is ascii85 encoded gif image
for(var r = 1; r < table.length; r++) {
   var img = getImage(table[r]['picture']);
   table[r]['picture'] = img;
}
```

See Also

Displaying Images in Table Cells, for a common application of getImage().

ISNULL(OBJECT)

Tests for null value.

Parameters

object	a report object

Example

```
if(isNull(Text1.text)) {
  // perform action when text for element Text1 is null
}
```

DOCINFO(STRING)

Returns information specified in the **Document Info** tab of the 'Report Properties' dialog box. The fields include the following:

- `report.title`
- `report.subject`
- `report.author`
- `report.keywords`
- `report.comments`
- `report.created`
- `report.modified`
- `report.format`
- `report.format` (available in *onPrint Handler* only).

Parameters

string	String containing document property

Example

```
var text = docInfo['report.title'];
```

See Also

Report Properties, in *Advanced Report Design*, to set document information.

LOG(STRING)

Prints a log message to the server log. If the script is running inside Style Studio, the log message is displayed in the Console panel.

Parameters

string	log message string

Example

```
log('onLoaded called');
```

See Also

Using the Console, for information about the Console panel.
Use log() to View Diagnostic Messages, for a common debugging application.

RUNQUERY(QUERY_NAME [,PARAMETERS])

Returns a result set as a two-dimensional array that can be assigned to a Table, Chart, or Section.

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, for common applications of runQuery().

SI.2 Element Object

The *Element* object is the base object of all other report objects. All other objects share its methods and properties.

SI.2.1 General Properties

This section lists general properties of the *Element* object.

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 Property, for more information about setting alignments.

BACKGROUND

Specifies the background color that fills the entire area of the element.

Type

```
java.awt.Color
```

Example

```
background = java.awt.Color.lightGray;
```

See Also

Color Property, for more information about setting color.

FONT

Specifies the font for this element.

Type
`java.awt.Font`

Example
```
font = new java.awt.Font("Arial", java.awt.Font.BOLD, 12);
```

See Also
Font Property, for more information about setting font.

FOREGROUND

Specifies the foreground (text) color of the element.

Type
`java.awt.Color`

Example
```
foreground = java.awt.Color.red;
```

See Also
Color Property, for more information about setting color.

INDENT

Specifies the indentation (in inches) if the element is the first on the line.

Type
`double`

Example
```
indent = 0.5;
```

KEEPWITHNEXT

Specifies that the element should be kept on the same page as the next element.

Type
`Boolean`

Example
```
keepWithNext = true;
```

SCHEDULEACTION(BOOLEAN[,EMAILS])

Specifies whether the report should be executed as part of a scheduled task. Set to `false` (for any element in the report) to override the scheduled task setting and suppress report execution.

Parameters

Boolean	true (default): execute task false: do not execute task

emails	Array of emails

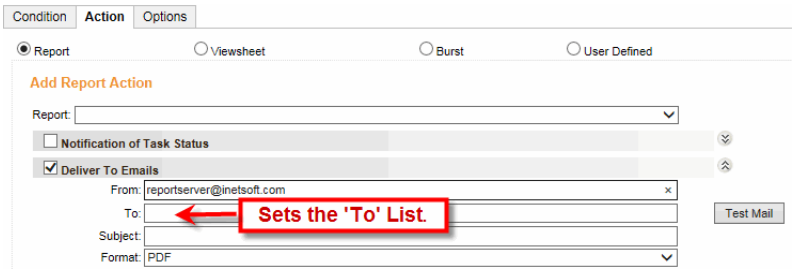
You can use the `scheduleAction` function to conditionalize the execution of a scheduled report task based on actual report data. For example, the following script uses the value in a text element to determine whether the report task should execute as scheduled.

Example

```
if (Text1.value > 10000) {
  Text1.scheduleAction(true)
}
else {
  Text1.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 'Text1' component exceeds the specified 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 (Text1.value > 10000) {
  emails = ['joe@inetsoft.com', 'sue@inetsoft.com',
'mark@inetsoft.com'];
}
else {
  emails = ['joe@inetsoft.com'];
}
Text1.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.
Scheduler, in *Advanced Server Administration*, for information on administrator scheduling.

Scheduling Dashboards and Reports, in *Using the Portal*, for information on end-user scheduling.

SECTIONID

The ID of the Section containing the element. If the element is not in a Section, then `sectionID` is undefined.

Type
`string`

Example
```
var table1 = report['sectionID'].table;
```

See Also
Sections, for scripting issues related to Sections.

SECTIONLEVEL

If the element is contained in a grouped Section, returns the grouping level of the element's sub-group. Otherwise, `sectionLevel` is undefined.

Type
`integer`

Example
```
var level = sectionLevel;
```

See Also
Sections, for scripting issues related to Sections.

SECTIONROW

The index of the Section row with which the element is associated. If the element is not in a Section, then `sectionRow` is undefined.

Type
`integer`

Example
```
var table1 = report['sectionID'].table;
var row = table1[sectionRow];
```

See Also
Sections, for scripting issues related to Sections.

SECTIONTYPE

The type of Section band that contains the element. The type values are '`Header`', '`Content`', and '`Footer`'. If the element is not in a Section, then `sectionType` is undefined

Type
```
string
```

Example
```
if(sectionType == "Footer") {
  ...
}
```

See Also
Sections, for scripting issues related to Sections.

SPACING

Specifies the line spacing (points) for a Text element.

Type
```
integer
```

Example
```
spacing = 1;
```

See Also
Text and TextBox, for scripting issues related to text.

TARGET

Specifies a target (anchor for a hyperlink) on this element.

Type
```
string
```

Example
```
target = text;
```

See Also
Target Property, for information about directing hyperlinks to a target.

VISIBLE

Specifies the visibility of the element. This property can be used to dynamically hide elements in a report.

Type
```
Boolean
```

Example
```
visible = parseInt(text) > 0;
```

See Also
Visibility Property, for more information and examples.

VALUE

Returns the integer value of an element within a Section.

Type
```
integer
```

Example
```
if(text54.value < 100) {
   ...
}
```

See Also
Sections, for scripting issues related to Sections.

POSITION

Specifies the position of an element within a Section.

Type
```
number array
```

Example
```
position = [80, position.y];
```

See Also
Sections, for scripting issues related to Sections.

FIELD

Returns the current value bound to an element within a Section. The `field` property uses the name of the bound data *field* (rather than the element's ID) to refer to the element's value. If the element that references the `field` array is not within a Section, then the `field` property is undefined.

Modifications to the field array have no effect on elements.

The `field` array provides read-only access to the values of *all* bound elements in the current band/row, and is accessible from every element inside the band.

The `field` array provides the *original* data bound to the element, not the string representation provide by the *Text Property*. Therefor, if a column has type Date, the `field` array returns a Date object.

Return
```
original value bound to an element in the Section
```

Example
```
field['customer_id'] + ':' + field['quantity'];
```

See Also
Text Property, for an alternative method of referring to text contents.

Formatting a Date, for information on formatting Date-type data as a string.
Sections, for scripting issues related to Sections.

HINTS

Specifies that a column added to a Section using script is included when the report is exported in CSV format. The value you specify for the property becomes the name of the column in the exported file.

Example

```
hints['csv.column'] = 'NewColumn';
```

This property must be set in the Section element's script, *not* in the script of an element *within* the Section.

See Also

Element Script, for information about adding script to report elements.
Sections, for scripting issues related to Sections.

SI.2.2 Tables and Sections

This section presents functions for Table and Section operations.

See Also

SI.22, *Section Object*, for additional functions pertaining to Sections.
Data Tables, for scripting issues related to Tables.
Formula Tables, for scripting issues related to Formula Tables.
Sections, for scripting issues related to Sections.

AVERAGE(COLUMN[, GROUP[,CONDITION]])

Calculates the average of values in the specified column inside a group. This function is only available to elements within a Section.

Parameters

column	column label of the column to summarize
group	column label of the group to summarize
condition	condition on the group to summarize

Return

```
average of all values in a group
```

Example

```
text = average("Quantity", "State", "field['Discount']>0");
```

See Also

average(cellRange), for the Table version.
SI.22, *Section Object*, for additional functions pertaining to Sections.
Sections, for scripting issues related to Sections.

CONCAT(COLUMN[, GROUP[,CONDITION]])

Concatenates all values in the specified column inside a group. This function is only available to elements within a Section.

Parameters

column	column label of the column to summarize
group	column label of the group to summarize
condition	condition on the group to summarize

Return

concatenation of all values in a group

Example

```
text = concat("Quantity", "State", "field['Discount']>0");
```

See Also

concat(cellRange), for the Table version.
SI.22, *Section Object*, for additional functions pertaining to Sections.
Sections, for scripting issues related to Sections.

CORRELATION(COLUMN, COLUMN2[, GROUP[,CONDITION]])

Calculates the correlation between two columns. This function is only available to elements within a Section.

Parameters

column	column label of the primary column
column2	column label of the column to correlate
group	column label of the group to summarize
condition	condition on the group to summarize

Return

correlation between the two columns

Example

```
text = correlation('Quantity','total','State',
   "field['Discount']>0");
```

See Also

correlation(tablelens,column1,column2 [,condition]), for the Table version.
SI.22, *Section Object*, for additional functions pertaining to Sections.
Sections, for scripting issues related to Sections.

COUNT(COLUMN[, GROUP[,CONDITION]])

Calculates the number of values in the specified column inside a group. This function is only available to elements within a Section.

Parameters

column	column label of the column to summarize

group	column label of the group to summarize
condition	condition on the group to summarize

Return

number of values in a group

Example

```
text = count("Quantity", "State", "field['Discount']>0");
```

See Also

count(cellRange), for the Table version.
SI.22, *Section Object*, for additional functions pertaining to Sections.
Sections, for scripting issues related to Sections.

COUNTDISTINCT(COLUMN[, GROUP[,CONDITION]])

Calculates the number of distinct values in the specified column inside a group. This function is only available to elements within a Section.

Parameters

column	column label of the column to summarize
group	column label of the group to summarize
condition	condition on the group to summarize

Return

number of distinct values in a group

Example

```
text = countDistinct("Quantity", "State",
   "field['Discount']>0");
```

See Also

countDistinct(cellRange), for the Table version.
SI.22, *Section Object*, for additional functions pertaining to Sections.
Sections, for scripting issues related to Sections.

COVARIANCE(COLUMN, COLUMN2[, GROUP[,CONDITION]])

Calculate the covariance of two columns. This function is only available to elements within a Section.

Parameters

column	column label of the column to summarize
column2	second column used in calculating covariance
group	column label of the group to summarize
condition	condition on the group to summarize

Return

covariance of the two columns

Example

```
text = covariance('Quantity','Total',"State",
   "field['Discount']>0");
```

See Also

covariance(tablelens,column, column2[, group[,condition]]), for the Table version.
SI.22, *Section Object*, for additional functions pertaining to Sections.
Sections, for scripting issues related to Sections.

MAX(COLUMN[, GROUP[,CONDITION]])

Calculates the maximum of values in the specified column inside a group. This function is only available to elements within a Section.

Parameters

column	column label of the column to summarize
group	column label of the group to summarize
condition	condition on the group to summarize

Return

maximum of all values in a group

Example

```
text = max("Quantity", "State", "field['Discount']>0");
```

See Also

max(cellRange), for the Table version.
SI.22, *Section Object*, for additional functions pertaining to Sections.
Sections, for scripting issues related to Sections.

MEDIAN(COLUMN[, GROUP[,CONDITION]])

Calculates the median number in the specified column inside a group. This function is only available to elements within a Section.

Parameters

column	column label of the column to summarize
group	column label of the group to summarize
condition	condition on the group to summarize

Return

median number of all values in a group

Example

```
text = median("Quantity", "State", "field['Discount']>0");
```

See Also

median(cellRange), for the Table version.
SI.22, *Section Object*, for additional functions pertaining to Sections.
Sections, for scripting issues related to Sections.

MIN(COLUMN[, GROUP[,CONDITION]])

Calculates the minimum of values in the specified column inside a group. This function is only available to elements within a Section.

Parameters

column	column label of the column to summarize
group	column label of the group to summarize
condition	condition on the group to summarize

Return

```
minimum of all values in a group
```

Example

```
text = min("Quantity", "State", "field['Discount']>0");
```

See Also

min(cellRange), for the Table version.
SI.22, *Section Object*, for additional functions pertaining to Sections.
Sections, for scripting issues related to Sections.

MODE(COLUMN[, GROUP[,CONDITION]])

Calculates the mode of values in the specified column inside a group. This function is only available to elements within a Section.

Parameters

column	column label of the column to summarize
group	column label of the group to summarize
condition	condition on the group to summarize

Return

```
mode of all values in a group
```

Example

```
text = mode("Quantity", "State", "field['Discount']>0");
```

See Also

mode(cellRange), for the Table version.
SI.22, *Section Object*, for additional functions pertaining to Sections.
Sections, for scripting issues related to Sections.

NTHLARGEST(N, COLUMN[, GROUP[,CONDITION]])

Calculates the *n*th largest value in the specified column inside a group. This function is only available to elements within a Section.

Parameters

n	the nth largest value to search
column	column label of the column to summarize
group	column label of the group to summarize
condition	condition on the group to summarize

Return

```
nth largest value in a group
```

Example

```
text = nthLargest(2, "Quantity","State",
   "field['Discount']>0");
```

See Also

nthLargest(n, cellRange), for the Table version.
SI.22, *Section Object*, for additional functions pertaining to Sections.
Sections, for scripting issues related to Sections.

NTHMOSTFREQUENT(N, COLUMN[, GROUP[,CONDITION]])

Calculates the *n*th most frequently occurring value in the specified column inside a group. This function is only available to elements within a Section.

Parameters

n	nth most frequent occurring value to search
column	column label of the column to summarize
group	column label of the group to summarize
condition	condition on the group to summarize

Return

```
nth most frequent occurring value in a group
```

Example

```
text = nthMostFrequent(2,"Quantity","State",
   "field['Discount']>0");
```

See Also

nthMostFrequent(n, cellRange), for the Table version.
SI.22, *Section Object*, for additional functions pertaining to Sections.
Sections, for scripting issues related to Sections.

NTHSMALLEST(N, COLUMN[, GROUP[,CONDITION]])

Calculates the *n*th smallest value in the specified column inside a group. This function is only available to elements within a Section.

Parameters

n	the nth smallest value to search
column	column label of the column to summarize
group	column label of the group to summarize
condition	condition on the group to summarize

Return

```
nth smallest value in a group
```

Example

```
text = nthSmallest(2,"Quantity","State",
   "field['Discount']>0");
```

See Also

nthSmallest(n, cellRange), for the Table version.
SI.22, *Section Object*, for additional functions pertaining to Sections.
Sections, for scripting issues related to Sections.

POPULATIONSTANDARDDEVIATION(COLUMN[, GROUP,CONDITION])

Calculates the population standard deviation of values in the specified column inside a group. This function is only available to elements within a Section.

Parameters

column	column label of the column to summarize
group	column label of the group to summarize
condition	condition on the group to summarize

Return

population standard deviation of all values in a group.

Example

```
text = populationStandardDeviation("Quantity", "State",
   "field['Discount']>0");
```

See Also

populationStandardDeviation(cellRange), for the Table version.
standardDeviation(column[, group[,condition]]), for sample version.
SI.22, *Section Object*, for additional functions pertaining to Sections.
Sections, for scripting issues related to Sections.

POPULATIONVARIANCE(COLUMN[, GROUP[,CONDITION]])

Calculates the population variance of values in the specified column inside a group. This function is only available to elements within a Section.

Parameters

column	column label of the column to summarize
group	column label of the group to summarize
condition	condition on the group to summarize

Return

population variance of all values in a group

Example

```
text = populationVariance("Quantity","State",
   "field['Discount']>0");
```

See Also

populationVariance(cellRange), for the Table version.
variance(column[, group[,condition]]), for the sample version.
SI.22, *Section Object*, for additional functions pertaining to Sections.
Sections, for scripting issues related to Sections.

PTHPERCENTILE(P, COLUMN[, GROUP[,CONDITION]])

Calculates the *p*th percentile, by group, of values in a specified column. The *p*th percentile is the value below which *p* percent of the records in the group fall. The `pthPercentile()` function is only available to elements within a Section.

Parameters

p	desired percentile value
column	column label of the column to summarize
group	column label of the group to summarize
condition	condition on the group to summarize

Return

p-th percentile of all values in a group

Example

```
text = pthPercentile(p,"Quantity", "State",
   "field['Discount']>0");
```

See Also

pthPercentile(tablelens,p,column[, group,condition]), for the Table version.
SI.22, *Section Object*, for additional functions pertaining to Sections.
Sections, for scripting issues related to Sections.

PRODUCT(COLUMN[, GROUP[,CONDITION]])

Calculates the product of values in the specified column inside a group. This function is only available to elements within a Section.

Parameters

column	column label of the column to summarize
group	column label of the group to summarize
condition	condition on the group to summarize

Return

product of all values in a group

Example

```
text = product("Quantity", "State", "field['Discount']>0");
```

See Also

product(cellRange), for the Table version.
SI.22, *Section Object*, for additional functions pertaining to Sections.
Sections, for scripting issues related to Sections.

STANDARDDEVIATION(COLUMN[, GROUP[,CONDITION]])

Calculates the (sample) standard deviation of values in the specified column inside a group. This function is only available to elements within a Section.

Parameters

column	column label of the column to summarize
group	column label of the group to summarize
condition	condition on the group to summarize

Return

```
standard deviation of all values in a group
```

Example

```
text = standardDeviation("Quantity","State",
"field['Discount']>0");
```

See Also

standardDeviation(cellRange), for the Table version.
populationStandardDeviation(column[, group,condition]), for population version.
SI.22, *Section Object*, for additional functions pertaining to Sections.
Sections, for scripting issues related to Sections.

SUM(COLUMN[, GROUP[,CONDITION]])

Calculates the sum of values in the specified column inside a group. This function is only available to elements within a Section.

Parameters

column	column label of the column to summarize
group	column label of the group to summarize
condition	condition on the group to summarize

Return

```
sum of all values in a group
```

Example

```
text = sum("Quantity", "State", "field['Discount']>0");
```

See Also

sum(cellRange), for the Table version.
SI.22, *Section Object*, for additional functions pertaining to Sections.
Sections, for scripting issues related to Sections.

VARIANCE(COLUMN[, GROUP[,CONDITION]])

Calculates the (sample) variance of values in the specified column inside a group. This function is only available to elements within a Section.

Parameters

column	column label of the column to summarize
group	column label of the group to summarize
condition	condition on the group to summarize

Return

```
variance of all values in a group
```

Example

```
text = variance("Quantity", "State", "field['Discount']>0");
```

See Also

variance(cellRange), for the Table version.
populationVariance(column[, group[,condition]]), for the population version.
SI.22, *Section Object*, for additional functions pertaining to Sections.
Sections, for scripting issues related to Sections.

WEIGHTEDAVERAGE(COLUMN, COLUMN2[, GROUP[,CONDITION]])

Calculates the weighted average of values in the specified columns inside a group. This function is only available to elements within a Section.

Parameters

column	column label of the column to summarize
column2	column label of the second column to summarize
group	column label of the group to summarize
condition	condition on the group to summarize

Return

```
weighted average of all values in a group
```

Example

```
text = weightedAverage("Total","Quantity","State",
   "field['Discount']>0");
```

See Also

weightedAverage(tablelens,column,column2 [,condition]]), for the Table version.
SI.22, *Section Object*, for additional functions pertaining to Sections.
Sections, for scripting issues related to Sections.

SI.2.3 Fixed Tables

This section describes functions that are provided for operations on fixed Tables.

See Also

SI.6, *Table Object* , for additional functions pertaining to Tables.
Data Tables, for scripting issues related to fixed Tables.

AVERAGE(CELLRANGE)

Calculates the average of values in the specified column inside a cell range. This function is only available to fixed Table elements.

Parameters

cellrange	the start and end of cellrange whose average is to be calculated.

Return

```
average of all values in a cell range
```

Example

```
text = average('[1,quantity]:[5,quantity]');
text = average('[1,0]:[5,0]');
```

See Also

average(tablelens,column[,condition]), for the *tableLens* version.
average(column[, group[,condition]]), for the Section version.
SI.6, *Table Object* , for additional functions pertaining to Tables.
Data Tables, for scripting issues related to fixed Tables.

CONCAT(CELLRANGE)

Concatenates all values in the specified column inside a cell range. This function is only available to fixed Table elements.

Parameters

cellrange start and end of cell range for concatenation

Return

```
concatenation of all values in a cellrange
```

Example

```
text = concat('[1,state]:[5,state]');
text = concat('[1,0]:[5,0]');
```

See Also

concat(column[, group[,condition]]), for the Section version.
SI.6, *Table Object* , for additional functions pertaining to Tables.
Data Tables, for scripting issues related to fixed Tables.

COUNT(CELLRANGE)

Calculates the number of non-empty values in the specified column inside a cellrange.This function is only available to fixed Table elements.

Parameters

cellrange the start and end of cell range of the column to be counted.

Return

```
number of values in a cellrange
```

Example

```
text = count('[1,state]:[1,state]');
text = count('[1,0]:[5,0]');
```

See Also

count(column[, group[,condition]]), for the Section version.
SI.6, *Table Object* , for additional functions pertaining to Tables.

Data Tables, for scripting issues related to fixed Tables.

COUNTDISTINCT(CELLRANGE)

Calculates the number of distinct values in the specified column inside a cell range. This function is only available to fixed Table elements.

Parameters

cellrange the start and end of cell range of the column to be counted (distinctly)

Return

number of distinct values in a cellrange

Example

```
text= countDistinct('[1,state]:[5,state]');
```

See Also

countDistinct(column[, group[,condition]]), for the Section version.
SI.6, *Table Object* , for additional functions pertaining to Tables.
Data Tables, for scripting issues related to fixed Tables.

MAX(CELLRANGE)

Calculates the maximum of values in the specified column inside a cellrange. This function is only available to fixed Table elements.

Parameters

cellrange the start and end of cell range whose aggregate is to be calculated

Return

maximum of all values in a group

Example

```
text = max('[1,state]:[5,state]');
```

See Also

max(column[, group[,condition]]), for the Section version.
SI.6, *Table Object* , for additional functions pertaining to Tables.
Data Tables, for scripting issues related to fixed Tables.

MEDIAN(CELLRANGE)

Calculates the median number in the specified column inside a cell range. This function is only available to fixed Table elements.

Parameters

cellrange the start and end of cell range whose median is to be calculated

Return

median number of all values in a cell range

Example

```
text = median('[1,price]:[5,price]');
```

See Also

median(column[, group[,condition]]), for the Section version.
SI.6, *Table Object* , for additional functions pertaining to Tables.
Data Tables, for scripting issues related to fixed Tables.

MIN(CELLRANGE)

Calculates the minimum of values in the specified column inside a cell range. This function is only available to fixed Table elements.

Parameters

cellrange the start and end of cell range whose minimum value is to be calculated

Return

minimum of all values in a cellrange

Example

```
text = min('[1,price]:[5,price]');
```

See Also

min(column[, group[,condition]]), for the Section version.
SI.6, *Table Object* , for additional functions pertaining to Tables.
Data Tables, for scripting issues related to fixed Tables.

MODE(CELLRANGE)

Calculates the mode of values in the specified column inside a cell range. This function is only available to fixed Table elements.

Parameters

cellrange the start and end of cell range whose mode is to be calculated

Return

mode of all values in a cellrange

Example

```
text = mode('[1,quantity]:[5,quantity]');
```

See Also

mode(column[, group[,condition]]), for the Section version.
SI.6, *Table Object* , for additional functions pertaining to Tables.
Data Tables, for scripting issues related to fixed Tables.

NTHLARGEST(N, CELLRANGE)

Calculates the *n*th largest value in the specified column inside a cell range. This function is only available to fixed Table elements.

Parameters

n	the nth largest value to search
cellrange	the start and end of cell range whose nth largest value is to be calculated

Return

nth largest value in a cellrange

Example

```
text=nthLargest(2, '[1,price]:[5,price]');
```

See Also

nthLargest(n, column[, group[,condition]]), for the Section version.
SI.6, *Table Object* , for additional functions pertaining to Tables.
Data Tables, for scripting issues related to fixed Tables.

NTHMOSTFREQUENT(N, CELLRANGE)

Calculates the *n*th most frequently occurring value in the specified column inside a cell range. This function is only available to fixed Table elements.

Parameters

n	the nth most frequent occurring value to search
cellrange	the start and end of cell range whose nth most frequent value is to be calculated

Return

nth most frequent occurring value in a cellrange

Example

```
text=
nthMostFrequent(2,'[1,state]:[5,state]');
```

See Also

nthMostFrequent(n, column[, group[,condition]]), for the Section version.
SI.6, *Table Object* , for additional functions pertaining to Tables.
Data Tables, for scripting issues related to fixed Tables.

NTHSMALLEST(N, CELLRANGE)

Calculates the *n*th smallest value in the specified column inside a cell range. This function is only available to fixed Table elements.

Parameters

n	the nth smallest value to search
cellrange	the start and end of cell range whose aggregate is to be calculated

Return

```
nth smallest value in a cellrange
```

Example

```
text = nthSmallest(2,'[1,price]:[5,price]');
```

See Also

nthSmallest(n, column[, group[,condition]]), for the Section version.
SI.6, *Table Object* , for additional functions pertaining to Tables.
Data Tables, for scripting issues related to fixed Tables.

POPULATIONSTANDARDDEVIATION(CELLRANGE)

Calculates the population standard deviation of values in the specified column inside a cellrange. This function is only available to fixed Table elements.

Parameters

cellrange	the start and end of cell range whose population standard deviation is to be calculated

Return

```
population standard deviation of all values in a cellrange.
```

Example

```
text=populationStandardDeviation('[1,sales]:[5,sales]');
```

See Also

populationStandardDeviation(column[, group,condition]), for the Section version.
standardDeviation(cellRange), for the sample version.
SI.6, *Table Object* , for additional functions pertaining to Tables.
Data Tables, for scripting issues related to fixed Tables.

POPULATIONVARIANCE(CELLRANGE)

Calculates the population variance of values in the specified column inside a cellrange. This function is only available to fixed Table elements.

Parameters

cellrange	the start and end of cell range whose population variance is to be calculated

Return

```
population variance of all values in a cellrange
```

Example

```
text=populationVariance('[1,sales]:[5,sales]');
```

See Also

populationVariance(column[, group[,condition]]), for the Section version.
variance(cellRange), for the sample version.
SI.6, *Table Object* , for additional functions pertaining to Tables.
Data Tables, for scripting issues related to fixed Tables.

PRODUCT(CELLRANGE)

Calculates the product of values in the specified column inside a cell range. This function is only available to fixed Table elements.

Parameters

cellrange the start and end of cell range whose price is to be calculated

Return

product of all values in a cellrange

Example

```
text = product('[1,price]:[5,price]');
```

See Also

product(column[, group[,condition]]), for the Section version.
SI.6, *Table Object* , for additional functions pertaining to Tables.
Data Tables, for scripting issues related to fixed Tables.

STANDARDDEVIATION(CELLRANGE)

Calculates the (sample) standard deviation of values in the specified column inside a cellrange. This function is only available to fixed Table elements.

Parameters

cellrange the start and end of cell range whose standard deviation is to be calculated

Return

standard deviation of all values in a cellrange

Example

```
text=standardDeviation("[1,price]:[5,price]");
```

See Also

standardDeviation(column[, group[,condition]]), for the Section version.
populationStandardDeviation(cellRange), for the population version.
SI.6, *Table Object* , for additional functions pertaining to Tables.
Data Tables, for scripting issues related to fixed Tables.

SUM(CELLRANGE)

Calculates the sum of values in the specified column inside a cell range. This function is only available to fixed Table elements.

Parameters

cellrange the start and end of cell range whose sum is to be calculated

Return

sum of all values in a cellrange

Example

```
text = sum("[1,quantity]:[5,quantity]");
```

See Also

sum(column[, group[,condition]]), for the Section version.
SI.6, *Table Object* , for additional functions pertaining to Tables.
Data Tables, for scripting issues related to fixed Tables.

VARIANCE(CELLRANGE)

Calculates the variance of values in the specified column inside a cell range. This function is only available to fixed Table elements.

Parameters

cellrange the start and end of cell range whose price is to be calculated

Return

variance of all values in a cellrange

Example

```
text = variance("[1,price]:[5,price]");
```

See Also

variance(column[, group[,condition]]), for the Section version.
populationVariance(cellRange), for the population version.
SI.6, *Table Object* , for additional functions pertaining to Tables.
Data Tables, for scripting issues related to fixed Tables.

SI.2.4 Global Objects

The functions in this section operate on a *tableLens* object. The '`Table1.Tablelens`' parameter in the examples below refers to the TableLens class of the Table element '`Table1`'.

AVERAGE(TABLELENS,COLUMN[,CONDITION])

Calculates the average of values in the specified column.

Parameters

tableLens	the tableLens object of the table
column	column label of the column to summarize
condition	condition on the column to summarize

Return

average of all values in a column

Example

```
text = average(Table1.tableLens, "Quantity", "Discount>0");
```

See Also

average(cellRange), for the Table version.

CONCAT(TABLELENS,COLUMN[,CONDITION])

Concatenates all values in the specified column.

Parameters

tableLens	the tableLens object of the table
column	column label of the column to summarize
condition	condition on the column to summarize

Return

concatenation of all values in a column

Example

```
text = concat(Table1.tableLens,"State","Discount>0");
```

See Also

concat(cellRange), for the Table version.

CORRELATION(TABLELENS,COLUMN1,COLUMN2 [,CONDITION])

Calculates the correlation between two columns.

Parameters

tableLens	the tableLens object of the table
column1	column label of the primary column
column2	column label of the column to correlate
condition	condition on the group to summarize

Return

correlation between the two columns

Example

```
text = correlation(Table1.tableLens,'Quantity','Total',
   "Discount>0");
```

See Also

correlation(column, column2[, group[,condition]]), for the Section version.

SI.6, *Table Object* , for additional functions pertaining to Tables.

Data Tables, for scripting issues related to fixed Tables.

COUNT(TABLELENS,COLUMN[,CONDITION])

Calculates the number of values in the specified column.

Parameters

tableLens	the tableLens object of the table
column	column label of the column to summarize
condition	condition on the column to summarize

Return

number of values in a column

Example

```
text = count(Table1.tableLens,'Quantity',"Discount>0");
```

See Also

count(cellRange), for the Table version.

COUNTDISTINCT(TABLELENS,COLUMN[,CONDITION])

Calculates the number of distinct values in the specified column.

Parameters

tableLens	the tableLens object of the table
column	column label of the column to summarize
condition	condition on the column to summarize

Return

number of distinct values in a column

Example

```
text=countDistinct(Table1.tableLens,'Quantity',
   "Discount>0");
```

See Also

countDistinct(cellRange), for the Table version.

COVARIANCE(TABLELENS,COLUMN, COLUMN2[, GROUP[,CONDITION]])

Calculates the covariance of two columns.

Parameters

tableLens	the tableLens object of the table
column	column label of the column to summarize
column2	second column used in calculating covariance
condition	condition on the group to summarize

Return

covariance of the two columns

Example

```
text =
covariance(Table1.tableLens,'Quantity','Total',
   "Discount>0");
```

See Also

covariance(column, column2[, group[,condition]]), for the Section version.

MAX(TABLELENS,COLUMN[,CONDITION])

Calculates the maximum of values in the specified column.

Parameters

tableLens	the tableLens object of the table
column	column label of the column to summarize
condition	condition on the column to summarize

Return

maximum of all values in a column

Example

```
text = max(Table1.tableLens,'Quantity',"Discount>0");
```

See Also

max(cellRange), for the Table version.

MEDIAN(TABLELENS,COLUMN[,CONDITION])

Calculates the median number in the specified column.

Parameters

tableLens	the tableLens object of the table
column	column label of the column to summarize
condition	condition on the column to summarize

Return

median number of all values in a column

Example

```
text = median(Table1.tableLens, 'Quantity', "Discount>0");
```

See Also

median(cellRange), for the Table version.

MIN(TABLELENS,COLUMN[,CONDITION])

Calculates the minimum of values in the specified column inside a group. This function is only available to elements in a section.

Parameters

tableLens	the tableLens object of the table
column	column label of the column to summarize

condition condition on the column to summarize

Return

minimum of all values in a column

Example

```
text = min(Table1.tableLens,'Quantity',"Discount>0");
```

See Also

min(cellRange), for the Table version.

MODE(TABLELENS,COLUMN[,CONDITION])

Calculate the mode of values in the specified column inside a group. This function is only available to elements in a section.

Parameters

tableLens the tableLens object of the table
column column label of the column to summarize
condition condition on the column to summarize

Return

mode of all values in a column

Example

```
text = mode(Table1.tableLens,'Quantity',"Discount>0");
```

See Also

mode(cellRange), for the Table version.

NTHLARGEST(TABLELENS,N,COLUMN[,CONDITION])

Calculate the *n*th largest value in the specified column.

Parameters

tableLens the tableLens object of the table
n the nth largest value to search
column column label of the column to summarize
condition condition on the column to summarize

Return

nth largest value in a column

Example

```
text = nthLargest(Table1.tableLens,2,'Quantity',
   "Discount>0");
```

See Also

nthLargest(n, cellRange), for the Table version.

NTHMOSTFREQUENT(TABLELENS,N,COLUMN [,CONDITION])

Calculate the *n*th most frequently occurring value in the specified column.

Parameters

tableLens	the tableLens object of the table
n	the nth most frequent value to search
column	column label of the column to summarize
condition	condition on the column to summarize

Return

nth most frequently occurring value in a column

Example

```
text =
nthMostFrequent(Table1.tableLens,2,'Quantity',
   "Discount>0");
```

See Also

nthMostFrequent(n, cellRange), for the Table version.

NTHSMALLEST(TABLELENS,N,COLUMN[,CONDITION])

Calculate the *n*th smallest value in the specified column.

Parameters

tableLens	the tableLens object of the table
n	the nth smallest value to search
column	column label of the column to summarize
condition	condition on the column to summarize

Return

nth smallest value in a column

Example

```
text = nthSmallest(Table1.tableLens,2,'Quantity',
   "Discount>0");
```

See Also

nthSmallest(n, cellRange), for the Table version.

POPULATIONSTANDARDDEVIATION(TABLELENS,COLUMN[,CONDITION])

Calculate the population standard deviation of values in the specified column.

Parameters

tableLens	the tableLens object of the table
column	column label of the column to summarize
condition	condition on the column to summarize

Return

population standard deviation of all values in a column.

Example

```
text = populationStandardDeviation(Table1.tableLens,
   'Quantity',"Discount>0");
```

See Also

populationStandardDeviation(cellRange), for the Table version.

POPULATIONVARIANCE(TABLELENS,COLUMN [,CONDITION])

Calculate the population variance of values in the specified column.

Parameters

tableLens	the tableLens object of the table
column	column label of the column to summarize
condition	condition on the column to summarize

Return

population variance of all values in a column

Example

```
text=
populationVariance(Table1.tableLens,'Quantity',
   "Discount>0");
```

See Also

populationVariance(cellRange), for the Table version.

PTHPERCENTILE(TABLELENS,P,COLUMN[, GROUP,CONDITION])

Calculate the *p*th percentile of values in the specified column inside a group.

Parameters

tableLens	the tableLens object of the table
p	the pth percentile to be calculated
column	column label of the column to summarize
condition	condition on the column to summarize

Return

p-th percentile of all values in a group

Example

```
text = pthPercentile(Table1.tableLens,99,'Quantity',
   "Discount>0");
```

See Also

pthPercentile(p, column[, group[,condition]]), for the Section version.

PRODUCT(TABLELENS,COLUMN[, CONDITION])

Calculate the product of values in the specified column.

Parameters

tableLens	the tableLens object of the table
column	column label of the column to summarize
condition	condition on the column to summarize

Return

product of all values in a column

Example

```
text = product(Table1.tableLens,"Quantity","Discount>0");
```

See Also

product(cellRange), for the Table version.

STANDARDDEVIATION(TABLELENS,COLUMN [,CONDITION])

Calculate the standard deviation of values in the specified column

Parameters

tableLens	the tableLens object of the table
column	column label of the column to summarize
condition	condition on the column to summarize

Return

standard deviation of all values in a column

Example

```
text = standardDeviation(Table1.tableLens,"Quantity",
   "Discount>0");
```

See Also

standardDeviation(cellRange), for the Table version.

SUM(TABLELENS,COLUMN[,CONDITION])

Calculate the sum of values in the specified column.

Parameters

tableLens	the tableLens object of the table
column	column label of the column to summarize
condition	condition on the column to summarize

Return

sum of all values in a column

Example

```
text = sum(Table1.tableLens,"Quantity","Discount>0");
```

See Also

sum(cellRange), for the Table version.

VARIANCE(TABLELENS,COLUMN[,CONDITION])

Calculate the variance of values in the specified column inside a group. This function is only available to elements in a section.

Parameters

tableLens	the tableLens object of the table

column	column label of the column to summarize
condition	condition on the column to summarize

Return

variance of all values in a column

Example

```
text = variance(Table1.tableLens,"Quantity","Discount>0");
```

See Also

variance(cellRange), for the Table version.

WEIGHTEDAVERAGE(TABLELENS,COLUMN,COLUMN2 [,CONDITION])

Calculate the weighted average of values in the specified columns inside a group. This function is only available to elements in a section.

Parameters

tableLens	the tableLens object of the table
column	column label of the column to summarize
column2	column label of the second column to summarize
condition	condition on the group to summarize

Return

weightedAverage of all values in a column

Example

```
text = weightedAverage(Table1.tableLens,"Total",
   "Quantity","Discount>0");
```

See Also

weightedAverage(column, column2[, group[,condition]]), for the Section version.

SI.3 Tab Object

A *Tab* element advances the printing to the next tab stop. The Tab element also serves as the superclass of the *Text Object*. The properties defined in the Tab element are also available in a Text element for controlling the behavior of tabs in the text string.

Super Class

Element Object

FILLSTYLE

The line style used to fill the tab space. Styles are defined in the *StyleReport Object*.

Type

integer

Example
```
fillStyle = StyleConstant.THIN_LINE;
```

RIGHTTAB

Specifies that the tab is a *right tab*. A right tab advances the printing to the next tab stop and right-aligns the following contents at that location.

Type
```
Boolean
```

Example
```
rightTab = true;
```

TABSTOPS

Specifies tab stop locations as an array. Locations are specified in inches from the left edge of the page printable area.

Type
```
number array
```

Example
```
tabStops = [1, 2.5, 4, 5.5, 7];
```

SI.4 Text Object

Text elements are the most commonly used report element. They can display simple lines of text or full paragraphs.

Super Class
Tab Object

See Also
Text Element, in *Advanced Report Design*, for basic information on Text elements.
Text and TextBox, for information on scripting Text.

HYPERLINK

Specifies a hyperlink to a report, element target, or a URL.

Type
```
String, or inetsoft.report.Hyperlink
```

Example
```
hyperlink = "Tutorial/Interactive";
```

See Also
Hyperlinks, in *Advanced Report Design*, for basic information on hyperlinks.

Using the onClick Handler, for other approaches to hyperlinking.

TEXTID

Specifies a Text ID for the text element. 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*.

Example

```
Text1.textID = 'Text1ID'
```

See Also

Localization/Internationalization in *Advanced Report Design* for more information on localization.

JUSTIFY

Specifies right-justification for text. When `true`, the right edge of the text is aligned evenly.

Type

`Boolean`

Example

```
justify = true;
```

ORPHANCONTROL

Specifies text paragraph widow/orphan control. When `true`, the report engine attempts to avoid single lines at the top or bottom of a page.

Type

`Boolean`

Example

```
orphanControl = false;
```

GROW

Specifies whether a text element (within a Section) can grow vertically to display contents that exceed the element's original size.

Type

`Boolean` `true: grow vertically to display contents`
`false: keep current size, and crop contents`

Example

```
grow = true;
```

See Also

Section Element, in *Advanced Report Design*, for information about Sections.

TEXT

The string displayed by the text element.

Type

```
string
```

Example

```
text = 'Hello.'
text = (new Date()).toString();
```

TEXTADVANCE

Specifies the amount of space (in points) to advance following the text element.

Type

```
integer
```

Example

```
textAdvance = 0;
```

SI.5 Heading Object

A heading object is a special Text element. Like a regular Text element, a heading object displays a block of text. In addition, a heading object has a *heading level*, which has two uses:

- The heading level determines the heading number (e.g., 1.2, 2.1.1., etc.) that is added to the heading text.
- The heading level is used to construct the Table of Contents, if the report contains one.

Super Class

`Text Object`

See Also

Heading and *Table of Contents*, in *Advanced Report Design*, for general information.

LEVEL

Specifies the heading level (top-level = 1).

Type

```
integer
```

Example

```
level = 2;
```

SI.6 Table Object

Table elements display tabular data. Tables can flow across pages, or horizontally break between pages. By default, table sizes (including column width and row heights) are automatically calculated based on the cell contents.

You can use built-in table styles to control display attributes such as font, color, and alignment. Therefore, a report application normally does not need to specify many table attributes in script.

Super Class

Element Object

See Also

Table Element, in *Advanced Report Design*, for general information on Tables.
SI.2.3, *Fixed Tables*, for additional functions pertaining to Tables.
Data Tables, for scripting issues related to fixed Tables.

SI.6.1 General Properties

This section presents general properties of the Table object.

FIXEDWIDTHS

Specifies table column widths as an array.

- When an automatic layout option is set for the table (e.g., 'Fit Page'), the values in the `fixedWidths` array are used to proportionally scale the columns of the table. The proportional width of the *i*th column is `fixedWidths[i]/sum(fixedWidths)`.
- When 'Manual Columns Widths' is set for the table, the values in the `fixedWidths` array provide the widths of all columns in pixels.

In both cases, the `fixedWidths` array should contain a value for each column in the table. To set the width of just a single column, use *colWidth* instead.

Type

```
integer array
```

Example

```
fixedWidths = [80, 100, 100, 90, 40];
```

See Also

Table Layout, in *Advanced Report Design*, for more information on setting column widths.

LAYOUT

Specifies the table layout option, which controls how the table is formatted. It must have one of the following values:

– TABLE_FIT_CONTENT
– TABLE_FIT_PAGE
– TABLE_EQUAL_WIDTH
– TABLE_FIT_CONTENT_1PP
– TABLE_FIT_CONTENT_PAGE

Type

`integer`

Example

```
layout = StyleConstant.TABLE_FIT_CONTENT;
```

See Also

Table Layout, in *Advanced Report Design*, for more information on the options.

ORPHANCONTROL

Specifies table row widow/orphan control. When `true`, the report engine attempts to avoid placing a single row at the top or bottom of a page.

Type

`Boolean` `true: avoid orphan rows`
`false: allow orphan rows`

Example

```
orphanControl = true;
```

COLUMNMATCHING

Specifies that table columns should match column widths from another table. This is useful when one table follows another in the report, and you want to align the columns of the second table with respect to the first.

Type

`two dimensional array`

Example

```
// Arbitrary matching:
columnMatching =
['tableElementId',[[0,0],[1,[1,2]],[[2,3],3]]];

// Exact (one-to-one) matching:
```

```
columnMatching =
['tableElementId',[[0,0],[1,1],[2,2],[3,3]]];
```

The mapping between columns of the adjusted table (i.e., the table being modified) and the reference table is an array that contains the element ID of the reference table followed by pairs of matched columns:

```
['RefId',[[col,Refcol],[col,Refcol],...,[col,Refcol]]]
```

In the above, `col` represents an individual index or arrays of indices from the adjusted table, and `Refcol` represents an individual index or arrays of indices from the reference table. For example, consider the following valid matches:

- `[[0,1,2],0]`: First three columns (0, 1, 2) are adjusted to span the width of the first column (0) in the reference table.
- `[[0,1,2],[0,1]]`: First three columns (0, 1, 2) are adjusted to span the combined width of the first two columns (0, 1) in the reference table.
- `[0,[0,1]]`: First column (0) is adjusted to span the combined width of the first two columns (0, 1) in the reference table.

Each column of each table must appear once in the mapping. Indices do not need to appear in order; however, the matched columns must be *alignable*. For example, the pairing `[[0,1,2],[0,1]]` is valid, but `[[0,1,2],[1,2]]` is not, because it does not permit the first column (0) of the reference table to be placed.

See Also

Matching Column Sizes in Two Tables, in *Advanced Report Design*, for more information.

FILLPAGE

Specifies how a table is presented in a page. When `true`, the table is stretched to the bottom of the page.

Type

`Boolean`

Example

```
fillPage = true;
```

ROWVISIBLE

Controls the visibility of the row in which it is placed. The `rowVisible` property is only available to cell formulas within a Freehand Table.

Type

`Boolean` `true: show corresponding row`

```
false: hide corresponding row
```

Example

```
rowVisible = false;
```

See Also

Creating a Table Using Freehand Operations, in *Advanced Report Design*, for more information.

REGIONVISIBLE

Controls the visibility of the region (header, body, etc.) in which it is placed. The `regionVisible` property is only available to cell formulas within a Freehand Table.

Type

```
Boolean      true: show corresponding region
             false: hide corresponding region
```

Example

```
regionVisible = false;
```

See Also

Creating a Table Using Freehand Operations, in *Advanced Report Design*, for more information.

ROW

Returns the index of the base table row corresponding to current Freehand Table cell. Hidden rows are not indexed. The `row` property is read-only, and available only to cell formulas within a Freehand Table.

Type

```
integer
```

Example

```
row;
```

To reference the current row index of the source data table prior to row-hiding, use dataRow().

DATAROW

Returns the index of the base table row corresponding to the current Freehand Table cell. Both hidden and visible rows are indexed. The `dataRow` property is read-only, and available only to cell formulas within a Freehand Table.

Type

```
integer
```

Example

```
dataRow;
```

To reference the current row index of the displayed output table (i.e., to ignore hidden rows), use row().

PADDING

Specifies the space (in pixels) between the table cell border and the table cell contents, using array of form `[top,left,bottom,right]`.

Type

`java.awt.Insets` or integer array

Example

```
padding = new java.awt.Insets(1, 2, 1, 2);
   or
padding = [1, 2, 1, 2];
```

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 as the second array index.

Type

two dimensional array

Example

```
var total = 0;
for(var row = 1; row < table.length; row++) {
  for(var col = 0; col < table.size; col++) {
    total = total + table[row][col];
  }
}
```

See Also

Accessing Table Data, for information about working with table data.

TABLE.LENGTH

Specifies the number of rows in a table, as displayed. This setting is overwritten by the number of rows returned by the query that is bound to the table. It counts the header rows as well as the data rows.

Type

integer

Example

```
table.length = 10;
```

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

Specifies the number of columns in a table, as displayed. This setting is overwritten by the number of columns returned by the query that is bound to the table.

Type

`integer`

Example

```
table.size = 10;
```

See Also

Accessing Table Data, for information about working with table data.
data.size, for number of columns prior to grouping and summarization.

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 as the second array index.

Type

`two dimensional array`

Example

```
var total = 0;
for(var row = 1; row < data.length; row++) {
   for(var col = 0; col < data.size; col++) {
      total = total + data[row][col];
   }
}
```

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. This setting is overwritten by the number of rows returned by the query that is bound to the table.

Type

`integer`

Example

```
if (data.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. This setting is overwritten by the number of columns returned by the query that is bound to the table.

Type

```
integer
```

Example

```
if (data.size > table.size) {
  Text1.text = 'Some table columns are hidden.';
}
```

See Also

table.size, for the width of post-aggregate (as-displayed) results table.
Accessing Table Data, for information about working with table data.

CELLROWBORDERCOLOR

Specifies the Table cell row border color as a two dimensional array. The attribute can be referenced using the row number as the first index and the column number as the second index.

Type

```
two dimensional array of java.awt.Color
```

Example

```
cellRowBorderColor[0][0] = java.awt.Color.red;
```

CELLCOLBORDERCOLOR

Specifies the Table cell column border color as a two dimensional array. The attribute can be referenced using the row number as the first index and the column number as the second index.

Type

```
two dimensional array of java.awt.Color
```

Example

```
cellColBorderColor[0][0] = java.awt.Color.red;
```

CELLROWBORDER

Specifies the Table cell row border line style as a two dimensional array. The attribute can be referenced using the row number as the first index and the column number as the second index.

Type

two dimensional array of *Line Styles* constants

Example

```
cellRowBorder[0][0] = StyleConstant.DOUBLE_LINE;
```

See Also

SI.41, *StyleReport Object*, for more information on constants.

CELLCOLBORDER

Specifies the Table cell column border line style as a two dimensional array. The attribute can be referenced using the row number as the first index and the column number as the second index.

Type

two dimensional array of *Line Styles* constants

Example

```
cellColBorder[0][0] = StyleConstant.DOUBLE_LINE;
```

See Also

SI.41, *StyleReport Object*, for more information on constants.

CELLINSETS

Specifies the Table cell gap space as a two dimensional array. The attribute can be referenced using the row number as the first index and the column number as the second index.

Type

two dimensional array of java.awt.Insets,
or array of four integers

Example

```
cellInsets[0][0] = [1, 2, 1, 2];
```

CELLSPAN

Specifies Table cell spanning as a two dimensional array. The attribute can be referenced using the row number as the first index and the column number as the second index.

Type

two dimensional array of java.awt.Dimension,

```
or array of row integers
```

Example

```
cellSpan[0][0] = [1, 2]; // span one column two rows
```

CELLFONT

Specifies the Table cell font as a two dimensional array. The attribute can be referenced using the row number as the first index and the column number as the second index.

Type

```
two dimensional array of java.awt.Font
```

Example

```
cellFont[0][0] = new java.awt.Font('Serif',
   java.awt.Font.BOLD, 12);
```

CELLALIGNMENT

Specifies the Table cell alignment as a two dimensional array. The attribute can be referenced using the row number as the first index and the column number as the second index.

Type

```
two dimensional array of Alignments constants
```

Example

```
cellAlignment[0][0] = StyleConstant.H_CENTER;
```

See Also

SI.41, *StyleReport Object*, for more information on constants.

CELLLINEWRAP

Specifies the Table cell line wrap settings as a two dimensional array. The attribute can be referenced using the row number as the first index and the column number as the second index.

Type

```
two dimensional array of Boolean
```

Example

```
cellLineWrap[0][0] = false;
```

CELLFOREGROUND

Specifies the Table cell foreground color as a two dimensional array. The attribute can be referenced using the row number as the first index and the column number as the second index.

Use ‘Edit Value’ in the Script Editor’s auto-complete menu to specify colors.

Type

```
two dimensional array of java.awt.Color or RGB values
```

Example

```
cellForeground[0][0] = java.awt.Color.green;
cellForeground[1][3] = [255,0,0]
```

CELLBACKGROUND

Specifies the Table cell background color as a two dimensional array. The attribute can be referenced using the row number as the first index and the column number as the second index.

Use ‘Edit Value’ in the Script Editor’s auto-complete menu to specify colors.

Type

```
two dimensional array of java.awt.Color or RGB values
```

Example

```
cellBackground[0][0] = java.awt.Color.green;
cellBackground[1][3] = [255,0,0]
```

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 number as the second index.

Type

```
two dimensional array of java.text.Format
```

Example

```
cellFormat[0][0] = new
java.text.DecimalFormat('$#,##0.00');
```

See Also

formatDate(), formatNumber(), for information on date/number formats.
Format Types, in *Advanced Report Design*, for general information.

CELLPRESENTER

Specifies the Table cell data presenter as a two dimensional array. The attribute can be referenced using the row number as the first index and the column number as the second index.

Type

```
two dimensional array of inetsoft.report.Presenter
```

Example

```
cellPresenter[0][0] =
‘inetsoft.report.painter.BooleanPresenter’;
```

See Also

Cell Presenter, in *Advanced Report Design*, for general information.

COLALIGNMENT

Specifies the Table column alignment as an array. Use the column number or column header as the index to reference the attribute.

Type

array of *Alignments* constants

Example

```
colAlignment['Total'] = StyleConstant.H_RIGHT;
```

See Also

SI.41, *StyleReport Object*, for more information on constants.

COLBACKGROUND

Specifies the Table column background as an array. Use the column number or column header as the index to reference the attribute.

Type

array of java.awt.Color

Example

```
colBackground['Total'] = java.awt.Color.green;
```

COLBORDER

Specifies the Table column border line style as an array. Use the column number or column header as the index to reference the attribute.

Type

array of *Line Styles* constants

Example

```
colBorder['Total'] = StyleConstant.DOUBLE_LINE;
```

See Also

SI.41, *StyleReport Object*, for more information on constants.

COLBORDERCOLOR

Specifies the Table column border color as an array. Use the column number or column header as the index to reference the attribute.

Type

array of java.awt.Color

Example

```
colBorderColor['Total'] = java.awt.Color.yellow;
```

COLFONT

Specifies the Table column font as an array. Use the column number or column header as the index to reference the attribute.

Type

```
array of java.awt.Font
```

Example

```
colFont['Total'] =
new java.awt.Font("Arial", java.awt.Font.PLAIN, 10);
```

COLFOREGROUND

Specifies the Table column foreground color as an array. Use the column number or column header as the index to reference the attribute.

Type

```
array of java.awt.Color
```

Example

```
colForeground['Total'] = java.awt.Color.red;
```

COLINSETS

Specifies the Table column gap space as an array. Use the column number or column header as the index to reference the attribute.

Type

```
array of java.awt.Insets
```

Example

```
// two points at left and right
colInsets['Total'] = [0, 2, 0, 2];
```

COLLINEWRAP

Specifies the Table column line wrap option as an array. Use the column number or column header as the index to reference the attribute.

Type

```
array of Boolean
```

Example

```
colLineWrap['Total'] = false;
```

COLWIDTH

Specifies a table column width (in pixels). Use the column number or column header as the index.

- When an automatic layout option is set for the table (e.g., 'Fit Page'), the `colWidth` value is used to proportionally scale the specified column of the table. The proportional width of the modified column is `colWidth/(colWidth+sum(other_column_widths))`.
- When 'Manual Columns Widths' is set for the table, the `colWidth` value specifies the final width of the column in pixels.

Type
```
integer (width in pixels)
```

Example
```
colWidth['Total'] = 50;
```

To set multiple column widths, use *fixedWidths*.

MAXCOLWIDTH

Specifies the maximum width for the columns. If a column exceeds the width, the contents of the cell are automatically wrapped. This property applies only to the 'Fit Content' and 'Fit Page' layouts.

Type
```
float
```

Example
```
maxColWidth = 50;
```

See Also
Table Layout, in *Advanced Report Design*, for information about Table layouts.

ROWALIGNMENT

Specifies the Table row alignment as an array. Use the row number as the index to reference the attribute.

Type
```
array of Alignments constants
```

Example
```
rowAlignment[0] = StyleConstant.H_RIGHT;
```

See Also
SI.41, *StyleReport Object*, for more information on constants.

ROWBACKGROUND

Specifies the Table row background as an array. Use the row number as the index to reference the attribute.

Type

```
array of java.awt.Color or RGB value
```

Example

```
rowBackground[1] = java.awt.Color.green;
rowBackground[1] = [0,0,255];
```

ROWBORDER

Affects the Table row border line style as an array. Use the row number as the index to reference the attribute.

Type

```
array of Line Styles constants
```

Example

```
rowBorder[0] = StyleConstant.DOUBLE_LINE;
```

See Also

SI.41, *StyleReport Object*, for more information on constants.

ROWBORDERCOLOR

Specifies the Table row border color as an array. Use the row number as the index to reference the attribute.

Type

```
array of java.awt.Color
```

Example

```
rowBorderColor[0] = java.awt.Color.yellow;
```

ROWFONT

Specifies the Table row font as an array. Use the row number as the index to reference the attribute.

Type

```
array of java.awt.Font
```

Example

```
rowFont[0] = new java.awt.Font("Arial",
java.awt.Font.PLAIN, 10);
```

ROWFOREGROUND

Specifies the Table row foreground color as an array. Use the row number as the index to reference the attribute.

Type

```
array of java.awt.Color
```

Example

```
rowForeground[0] = java.awt.Color.red;
```

ROWINSETS

Specifies the Table row gap space as an array. Use the row number as the index to reference the attribute.

Type

```
array of java.awt.Insets
```

Example

```
// two points at left and right
rowInsets[0] = [0, 2, 0, 2];
```

ROWLINEWRAP

Species the Table row line wrap option as an array. Use the row number as the index to reference the attribute.

Type

```
array of Boolean
```

Example

```
rowLineWrap[0] = false;
```

ROWHEIGHT

Specifies the table row heights as an array, with row number as the array index. The following rules apply:

- A positive integer value specifies a fixed row height in pixels.
- A value of -1 specifies automatic row height calculation (default).
- A negative integer value (other than -1) specifies a fixed *minimum* row height in pixels. In this case, the row height is calculated automatically but constrained to be no smaller than the specified value.
- A value of 0 specifies that the row should be hidden (i.e., zero height).

Type

```
array of integers
```

Examples

```
rowHeight[1] = -50; // minimum height of 50 for row 1
rowHeight[2] = -1;  // auto-height for row 2
rowHeight[3] = 0;   // hide row 3
rowHeight[4] = 99;  // fixed height of 99 for row 4
```

FORMAT

Specifies the format of a Table column.

Type

array of java.text.Format

Example

```
format['price'] =
   java.text.NumberFormat.getCurrencyInstance();
```

See Also

formatDate(), formatNumber(), for information on date/number formats.
Format Types, in *Advanced Report Design*, for general information.

ROWFORMAT

Specifies the format of a Table row.

Type

array of java.text.Format

Example

```
rowFormat[0] = java.text.DateFormat.getDateInstance();
```

See Also

formatDate(), formatNumber(), for information on date/number formats.
Format Types, in *Advanced Report Design*, for general information.

HYPERLINK

Add hyperlinks to a Table column. Hyperlinks can point to a report, element target, or a URL.

Type

array of string, or inetsoft.report.Hyperlink

Example

```
hyperlink['State'] = "Tutorial/Interactive";
```

See Also

Hyperlinks, in *Advanced Report Design*, for basic information on hyperlinks.
Using the onClick Handler, for other approaches to hyperlinking.

HIGHLIGHTED(NAME)

Identifies whether a highlight with the specified name has been applied to the table (i.e., whether the highlight condition has been satisfied.)

Parameter

name name of the highlight

Example

```
if(Table1.highlighted.lowSales) {
```

```
  log('lowSales highlight was applied')
}
```

See Also

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

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 Report Design*, for basic information on hyperlinks.

Using the onClick Handler, for other approaches to hyperlinking.

GETTEXTID(COL)

Returns the `textID` assigned to the column header for table column `col`. See *setTextID(col, textID)* to assign a Text ID using script.

Parameter

col	A column name or one of the following strings: '#Summary' to set TextID for summary header '#GrandTotal' to set TextID for grand total header

Example

```
Table1.getTextID("#Summary");
Table1.getTextID("#GrandTotal");
Table1.getTextID("Employee");
```

See Also

Localization/Internationalization in *Advanced Report Design* for more information on localization.

SETTEXTID(COL, TEXTID)

Assigns the specified `textID` to the column header for table column `col`. 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*.

Parameter

```
col          A column name or one of the following strings:
               '#Summary' to set TextID for summary header
               '#GrandTotal' to set TextID for grand total
header
textID       String identifier to be mapped in SreeBundle
file
```

Example

```
Table1.setTextID("#Summary", "SummaryID");
Table1.setTextID("#GrandTotal", "GrandTotalID");
Table1.setTextID("Employee", "EmployeeColumnID");
```

See Also

Localization/Internationalization in *Advanced Report Design* for more information on localization.

PRESENTER

Adds a presenter to a column of the Table.

Type

```
array of inetsoft.report.Presenter
```

Example

```
presenter[2] =
newInstance('inetsoft.report.painter.IconCounterPresenter')
```

Note that the '*newInstance(name)*' method is used to create the object instead of the '`new`' operator, because otherwise the `painter` package will confuse the JavaScript class loader.

See Also

Cell Presenter, in *Advanced Report Design*, for general information.

ROWPRESENTER

Adds a presenter to a row of the Table.

Type

```
array of inetsoft.report.Presenter
```

Example

```
rowPresenter[0] =
  newInstance('inetsoft.report.painter.HeaderPresenter');
```

Note that the '*newInstance(name)*' method is used to create the object instead of the '`new`' operator, because otherwise the `painter` package will confuse the JavaScript class loader.

See Also

Cell Presenter, in *Advanced Report Design*, for general information.

SIZE

Sets the size of the Table within a Section.

Type

`inetsoft.report.Size`

Example

```
size = [8,4];
```

See Also

Section Element, in *Advanced Report Design*, for information about Sections.

QUERY

Specifies the query, Worksheet, or data model bound to the Table element.

Type

`string`

Example

```
// Query:
Table1.query = 'customers';
Table1.query = 'Sample Queries/Fast Go Game'

// Global Worksheet:
Table1.query = 'ws:global:State Stats';
Table1.query = 'ws:global:Sales/Sales Explore';

// Local Worksheet:
Table1.query = "ws:Sales Explore";

// User Worksheet:
Table1.query = "ws:Annie:Sales Explore";

// Data Model:
Table1.query = "Orders/logicModel::Return Model"
```

See Also

Binding Queries, for general information on the 'query' property.
Creating a Local Query, in *Advanced Data Modeling*, for information about local queries.
Creating a Local Worksheet, in *Advanced Data Mashup*, for information about local Worksheets.
Controlling Access to an Asset, in *Advanced Data Mashup*, for information about user Worksheets.
query, for the equivalent Chart property.
query, for the equivalent Section property.

GROW

Specifies whether an element (inside a Section) can grow vertically in the event that the element's contents exceed its original size.

Type
`Boolean`

Example
```
grow = true;
```

See Also
Section Element, in *Advanced Report Design*, for information about Sections.

TABLELENS

Specifies the Table's TableLens object, which can be used to access and modify the table attributes.

Type
`inetsoft.report.lens.AttributeTableLens`

Example
```
tableLens = new inetsoft.report.lens.DefaultTableLens(2,2);
tableLens.setForeground(0, 2, java.awt.Color.red);
```

TABLEADVANCE

Specifies the amount of vertical space to advance following the Table. If this property is set to 0, the next element will touch the bottom of the Table.

Type
`integer`

Example
```
tableAdvance = 0;
```

TABLEWIDTH

Specifies a fixed width (in inches) for a Table. This value overrides the automatic table width calculation.

Type
`double`

Example
```
tableWidth = 5.5;
```

SI.6.2 The Table bindingInfo Property

The Table *bindingInfo* property provides access to functions controlling the data binding of a table element. This includes changes to hidden columns, summary columns, group columns, and option settings. The following sections introduce the functions provided by the *bindingInfo* property.

See Also
Chart bindingInfo Property, for the corresponding Chart functions.
The Section bindingInfo Property, for the corresponding Section functions.
Binding Data to a Table Element, in *Advanced Report Design*, for information on data binding.

BINDINGINFO.ADDGROUPCOL(COLNAME)

Creates grouping for the specified Table column.

Parameter

colName name of the group column

Example

```
bindingInfo.addGroupCol('state');
```

See Also
bindingInfo.removeGroupCol(colName), for the inverse operation.
bindingInfo.setColorField(col, type), for color grouping in a Chart.
bindingInfo.setShapeField(col, type), for shape grouping in a Chart.
bindingInfo.setSizeField(col, type), for size grouping in a Chart.

BINDINGINFO.SETDATELEVEL(COLNAME, LEVEL)

Sets the grouping level for a grouped date column.

Parameter

colName name of the grouped date column
level grouping level, e.g., StyleConstant.QUARTER_DATE_GROUP

Example

```
// Group by two months (Jan-Feb, Mar-Apr, May-Jun, etc.)
bindingInfo.setDateLevel('Order.Date',StyleConstant.MONTH_D
ATE_GROUP);
bindingInfo.setDateInterval('Order.Date',2);
```

See Also
bindingInfo.setDateInterval(colName, value), to set the grouping interval.
bindingInfo.addGroupCol(colName), to add a new grouping column.

BINDINGINFO.SETDATEINTERVAL(COLNAME, VALUE)

Sets the grouping interval for a grouped date column, i.e., the number of date units at the specified date grouping level to be included in each group.

Parameter

colName	name of the grouped date column
value	number of sequential date units to group

Example

```
// Group by two months (Jan-Feb, Mar-Apr, May-Jun, etc.)
bindingInfo.setDateLevel('Order.Date',StyleConstant.MONTH_D
ATE_GROUP);
bindingInfo.setDateInterval('Order.Date',2);
```

See Also

bindingInfo.setDateLevel(colName, level), to set the grouping level.
bindingInfo.addGroupCol(colName), to add a new grouping column.

BINDINGINFO.ADDHIDDENCOLUMN(COLNAME)

Hides the specified Table column from view.

Parameter

colName	the column name

Example

```
bindingInfo.addHiddenColumn('state');
```

See Also

bindingInfo.removeHiddenColumn(colName), for the inverse operation.

BINDINGINFO.ADDSUMMARYCOL(COLNAME)

Creates summarization for the specified Table column. A column can be used multiple times for summarization.

Parameter

colName	column name on which to summarize

Example

```
bindingInfo.addSummaryCol('Price');
```

See Also

bindingInfo.removeSummaryCol(colName), for the inverse operation.

BINDINGINFO.GETCOLUMNORDER(COLNAME)

Returns the sort order of the specified column.

Parameter

colName	name of column to test for sorting

Return

0:	if the column is unsorted
1:	if the column is sorted ascending
2:	if the column is sorted descending

Example

```
bindingInfo.getColumnOrder('Employee');
```

See Also

bindingInfo.getColumnOrder(col), for the equivalent Chart function.

BINDINGINFO.GETFORMULA(I)

Returns the summarization formula for the specified column.

Parameter

i index of the column whose summary formula is desired

Return

The formula in String format

Example

```
bindingInfo.getFormula(2);
```

See Also

bindingInfo.getFormula(col,binding), for the equivalent Chart function.

BINDINGINFO.GETGROUPCOL(I)

Returns the column name of the specified grouping field.

Parameter

i index of the group column

Return

Column name of the ith group column

Example

```
bindingInfo.getGroupCol(1);
```

BINDINGINFO.GETGROUPCOLCOUNT()

Returns the total number of grouping columns.

Return

number of columns that are grouped

Example

```
bindingInfo.getGroupColCount();
```

BINDINGINFO.GETGROUPORDER(COLNAME)

Returns the sort order for a grouping column.

Parameter

colName column whose grouping order is desired

Return

```
Integer value representing the grouping order:
   1: Ascending order
   2: Descending order
   4: Original order
```

Example

```
bindingInfo.getGroupOrder('State');
```

BINDINGINFO.GETSUMMARYCOL(I)

Returns the column name of the specifies summarization field.

Parameter

```
i      index of the summary column
```

Return

```
Column name of the ith summary column
```

Example

```
bindingInfo.getSummaryCol(1);
```

BINDINGINFO.GETSUMMARYCOLCOUNT()

Returns the total number of summary columns.

Return

```
number of columns that are summarized
```

Example

```
bindingInfo.getSummaryColCount();
```

BINDINGINFO.GETTOPN(GROUPCOL)

Returns the “top N” setting for the specified grouping column.

Parameter

```
groupCol      group column whose TopN setting is desired
```

Return

```
integer, the TopN value
```

Example

```
bindingInfo.getTopN('State');
```

See Also

bindingInfo.getTopN(col), for the equivalent Chart function.
bindingInfo.setTopN(groupCol, n), for the inverse operation.
Grouping and Summarization in Data Binding Wizard, in *Advanced Report Design*, for details on Top-N filtering.

BINDINGINFO.GETTOPNGROUPCOL(GROUPCOL)

Returns the 'Top N' group column name. This can be used to test if the 'Top N' of a group column exists.

Parameter

groupCol	the specified group column

Return

The group column name if exists, null otherwise

Example

```
bindingInfo.getTopNGroupCol('State');
```

BINDINGINFO.GETTOPNSUMMARYCOL(GROUPCOL)

Returns the 'Top N' summary column.

Parameter

groupCol	the specified group column

Return

The summary column name if exists, null otherwise

Example

```
bindingInfo.getTopNSummaryCol('State');
```

See Also

bindingInfo.getTopNSummaryCol(col), for the equivalent Chart function.

BINDINGINFO.ISSUMMARIZE(COLNAME)

Returns true if a grouping column is summarized.

Parameter

colName	name of a grouped column

Return

true	if grouping column is currently summarized
false	if grouping column is not summarized

Example

```
bindingInfo.isSummarize('State');
```

See Also

Walkthrough: Adding Table Grouping using the Wizard, in *Advanced Report Design*, for information on the 'Summarize Group' option of the 'Data Binding' dialog box.

BINDINGINFO.ISTOPNREVERSE(GROUPCOL)

Returns true if specified ranking is 'Bottom N' rather than 'Top N'.

Parameter

groupCol	the specified group column

Return

true	if topN reverse setting is true
false	if not found, or not reverse

Example

```
bindingInfo.isTopNReverse('State');
```

See Also

bindingInfo.isTopNReverse(col), for the equivalent Chart function.
bindingInfo.setTopNReverse(groupCol, bottom), for the inverse operation.
Grouping and Summarization in Data Binding Wizard, in *Advanced Report Design*, for details on Top-N filtering.

BINDINGINFO.REMOVEALLGROUPCOLS()

Removes grouping from all columns.

Example

```
bindingInfo.removeAllGroupCols();
```

See Also

bindingInfo.addGroupCol(colName), for the inverse operation.

BINDINGINFO.REMOVEGROUPCOL(COLNAME)

Removes grouping from the specified column.

Parameter

colName	Name of column to be ungrouped

Example

```
bindingInfo.removeGroupCol('State');
```

See Also

bindingInfo.addGroupCol(colName), for the inverse operation.

BINDINGINFO.REMOVEHIDDENCOLUMN(COLNAME)

Displays the specified column.

Parameter

colName	name of column to display

Example

```
bindingInfo.removeHiddenColumn('customer','state');
```

See Also

bindingInfo.addHiddenColumn(colName), for the inverse operation.

BINDINGINFO.REMOVESUMMARYCOL(COLNAME)

Removes summarization from the specified column.

Parameter

colName	Name of column to be un-summarized

Example

```
bindingInfo.removeSummaryCol('Price');
```

See Also

bindingInfo.addSummaryCol(colName), for the inverse operation.

BINDINGINFO.SETCOLUMNORDER(COLNAME,ORDER)

Sets the sort order of the specified column.

Parameter

colName	The name of column to sort
order	The sort order for the specified column: 0: remove sorting 1: sort ascending 2: sort descending

Example

```
bindingInfo.setColumnOrder('Employee',2);
// sort 'Employee' column descending
```

See Also

bindingInfo.setColumnOrder(col,order,[sortCol]), for the equivalent Chart function.
bindingInfo.getColumnOrder(colName), for the inverse operation.

BINDINGINFO.SETCOLUMNSEQUENCE(SEQ)

Arranges the Table columns in the left-to-right sequence specified. Omitted columns are not displayed.

Parameter

seq	A string specifying the desired column sequence

The parameter form 'idx:[ix1,ix2,...]' sets the sequence using column indices ix1,ix2,...

The parameter form 'name:[col1,col2,...]' sets the sequence using column names col1,col2,...

Example

```
bindingInfo.setColumnSequence('idx:[2,1,0]');
// Place third column first, and first column third

bindingInfo.setColumnSequence('name:[Col2,Col3,Col1]');
// Place column 'Col2' first, 'Col3' second, 'Col1' third
```

See Also

Specifying Columns in Data Binding Wizard, in *Advanced Report Design*, for information on ordering.

BINDINGINFO.SETCOLUMNVISIBLE(COLNAME,VALUE)

Sets the visibility of the specified column.

Parameter

colName	the column name
value	true/false to show/hide column

Example

```
bindingInfo.setColumnVisible('state',false);
```

See Also

bindingInfo.addHiddenColumn(colName), for an alternative approach.

BINDINGINFO.SETFORMULA(COL, FORMULA)

Sets the summarization formula for the specified column.

Parameter

col	The index of the column to be summarized
formula	The summary formula

Example

```
bindingInfo.setFormula(5, 'Sum<0>');
```

See Also

bindingInfo.setFormula(col, formula, binding), for the equivalent Chart function.
bindingInfo.getFormula(i), for the inverse operation.

BINDINGINFO.SETGROUPORDER(COLNAME, ORDER)

Sets the sort order of the specified grouping column.

Parameter

colName	column whose grouping order is to be set
order	order of this group column: StyleConstant.SORT_ASC: Ascending order StyleConstant.SORT_DESC: Descending order StyleConstant.SORT_NONE: Original order

Example

```
bindingInfo.setGroupOrder('State',
StyleConstant.SORT_DESC);
```

See Also

bindingInfo.getGroupOrder(colName), for the inverse operation.

BINDINGINFO.SETSUMMARIZE(COLNAME, SUM)

Specifies whether summaries should be displayed for the indicated grouping column.

Parameter

colName	name of the group column to be summarized
sum	true: show summarization for group false: do not show summarization

Example

```
bindingInfo.setSummarize('State', true);
```

See Also

bindingInfo.addSummaryCol(colName), to specify the column to be summarized.

BINDINGINFO.SETTOPN(GROUPCOL, N)

Sets the 'Top N' value for the specified grouping column.

Parameter

groupCol	the specified group column name
n	topN integer value

Example

```
bindingInfo.setTopN('State', 10);
```

See Also

bindingInfo.setTopN(col,N), for the equivalent Chart function.
bindingInfo.getTopN(groupCol), for the inverse operation.
Grouping and Summarization in Data Binding Wizard, in *Advanced Report Design*, for details on Top-N filtering.

BINDINGINFO.SETTOPNGROUPCOL(GROUPCOL1, GROUPCOL2)

Transfers the 'top-N' ranking from one grouping column to another grouping column.

Parameter

groupCol1	a grouped column with existing topN ranking
groupCol2	a grouped column to receive topN ranking

Example

```
bindingInfo.setTopNGroupCol('State','City');
```

See Also

bindingInfo.setTopN(groupCol, n), to specify ranking.
Grouping and Summarization in Data Binding Wizard, in *Advanced Report Design*, for details on Top-N filtering.

BINDINGINFO.SETTOPNREVERSE(GROUPCOL, BOTTOM)

Sets reverse 'Top N' ranking (i.e., 'Bottom N' ranking) for the specified grouping column.

Parameter

groupCol	the specified group column
bottom	true: reverse (Bottom N) ranking

Example

```
bindingInfo.setTopNReverse('State',true);
```

See Also

bindingInfo.setTopNReverse(col,boolean), for the equivalent Chart function.
bindingInfo.isTopNReverse(groupCol), for the inverse operation.

BINDINGINFO.SETTOPNSUMMARYCOL(GROUPCOL, COL)

Assigns the summary column to use for 'Top N' ranking of the indicated grouping column.

Parameter

groupCol	the specified group column name
col	the summary column name

Example

```
bindingInfo.setTopNSummaryCol('State', 'Price');
```

See Also

bindingInfo.setTopNSummaryCol(col,agg), for the equivalent Chart function.
bindingInfo.getTopNSummaryCol(groupCol), for the inverse operation.

BINDINGINFO.SETOPTION(NAME, VALUE)

Specifies a grouping option.

Parameter

name	the option name
value	the option value

Example

```
Table1.bindingInfo.setOption('grandLabel', 'Total');
```

The valid option/value pairs are shown below.

OPTION NAME	OPTION VALUE
grandLabel	Specify the label of the grand total. **Type** String **Example** Table1.bindingInfo.setOption ('grandLabel','Total');
grandTotal	Specify whether or not to add a grand total row. **Type** Boolean **Example** Table1.bindingInfo.setOption ('grandTotal',true);
inPlaceHeader	Specify whether or not to use in-place group header (without additional row). **Type** Boolean **Example** Table1.bindingInfo.setOption ('inPlaceHeader',true);
showGroupCols	Hide / unhide the group column contents. **Type** Boolean **Example** Table1.bindingInfo.setOption ('showGroupCols',false);
summaryLabel	Set the summary row label. **Type** string **Example** Table1.bindingInfo.setOption ('summaryLabel', 'Summary');
summaryOnly	Set to true in order to display only the summary data. **Type** Boolean **Example** Table1.bindingInfo.setOption ('summaryOnly', true);
setKeepHierarchy	Set to true to keep group hierarchy when only the summary data is being displayed. **Type** Boolean **Example** Table1.bindingInfo.setOption ('setKeepHierarchy', true);
setMergeGroup-Cells	Set to true to merge all the cells from the same group into one span cell. **Type** Boolean **Example** Table1.bindingInfo.setOption ('setMergeGroupCells',true);

See Also

Grouping and Summarization in Data Binding Wizard, in *Advanced Report Design*, for details on options.

SI.6.3 The groupedTable Property

The *groupedTable* property provides functions that return information about group boundaries.

GROUPEDTABLE.GETAVAILABLELEVELS(ROW)

Returns the available grouping levels for the specified row.

Parameter

row The specified row

Return

ScriptbleObject

Example

```
groupedTable.getAvailableLevels(2);
```

GROUPEDTABLE.GETGROUPFIRSTROW(ROW)

Returns the first row from the highest available grouping level at the specified row.

Parameter

row The specified row

Return

integer, -1 if not available

Example

```
groupedTable.getGroupFirstRow(2);
```

GROUPEDTABLE.GETGROUPLASTROW(ROW)

Returns the last row from the highest available grouping level at the specified row.

Parameter

row The specified row

Return

integer, -1 if not available

Example

```
groupedTable.getGroupLastRow(2);
```

GROUPEDTABLE.GETGROUPFIRSTROWWITHLEVEL(ROW, LEVEL)

Returns the first row at the specified row and grouping level.

Parameter

```
row    The specified row
level  The specified level
```

Return

```
integer, -1 if not available
```

Example

```
groupedTable.getGroupFirstRowWithLevel(2, 1);
```

GROUPEDTABLE.GETGROUPLASTROWWITHLEVEL(ROW, LEVEL)

Returns the last row at the specified row and grouping level.

Parameter

```
row    The specified row
level  The specified level
```

Return

```
integer, -1 if not available
```

Example

```
groupedTable.getGroupLastRowWithLevel(2, 1);
```

SI.7 Freehand Table Object

In addition to all of the properties of a regular *Table Object* , the Freehand Table has one property uniquely associated with it.

REGIONROW

Returns (read-only) the value of the current row index in the base table of a Freehand table.

Type

```
integer
```

Example

```
regionRow;
```

SI.8 Painter Object

The Painter object serves as the base for all float elements: *TextBox Object* , *Field Object* , *Chart Object*, Image Object, etc. It defines common properties for positions and sizes of float elements.

Super Class

Element Object

ANCHOR

Specifies the position of the element relative to the last element and the edge of the page.

Type

```
inetsoft.report.Position, or number array
```

Example

```
anchor = new inetsoft.report.Position(0.5, 1);
  or
anchor = [0.5, 1];
```

See Also

Property Dialog Boxes, in *Advanced Report Design*, for more information about anchors.

HYPERLINK

Specifies a hyperlink to a report, element target, or a URL.

Type

```
string or inetsoft.report.Hyperlink
```

Example

```
hyperlink = "Tutorial/Ad Hoc";
```

See Also

Hyperlinks, in *Advanced Report Design*, for basic information on hyperlinks.
Using the onClick Handler, for other approaches to hyperlinking.

IMAGE

Assigns an image (GIF or JPEG) to a Painter.

Type

```
inetsoft.report.Painter
```

Example

```
image = getImage("/inetsoft/report/design/images/
center.gif");
```

LAYOUT

Sets the layout of the Painter element, specifically how the element is printed if it is larger than the current available space on the page. the property has a value of `PAINTER_BREAKABLE` or `PAINTER_NON_BREAK`.

Type

```
integer
```

Example

```
layout = StyleConstant.PAINTER_BREAKABLE;
```

See Also

SI.41, *StyleReport Object*, for more information on constants.

MARGIN

Specifies the margin (in points) around the Painter element.

Type

`java.awt.Insets, or integer array`

Example

```
margin = new java.awt.Insets(1, 1, 1, 1);
  or
margin = [1, 1, 1, 1];
```

ROTATION

Specifies the angle to rotate the Painter element.

Type

`degrees from 0 to 360`

Example

```
rotation = 90;
```

SIZE

Specifies the size (inches) of the Painter element.

Type

`inetsoft.report.Size, or number array`

Example

```
size = new inetsoft.report.Size(2, 1);
  or
size = [2, 1];
```

WRAPPING

Specifies how contents wrap around the Painter element. Wrapping options are defined in the *StyleReport Object*:

– WRAP_NONE
– WRAP_LEFT
– WRAP_RIGHT
– WRAP_BOTH
– WRAP_TOP_BOTTOM

Type
integer

Example
```
wrapping = StyleConstant.WRAP_RIGHT;
```

See Also
SI.41, *StyleReport Object*, for more information on constants.

SI.9 TextBox Object

A *TextBox* element defines a rectangular area on the page in which to display a text paragraph. It can have borders, and can be positioned on a page using an anchor.

Super Class
Painter Object

See Also
Property Dialog Boxes, in *Advanced Report Design*, for more information about anchors.
Text and TextBox, for additional information on text presentation elements.

BORDER

Specifies the border around the TextBox as a *StyleReport Object* line style. This property is overridden by the '*borders*' property, if set.

Type
Line Styles integer

Example
```
border = StyleConstant.DOUBLE_LINE;
```

See Also
SI.41, *StyleReport Object*, for more information on constants.
Text and TextBox, for additional information on text presentation elements.

BORDERS

Specifies the individual borders for each side of the TextBox, as Insets object or array of form `[top,left,bottom,right]`. If set, this property overrides the '*border*' property setting.

Type
`java.awt.Insets`, or integer array of *Line Styles*

Example
```
borders =
[StyleConstant.DOUBLE_LINE,StyleConstant.NO_BORDER,
StyleConstant.THIN_LINE,StyleConstant.NO_BORDER];
```

See Also

Text and TextBox, for additional information on text presentation elements.

BORDERCOLOR

Specifies the TextBox border color.

Type

`java.awt.Color or integer RGB value`

Example

```
borderColor = 0xFF00FF;
```

See Also

Text and TextBox, for additional information on text presentation elements.

CORNERSIZE

Specifies the size of the corner (*x* radius and *y* radius) for a rounded-corner TextBox.

Type

`integer array: [x radius, y radius]`

Example

```
cornerSize = [30,50];
```

See Also

shape, for information on changing corner style using script.
TextBox, in *Advanced Report Design*, for information on changing corner style.
Text and TextBox, for additional information on text presentation elements.

TEXTID

Specifies a Text ID for the TextBox element. 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*.

Example

```
Text1.textID = 'Text1ID'
```

See Also

Localization/Internationalization in *Advanced Report Design* for more information on localization.

JUSTIFY

Specifies that text lines are aligned along the right edge.

Type
Boolean

Example
```
justify = true;
```

See Also
Text and TextBox, for additional information on text presentation elements.

PADDING

Specifies the size of the gap (in points) between the TextBox border and text contents.

Type
java.awt.Insets or integer array

Example
```
padding = new java.awt.Insets(1, 2, 1, 2);
   or
padding = [1, 2, 1, 2];
```

See Also
Text and TextBox, for additional information on text presentation elements.

SHAPE

Specifies the shape of the textbox as `BOX_RECTANGLE` or `BOX_ROUNDED_RECTANGLE`.

Type
integer

Example
```
shape = StyleConstant.BOX_ROUNDED_RECTANGLE;
```

See Also
SI.41, *StyleReport Object*, for more information on constants.
Text and TextBox, for additional information on text presentation elements.

SHADOW

Adds a drop-shadow to a TextBox.

Type
Boolean

Example
```
shadow = true;
```

See Also
Text and TextBox, for additional information on text presentation elements.

TEXT

Specifies the text contents to be displayed inside the TextBox.

Type
```
string
```

Example
```
text = 'Hello\nWorld';
```

See Also
Text and TextBox, for additional information on text presentation elements.

TEXTALIGNMENT

Specifies the text alignment with the TextBox. It should specify a bitwise OR of the *Alignments* values defined in the *StyleReport Object*.

Type
```
Alignments integer
```

Example
```
textAlignment = StyleConstant.H_CENTER;
```

See Also
SI.41, *StyleReport Object*, for more information on constants.
Text and TextBox, for additional information on text presentation elements.
Alignment Property, for more information on setting alignment.

SI.10 Chart Object

The Chart element presents a dataset as a visual graph.

Super Class
```
Painter Object
```

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.

SI.10.1General Properties

This section presents general Chart properties. See Appendix CR:, *Chart Script Reference* for full list of Chart properties.

QUERY

Specifies the query, Worksheet, or data model bound to the Chart element.

Type

`string`

Example

```
// Query:
Chart1.query = 'customers';
Chart1.query = 'Sample Queries/Fast Go Game'

// Global Worksheet:
Chart1.query = 'ws:global:State Stats';
Chart1.query = 'ws:global:Sales/Sales Explore';

// Local Worksheet:
Chart1.query = "ws:Sales Explore";

// User Worksheet:
Chart1.query = "ws:Annie:Sales Explore";

// Data Model:
Chart1.query = "Orders/logicModel::Return Model"
```

See Also

Binding Queries, for general information on the 'query' property.
Creating a Local Query, in *Advanced Data Modeling*, for information about local queries.
Creating a Local Worksheet, in *Advanced Data Mashup*, for information about local Worksheets.
Controlling Access to an Asset, in *Advanced Data Mashup*, for information about user Worksheets.
Modifying a Chart Data Binding, for an example of data binding.
query, for the equivalent Table property.
query, for the equivalent Section property.

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 creating a new graph.

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")
```

See Also

Hyperlinks, in *Advanced Report Design*, for basic information on hyperlinks.

Using the onClick Handler, for other approaches to hyperlinking.

GETTEXTID(TITLE)

Returns the Text ID assigned to the specified chart `title`. See *setTextID(title, textID)* to assign a Text ID using script.

Parameter

title	One of the following strings: 'X Axis Title' 'Y Axis Title' 'Secondary X Axis Title' 'Secondary Y Axis Title'

Example

```
Chart1.getTextID("X Axis Title");
```

See Also

Localization/Internationalization in *Advanced Report Design* for more information on localization.

SETTEXTID(TITLE, TEXTID)

Assigns the specified `textID` to the specified chart `title`. 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*.

Parameter

title	One of the following strings: 'X Axis Title' 'Y Axis Title' 'Secondary X Axis Title' 'Secondary Y Axis Title'
textID	String identifier to be mapped in SreeBundle file

Example

```
Chart1.setTextID("X Axis Title","XTitleID");
```

See Also

Localization/Internationalization in *Advanced Report Design* for more information on localization.

HIGHLIGHTED(NAME)

Identifies whether a highlight with the specified name has been applied to the chart (i.e., whether the highlight condition has been satisfied.).

Parameter

name	name of the highlight

Example

```
if(Chart1.highlighted.lowSales) {
  log('lowSales highlight was applied')
}
```

See Also

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

SIZE

Specifies the chart size in page units (with resolution fixed at 72 pixels per inch).

Type

Number	Array: [width, height]

Example

```
Chart1.size = [6,4];
```

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 use the code "`<br>`" to insert a line break into the tooltip, but other HTML markup is not supported.

Type

```
String              The tooltip to display
```

Example

```
Chart1.toolTip = "Company: {0} <br>Total Sales:
{1,number,$##,###.00}";
```

SI.11 Field Object

Field object is the base class for all other form elements, including button, image button, choice, text field, and text area.

Super Class

```
Painter
```

FORM

The name of the form to which this field belongs. The form name is used as the request name in the request object. It also serves to group all fields into one form. Fields sharing the same form name are considered part of the same form and are added to the request object when this form is submitted.

Type

```
string
```

Example

```
if(form == 'quarterly') {
   selectedItem = "Quarterly";
}
```

NAME

Sets the field name. The field name is used to construct a name value pair in a request object when a form is submitted.

Type

```
string
```

Example

```
name = "firstname";
```

SI.12 Button Object

The Button element can be used inside a form for submitting the form. (See *Button* in the *Advanced Report Design*.) When a button is clicked on a viewer, the form is submitted as a request object to the server. A replet can handle the request using an onClick JavaScript handler or a Java RequestListener.

Super Class
```
Field
```

TEXT

Determines the Button label string.

Type
```
string
```

Example
```
text = "OK";
```

SI.13 ImageButton Object

The ImageButton is similar to a button element. (See *Image Button* in the *Advanced Report Design*.) Instead of displaying a text label inside the button, an ImageButton displays an image icon.

Super Class
```
Field
```

RESOURCE

Affects the resource name of the image icon. The resource should be reachable on the CLASSPATH and point to a valid image file in one of the formats supported by Java (e.g., gif, jpg).

Type
```
string
```

Example
```
resource = "/app/images/go.gif";
```

SI.14 DateCombo Object

The DateCombo element can be used to display a date in a form. (See *Date Combo* in the *Advanced Report Design*.) It provides a calendar popup which can be used to select a date.

Super Class
```
Field
```

PROMPTTIME

This enables prompting for time in a DateCombo Element.

Type
```
Boolean
```

Example
```
promptTime = true;
```

SI.15 CheckBox Object

The CheckBox element adds an interactive checkbox to a form. (See *Check Box* in the *Advanced Report Design*.) It has a state of on or off. The checkbox value is added to the form request object when the form is submitted.

Super Class
```
Field
```

SELECTED

Affects the checkbox selection state. If the checkbox is selected, this property is true.

Type
```
Boolean
```

Example
```
selected = ((new Date()).getDay() == 1);
```

TEXT

Determines the checkbox label. The label is displayed at the right side of a checkbox icon.

Type
```
string
```

Example
```
text = "Last 30 days";
```

SI.16 Choice Object

A Choice element is a drop-down list for selecting a single item from a list of choices. (See *Choice* in the *Advanced Report Design*.) This element can be used inside a form for specifying a form value.

Super Class
```
Field
```

CHOICES

Determines the list of items to use in the drop-down list.

Type

`array`

Example

```
choices = ["Apple", "Orange", "Banana"];
```

SELECTEDITEM

This is the currently selected item in the list.

Type

`any object (Should be the same type as the items in choices array)`

Example

```
selectedItem = 'Orange';
```

VALUES

Determines the values that correspond to the choices in the drop-down list; `values[i]` is submitted when `choices[i]` is selected.

Type

`array`

Example

```
values = [1, 2, 3];    // numeric values
values = ['Robert', 'Sue', 'Eric'];    // string values
```

SI.17 TextField Object

This is an interactive form element for entering a single line of text data. (See *Text Field* in *Advanced Report Design*.) If a form contains a single TextField element, hitting enter key in the TextField also submits the form.

Super Class

`Field`

COLS

This is the number of columns in the TextField. This value is used to calculate the preferred size of a TextField element. It is ignored if the ‘size’ property has been set.

Type

`integer`

Example

```
cols = Math.max(cols, 15);
```

TEXT

This determines the text contents of a TextField.

Type

```
string
```

Example

```
if(text == "") {
   text = "default";
}
```

MAXLENGTH

This is used to set the maximum number of characters allowed in the TextField. (The limit is enforced in the deployed report, not in the Style Studio preview.)

Type

```
integer
```

Example

```
maxLength=10;
```

SI.18 TextArea Object

A TextArea is an interactive form element for entering multiple lines of text data. (See *Text Area* in *Advanced Report Design*.)

Super Class

```
Field
```

COLS

Affects the number of columns in a TextArea. This value is used to calculate the width of the field if the ‘size’ property is not set.

Type

```
integer
```

Example

```
cols = 20;
```

ROWS

Affects the number of rows in a TextArea. This value is used to calculate the height of the field if the ‘size’ property is not set.

Type
```
integer
```

Example
```
rows = 5;
```

TEXT

The text contents of a TextArea.

Type
```
string
```

Example
```
text = "First line\nsecond line";
```

SI.19 List Object

A List is an interactive form element for selecting one or more options. (See *List* in *Advanced Report Design*.)

Super Class
```
Field
```

The following properties are available:

CHOICES

Determines the set of labels displayed in the list.

Type
```
array
```

Example
```
choices = ["Apple", "Orange", "Banana"];
```

SELECTEDITEMS

Returns array of selected list values (not labels).

Type
```
any object (Should be the same type as the items in choices
array)
```

Example
```
selectedItem2 = List1.selectedItems[1];
```

VALUES

Determines the set of values corresponding to the labels in the list.

Type
array

Example
```
values = ["Apple", "Orange", "Banana"];
```

SI.20 CondPageBreak Object

Conditional page break defines a page break which depends on the amount of space left on the page. If the space is less than the conditional height, the printing advances to the next page. Otherwise the page break is ignored.

Super Class
Element

CONDHEIGHT

Affects the conditional height in inches.

Type
double

Example
```
condHeight = 2.5;
```

SI.21 Newline Object

Newline object advances printing to the next line. (See *Newline* in *Advanced Report Design*.)

Super Class
Element

COUNT

The number of lines that the Newline should advance.

Type
integer

Example
```
Newline1.count = 2;
// must be set before Newline element is rendered
```

SI.22 Section Object

A Section is normally used to display tabular data by repeating the section bands. The elements inside a Section can be accessed individually as their own objects. The Section element itself only defines a property for accessing the table binding.

Super Class

```
Element
```

See Also

Sections, for scripting issues related to Sections.
Tables and Sections, for additional functions pertaining to Sections.

SI.22.1General Properties

This section presents general properties of the Section object.

TABLELENS

Affects table data binding. This affects binding of the section to a TableLens object.

Type

```
tableLens object
```

Example

```
tableLens = new inetsoft.report.lens.DefaultTableLens(2,2);
```

TABLE

The data table associated with this section. It is a two-dimensional array holding the table data.

Type

```
two dimensional array
```

Example

```
for(var row = 0; row < table.length; row++) {
   for(var col = 0; col < table[row].length; col++) {
      // table[row][col] contains the data at that cell
   }
}
```

DATA

Affects the original table data. If no grouping or other summarization is defined for this section element, this object is the same as the 'table' property. Otherwise, it holds the original data of the table, while 'table' contains the summarized data. This is a two dimensional array holding the table cell data. To reference the data use the row number as the first array index and column number as the second array index.

Type

```
two dimensional array
```

Example

```
for(var row = 0; row < data.length; row++) {
   for(var col = 0; col < data[row].length; col++) {
```

```
        ...
    }
}
```

QUERY

Specifies the query, Worksheet, or data model bound to the Section element.

Type

`string`

Example

```
// Query:
Section1.query = 'customers';
Section1.query = 'Sample Queries/Fast Go Game'

// Global Worksheet:
Section1.query = 'ws:global:State Stats';
Section1.query = 'ws:global:Sales/Sales Explore';

// Local Worksheet:
Section1.query = "ws:Sales Explore";

// User Worksheet:
Section1.query = "ws:Annie:Sales Explore";

// Data Model:
Section1.query = "Orders/logicModel::Return Model";
```

See Also

Binding Queries, for general information on the ‘query’ property.
Creating a Local Query, in *Advanced Data Modeling*, for information about local queries.
Creating a Local Worksheet, in *Advanced Data Mashup*, for information about local Worksheets.
Controlling Access to an Asset, in *Advanced Data Mashup*, for information about user Worksheets.
Modifying a Chart Data Binding, for an example of data binding.
query, for the equivalent Chart property.
query, for the equivalent Table property.

BAND

The section bands in this Section. It is an associated array. Each band can be referenced using its index or its name, such as ‘header1’, ‘footer1’, ‘content’, etc.

Type

`associative array`

Example

```
band['content'].pageAfter = true;
```

SI.22.2The Section bindingInfo Property

The Section 'bindingInfo' property provides access to functions controlling the Section element's data binding. This includes hidden columns, summary columns, group columns, and option settings.

See *The Table bindingInfo Property* and *Chart bindingInfo Property* sections for the corresponding Table and Chart properties.

BINDINGINFO.ADDGROUPCOL(COLNAME)

Add a group column.

Parameter

colName name of the group column

Example

```
bindingInfo.addGroupCol('State');
```

BINDINGINFO.ADDHIDDENCOLUMN(COLNAME)

Hide a column.

Parameter

colName the column name

Example

```
bindingInfo.addHiddenColumn('customer');
```

BINDINGINFO.ADDSUMMARYCOL(COLNAME)

Add a summary column. A column can be used multiple times for summarization.

Parameter

colName column name on which to summarize

Example

```
bindingInfo.addSummaryCol('Price');
```

BINDINGINFO.GETFORMULA(I)

Get the summary formula for a column.

Parameter

i index of the column whose summary formula is desired

Return

The formula in String format

Example

```
bindingInfo.getFormula(2);
```

BINDINGINFO.GETGROUPCOL(I)

Get the group column name.

Parameter

i index of the group column

Return

Column name of the ith group column

Example

```
bindingInfo.getGroupCol(1);
```

BINDINGINFO.GETGROUPCOLCOUNT()

Get the total number of columns that are grouped.

Return

integer number of columns that are grouped

Example

```
bindingInfo.getGroupColCount();
```

BINDINGINFO.GETGROUPORDER(COLNAME)

Get the ordering of a group column.

Parameter

colName column whose grouping order is desired

Return

The order of the group in integer

Example

```
bindingInfo.getGroupOrder('State');
```

BINDINGINFO.GETSUMMARYCOL(I)

Get the summary column name.

Parameter

i index of the summary column

Return

Column name of the ith summary column

Example

```
bindingInfo.getSummaryCol(1);
```

BINDINGINFO.GETSUMMARYCOLCOUNT()

Get the total number of columns that are summarized.

Return
```
integer number of columns that are summarized
```

Example
```
bindingInfo.getSummaryColCount();
```

BINDINGINFO.GETTOPN(GROUPCOL)

Get the TopN setting.

Parameter
```
groupCol      group column name whose TopN setting is
desired
```

Return
```
The TopN value
```

Example
```
bindingInfo.getTopN('State');
```

BINDINGINFO.GETTOPNGROUPCOL(GROUPCOL)

Get the TopN group column. This function can be used to test if the TopN of a group column exists.

Parameter
```
groupCol      the specified group column
```

Return
```
The group column name if exists, null otherwise
```

Example
```
bindingInfo.getTopNGroupCol('State');
```

BINDINGINFO.GETTOPNSUMMARYCOL(GROUPCOL)

Get the TopN summary column.

Parameter
```
groupCol      the specified group column
```

Return
```
The summary column name if exists, null otherwise
```

Example
```
bindingInfo.getTopNSummaryCol('State');
```

BINDINGINFO.ISSUMMARIZE(COLNAME)

Use to check if a group should be summarized.

Parameter

colName name of the group column that needs to be checked for summarization

Example

```
bindingInfo.isSummarize('State');
```

BINDINGINFO.ISTOPNREVERSE(GROUPCOL)

Get the TopN reverse setting.

Parameter

groupCol the specified group column

Return

true if TopN reverse setting is true
false if TopN reverse setting is false or not found

Example

```
bindingInfo.isTopNReverse('State');
```

BINDINGINFO.REMOVEALLGROUPCOLS()

Remove all of the group columns.

Example

```
bindingInfo.removeAllGroupCols();
```

BINDINGINFO.REMOVEGROUPCOL(COLNAME)

Remove the specified group column.

Parameter

colName The name of the group column to be removed

Example

```
bindingInfo.removeGroupCol('State');
```

BINDINGINFO.REMOVEHIDDENCOLUMN(COLNAME)

Remove the hidden column.

Parameter

colName the column name

Example

```
bindingInfo.removeHiddenColumn('customer','state');
```

BINDINGINFO.REMOVESUMMARYCOL(COLNAME)

Remove the specified summary column.

Parameter

colName	The name of the summary column to be removed

Example

```
bindingInfo.removeSummaryCol('Price');
```

BINDINGINFO.SETFORMULA(COL, FORMULA)

Set the summary formula for a column.

Parameter

col	The index of the column to be summarized
formula	The summary formula for the column

Example

```
bindingInfo.setFormula(5, 'Sum<0>');
```

BINDINGINFO.SETGROUPORDER(COLNAME, ORDER)

Set the ordering of a group column.

Parameter

colName	column whose grouping order is to be set
order	order of this group column: StyleConstant.SORT_ASC: Ascending order StyleConstant.SORT_DESC: Descending order StyleConstant.SORT_NONE: Original order

Example

```
bindingInfo.setGroupOrder('State',
StyleConstant.SORT_DESC);
```

BINDINGINFO.SETSUMMARIZE(COLNAME, SUM)

Set whether a group (specified as group column) should be summarized.

Parameter

colName	name of the group column to be summarized
sum	Boolean value specifying whether to summarize

Example

```
bindingInfo.setSummarize('State', true);
```

BINDINGINFO.SETTOPN(GROUPCOL, N)

Set the TopN value for a group column.

Parameter

groupCol	the specified group column name

n	TopN integer value

Example

```
bindingInfo.setTopN('State', 10);
```

BINDINGINFO.SETTOPNGROUPCOL(GROUPCOL, COL)

Set the group to extract based on the grouping column.

Parameter

groupCol	the original group column
col	the new group column

Example

```
bindingInfo.setTopNGroupCol('State', 'Price');
```

BINDINGINFO.SETTOPNREVERSE(GROUPCOL, BOOLEAN)

Set to true to get bottom N groups.

Parameter

groupCol	the specified group column
bottom	true is reverse

Example

```
bindingInfo.setTopNReverse('State',true);
```

BINDINGINFO.SETTOPNSUMMARYCOL(GROUPCOL, COL)

Set the column to use for comparison when creating the TopN table.

Parameter

groupCol	the specified group column name
col	the summary column name

Example

```
bindingInfo.setTopNSummaryCol('State', 'Price');
```

SI.23 Section Band Object

A basic section consists of three Section Bands, a ‘Header’ band, a ‘Content’ band, and a ‘Footer’ band. The ‘Header’ and ‘Footer’ bands are used for summarized data. The ‘Content’ band always shows the detail record.

All properties in a section are associated with the section bands. The properties of a section band can be accessed using the Report Explorer (see *Report Explorer View* in the *Advanced Report Design*). Select a band on the Report Explorer element tree to display all properties for that band in the right-side Property pane.

ATBOTTOM

Determines if the Section Band is printed at the bottom of the page. Does not apply to main header, `band['header']`.

Type

`Boolean`

Example

```
band['header1'].atBottom = true;
band[0].atBottom = true;
```

BACKGROUND

Sets the Section Band background color.

Type

`java.awt.Color or integer value of RGB`

Example

```
band['header1'].background = java.awt.Color.green;
```

BREAKABLE

Controls if the Section Band can span across pages.

Type

`Boolean`

Example

```
band['header1'].breakable = true;
```

HEIGHT

Affects the height of the Section Band in pixels.

Type

`integer`

Example

```
band['header1'].height = 10;
```

SHRINKTOFIT

Setting this property to true, the Section Band will shrink in height if the elements in the band do not occupy the full size allocated to them at design time.

Type

`Boolean`

Example

```
band['header1'].shrinkToFit = true;
```

VISIBLE

If a Section Band is not visible, it will not be printed.

Type

```
Boolean
```

Example

```
band['header1'].visible = false;
```

PAGEBEFORE

If this property is true, this Section Band always starts at a new page.

Type

```
Boolean
```

Example

```
band['header1'].pageBefore = true;
```

PAGEAFTER

If this property is true, the report always advances to a new page after this Section Band.

Type

```
Boolean
```

Example

```
band['footer1'].pageAfter = true;
```

REPEATHEADER

Controls if this Section Band is repeated at the top of every page.

Type

```
Boolean
```

Example

```
band['header1'].repeatHeader = false;
```

SETPRESENTER(STRING, PRESENTER)

This option sets a presenter for a field. 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

string	field name
presenter	Presenter object

Example

```
setPresenter("Quantity",
newInstance('inetsoft.report.painter.BarPresenter'));
```

SUPPRESSBLANK

If this option is true, the Section Band is ignored if all elements in the band are blank.

Type

```
Boolean
```

Example

```
band['header1'].suppressBlank = true;
```

UNDERLAY

If this option is true, the next Section Band is printed on top of this band.

Type

```
Boolean
```

Example

```
band['header1'].underlay = true;
```

TOPBORDER

This property controls the top border of a Section Band, which can be any one of the available border styles.

Type

```
integer
```

Example

```
band['header1'].topBorder = StyleConstant.THIN_LINE;
```

LEFTBORDER

This property controls the left border of a Section Band, which can be one of the available border styles.

Type

```
integer
```

Example

```
band['header1'].leftBorder = StyleConstant.THIN_LINE;
```

BOTTOMBORDER

This property controls the bottom border of a Section Band, which can be one of the available border styles.

Type

```
integer
```

Example

```
band['header1'].bottomBorder = StyleConstant.THIN_LINE;
```

RIGHTBORDER

This property controls the right border of a Section Band, which can be one of the available border styles.

Type

```
integer
```

Example

```
band['header1'].rightBorder = StyleConstant.THIN_LINE;
```

SI.24 Separator Object

A Separator is a horizontal line across a page. It can be drawn using any one of the line styles supported by Style Intelligence.

Super Class

```
Element
```

SEPARATORADVANCE

Sets the amount of vertical space to advance below a Separator.

Type

```
integer
```

Example

```
separatorAdvance = 0;
```

STYLE

The Separator line style, specified as one of the line styles defined in the *StyleConstant Object*.

Type

```
integer
```

Example

```
style = StyleConstant.ULTRA_THIN_LINE;
```

SI.25 Space Object

A Space element defines horizontal white space. The spacing is defined as points.

Super Class

```
Element
```

SPACE

Affects the number of points to advance to the right.

Type

```
integer
```

Example

```
space = 20;
```

SI.26 TOC Object

The TOC element defines a Table Of Contents in a report. The items in a TOC element are automatically discovered from heading elements. A report only needs to specify the style of the TOC.

Super Class

```
Element
```

TOC

Determines TOC style. Options are any of the pre-defined styles defined in the TOC global object.

```
TOC.DEFAULT
TOC.CLASSIC
TOC.DISTINCTIVE
TOC.FANCY
TOC.MODERN
TOC.FORMAL
TOC.SIMPLE
```

Type

```
inetsoft.report.TOC
```

Example

```
toc = TOC.SIMPLE;
```

SI.27 Bean Object

A Report Bean is a reusable, customizable composite report element. The Bean object represents a report bean inside a report. Report Bean's

properties are defined when a report bean is created in Style Studio. The parameters defined on the bean can be accessed through the bean object.

Super Class

```
Element
```

See Also

Report Beans, for information and examples on scripting report beans.
Report Bean, in *Advanced Report Design*, for examples of bean usage.

BEAN PROPERTIES

The Bean properties can be accessed using the regular property syntax or using the property name as the index on the Bean element.

Type

```
Type of the property depends on how the property is defined
```

Example

```
Bean1.title = 'Example';
Bean1['report title'] = 'Report Bean Example';
```

BEANNAME

The name of the Bean used in this Bean element. This property can be used to select a bean from the Bean library to be used in this element.

Type

```
string
```

Example

```
beanName = 'title bean';
```

ELEMENTS

Provides access (from the parent report) to elements contained within the bean.

Example

```
Bean1.elements.Text1.text= 'hello';
Bean1.elements.Table1.table[0][0] = 'Order ID';
```

LINEAFTER

Determines whether the element in the report following the bean appears on a new line.

Type

```
Boolean
```

Example

```
lineAfter = false;
```

SI.28 XType Data Types

The XType object contains all data type constants. The constants are used to specify the data type of a parameter when defining report parameters. The following types are defined in the XType object.

TYPE	DESCRIPTION
BOOLEAN	true or false
BYTE	single byte number
CHAR	character
DATE	date with no time component
DOUBLE	double number
ENUM	user defined string constants
FLOAT	float number
INTEGER	integer number
LONG	long integer number
SHORT	short integer number
STRING	text string
TIME	time
TIME_INSTANT	time and date

SI.29 Toolbar Object

The 'Toolbar' object allows you to selectively enable or disable button in the report toolbar (in the User Portal). It offers two functions, `setActionVisible()` and `isActionVisible()`, which are described in the following sections.

TOOLBAR.ISACTIONVISIBLE(NAME)

Returns the visibility of the specified report button.

Parameter

name Name of a custom or default button:
"PAGE DISPLAY"
"FIND"
"FIND_NEXT"
"PDF EXPORT"
"EMAIL"
"PRINT"
"SERVER PRINT"
"REPORT EXPLORER"
"EXPORT"
"REFRESH"
"RETURN"
"CUSTOMIZE"
"SAVE AS"
"AD HOC"
"ZOOM"
"REPORT NAME"

```
"CLOSE"
```

Example

```
Toolbar.isActionVisible("REPORT EXPLORER");
```

TOOLBAR.SETACTIONVISIBLE(NAME,BOOLEAN)

Sets the visibility of the specified report button. Use `'ALL'` as the '`name`' parameter to apply to all buttons.

Parameter

name Name of a custom or default button:
"PAGE DISPLAY"
"FIND"
"FIND_NEXT"
"PDF EXPORT"
"EMAIL"
"PRINT"
"SERVER PRINT"
"REPORT EXPLORER"
"EXPORT"
"REFRESH"
"RETURN"
"CUSTOMIZE"
"SAVE AS"
"AD HOC"
"ZOOM"
"REPORT NAME"
"CLOSE"
"ALL" or "all"

boolean true, to make the button visible
false, to make the button invisible

Example

```
Toolbar.setActionVisible("REPORT EXPLORER",false);
```

SI.30 PageBreak Event Object

A PageBreak Event object is available in page break listener scripts. The event object can be used to check the information on the current page, such as the first element on the page and the last element on the page.

Note: A **pageEvent** object is available and can be accessed by scripts in the header and footer to access PageBreak event information. It contains the same information as the PageBreak event but can be accessed outside the PageBreak listener.

See Also
onPageBreak Handler, for an illustration of using the PageBreak handler.

ELEMENTID

The element ID of the last element on the page.

Type

```
string
```

Example

```
if(event.elementID == 'Table1') {
   ...
}
```

REGION

If the last element is a table element, this is the table region of the table on the current page.

x	left-most column index
y	top-most row index
width	number of columns
height	number of rows

Type

java.awt.Rectangle

Example

```
for(var row = event.region.y; row < event.region.y +
event.region.height; row++) {
   for(var col = event.region.x; col < event.region.x +
      event.region.width; col++) {
      ...
   }
}
```

FIRSTELEMENTID

The element ID of the first element on the page.

Type

string

Example

```
if(event.firstElementID == 'Table1') {
   ...
}
```

FIRSTREGION

If the first element is a table element, this is the table region of the table on the current page.

Type

java.awt.Rectangle

LASTPAGE

This property is true if this is the last page of the return.

Type

Boolean

Example

```
if(event.lastPage) {
   ...
}
```

PAGEINDEX

The current page number (first page = 0).

Type

```
number
```

Example

```
Text1.text = event.pageIndex;
```

SI.31 Selection Event Object

A Selection Event object is accessible using the global ‘event’ object in an onClick event handler. The event can be used to query event information.

ELEMENTID

Affects the ID of the element where the event occurred.

Type

```
string
```

Example

```
var id = event.elementID;
```

ITEM

Affects the location within the element where the mouse is clicked. This is an EventPoint object, where the x value is the column index and the y value is the row index.

Type

```
inetsoft.report.event.EventPoint
```

Example

```
if(event.item.y == 0) {
   ...
}
```

SI.32 Request Event Object

A Request Event is accessible in an *onClick Handler* script that is attached to a Form field element. The Request element contains the values submitted from a form. It is stored in the global ‘event’ object.

REPLETREQUEST

This is the request object containing all form values.

Type
```
inetsoft.sree.RepletRequest
```

Example
```
showReplet("report/details", event.repletRequest);
```

SI.33 Query Parameter Object

If a report has embedded query binding, the parameters used by the queries are accessible as properties in a '*parameter*' object. Each parameter is a property in the 'parameter' object and can be accessed using regular JavaScript object property notation.

```
text = "Customer list for " + parameter.state;
```

The parameter values can be modified by *onLoad Handler* script. The modifications made in an onLoad script will be used when the queries are executed. Modifications made in all other scripts are not effective in queries.

SI.34 RepletRequest Object

A report is created with optional replet request parameters. The parameters may be defined when a replet is registered with a server, or interactively entered by end users when the report is created. The request parameters can be accessed inside a script like the query parameters. The length of request parameters is given by *request.length*.

The report creation request parameters are stored in a JavaScript object, 'request'. The parameters can be accessed as properties in the object.

```
if(request.chartType == 'BAR') {
   Chart1.chartStyle = Chart.CHART_BAR;
}
```

The request parameter values are read-only. Modifying the parameter values has no effect on the report creation process.

See Also

parameter, for an alternate method of accessing report parameters.

SI.35 ReportSheet Object

All scripts in a report can access the Java ReportSheet object as a top-level object: 'reportsheet'. All public methods in ReportSheet class are available.

Name
```
reportsheet
```

Type
```
inetsoft.report.ReportSheet
```

Example

```
reportsheet.background = java.awt.Color.lightGray;
```

SI.36 Style Intelligence Global Object

Style Intelligence scripts can access a set of functions only available in the server environment. The functions are generally used in *onClick Handler* scripts to create report viewer commands. In the Script Editor 'Functions' tree, these are listed under 'StyleReportEE/Global'.

SHOWREPLET(REPLET, REQUEST)

This function creates a command to display a report, with optional report request parameters. It should only be used in the onClick script.

Parameter

replet	path/name of report in repository
request	an array of parameter name-value pairs

Example

```
showReplet("sales/monthly", [['month',3], ['type','all']]);
```

SHOWURL(URL, TARGET)

This function creates a command to display a document at the specified URL. Target frame name is optional.

Parameter

url	URL string
target	target frame name

Example

```
showURL("http://www.inetsoft.com")
```

PROMPTPARAMETERS(REQNAME)

This function creates a command to prompt for parameter values for the specified request. The parameters must be declared in a replet explicitly.

Parameter

reqname	request name

Example

```
promptParameters('customize');
```

SENDREQUEST(REQUEST)

This function creates a command to generate a request to a replet.

Parameter

request	an array of pairs containing parameter name-value pairs

Example

```
sendRequest([['name','value']]);
```

REFRESH()

This function creates a command that requests the viewer to refresh the report. This causes the report to be re-generated on the server.

Example

```
refresh();
```

SCROLLTO(EID, ROW)

This function creates a command to scroll the report to the specified location.

Parameter

```
eid    target element ID
row    if the element is a table, the target row number
```

Example

```
scrollTo('Table1', 200);
```

SHOWSTATUS(MESSAGE)

This function creates a command to show a message on the status bar of the viewer.

Example

```
showStatus('This is a status');
```

PVIEWSHEET

For a report which is accessed via drilldown (hyperlink or auto-drill) from a Viewsheet, the `pviewsheet` property provides access to the current settings of components within the parent Viewsheet.

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 by using the following syntax within report script:

```
pviewsheet.Company.selectedObjects
```

See Also

Viewsheet Object Reference, in *Advanced Dashboard Scripting*, for a list of accessible Viewsheet component properties.

pviewsheet, in *Advanced Dashboard Scripting*, to access component settings from within a drilldown Viewsheet.

Auto-Drilldown, in *Advanced Data Modeling*, for information on configuring data-level hyperlinks.
Hyperlinks, in *Advanced Dashboard Design*, for information on Viewsheet drilldowns.

SI.37 Replet Object

In the User Portal, a report script can access its own instance of BasicReplet. This allows a script to add interactive features to the report without implementing *onClick Handler*. You can access the replet object with the global JavaScript property '`replet`'.

PARAMETER

Used to access a report parameter definitions. Use `parameter.length` to find the number of report parameters.

Type
```
array of parameters
```

Examples
```
parameter.state = 'NJ';
parameter.companies[0] = 'CA';
parameter['create'].addBoolean('Include All');
```

See Also
HTTP Request, Session, and Principal, for information on accessing http session.
RepletParameters Object, for special user and principal parameters.

SORTONHEADER

If this property is true, actions are automatically generated to sort a table when a column header is clicked.

Type
```
Boolean
```

Example
```
sortOnHeader = true;
```

PAGESIZE

Sets the page size of this report.

Type
```
page size in inches, an instance of inetsoft.report.Size
```

Example
```
pageSize = StyleConstant.PAPER_LEGAL;
```

ORIENTATION

Sets the orientation of the report, either landscape or portrait.

Type

```
orientation          StyleConstant.LANDSCAPE or
                     StyleConstant.PORTRAIT
```

Example

```
orientation = StyleConstant.LANDSCAPE;
```

PRINCIPAL

The user ID of the user who opened this report. It can be used to add a parameter to a query to perform a query based on user ID.

Type

```
string
```

Example

```
parameter['user_id'] = replet.principal;
```

ROLES

The roles assigned to the end user. A user may have multiple roles. Roles are presented in an array.

Type

```
array of strings
```

Example

```
for(var r = 0; r < roles.length; r++) {
   if(roles[r] == 'region_manager') {
      summaryTable.visible = true;
   }
}
```

ADDLINK(EID, [ITEM], REPLET [,REQUEST])

Adds a hyperlink to the specified element which targets another report.

Parameter

```
eid          element ID
item         row/column specifier,
             an inetsoft.report.event object
replet       report name
request      report request,
             an inetsoft.sree.RepletRequest instance,
             or a parameter/value array (array of pairs)
```

Example

```
replet.addLink('Text1', 'Orders/Order Details');
```

Example

```
var req = new inetsoft.sree.RepletRequest("customize");
req.setParameter("name", "value");
replet.addLink("Text2",
   new inetsoft.report.event.EventPoint.at(new
   java.awt.Point()), "report/details", req);

// this is equivalent to
replet.addLink("Text2", [["name", "value"]]);
```

ADDLINKURL(EID, ITEM, URL, TARGET)

Adds a hyperlink to an element, which opens a URL in a specified window.

Parameter

eid	element ID
item	row/column specifier, an inetsoft.report.event.EventPoint object
url	a URL string
target	A new frame name or predefined HTML target, e.g., _blank, _self, _parent, _top.

Example

```
replet.addLinkURL("home", undefined,
   'http://www.inetsoft.com', "frameName");
```

ADDREQUEST(EID, ITEM, REQUEST)

Adds a request link for an element (item). If the element/item is clicked, the request is sent to the replet as a RequestEvent.

Parameter

eid	element ID
item	row/column specifier, a inetsoft.report.event.EventPoint object
request	report request object, inetsoft.sree.RepletRequest

Example

```
replet.addRequest("link", undefined, [["name", "value"]]);
```

ADDSTATUS(EID, ITEM, MESSAGE)

Associates a status message with an element. The ‘message’ parameter is displayed as a tooltip when the user moves the mouse over the element. The message is also displayed in the browser status bar if permitted by browser security. (Modern browsers disable status bar scripting by default.)

Parameter

eid	element ID
item	row/column specifier, a inetsoft.report.event.EventPoint object
message	status message

Example

```
replet.addStatus("Title", undefined, "This is an example
report");
```

See *Adding Tooltips to Elements* for further examples.

ADDPOPUPMENU(EID, ITEM, ITEMS)

Adds a popup menu for an element. The menu is popped when a menu trigger mouse button is pushed inside the element (item). When a popup menu is added to an element, the action to be performed should also be specified. The actions are specified using addMenuAction() function.

Parameter

eid	element ID
item	row/column specifier, a inetsoft.report.event.EventPoint object
items	an array of menu item strings

Example

```
replet.addPopupMenu("Table1",
   new inetsoft.report.event.EventPoint(0, -1), ["Sort"]);
```

ADDMENUACTION(EID, ITEM, MENUITEM, ACTION)

Adds a menu action to an element. The action is performed when a menu item associated with the element is selected.

Parameter

eid	element ID
item	row/column specifier, a inetsoft.report.event.EventPoint object
menuitem	menu item string
action	an action command, inetsoft.sree.Repletcommand

Example

```
replet.addMenuAction("Text1", undefined, "Details",
showReplet("report/details"));
```

ADDFORMLINK(FORM, REPLET)

Adds a hyperlink from a form to a replet. When the form is submitted, the request is used to create the specified replet.

Parameter

form	form name
replet	path/name of report

Example

```
replet.addFormLink("Form1", "report/details");
```

REMOVEALLACTIONS()

Removes all the actions (i.e., hyperlink and status actions) associated with a replet.

Example

```
replet.removeAllActions();
```

SI.38 ParentReport Object

The nested scope of the bean and sub-report scripting environment results in any variable or element in a bean or a sub-report hiding the objects with same names in the parent report. However, such objects in the parent report can be accessed using the 'parentReport' object. For example, say that a bean has a text element whose id is set to 'Title' and its parent report also has a text element with the id equal to 'Title'. The 'Title' text element in the parent report can be accessed using

```
parentReport.Title
  or
parentReport['Title']
```

SI.39 RepletParameters Object

A RepletParameters object contains the replet parameter definitions. It is accessed through `replet.parameters.create`, and can be used to add or modify replet parameter definitions.

The `replet.parameters.create.parameter` array is available as a global property called `parameter`. Here is a list of system parameters that are available in the global `parameter` array:

PARAMETER NAME	PURPOSE
USER	The string containing the user id.
ROLES	An array containing the role names. This can be used with the 'one-of' condition operator.
__principal__	Retrieves the SRPrincipal object in both report script and VPM triggers. Works in scheduled reports too.
__service_request__	Gets the HttpServiceRequest, only valid in a report script when run live by a user.
Replet.RunType	Returns the context of this report: live, pregenerated, or archive (if the SRO is re-executed due to ad hoc changes)

ADDBOOLEAN(STRING, BOOLEAN)

Adds a Boolean parameter to the parameter list. A Boolean parameter is prompted using a checkbox.

Parameter

```
string      parameter name
boolean     default Boolean value
```

Example

```
replet.parameters['create'].addBoolean('showChart', true);
```

ADDCHOICE(STRING, OBJECT, ARRAY)

Adds a choice parameter to the parameter list. A choice parameter is prompted using a combo box.

Parameter

```
string        parameter name
object        default value
array         array of items to select from
```

Example

```
replet.parameters['create'].addChoice('border', 0,
['Thin Line', 'Double Line']);
```

ADDDATE(STRING, DATE)

Adds a date parameter to the parameter list. A date parameter is prompted using a date combo box.

Parameter

```
string        parameter name
date          default date value
```

Example

```
replet.parameters['create'].addDate('start_time', new
Date());
```

ADDDATETIME(STRING, DATE)

Adds a date/time parameter to the parameter list. A date/time parameter is prompted using a combo box or a text field.

Parameter

```
string        parameter name
date          default date/time value
```

Example

```
replet.parameters['create'].addDate('start_time', new
Date());
```

ADDLIST(STRING, DEFAULT, LIST)

Adds a list parameter to the parameter list. A list is presented as a multi-selection list on the prompt window.

Parameter

```
string        parameter name
default       array of default selections
list          array of selection items
```

Example

```
replet.parameters['create'].addList('type', ['Grouping'],
['Grouping', 'Summarize']);
```

ADDOPTION(STRING, DEFAULT, LIST)

Adds a group of options to the parameter list. An option group is presented as a group of checkbox controls.

Parameter

string	parameter name
default	array of default selections
list	array of selection items

Example

```
replet.parameters['create'].addOptions('type',
['Grouping'], ['Grouping', 'Summarize']);
```

ADDPARAMETER(STRING, DEFAULT, FORMAT)

Adds a parameter with optional formatting. The parameter is prompted using a text field. If the format is specified, it is used to convert the object to and from its string representation.

Parameter

string	parameter name
default	parameter default value
format	java.text.Format object for formatting object

Example

```
replet.parameters['customize'].addParameter('curr',
   new java.util.Date(),
   new java.text.SimpleDateFormat('yyyy-MM-dd'));
```

ADDPASSWORD(STRING)

Adds a password parameter to the parameter list. A password is prompted using a text field with hidden text.

Parameter

string	parameter name

Example

```
replet.parameters['create'].addPassword('password');
```

ADDRADIO(STRING, DEFAULT, LIST)

Adds a radio button group parameter to the parameter list. The parameter is presented as a group of radio buttons.

Parameter

string	parameter name
default	default selection

list	array of selection items

Example

```
replet.parameters['create'].addRadio('type', 'Grouping',
['Grouping', 'Summarize']);
```

ADDTEXTAREA(STRING, DEFAULT, ROWS, COLS)

Adds a text area parameter to the parameter list. The parameter is prompted using a multi-line text area control.

Parameter

string	parameter name
default	default text
rows	number of rows
cols	number of columns

Example

```
replet.parameters['create'].addTextArea('comments', '', 5,
40);
```

ADDTIME(STRING, DATE)

Adds a time parameter to the parameter list. The parameter is prompted using a time spinner or a text field.

Parameter

string	parameter name
date	default value

Example

```
replet.parameters['create'].addTime('time', new Date());
```

CLEAR()

Removes all parameter definitions.

Example

```
replet.parameters['create'].clear();
```

SETALIAS(NAME, ALIAS)

Sets the parameter alias. The alias is used as the label of the parameter on the prompting window.

Parameter

name	parameter name
alias	parameter alias

Example

```
replet.parameters['create'].setAlias('time', 'Start Time');
```

SETVISIBLE(STRING, BOOLEAN)

Sets the visibility of a parameter. If a parameter is hidden, it is not prompted on the parameter window. The default value is used as the value of the parameter.

Parameter

string	parameter name
boolean	visibility flag

Example

```
replet.parameters['create'].setVisible('time', false);
```

SI.40 PDF Security Provider Subsets

This is the subset required in the report's *onLoad Handler* script when used in conjunction with an appropriate PDF security provider, and accessed through the docInfo object.

The following properties are available:

- `pdf.password.owner` – password that provides owner access to document (as entered in Adobe software).
- `pdf.password.user` – password that provides user access to document (as entered in Adobe software); required to use any security property.
- `pdf.permission.add` [`true/false`] – provides the ability to add/modify; maps to "Commenting" feature.
- `pdf.permission.change` [`true/false`] – provides the ability to change anything; maps to "Changing the Document and Document Assembly" feature.
- `pdf.permission.copy` [`true/false`] – provides the ability to copy; maps to "Content Copying or Extraction" and "Content Extraction for Accessibility" features.
- `pdf.permission.print` [`true/false`] – provides the ability to print; maps to "Printing" feature.

To use these security features, 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.

SI.41 StyleReport Object

The *StyleReport* object has been deprecated. Please use the *StyleConstant Object* instead.

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

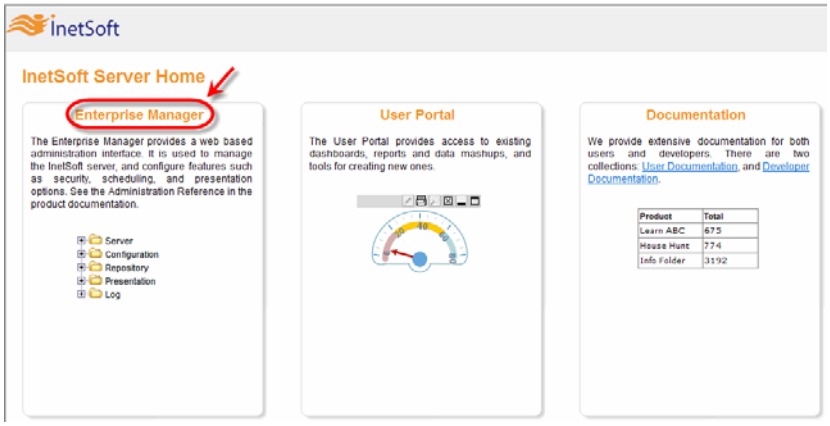
Enterprise Manager is a web-based tool that allows you to administer the report server. To launch the Enterprise Manager in a Microsoft Windows environment, follow these steps:

1. In the Windows *Start* menu, select *Style Intelligence* > *Style Intelligence Server.*

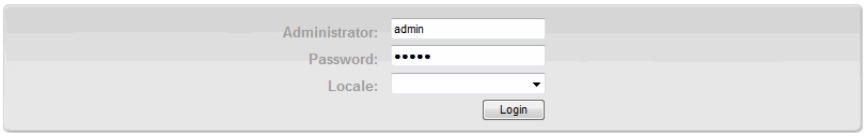
   This launches the default Tomcat web server on the local machine.

2. Point a web browser to the Style Intelligence webapp homepage with default URL `http://localhost:8080/sree/`.

3. Click the 'Enterprise Manager' link. This opens the Enterprise Manager login screen.



4. 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:

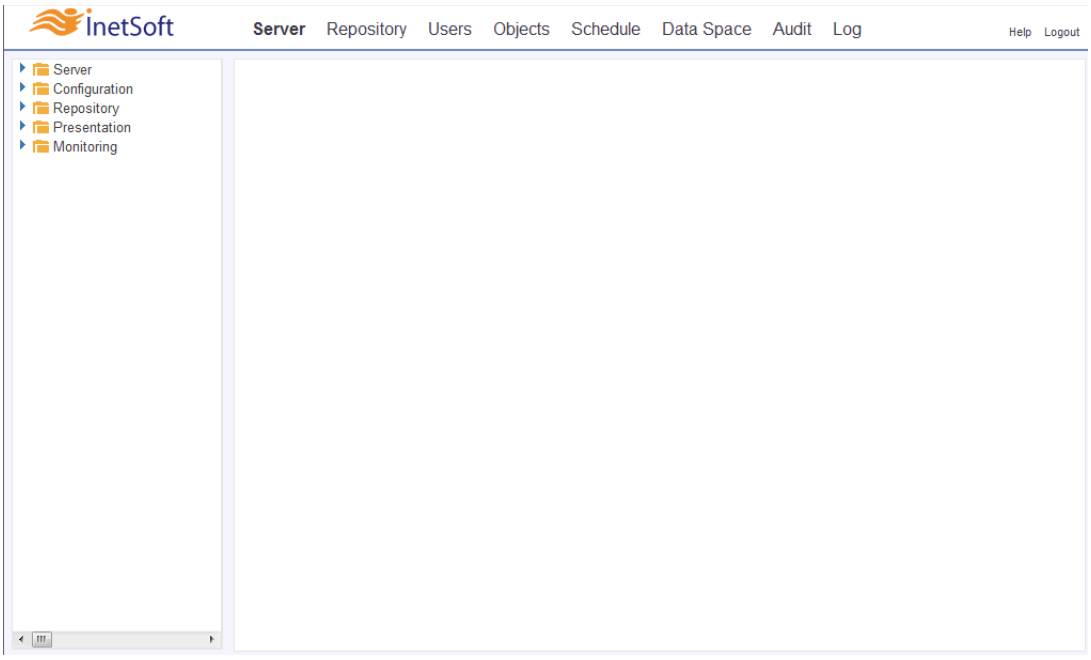
```
http://{servername}:{port}/sree/EnterpriseManager
```

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.
- *asset.dat* or *asset.dat.d* directory: Contains the XML specification for SQL queries, database connections, data models, and Virtual Private Models (VPMs), Data Worksheets, and Viewsheets.
- *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:

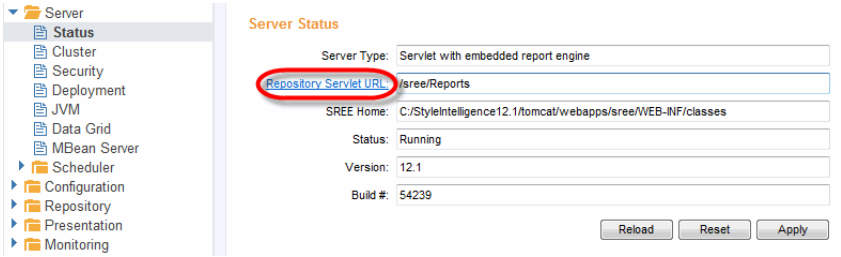
- `dashboard-layouts.xml` (required for end-users to construct dashboards)

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

Specify the location of the reusable component library file 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 Repository Tab

The **Repository** 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 **Repository** 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*.

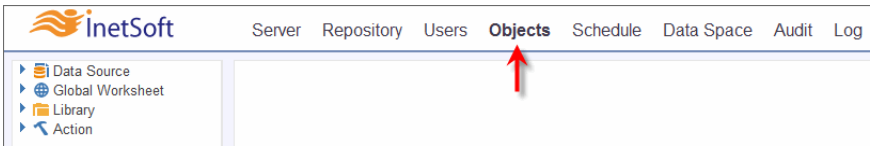
See Also

Specifying a Security Provider, for information about setting a security provider.

2.4 Objects Tab

Note: Report permissions are managed under the *Repository 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

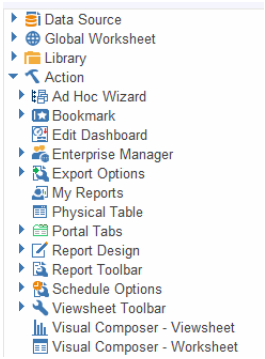
Manage Data Worksheet permissions.

Library

Manage permission for reusable components stored in the library (*stylereport.srl.d*): 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.
- **Edit Mobile Devices**: Permission to edit device classes or create new device classes for dashboard layouts. See *Defining a New Device Class* in *Advanced Dashboard Design* for more information on device classes.
- **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.
  - **Repository**: Provides access to the **Repository** tab in Enterprise Manager. (See *Repository 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.

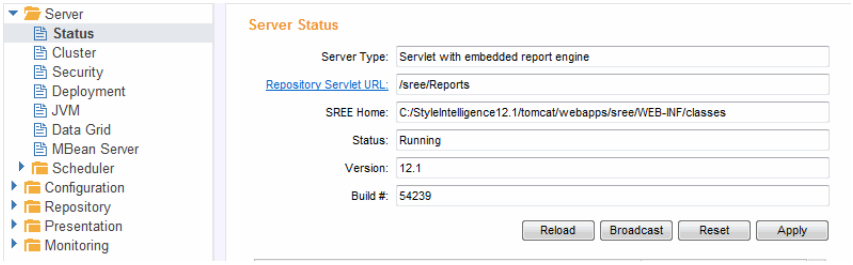
3 Server Environment

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



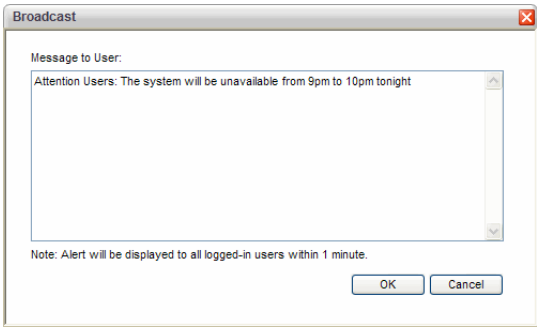
Note: The 'Restart' button does **not** restart the server. It merely refreshes the server properties by reloading the *sree.properties* file. In most cases these properties are refreshed *automatically* when *sree.properties* is modified, so there is no need to use the 'Restart' button.

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.



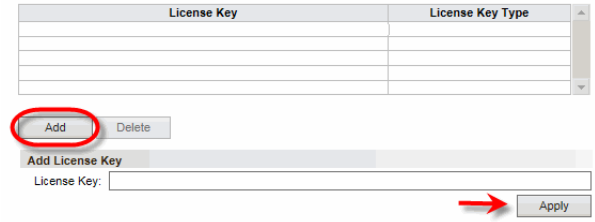
When you have entered the desired message, click ‘OK’. The message will be displayed to active users within about one minute after sending. This can be used to warn of system-wide scheduled maintenance.

Administering License Keys

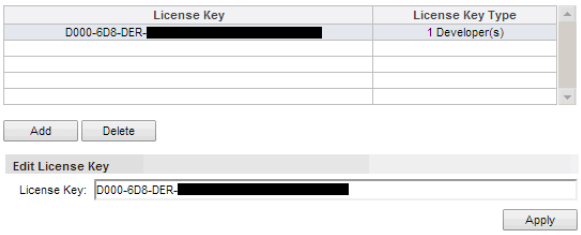
To add or modify a License Key in Enterprise Manager, open the ‘Server’ > ‘Status’ page (under the **Server** tab).

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

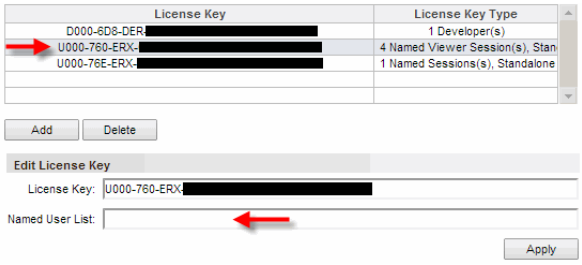
Configuring Server Clustering, for information on setting licenses in a clustered environment.

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.

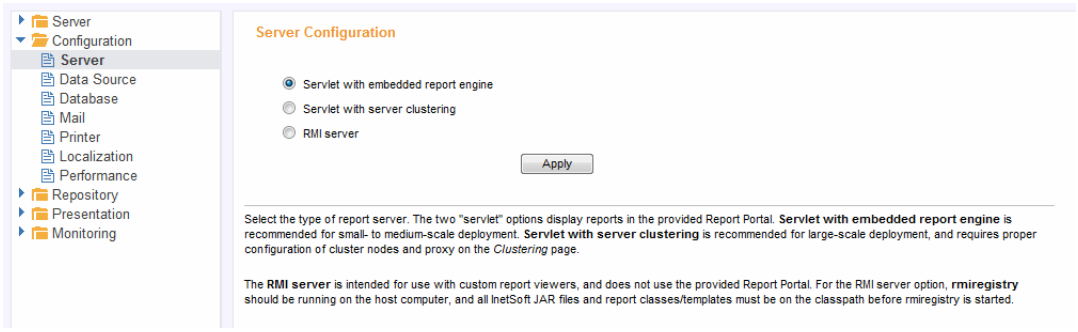
You can assign the ‘Designer’ role to more users than the maximum number allowed by the license. If the maximum number of licensed users is reached, the next named user who attempts to open Visual Composer will receive a notification that all licenses are in use.

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
```

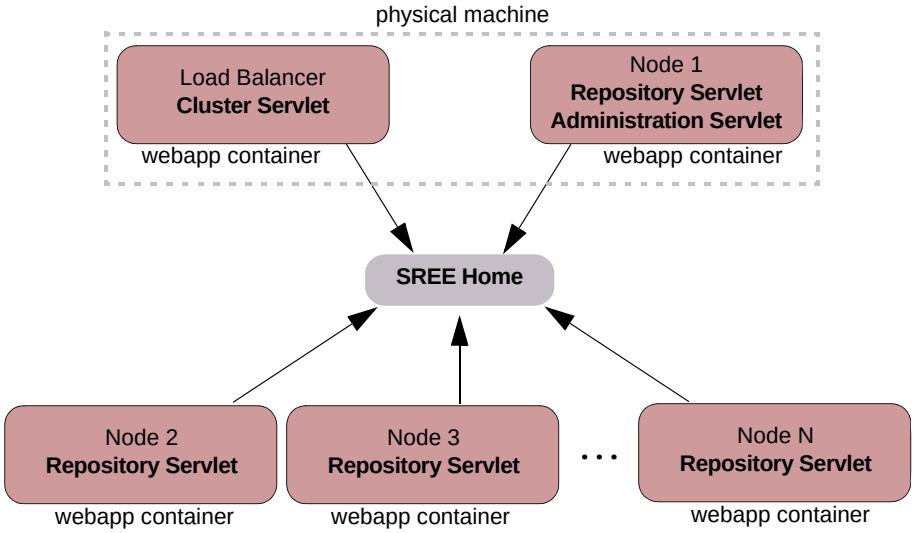
You can stop and restart the RMI server using the ‘Stop’ and ‘Restart’ buttons on the ‘Server Status’ page.

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 balancer automatically directs the server requests to available cluster nodes to achieve maximum throughput.

A typical cluster configuration is shown below. Note that the load balancer can be installed on the same physical machine as Node 1 if desired. In most cases a hardware load balancer is used in place packaged load balancer application.



It is highly recommended that you discuss your clustering needs with an InetSoft technical resource before you install cluster nodes or proxy. The following section provide information about managing the cluster once the nodes have been correctly installed.

3.3.1 Defining the Cluster Nodes

Once you have installed the cluster nodes and load balancer, 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.

3.3.2 Assigning Machine-Specific Properties

The nodes in a cluster all share a common server repository, 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*.

3.3.3 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*.

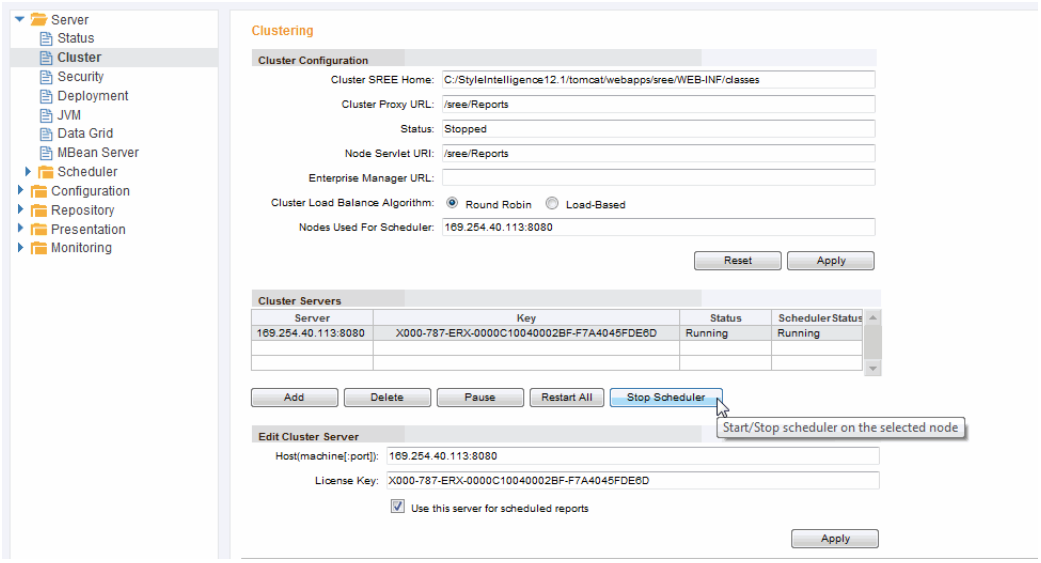
Running the Scheduler on a Cluster Node

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.

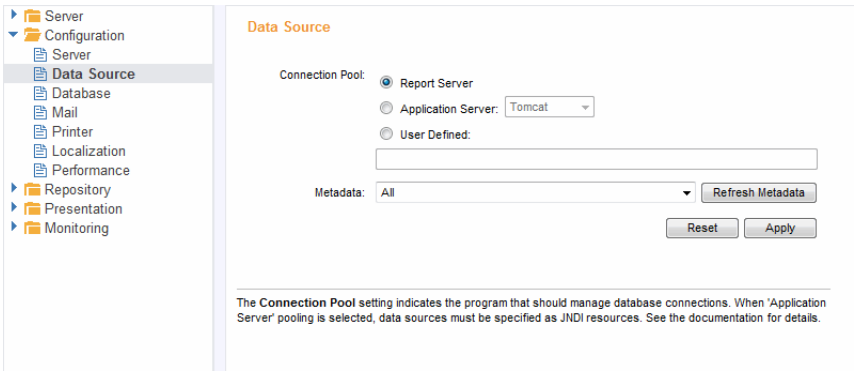
3. Press the 'Start Scheduler' button to start the Scheduler on the selected node.



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' > 'Data Source' page allows you to configure connection pooling and refresh metadata.



3.4.1 Connection Pooling

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.

3.4.2 Refreshing Metadata

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.

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.

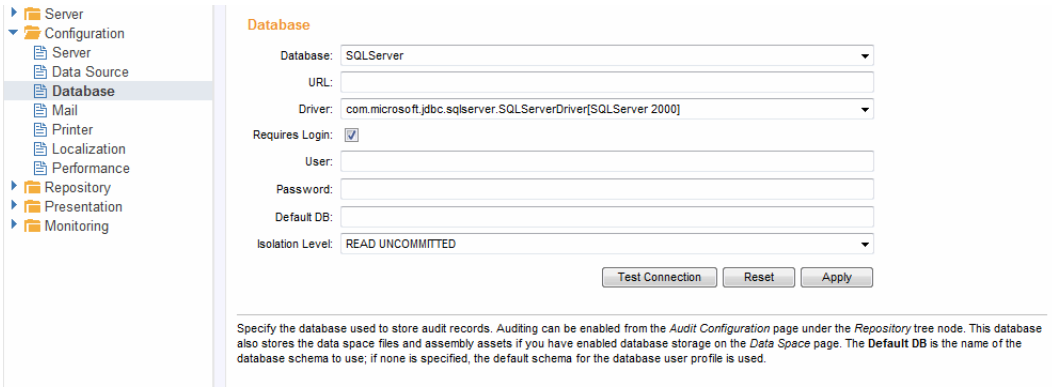
You can use a database for persistent storage of the following items:

- Data space (report templates, data sources, queries, Data Worksheets, etc.)
- Asset repository (*asset.dat*/*asset.dat.d*)
- 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 3. 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.

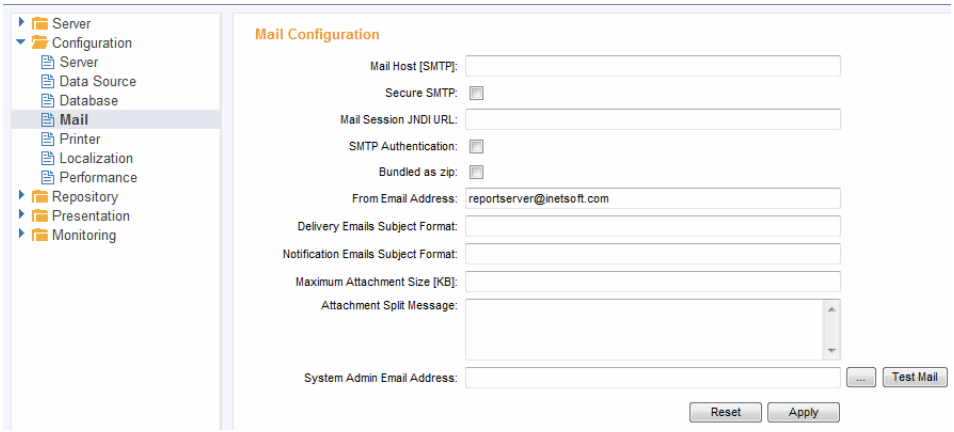
3.6 Configuring Server Email

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 4. 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

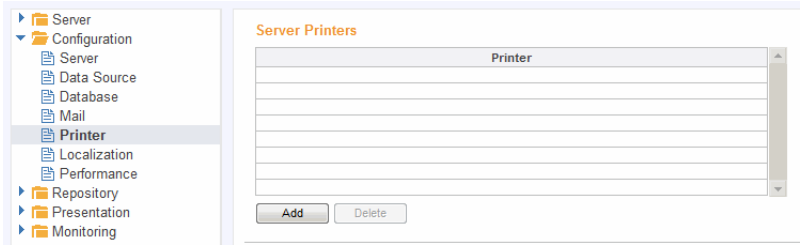
The 'Maximum Attachment Size [kB]' sets an upper limit for the size of an exported report file sent as a single attachment. If exceeded, the file is split into multiple files (binary split) and delivered via multiple emails. It is the responsibility of the recipient to concatenate all the fragments into 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 5. 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

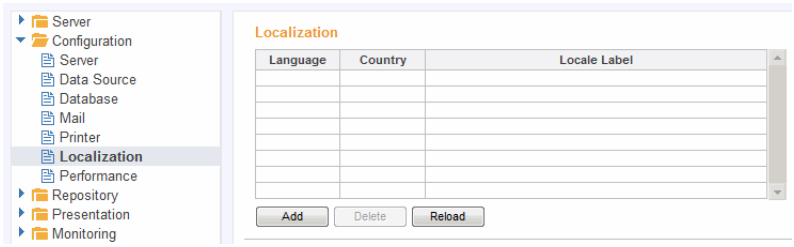
To implement localization at the Portal-level, report-level, or Viewsheet level, first register the desired locales in the 'Localization' page of the Enterprise Manager:

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 6. 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

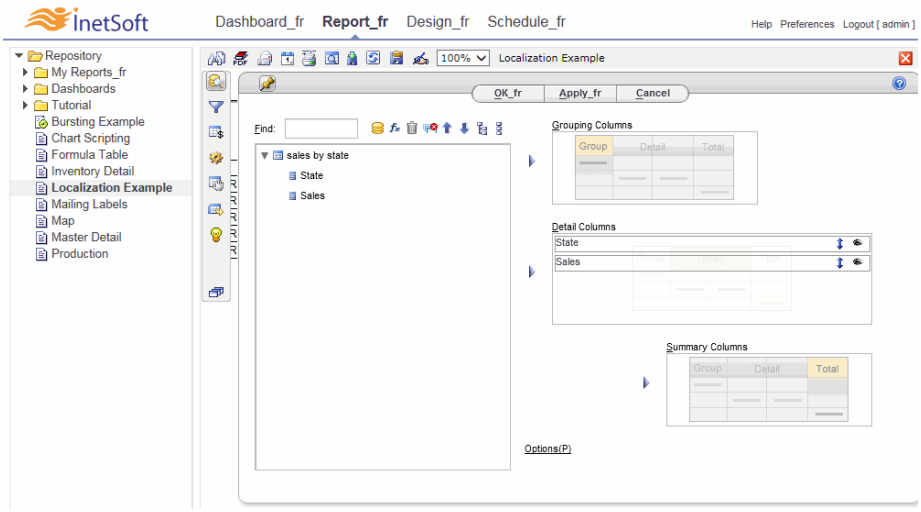
For example, the French Canadian localization file mentioned above might contain the following lines of text, where, for purposes of illustration, "`Repository_fr`" is used as the French translation of "`Repository`," and so on.

```
Repository = Repository_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 7. 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 *Localizing/ Internationalizing a Dashboard* 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

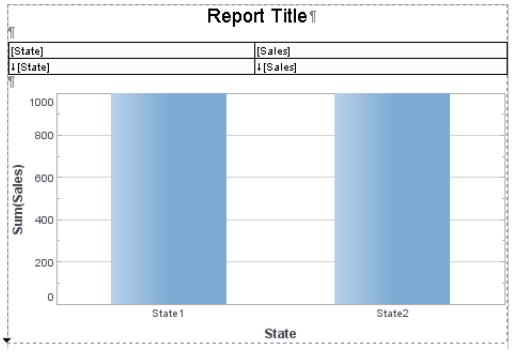
In this example, you will configure a report for French Canadian localization.You will also localize a data source name and a query name.

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’ > ‘Sample Queries’ > ‘sales by state’ query.

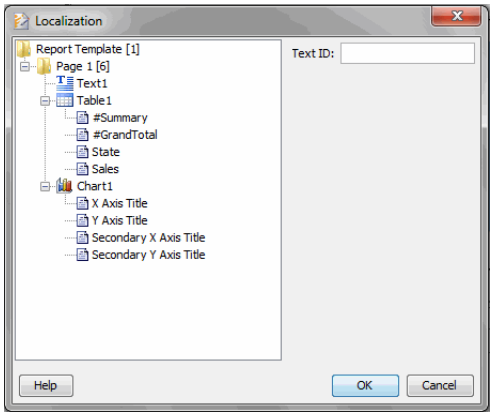
4. Add a bar chart, and bind it also to the ‘Orders’ > ‘Sample Queries’ > ‘sales by state’ query. In the **Chart** tab of the ‘Data Binding’ dialog box, add ‘State’ on the X-axis and ‘Sales’ on the Y-axis.



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.

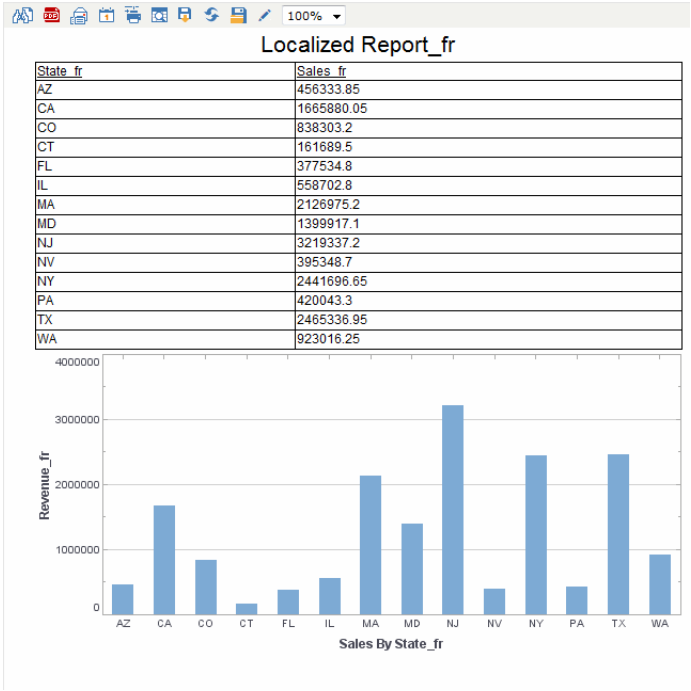
11. In the ‘Localization’ page of the Enterprise Manager, add a new locale.

    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.

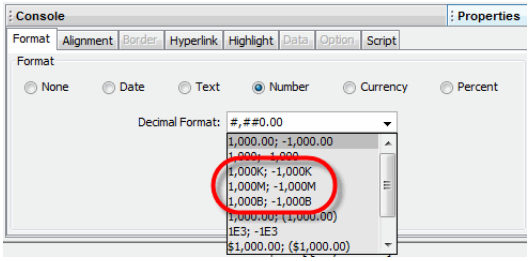
Figure 8. Report with localization applied.



¢

3.8.4 Defining Custom Number Format Multipliers

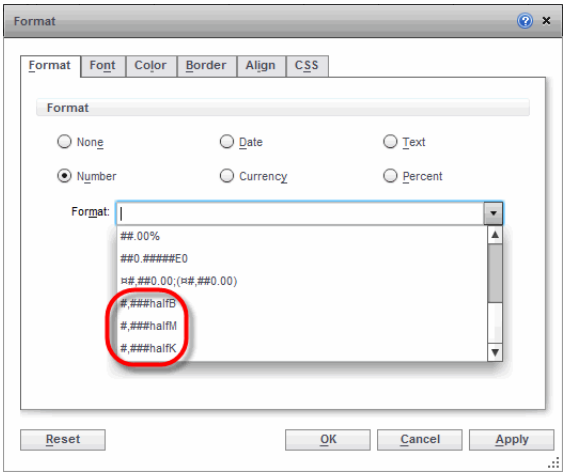
When applying a number format, users can add a suffix “K”, “M”, or “B” to the pattern string to automatically divide the numerical value by 10^3, 10^6, and 10^9, respectively.



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

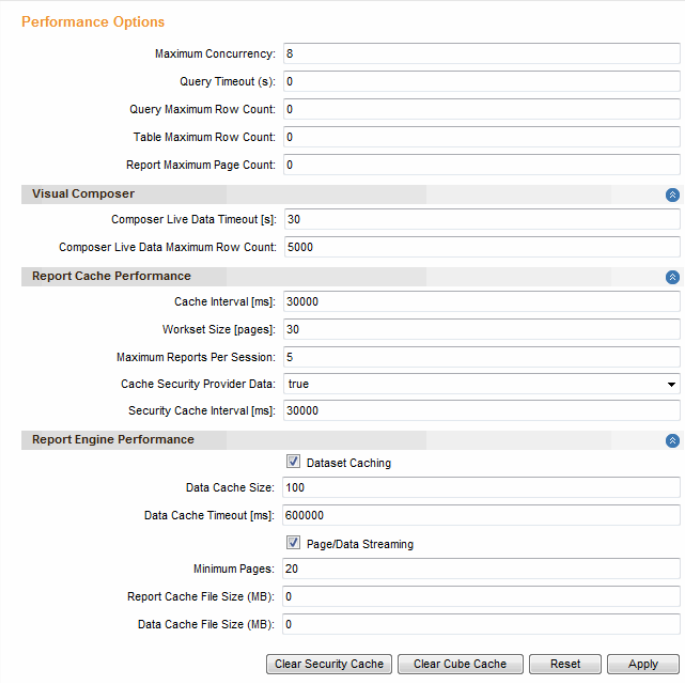
Number Format, in *Advanced Dashboard Design*, for information about formatting a Viewsheet.

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 9. 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 reports that can be executed simultaneously.
- **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.

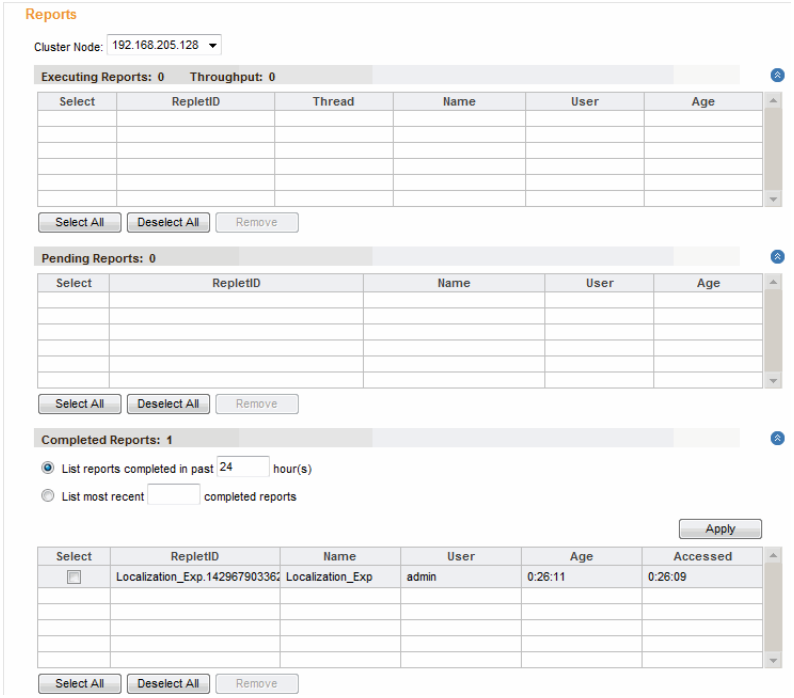
3.10.2 Reports

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.

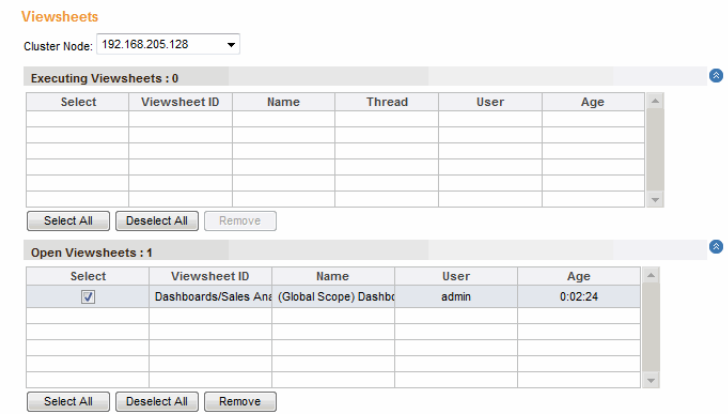
Configuring Remote Monitoring, to accessing data from remote machine.

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.

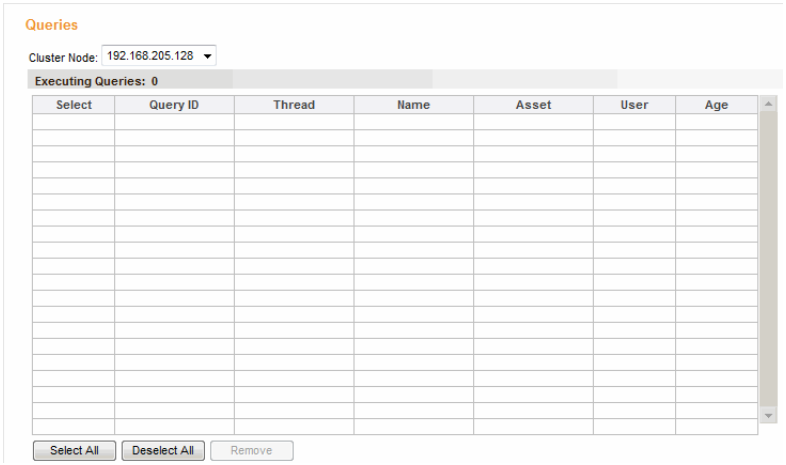
To terminate a Viewsheet, click the ‘Select’ box next to the Viewsheet, and then click the ‘Remove’ button below the table. The ‘Remove’ button has the following effect for the different Viewsheet states:

- 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.

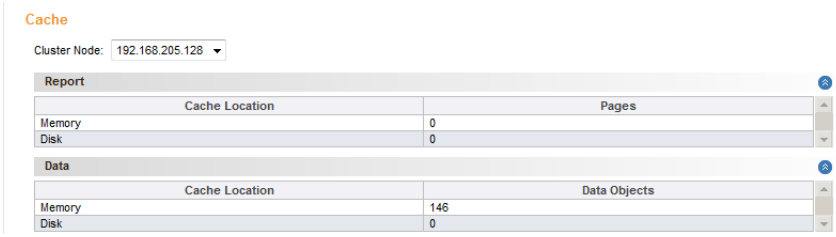
See Also

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.

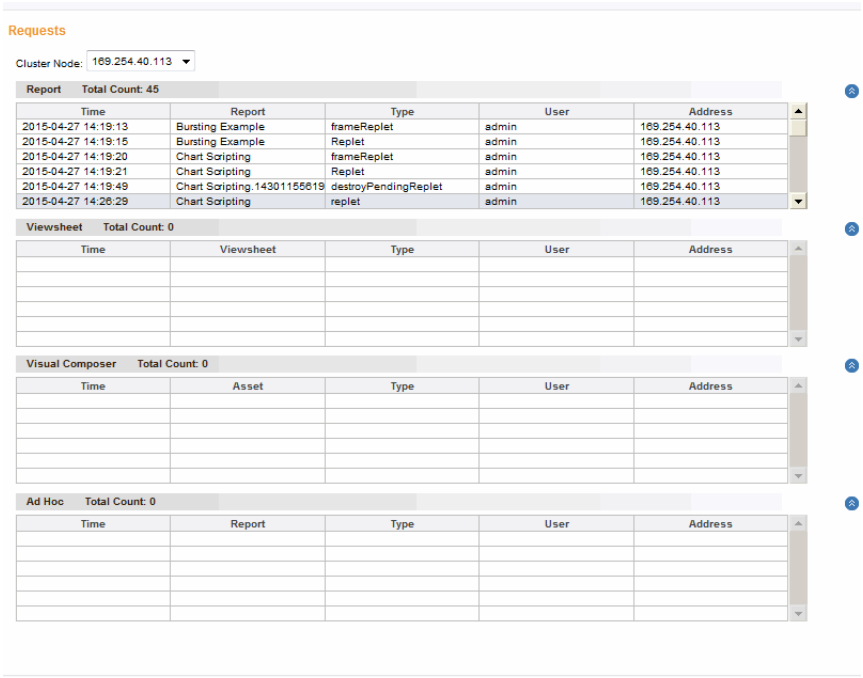
Configuring Remote Monitoring, to accessing data from remote machine.

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

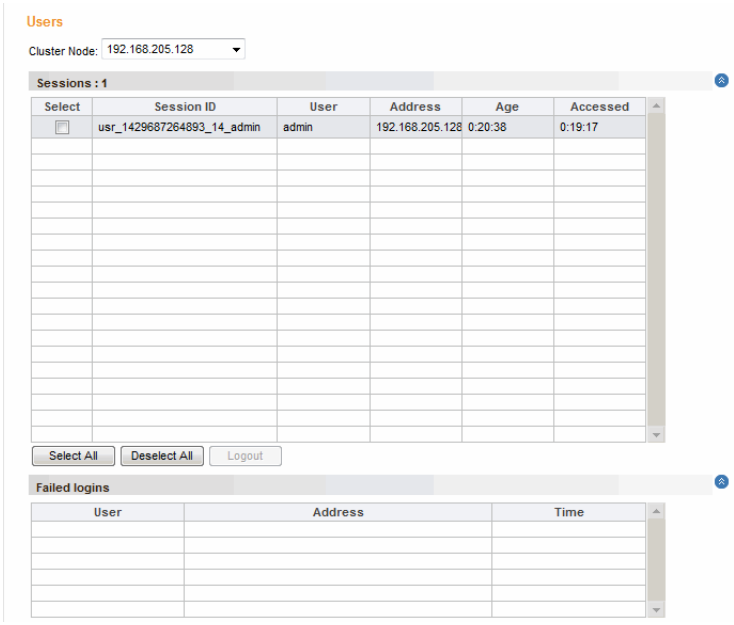
Performance Options and Safeguards, to place limits on server activity.
Configuring Remote Monitoring, to accessing data from remote machine.

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.

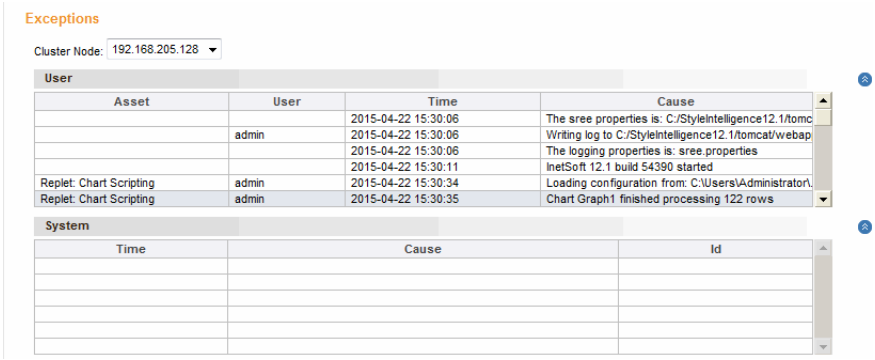
Configuring Monitoring Resolution, for information on setting the monitoring level.

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

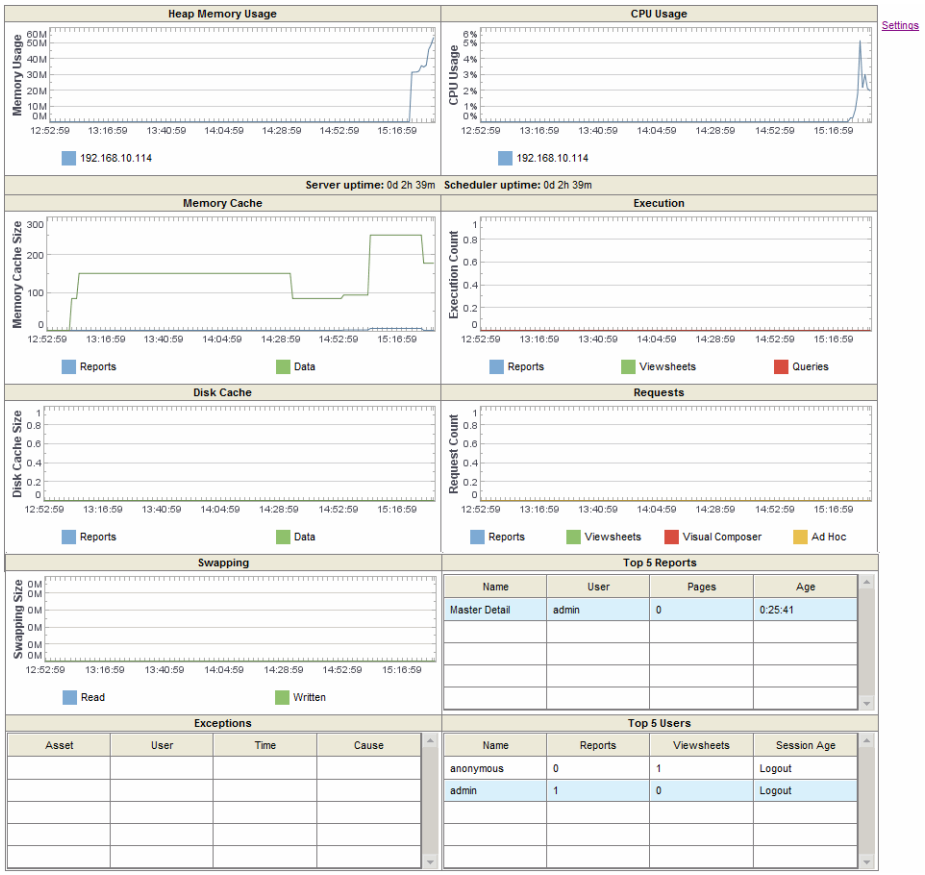
Configuring Remote Monitoring, to accessing data from remote machine.

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

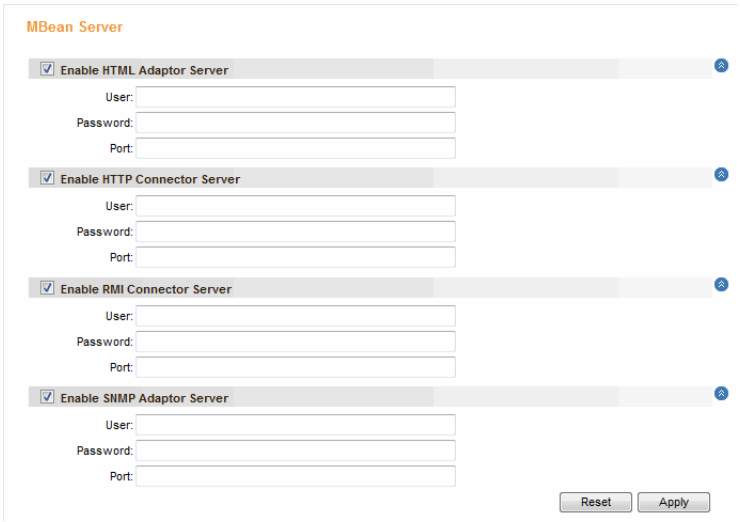
See Also

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.
```

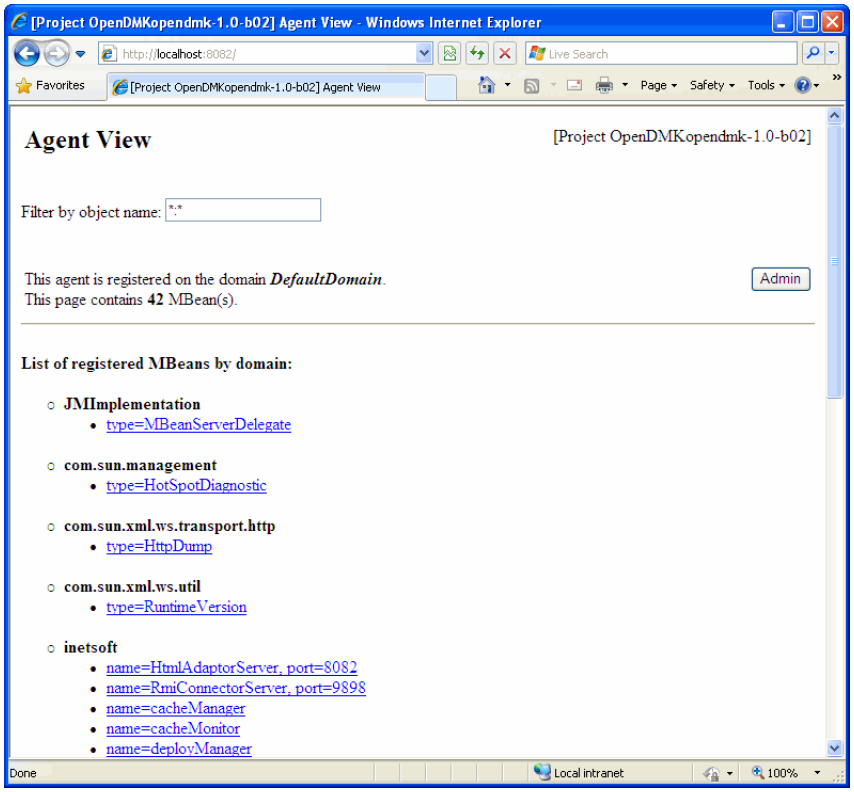
    b. Enter a username and password for the connection (default: admin/admin).

    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’.

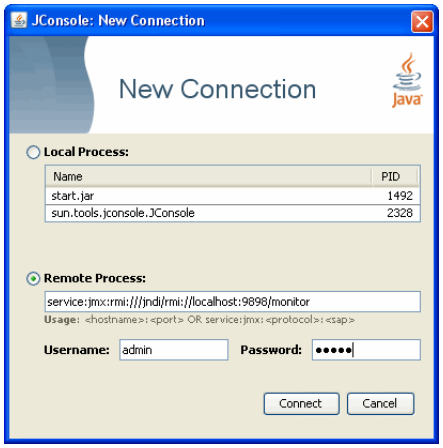
```
Note: User/password is unique to this module and is not
authenticated by the security provider.
```

    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.



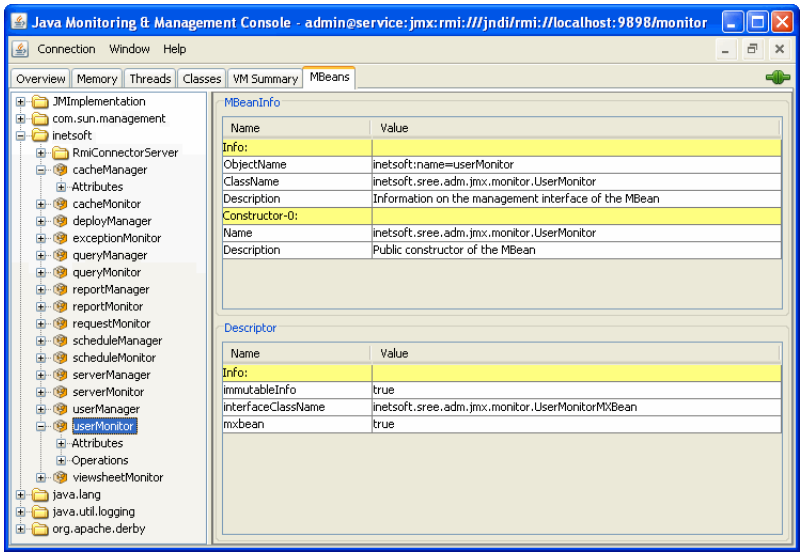
4. In the large text field, enter the following connection string, where `{hostname}` is the server name and `{port}` is the specified RMI port (default: 9898).

   ```
   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.

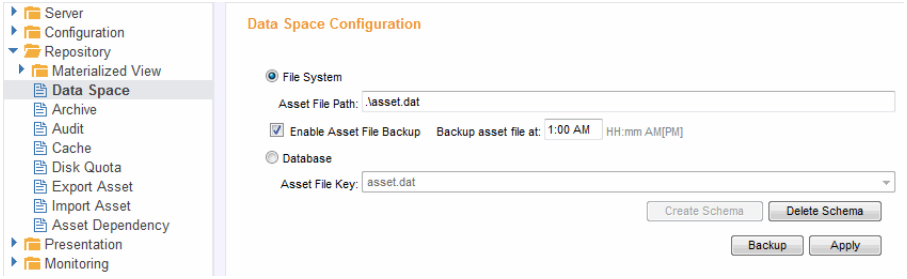
4.1 Data Space

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 10. 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 or *asset.dat.d* directory 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

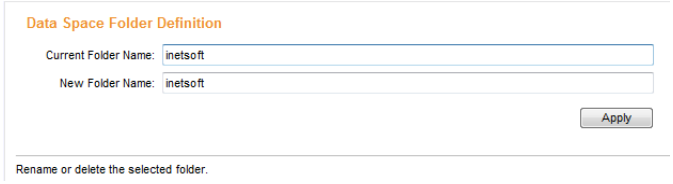
All files and folders in the ‘Data Space’ can be managed through the **Data Space** tab.

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 11. Data Space Folder Definition

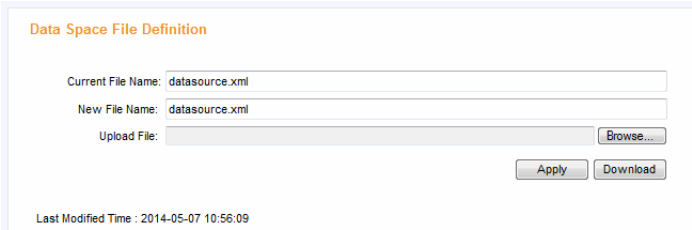


Managing Files

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 12. 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.

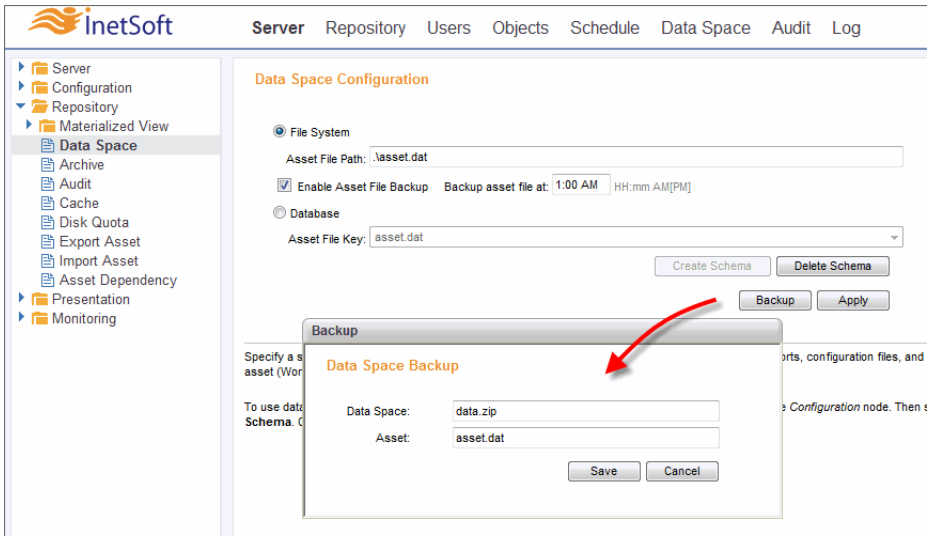
3. In the 'Data Space' field, specify the backup file name, e.g., `data.zip`.

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 13. 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 **Repository** 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.

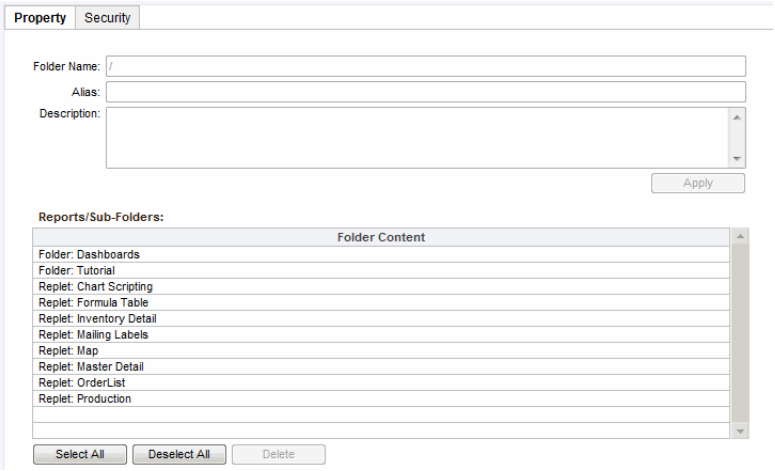
4.2.1 The Repository Page

From the 'Repository' node, under the **Repository** 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 14. 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:

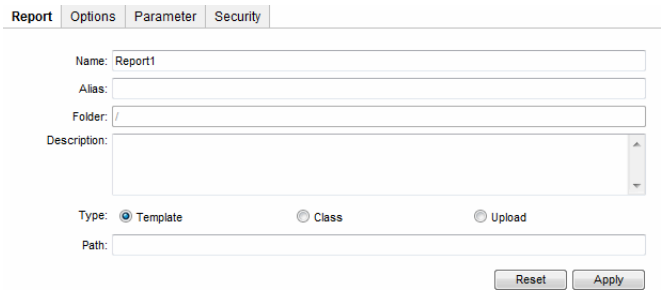
REPORT NAME	REPORT ALIAS
report1	First Report
report2	Middle Report
report3	Last Report

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 **Repository** tab to access the report definition page.

Figure 15. 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 *Report Export Menu Options* setting under the 'Presentation' > 'Report' node.

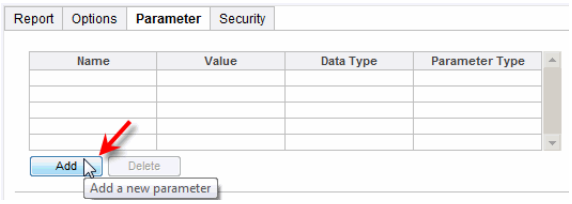
Parameter Tab

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.

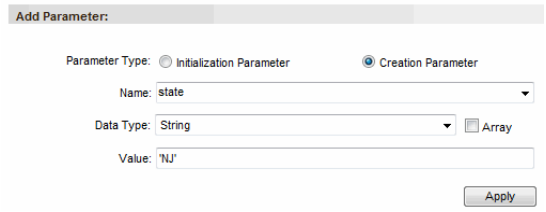
2. Select 'Initialization Parameter' or 'Creation Parameter'.

   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*.)

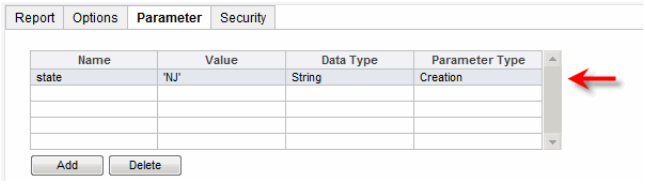
3. In the 'Name' field, select an existing report parameter or manually enter the name of a report parameter.

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 *Interacting with Reports*.

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

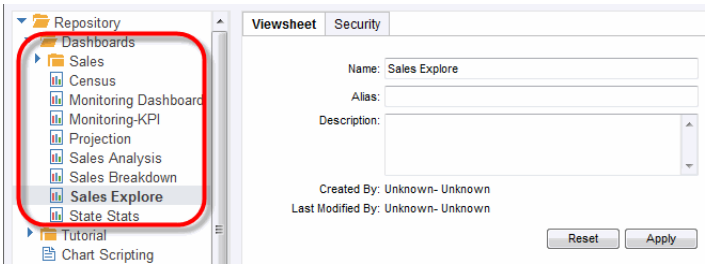
Select the **Security** tab to manage report permissions. By default, a report is set to inherit security permissions from its parent, which is the folder it resides in. This default behavior can be modified and a report may have its own security information setup, independent of its parent. Once set 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.



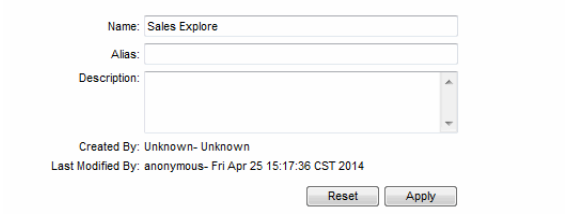
The definitions of all Viewsheets are stored in the *asset.dat* file or *asset.dat.d* directory. To add a Viewsheet to the Repository, use Visual Composer to save the Viewsheet into a global scope. See *Advanced Dashboard Design* for more details.

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 16. Viewsheet Definition Page



The following options are available

Table 6. Viewsheet Definition – Viewsheet Options

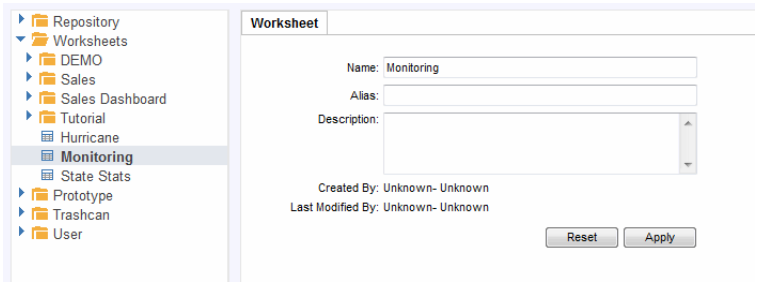
FIELD	DESCRIPTION
Name	The name under which the Viewsheet should be listed in the Repository. Certain characters (e.g., "/") are not permitted in names.
Alias	The Viewsheet name displayed to the user in the Portal.
Description	A brief description to be displayed as a tooltip when the user hovers the mouse over the Viewsheet in the Portal or Visual Composer Repository trees.

Security Tab

Select 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

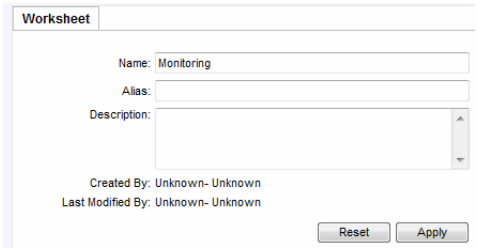
Data Worksheets appear in the Repository under the 'Worksheet' node.



The definitions of all Data Worksheets (other than local Worksheets embedded within individual reports) are stored in the *asset.dat* file or *asset.dat.d* directory. To add a Worksheet to the Repository, use 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 17. 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:

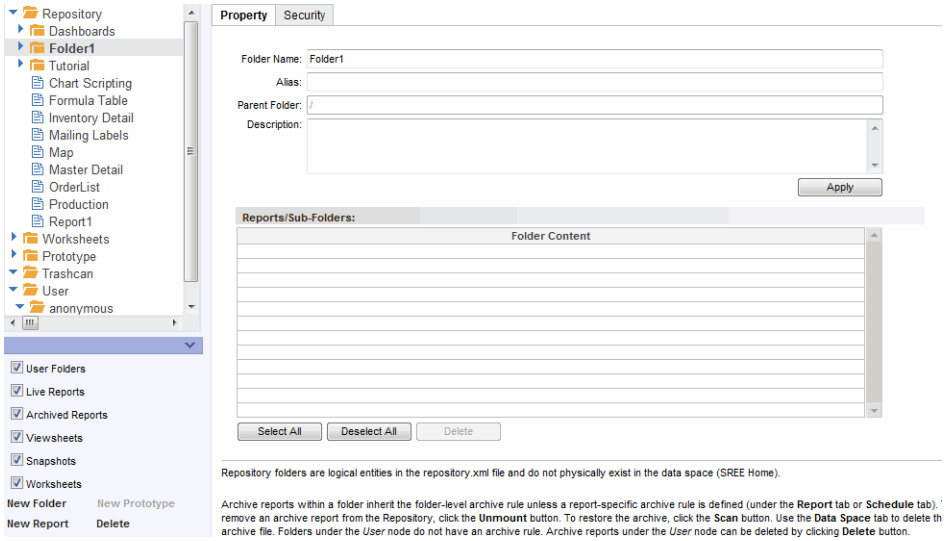
1. In the navigation tree, select a parent folder for the new folder. The parent folder can be the root 'Repository' node or any other folder. (You can change the parent folder later, if needed.)

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.

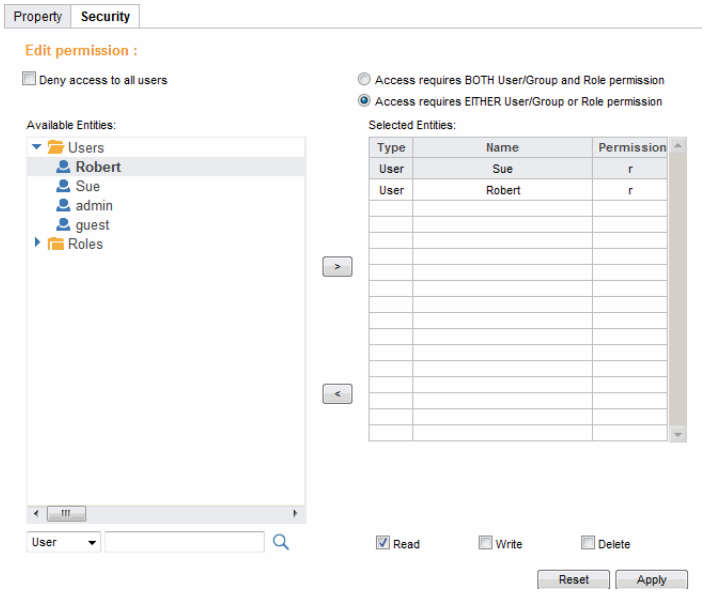
1. Open the Enterprise Manager and enable the 'Default Security Provider'. Complete instructions for this step and the next step may be found within the *Configuring the Default Security Provider* section.

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 **Repository** tab, click the 'Repository' node and select the **Security** tab.

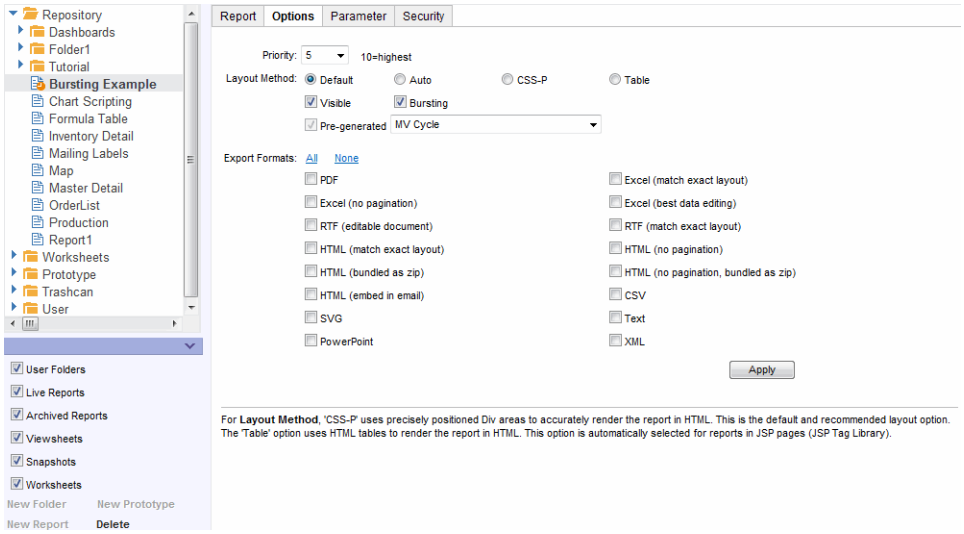
4. Deselect the 'Deny access to all users' checkbox.

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.



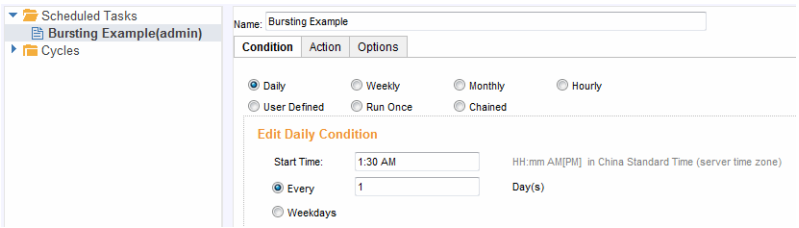
8. Access the ‘Server’ > ‘Scheduler’ page from under the **Server** tab and ensure that the Scheduler is running. If not, click the ‘Start’ button.

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.



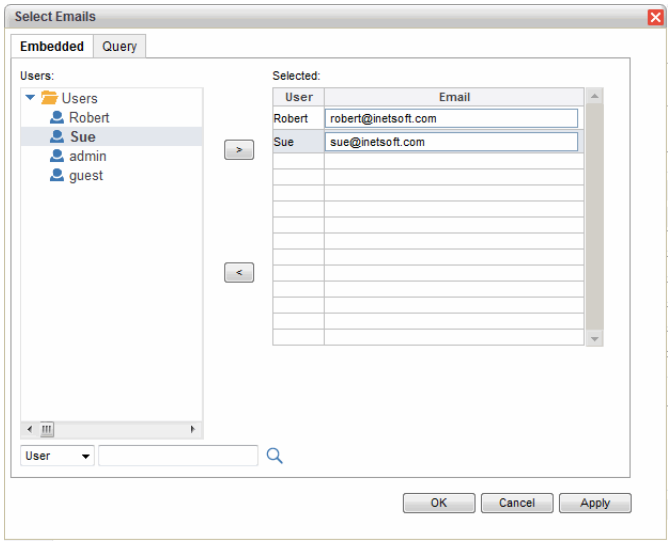
13. Select the **Action** tab and choose ‘Burst’ as the Action Type.

14. In the ‘Report’ drop down list, select ‘Bursting Example.’

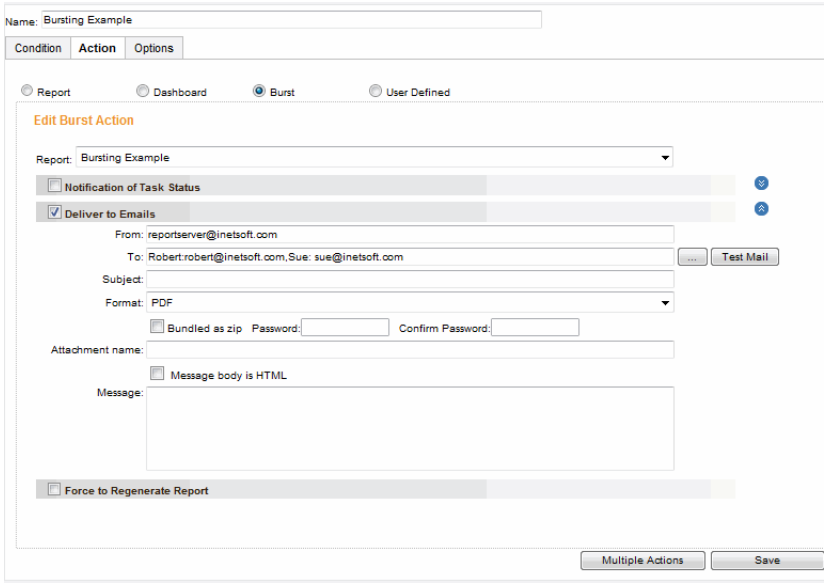
Note: The 'Select Emails' dialog box is available only when the 'Execute As' property (**Options** tab) specifies an administrator.

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 **Repository** tab. The 'User' 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.

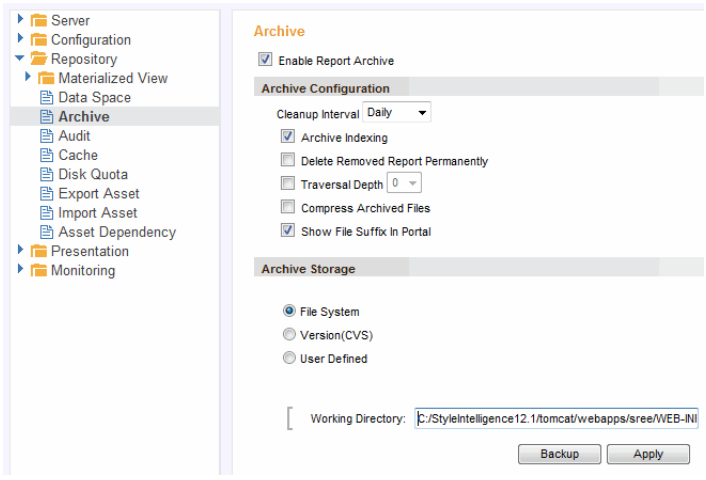
However, if an application has many very large reports, dynamically generating the reports on demand may not be a feasible solution. To avoid repeatedly generating the reports, a report archive for the application can be set up to save and reuse generated reports.

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 18. 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.

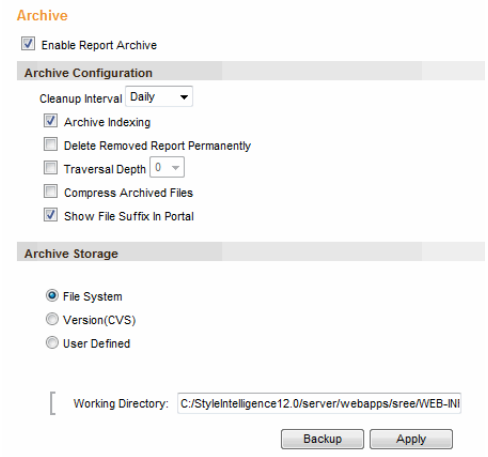
Administering Archived Reports, for information on managing archived reports.

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.

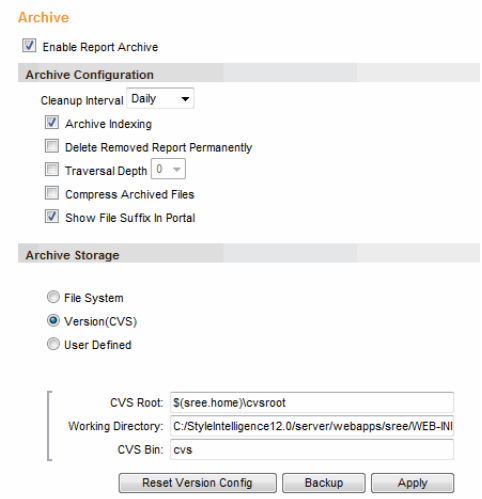
2. Expand the 'Repository' node and select 'Archive' on the Navigation tree.

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 **Repository** 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.
```

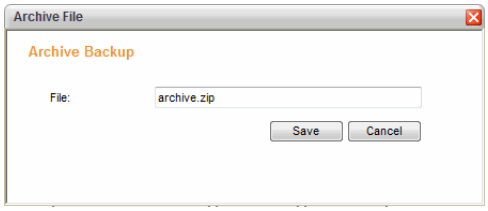
- **Traversal Depth**: The 'Traversal Depth' option allows you to set the number of levels of reports that can be followed starting from the hyperlinks within the archived report. This option only applies to the 'Generated Report' output format. A 'Traversal Depth' of 0 indicates that the reports targeted 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 **Repository** 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.

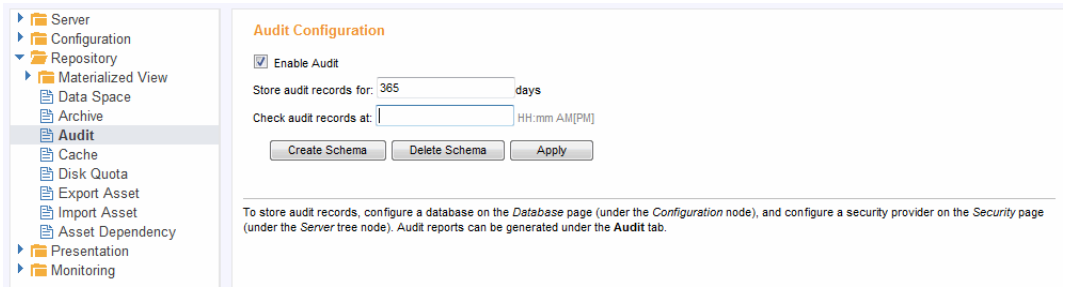
4. Enter a value in the ‘Check audit records at’ field to specify the time of day at which obsolete audit records should be deleted.

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 19. 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 20. Audit Page

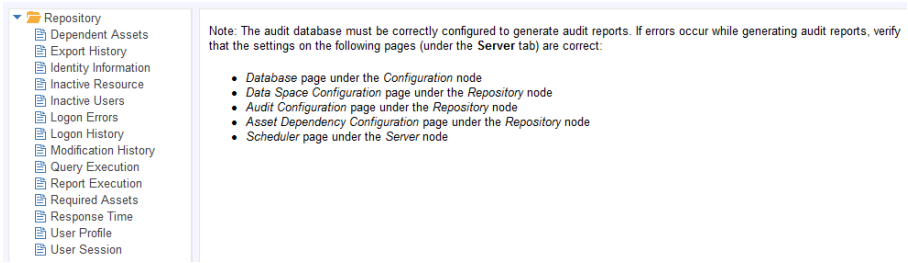


Table 8. Type of information under each Audit Log node

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.

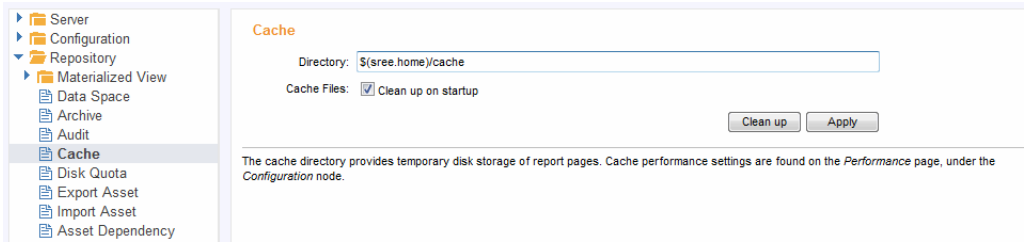
NODE NAME	TYPE OF INFORMATION
Query Execution	Report of queries, data models, and Data Worksheets executed over a selected period.
Report Execution	Report of reports executed over a selected period.
Required Assets	Report of assets that are required by a selected asset.
Response Time	Report of report 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 21. 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* in *Managing the Server*. 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/tomcat/webapps/sree/
WEB-INF/lib/bisuite.jar"/>
    <path path="C:/StyleIntelligence/tomcat/webapps/sree/
WEB-INF/lib/etools.jar"/>
    <path path="C:/StyleIntelligence/tomcat/webapps/sree/
WEB-INF/lib/derby.jar"/>
    <path path="C:/StyleIntelligence/tomcat/webapps/sree/
WEB-INF/lib/derbyclient.jar"/>
    <path path="C:/StyleIntelligence/tomcat/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/tomcat/
webapps/sree/WEB-INF/classes" forcetooverwrite="false"/>
  </target>

  <!-- Import from Database to File System -->
  <target name="ImportDBToFS">
    <importdbtofs home="C:/StyleIntelligence/tomcat/
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.

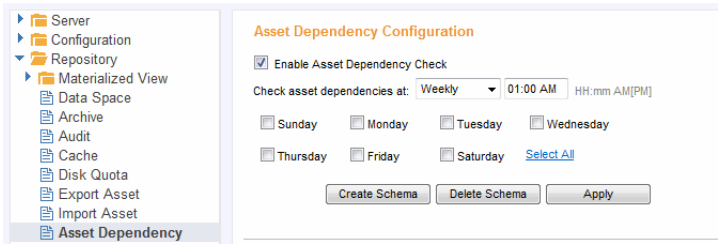
2. Select 'Daily' or 'Weekly' in the 'Check asset dependencies at' field. For the 'Daily' option, specify an execution time. For the 'Weekly' option, specify an execution time and select a set of days on which the task should execute.

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 22. 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:

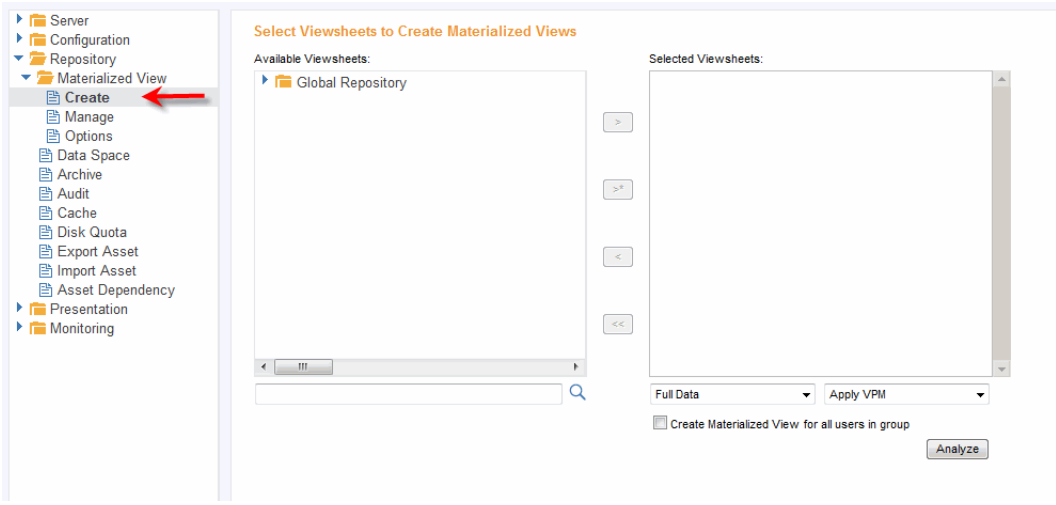
1. Log into Enterprise Manager.

   See *Distributing Materialized Views (Data Grid)* for more information about data grid configurations.

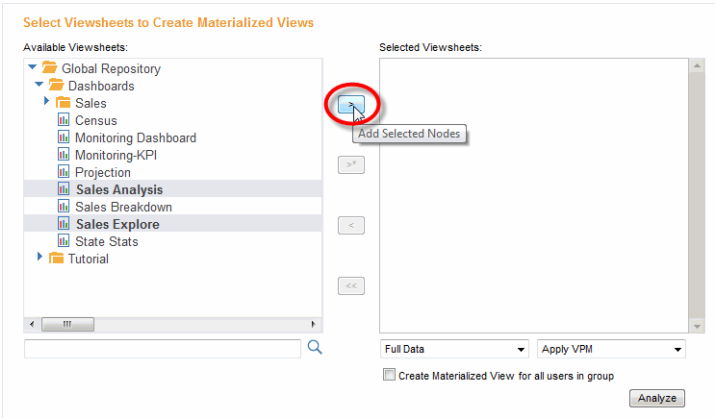
   If you are using a data grid, you must log into Enterprise Manager on the Data Server machine. (You cannot create a materialized view by using Enterprise Manager on the Report Server machine, unless that machine also hosts the Data Server.) For example, if *Machine1* contains the Report Server, *Machine2* 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
   ```

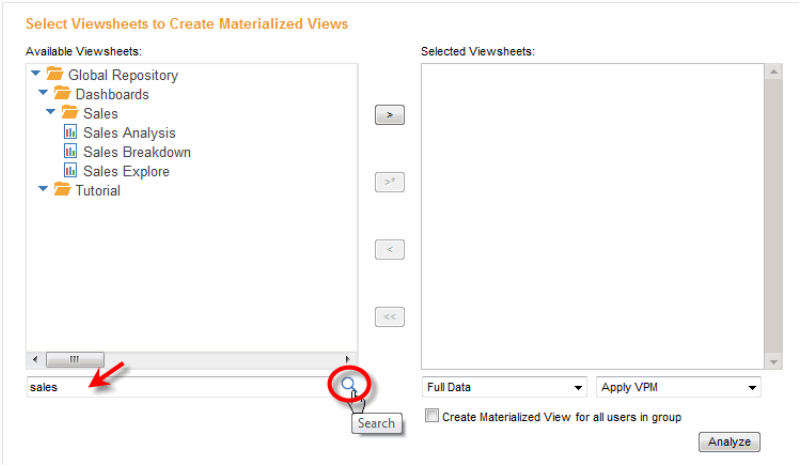
2. Expand the 'Repository' > 'Materialized View' node under the **Server** tab, and select 'Create'.



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.



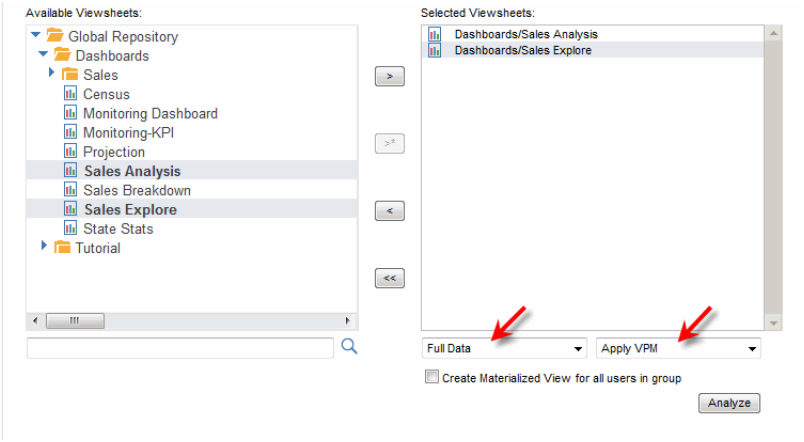
4. From the menu below the ‘Selected Viewsheets’ panel, choose either ‘Full Data’ or ‘Minimum Data’.

   ‘Full Data’ materializes the entire Worksheet Data Block associated with a materializable Viewsheet component. ‘Minimum Data’ restricts the materialization to include only columns of the Data Block that are actually supplying data to a Viewsheet Selection component, but only in cases when the Selection component is the sole user of this data. In cases when data in a Data Block is accessed by multiple Viewsheet components, full data is used automatically.

   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.

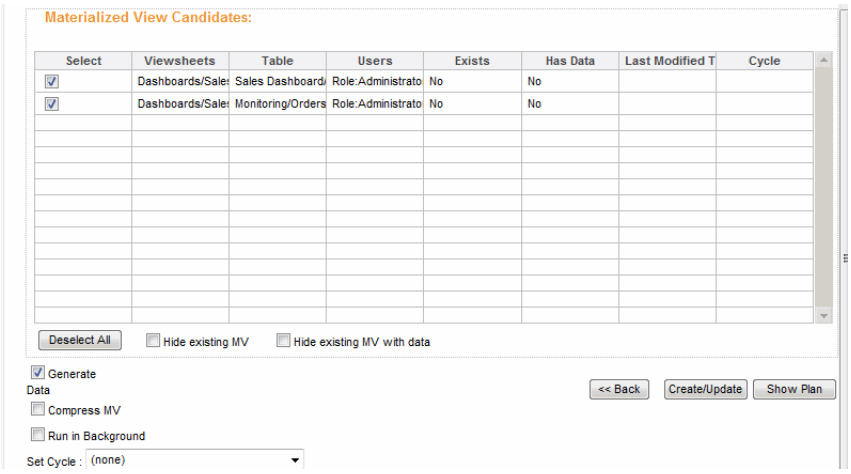
See *Virtual Private Model Security* in *Advanced Data Modeling* for details.

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’.

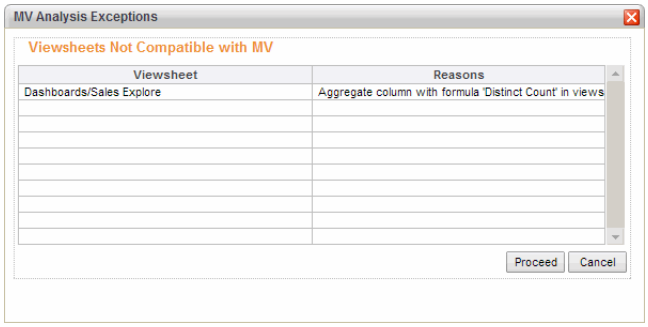


Only Viewsheets containing aggregated data can be considered candidates for materialized views.

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



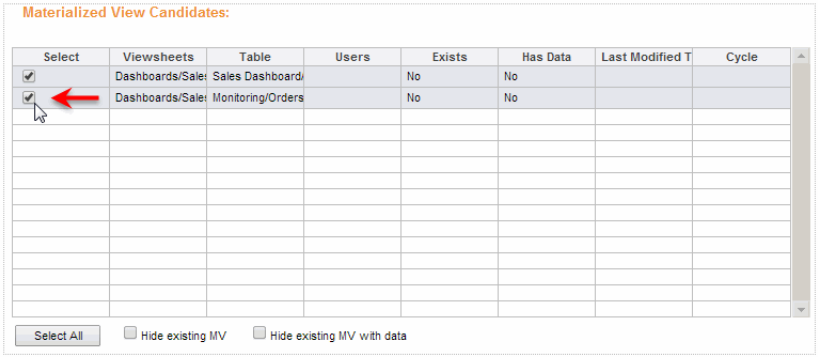
If there are design aspects of a Viewsheet or Data Worksheet that prevent the generation of a materialized view, the 'MV Analysis Exceptions' dialog box is shown.



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.

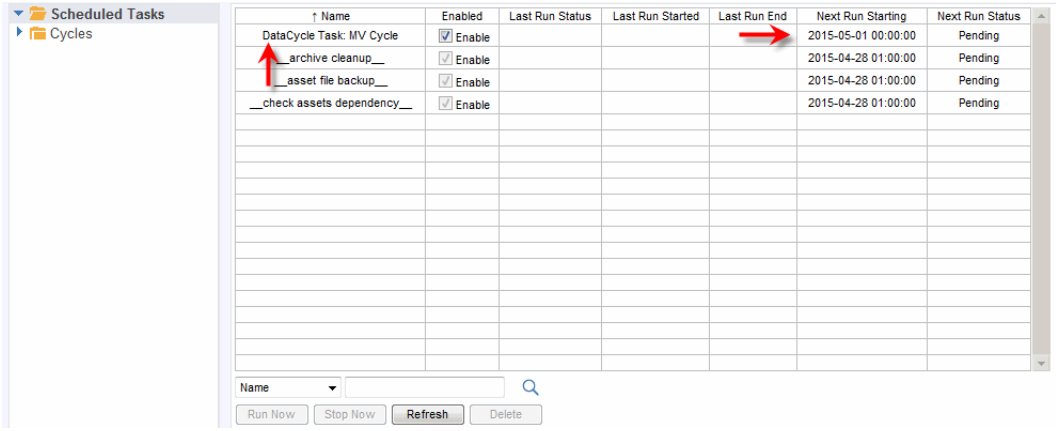
8. Check the boxes in the ‘Select’ column to select the Viewsheets for which you want to create a materialized view. To select all Viewsheets in the table, press the ‘Select All’ button below the table.



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.

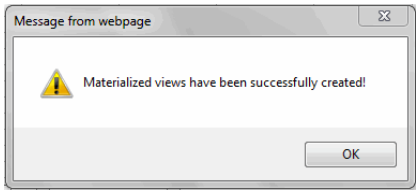
9. In the ‘Select Cycle’ column of the table, choose the cycle on which the materialized view should be regenerated. (See the *Scheduler Cycle* section for more information about defining cycles.)

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:

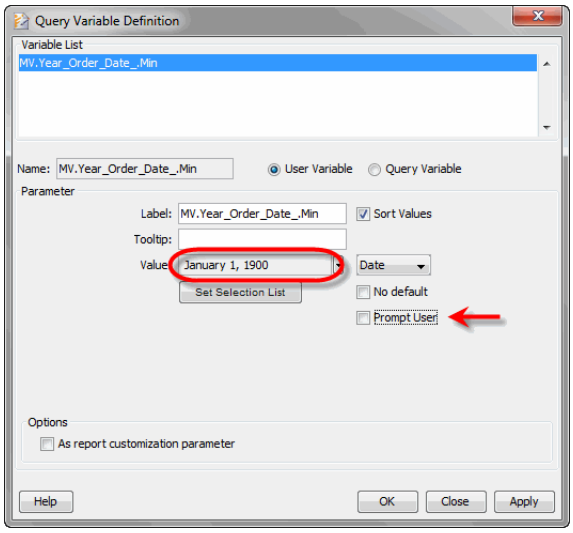
1. Using the Query Wizard in Style Studio, add the desired update condition to the query definition itself. For example:

   ```
   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

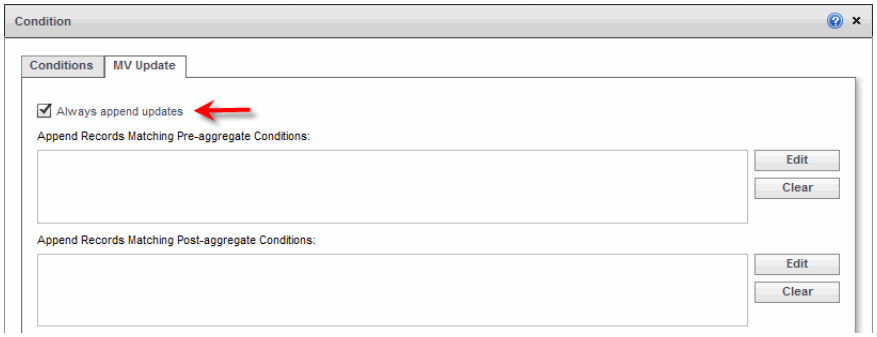
Advanced Data Mashup for more information about these parameters.)

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. Note that this assumes

that the tables being queried contain **only** new data to 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

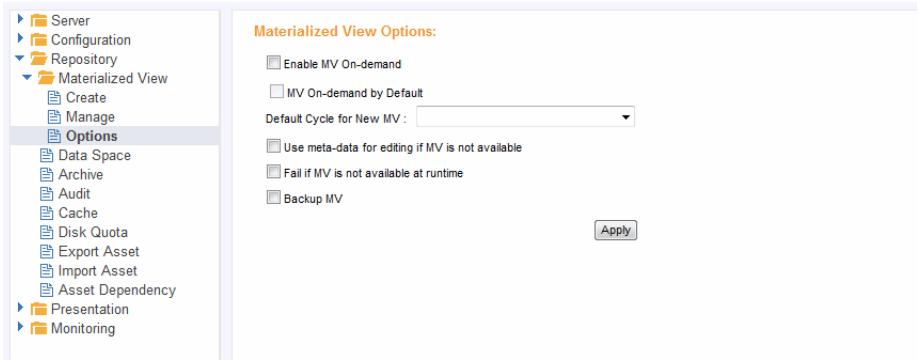
For multi-tenant implementations utilizing the multi-connection feature, materialized views are created based on user/group connection assignments. For this reason, if you set a Viewsheet's permissions by *role* (on the Viewsheet's **Security** tab), you should also add individual permissions for users 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.

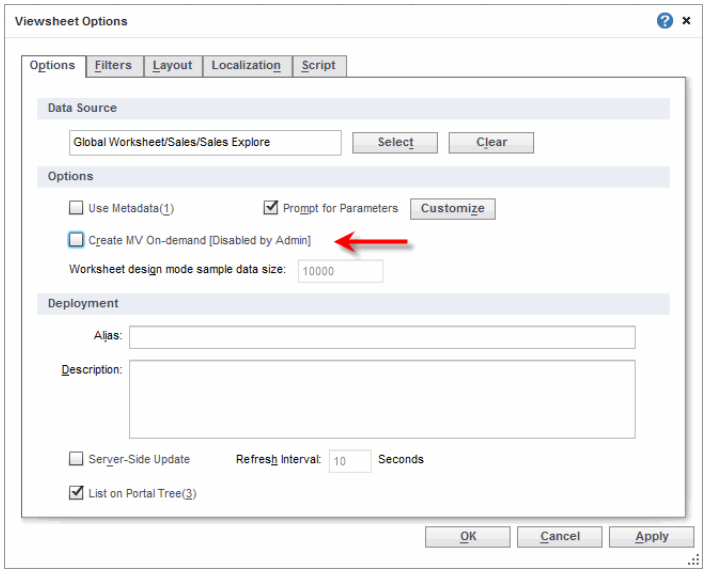
4.9.4 Materialized View Options

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 Dashboard 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.

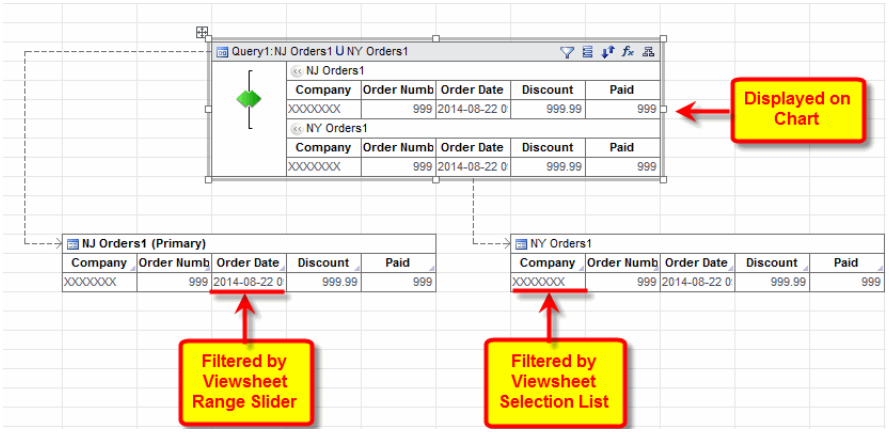
See Also

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.

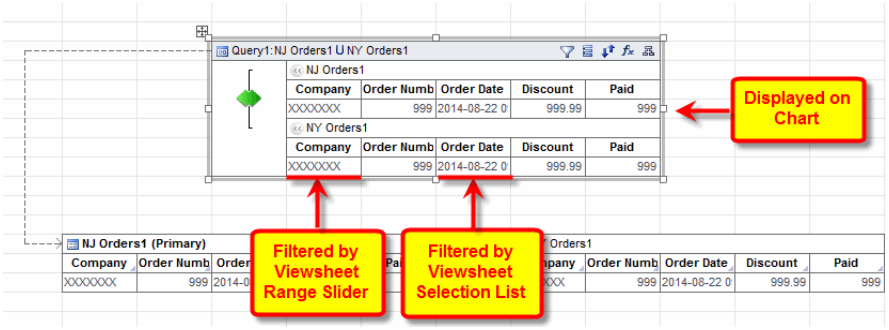
In order for the underlying Data Block to be materialized, any Viewsheet selections (e.g., Selection Lists) that filter the Data Block must be applied directly to the Data Block itself, rather than to any precursor data blocks. For example, consider the following Worksheet:



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:



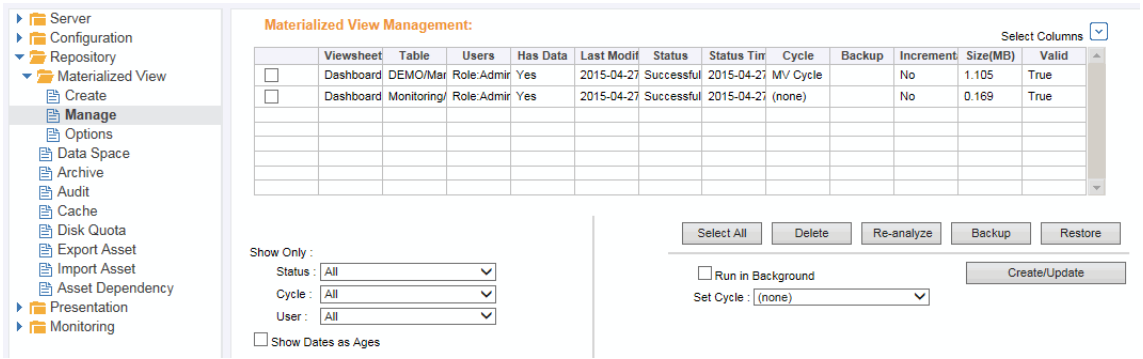
The automatic transformation is internal, and does not alter the actual Viewsheet.

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.

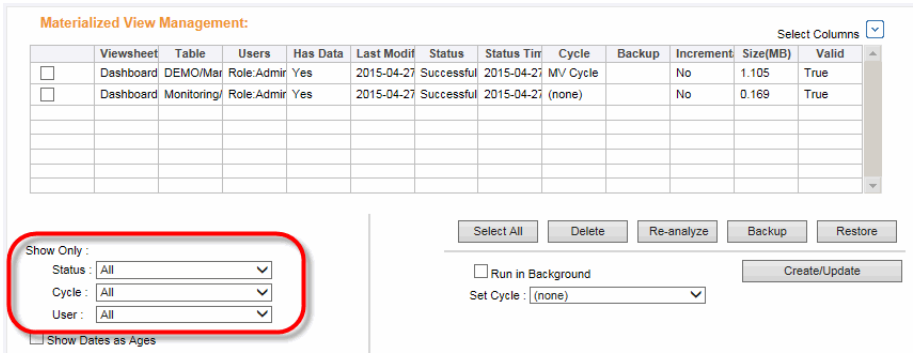
In many of these instances, you may be able to facilitate materialization by “moving the selection up.” To do this, evaluate the placement of your selection components, and try to relocate them (i.e., move them up) to filter the *resultant* Data Block rather than the precursor Data Blocks. By doing this you may enable Enterprise Manager to materialize the problem Data Block.

4.9.7 Managing Materialized Views

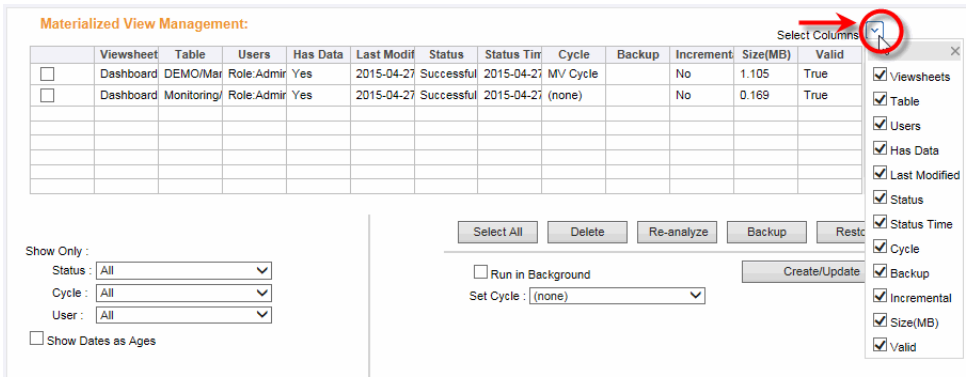
Manage your existing materialized views from the ‘Repository’ > ‘Materialized View’ > ‘Manage’ page under the **Server** tab.



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’.
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.

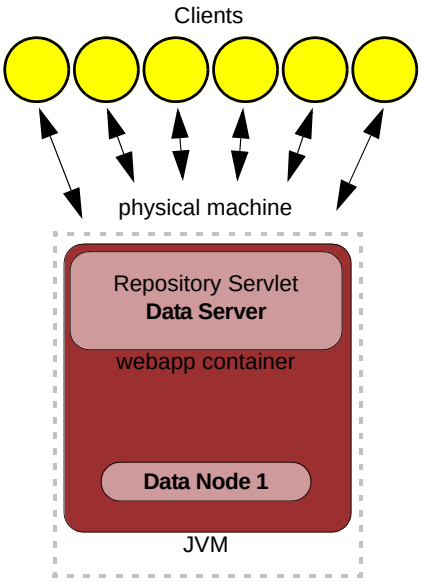
You can configure the Report Server, Data Server, and Data Node in various ways, as described in the following sections. Note that the Data Server and a Data Node *cannot* reside on the same machine, except in the default installation (*Report Server, Data Server, and Data Node on Same Machine (Default)*).

It is highly recommended that you discuss your needs with an InetSoft technical resource before you undertake installation of a data grid. The following sections discuss different arrangements for the data nodes and server.

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 23. Report Server, Data Server, and Data Node on one machine



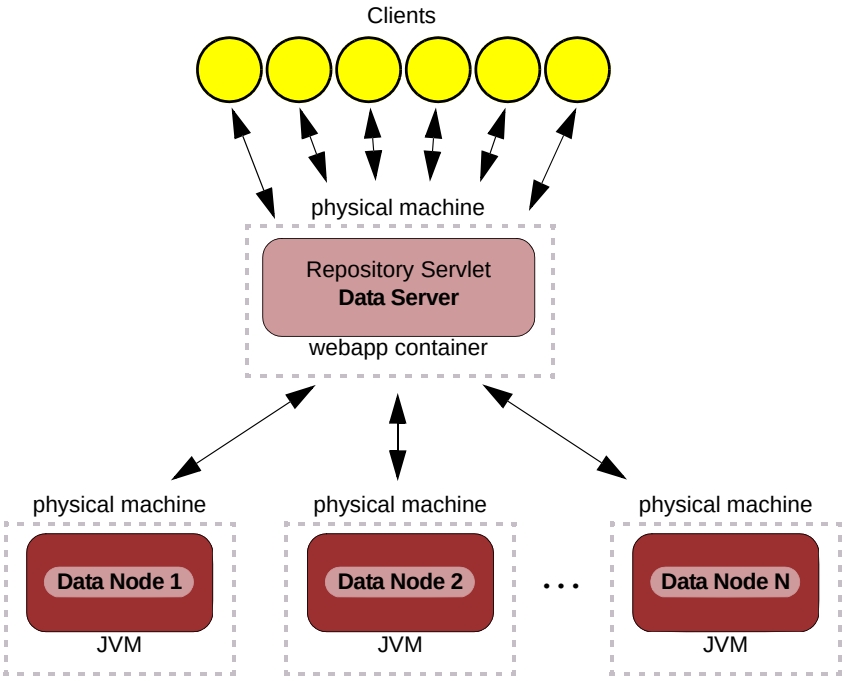
Viewsheet materialization with this default configuration generally delivers an improvement in performance over non-materialized Viewsheets. If performance is satisfactory, you do not need to further modify any settings. However, you can generally obtain better performance for materialized Viewsheets 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

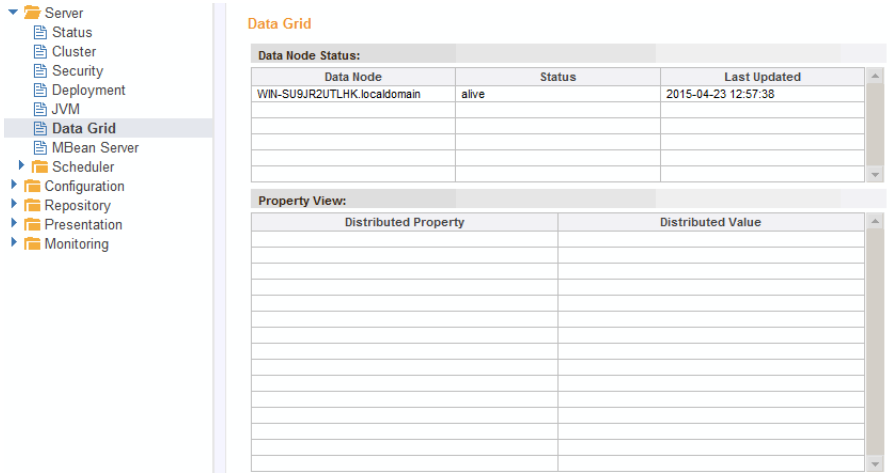
You can install the Report Server and Data Server on a single machine, and distribute the Data Nodes across a different set of machines.

Figure 24. Report and Data Server on Machine 1, Data Nodes on Machines 2-N



It is highly recommended that you discuss your needs with an InetSoft technical resource before you undertake installation of a data grid.

After installing the data grid, open Enterprise Manager on the Report Server machine. Open the 'Server' > 'Data Grid' page under the **Server** tab. Verify that all Data Node machines you have started are shown with "alive" status.

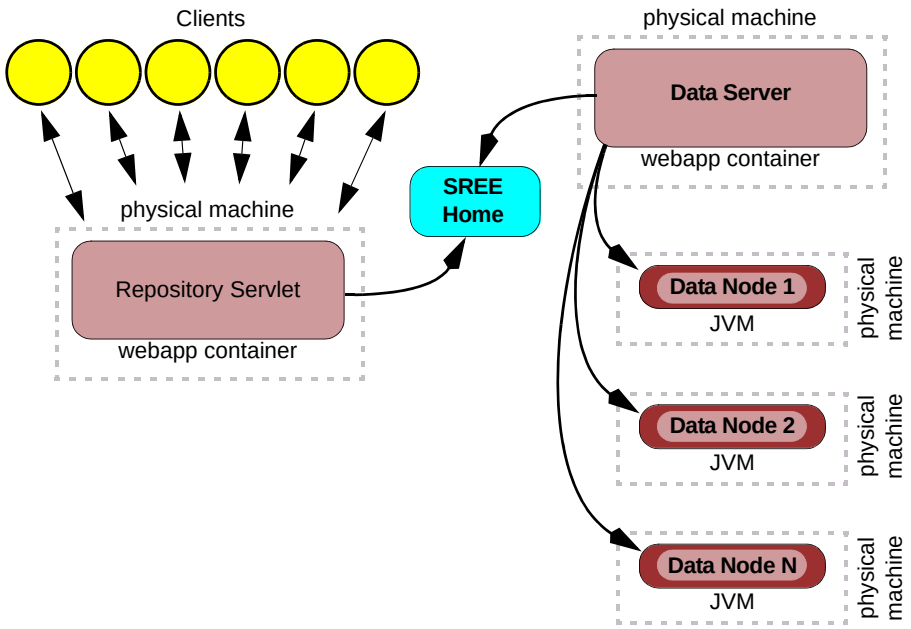


To create a materialized view in this data grid environment, use Enterprise Manager on the Report/Data Server machine. See *Creating a Materialized View*.

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 25. Report Server on Machine 1, Data Server on Machine 2, Data Nodes on Machines 3-N

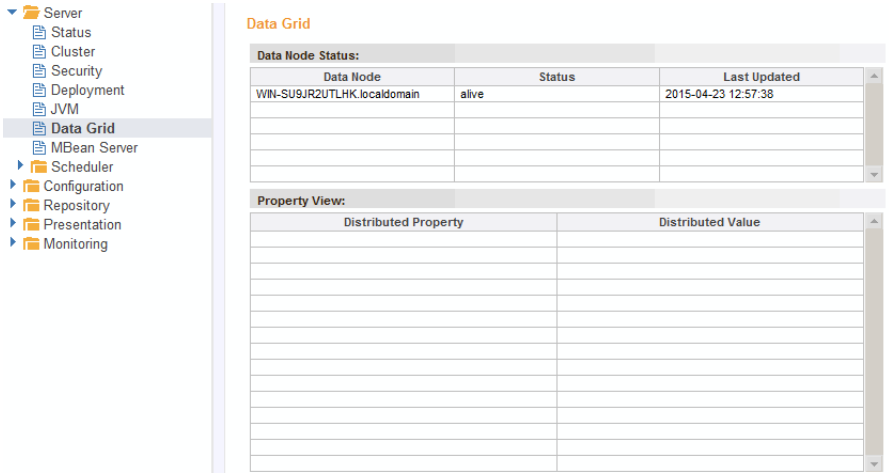


It is highly recommended that you discuss your needs with an InetSoft technical resource before you undertake installation of a data grid.

Enterprise Manager on the Report Server machine cannot display node status.

After you have installed the data grid, 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 26. The Data Grid page.



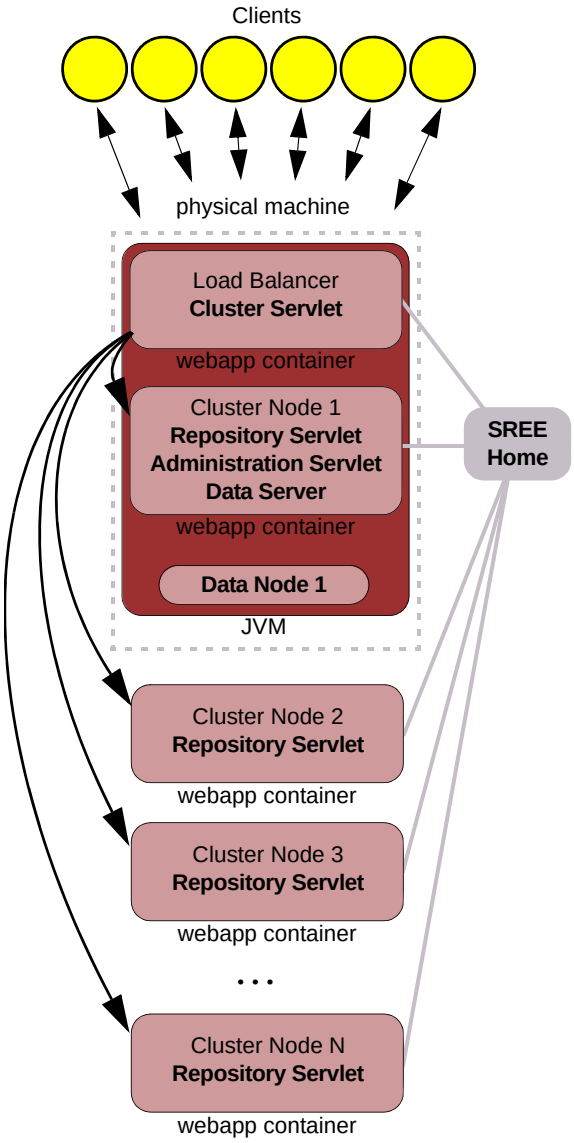
To create a materialized view in this data grid environment, you must log into Enterprise Manager on the Data Server machine. (You cannot create a materialized view by using Enterprise Manager on the Report Server machine.) See *Creating a Materialized View* for further information.

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 27. Report Server, Data Server, and Data Node on one machine, in clustered environment

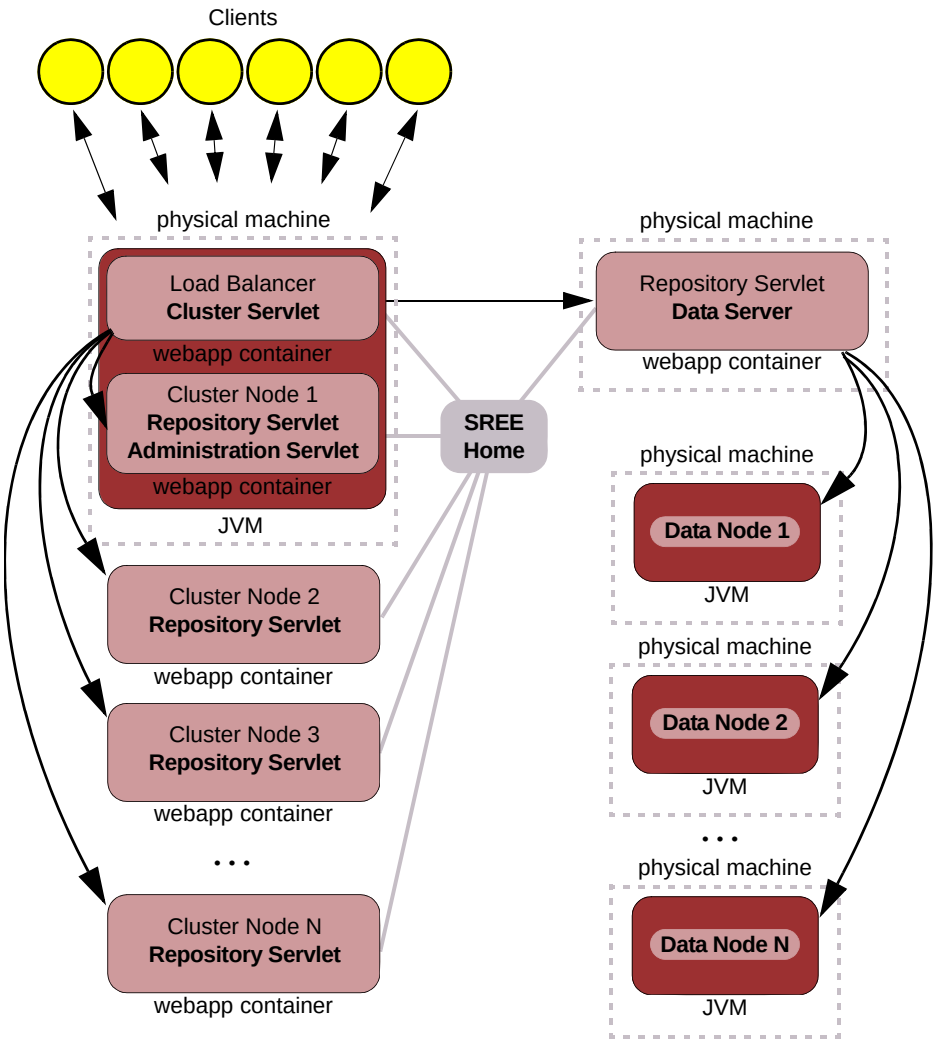


To obtain better performance, distribute the materialized view across multiple machines as described in *Distributed Materialization in a Clustered Environment*. To create a materialized view using Enterprise Manager, see *Creating a Materialized View*.

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 28. 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. 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.

Preparing the LDAP Server

Before you configure the LDAP security provider in Style Intelligence, read the recommendations below:

1. Active Directory schemas can contain a large number of objects, so it is important to specify the portions of the schema that will be accessed by Style Intelligence. Browse your LDAP schema with a tool such as Apache Directory Studio to familiarize yourself with the objects in your LDAP schema.

2. It is recommended that you add new security groups into your LDAP schema to be used when defining security for Style Intelligence (e.g., InetSoftAdmin, InetSoftUser, InetSoftDeveloper, etc.). Add the appropriate users to each of these groups.

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

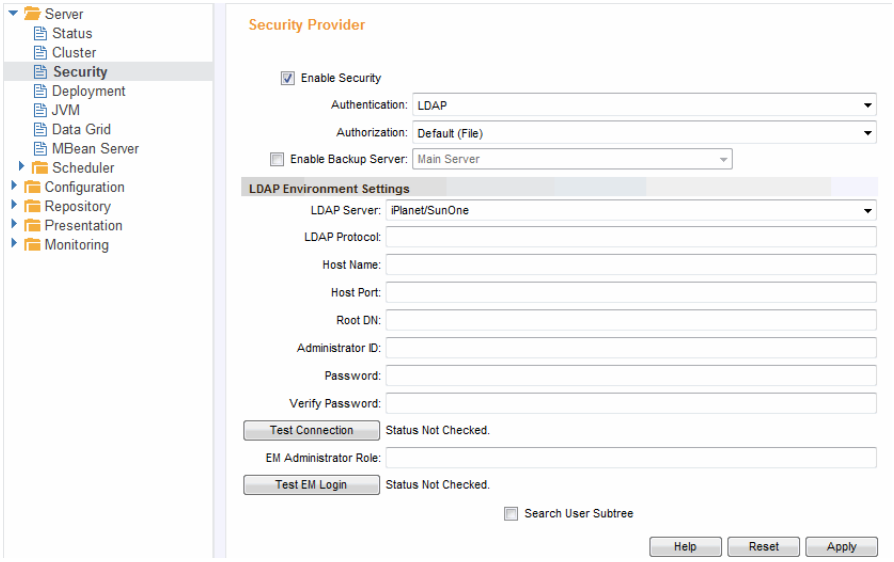
- Since directory servers do not currently support roles fully, groups in the LDAP schema are mapped to roles in Style Intelligence.
- If the administrator password on the LDAP server is changed, you will not be able to log into Enterprise Manager in the usual way. Instead, log into Enterprise Manager 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 29. 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 3 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 3. 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	inetsoftreplet	Directory String(multi-valued)

- objectclass inetsoftreplet
  oid inetsoftreplet-oid
  superior top

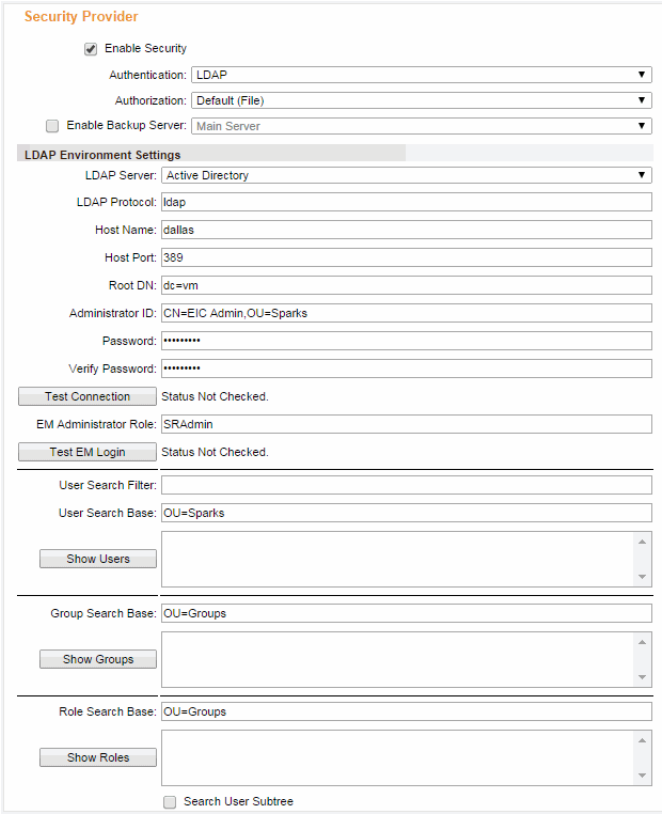
- requires
  cn,

- 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.)

Figure 30. Sample LDAP Environment Setup for Active Directory




```
LDAP is recommended for authentication but not for authorization
because authorization requires modification of the LDAP schema.
```

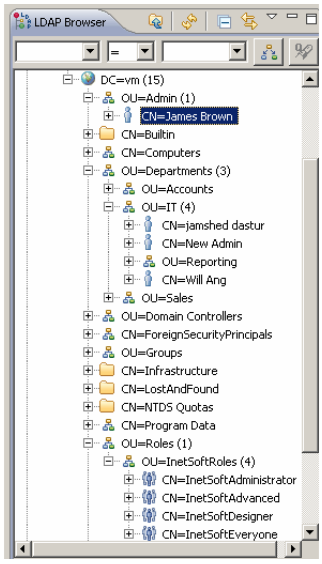
When configuring an Active Directory server to perform authentication for Style Intelligence, take note of the following points:

- 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 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.

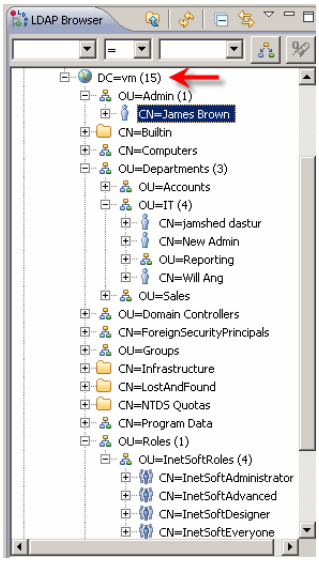
See *Sample Active Directory Setup* for a walk-through of setting up Active Directory as security provider.

Sample Active Directory Setup

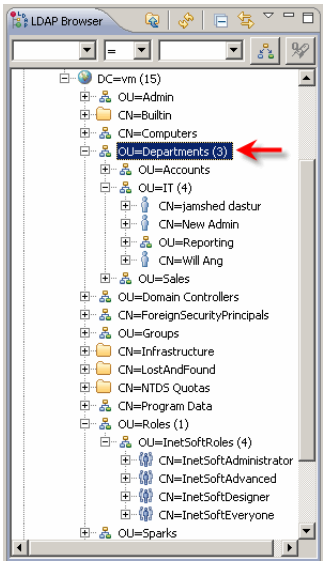
In the following example, consider the following Active Directory schema. The bullets below indicate how to fill out the 'Security Provider' page for this schema.



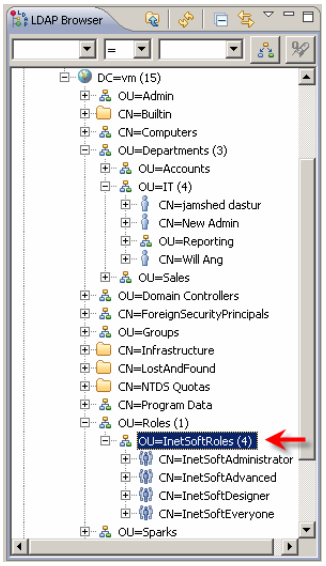
- **Root DN**: The 'Root DN' (here, "`dc=vm`") is automatically appended to all units within Style Intelligence and need not be included in any other entry.



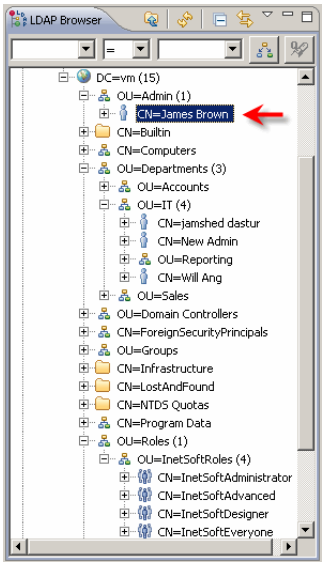
- **User Search Base/Group Search Base**: To search all units below the 'Departments' unit, set 'User Search Base' and 'Group Search Base' to "`ou=Departments`". (If you want to include only the 'Accounts' and 'Sales' units, set "`ou=Accounts,ou=Departments;ou=Sales,ou=Departments`".)



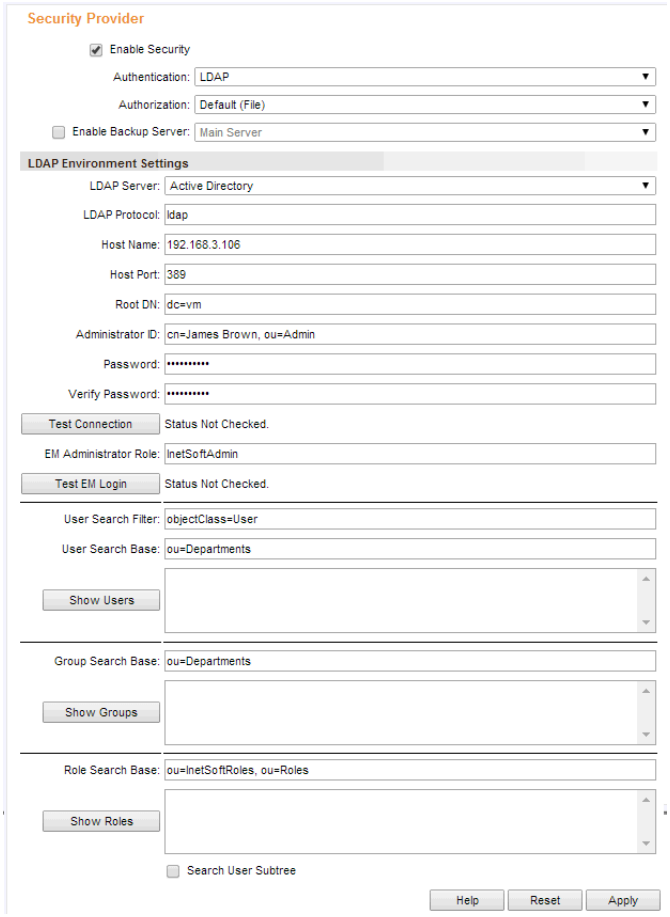
- **Role Search Base**: To search the 'InetSoftRoles' unit, set 'Role Search Base' to "`ou=InetSoftRoles,ou=Roles`".



- **Administrator ID**: The 'Administrator ID' in this case is "`cn=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*.

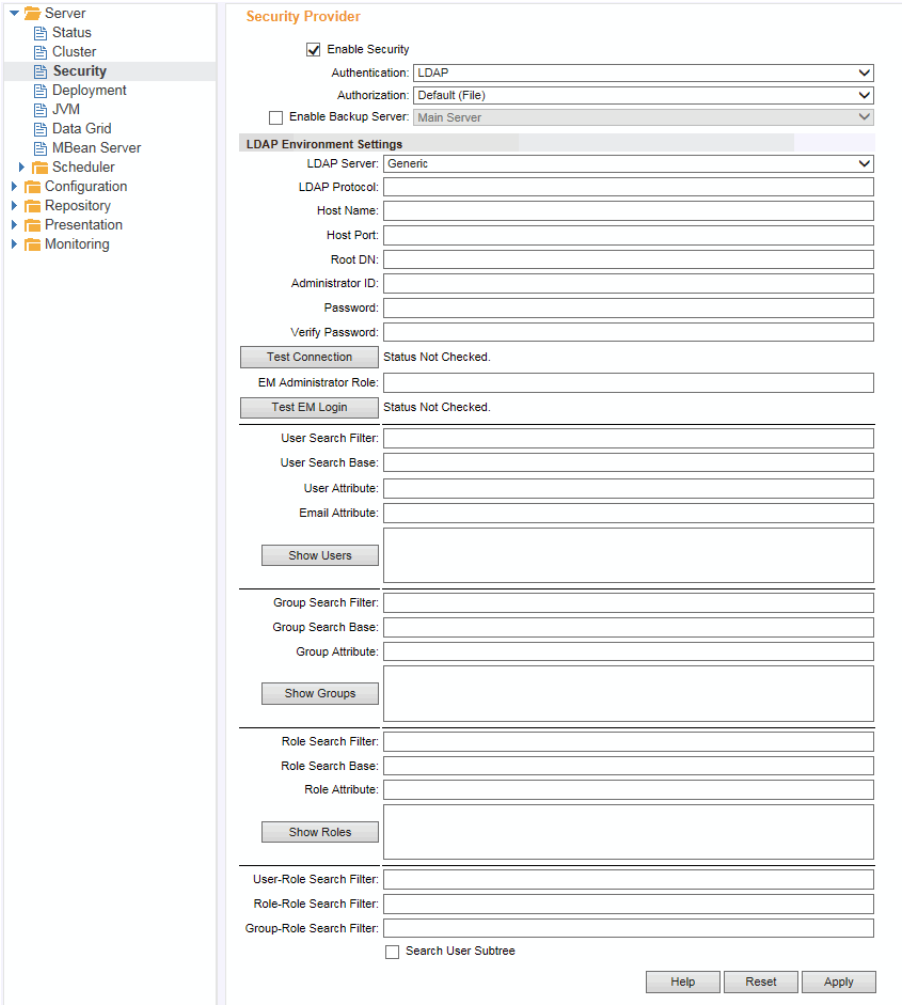
To use an Active Directory server for authorization, you should enable writing to the schema. To do this, follow the steps below:

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 4. 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`

PROPERTY NAME	DESCRIPTION
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
   }

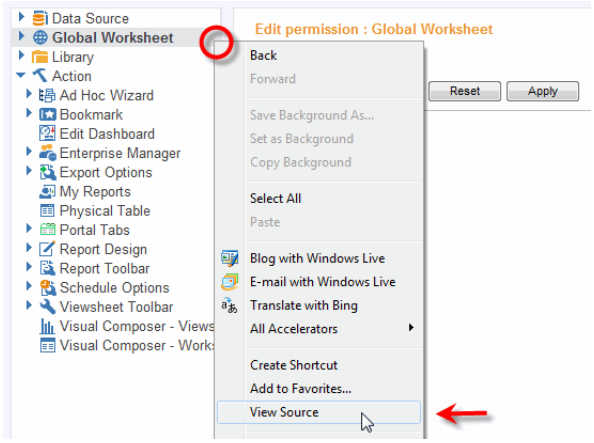
}
```

```
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"
Repository 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

Note: Users, groups, and roles can be created and configured in the Enterprise Manager only when using the 'Default Security Provider'. When 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.

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.

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

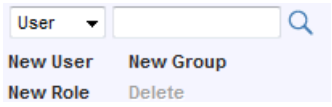
To find specific users, roles, or groups, follow the steps below:

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 31. 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:

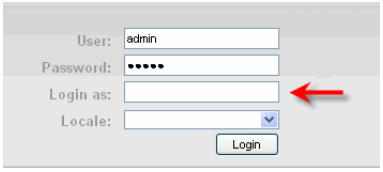
```
login.loginAs=on
```

2. In Enterprise Manager, on the ‘Server’ > ‘Status’ page (under the **Server** tab), press the ‘Reload’ 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 **Repository** 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 **Repository** 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 **Repository** tab tree.
Viewsheet	The repository folder containing the Viewsheet on the **Repository** tab tree.
Folder	The repository folder containing the folder on the **Repository** 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

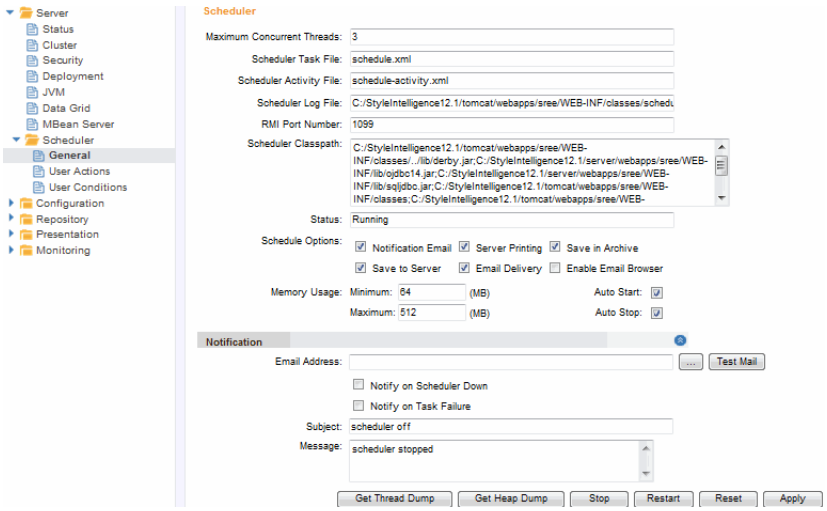
The ‘Scheduler’ is an application that facilitates scheduled batch report and Viewsheet generation. An administrator or an end user (with scheduler permissions) can access this tool via the Enterprise Manager or the User Portal respectively. You can create one or more scheduled ‘tasks’ for which you specify a time condition, reports or Viewsheets to be generated, delivery mechanism (email, print, save to server, 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.

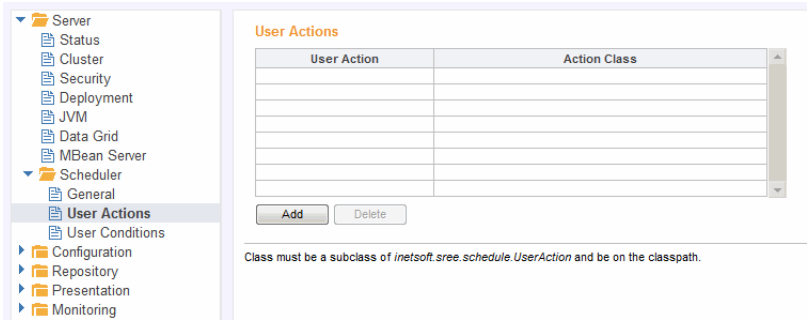
6.1 Configuring the Scheduler

The ‘Scheduler’ can be configured using the Enterprise Manager. Click the **Server** tab and select the ‘Server’ > ‘Scheduler’ > ‘General’ page on the navigation tree.

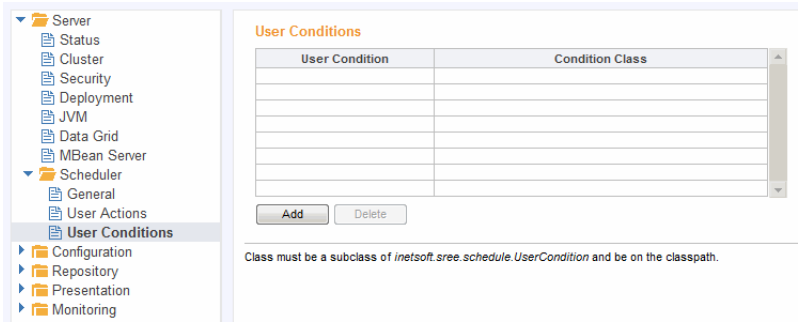
Figure 32. Scheduler Configuration Page



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

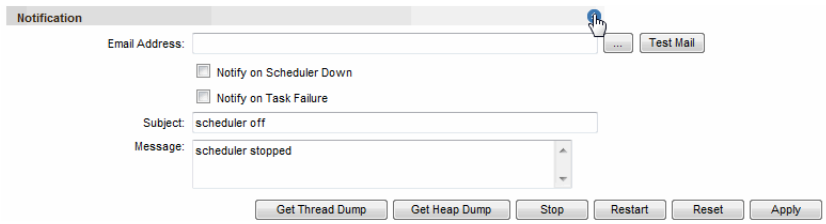
The table below lists the Scheduler properties.

Table 5. 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 server, 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.

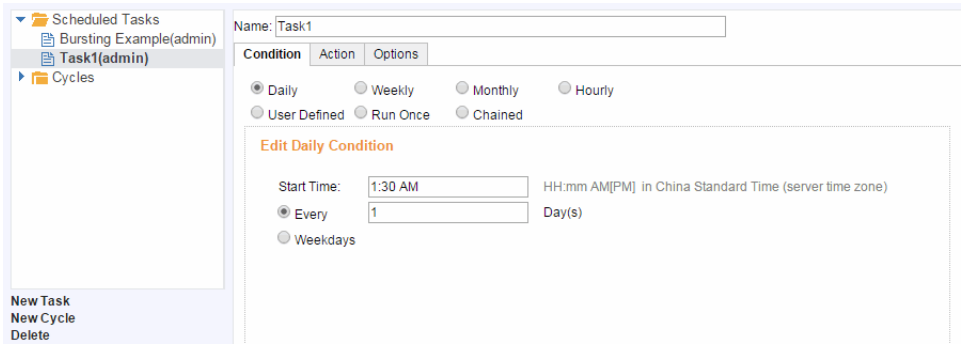
4. Specify the condition's fully qualified class name in the **Condition** tab of the task definition.

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 33. 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 6. 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 (embed in 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. To use HTML in the message body, enable the 'Message body is HTML' option. 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.

OPTION	DESCRIPTION
Save to Server	Save the report on the file system or FTP server in any of the following formats: PDF, HTML, Excel, PowerPoint, RTF, SVG, CSV, Text, XML. To save to file system, 'Path' should specify a valid absolute path, including filename. (The filename extension is added automatically.) To save to FTP server, 'Path' should specify a valid URL in the following format: `(s)ftp://<username>:<password>@<hostname>:<port>/<filename>`, where `password` and `port` are optional. For SFTP connections, the known hosts file is picked up automatically from the user's home directory in the*.ssh/known_hosts* file. Before running the scheduled task, populate the *known_hosts* file by first connecting (SSH) to the target server from each InetSoft server node. To change the *known_hosts* file location, use property `ftp.knownhosts.path`. 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
```

Dashboard Action

A Dashboard action executes one or more dashboards (Viewsheets), and delivers them as emails or simply transmits a notification. The following table explains the available delivery options.

Table 7. Delivery Options

OPTION	DESCRIPTION
Deliver to Emails	Send a dashboard to other people as an email attachment. The dashboard 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. To use HTML in the message body, enable the 'Message body is HTML' option.
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.

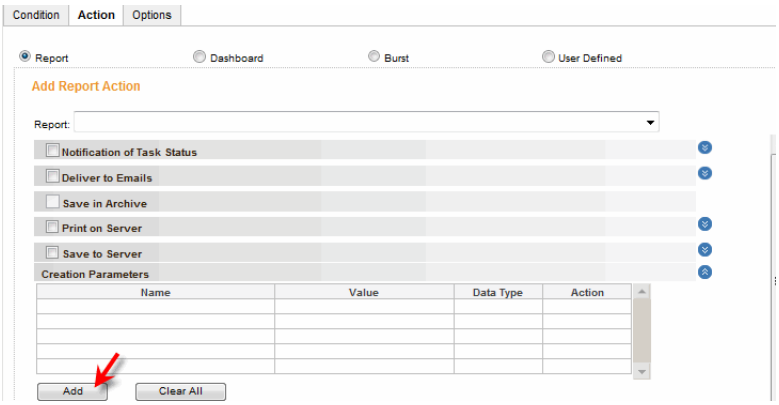
Specifying Multiple Actions

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.

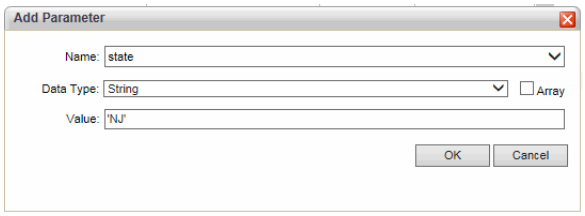


   This opens the 'Add Parameter' dialog box.

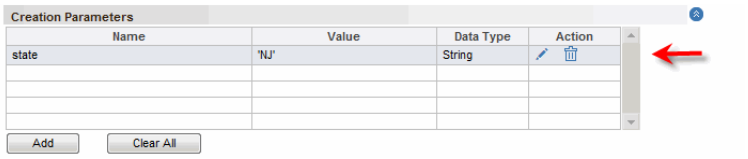
2. In the 'Name' field, select an existing parameter or manually enter the name of a parameter.

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

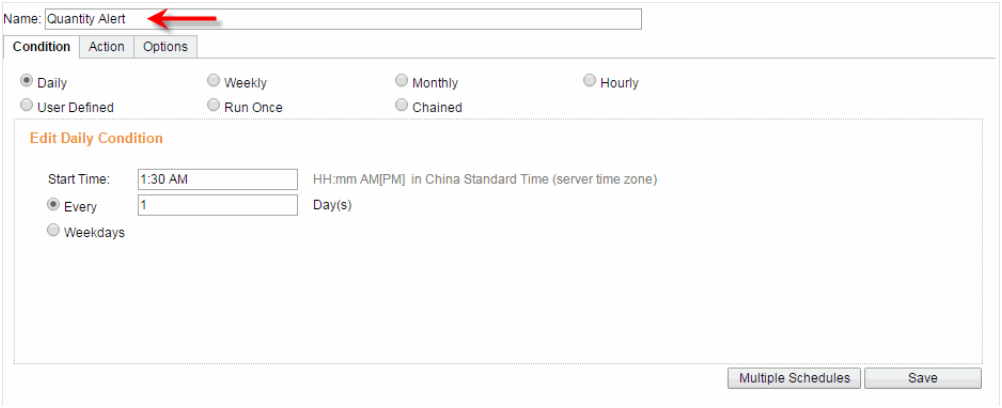
An alert is a notification that a specified event has occurred, for example, that total sales have fallen below some critical level. You can set an alert based on highlight conditions in a report or Viewsheet, or based on Viewsheet Output component ranges.

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

1. Select the **Schedule** tab in the Enterprise Manager.

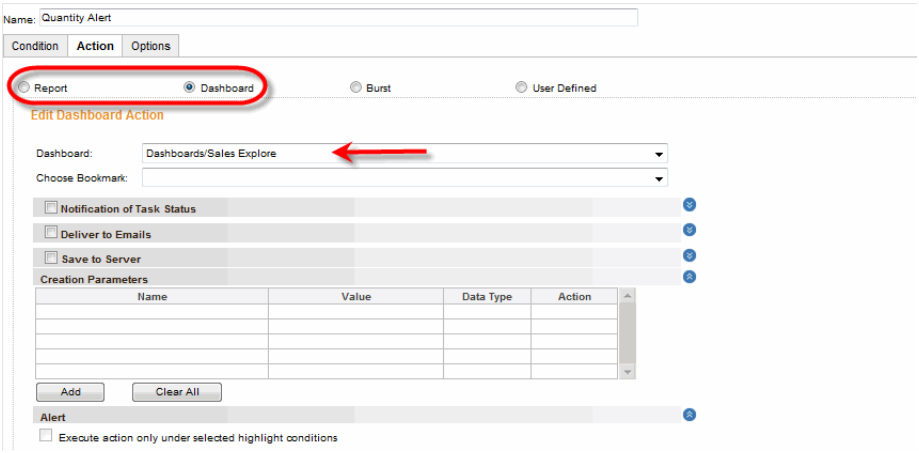
2. Press the ‘New Task’ button below the Schedule table. This creates a new task with the name ‘Task1’, and opens the Editor.

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.

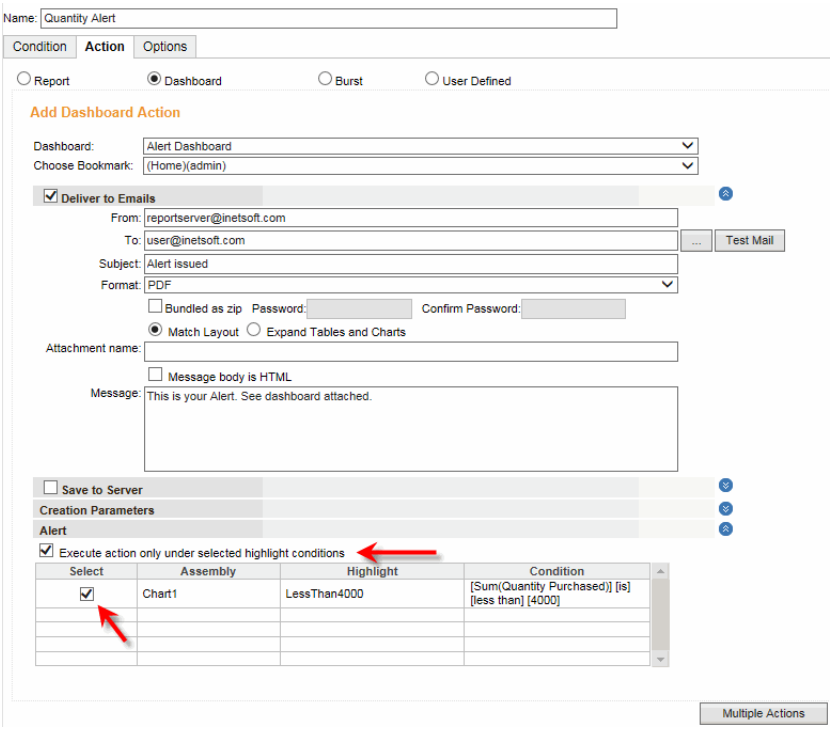


6. Select the actions that you want to perform if the alert is triggered. Generally, you will at least select the 'Deliver To Emails' option. (See *Scheduler Actions* for more about the available actions.) Enter a 'Subject' and 'Message' appropriate for the nature of the alert.

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.

See *Setting an Alert*, in *Using the Portal*, for information on how an end-user can set an alert. See *Automated Alerts* for information about setting an alert based on conditions that are not defined in a highlight.

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.

Gauge 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.

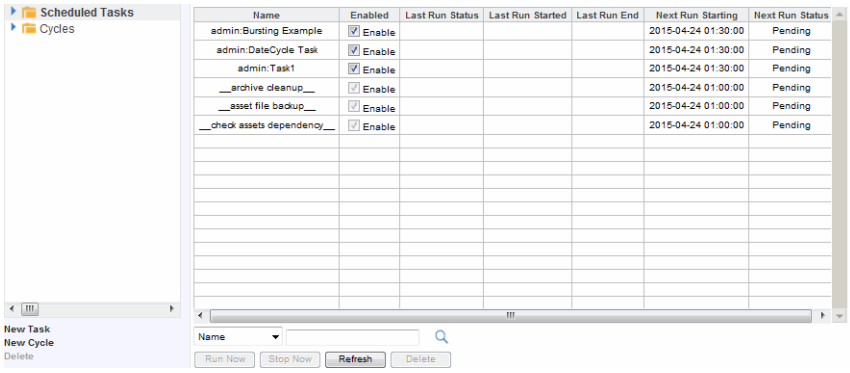
- **Delete if not scheduled to run again**: If a task is scheduled to run once, this option will delete it from the system once it runs.
- **Start From and Stop On**: A task will only execute within the specified date range.
- **Execute As**: A task can be executed as a user, in which case user permissions 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 34. Scheduled Tasks Page



6.5 Scheduler Cycle

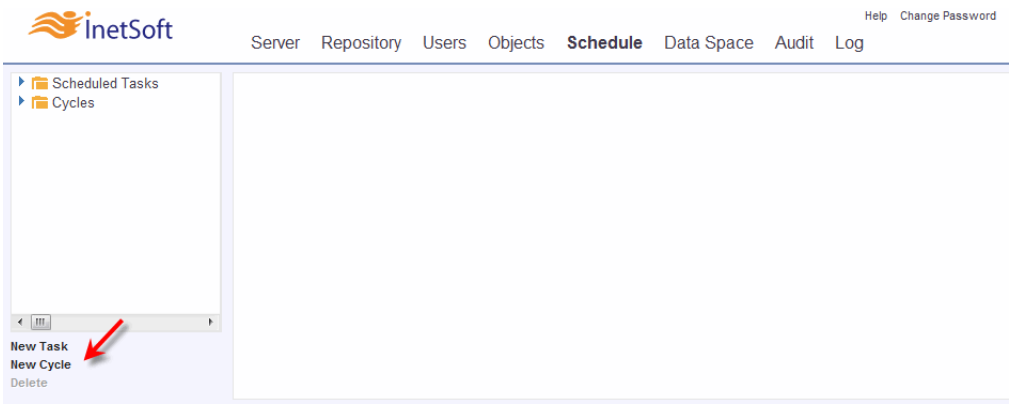
A scheduler cycle is a time condition that can be used to automate materialized view generation or report pre-generation. For more

information about materialized views, see *Materialized Views*. For more information about report pre-generation, see *Configuring Reports*.

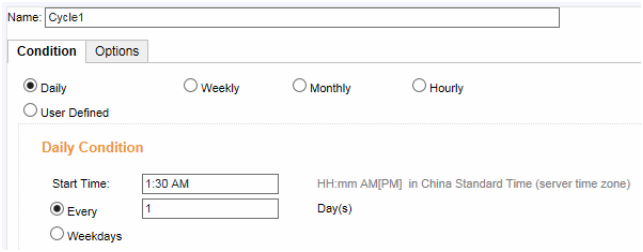
To create a new cycle, follow the steps below:

1. Select the **Schedule** tab in Enterprise Manager.

2. Press the 'New Cycle' link at the bottom of the left panel to create a new cycle.



3. In the 'Name' field enter a name for the cycle, and set the cycle time condition. See *Scheduler Conditions* for details on the different time conditions available.

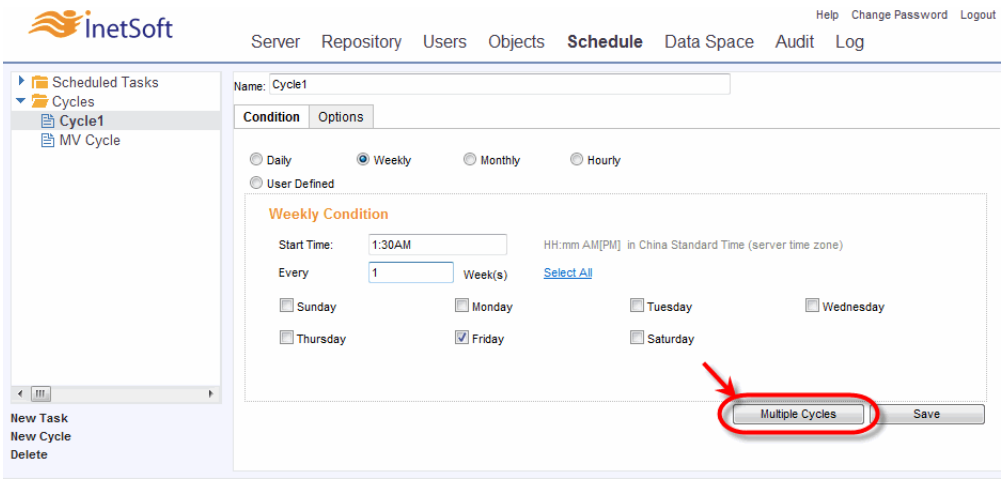


4. Press the 'Save' button at the bottom of the panel.

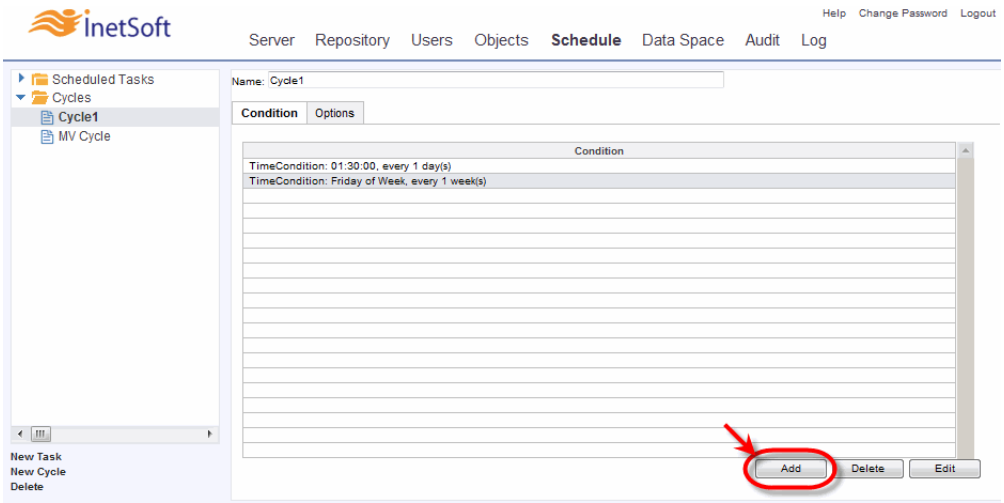
To create multiple conditions on a single cycle, follow the steps below:

1. Create a new cycle as described above.

2. Press the 'Multiple Cycles' button at the bottom of the window.



3. Press the ‘Add’ button to add a new time condition.



4. Specify the time condition as described previously.

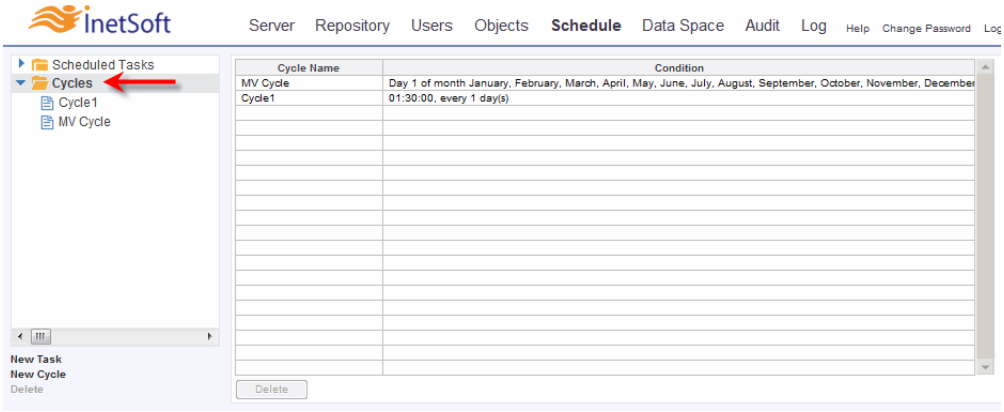
5. To edit one of the multiple conditions, select the condition in the ‘Condition’ table, and press ‘Edit’.

6. To delete one of the multiple conditions, select the condition in the ‘Condition’ table, and press ‘Delete’.

To delete a cycle, follow the steps below:

1. Select the **Schedule** tab in Enterprise Manager.

2. Click on the ‘Cycles’ folder in left panel.



3. Select the cycle you want to delete in the table.

4. Press the 'Delete' button below the table.

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 easily create an automated alert based on existing highlight conditions in a report or dashboard. See *Setting an Alert* for more information.

You can also implement an *automated alert* based on arbitrary conditions 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.

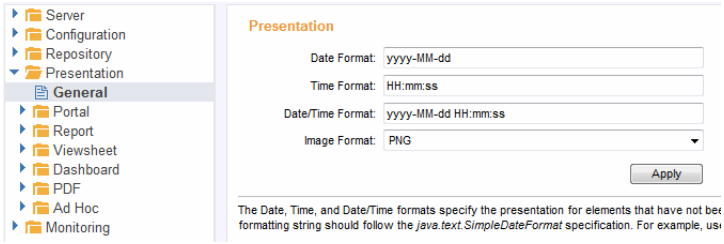
7.1 General Presentation

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 35. The ‘Presentation’ > ‘General’ Page



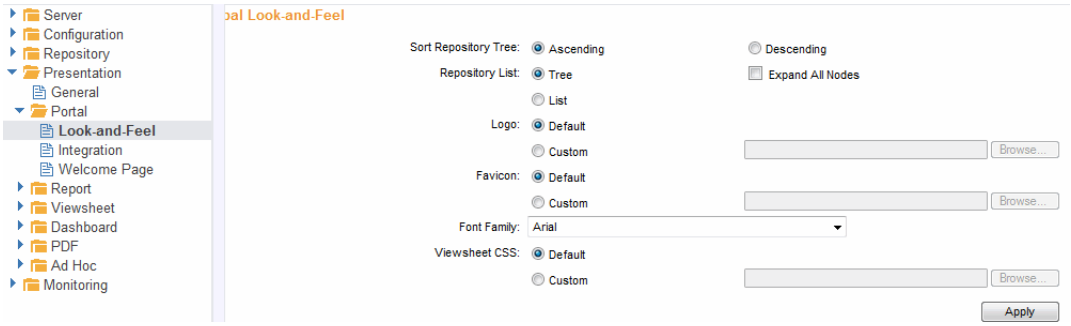
Under the ‘Presentation’ node are options to configure the appearance of Portal items, presentation of web-based reports, and PDF export options, respectively.

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 *Creating Portal CSS Styles* for more details.

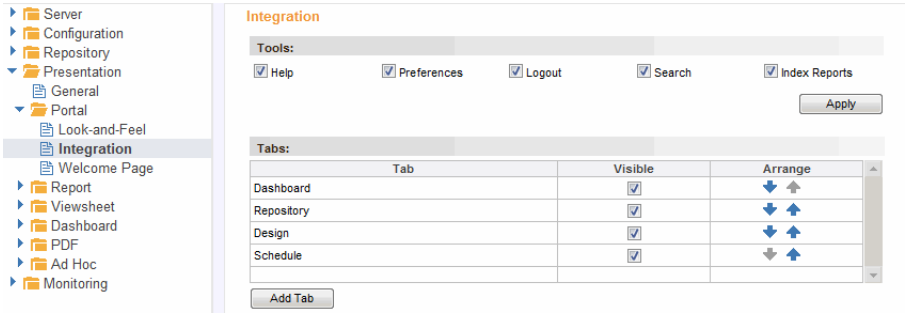


- **Sort Repository Tree**: Specifies the sort order of items in the Repository listing.
- **Repository List**: Controls the Portal's Repository style. The default is 'Tree', which displays Repository folders and reports in an hierarchical structure. The 'Expand All Nodes' option pre-expands all of the 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 dashboard and report component styles. The CSS file can specify styles for alignment, font, border, and color. See 7.8 *Styling with CSS* for more information.

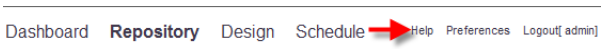
7.2.2 Integration

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 36. 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.

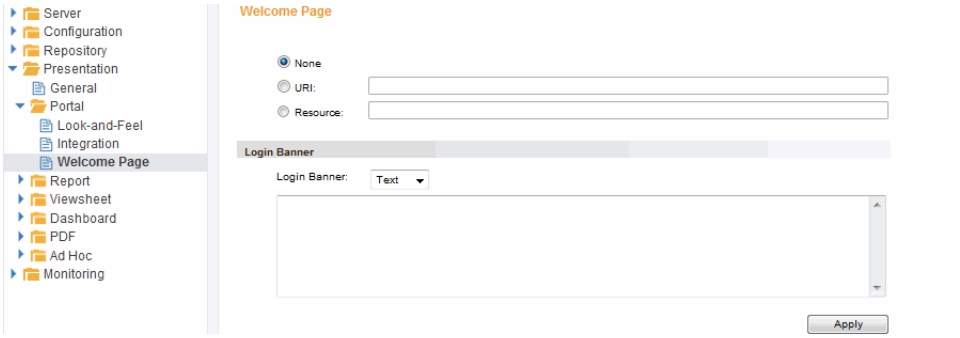


The ‘Logout’ option is only in effect when security is enabled. The ‘Search’ option is only in effect when archiving is turned on, and enables complete searching within archived reports. The ‘Index Reports’ option enables searching of non-archived reports as well, but searches are limited to the report name, alias, description, keywords, and comments.

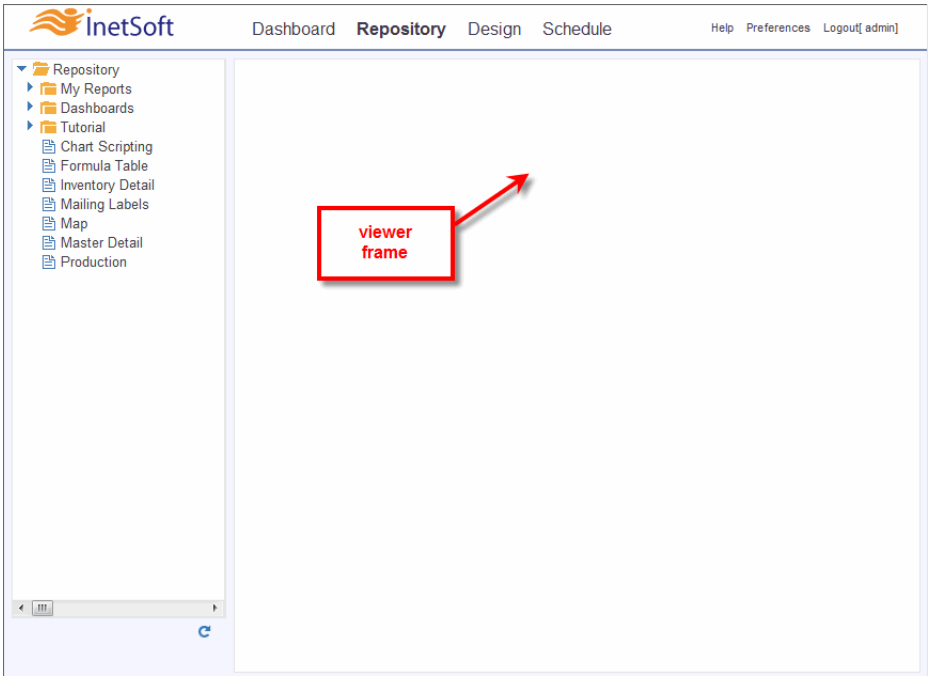
See Also
Archive, for more information about archiving.
Advanced Server Integration, for more advanced approaches to web integration.

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.



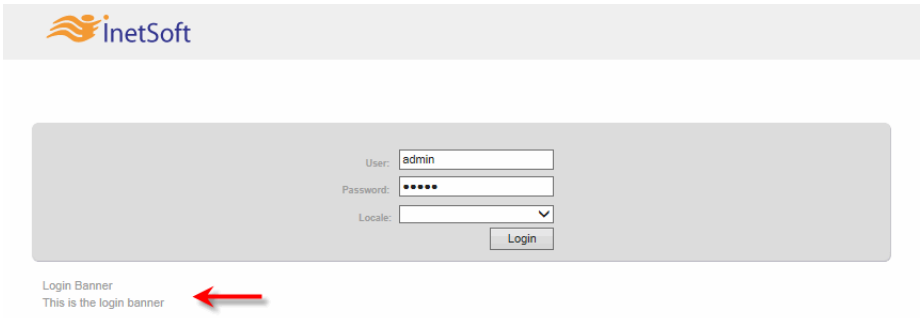
If you do not specify a welcome page, the viewer frame will remain blank.



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

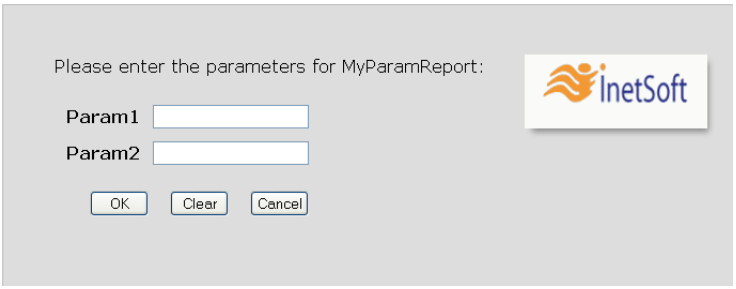
The ‘Parameter Dialog HTML Template’ field specifies an HTML template file (as a resource) to be used for the default parameter prompting screen. The template can contain regular HTML as well as three optional tags, `$(report.parameters)`, `$(report.title)`, and `$(report.description)`. These tags correspond, 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*.

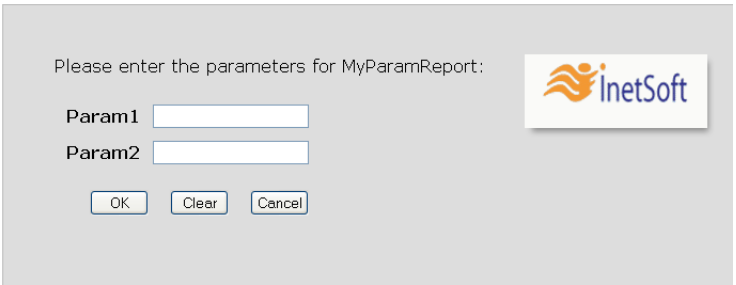
A sample CSS file which corresponds to the custom HTML template in the *Parameter Dialog HTML Template* section is shown below:

```
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:

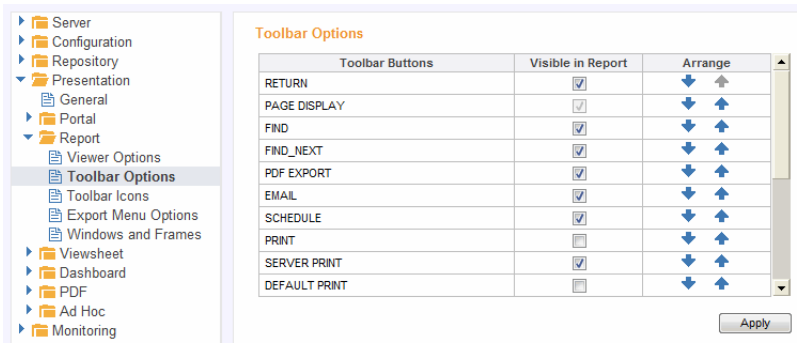
DIALOG	CLASSES OR IDS
all	body{}, select{}, option{}, label{}, input.button{}, input.textbox{}
Find	#find_scope{}, #find_column{}, #find_operator{}, #find_logicOperator{}, #find_value{}, #find_moreButton{}, #find_fewerButton{}, #find_findButton{}, #find_clearButton{}, #find_cancelButton{}, #find_body{}
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 37. 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 38. 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.

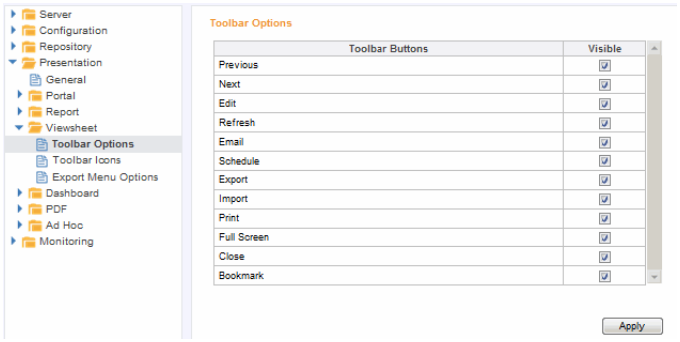
7.4 Viewsheet

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 39. Viewsheet Toolbar Options



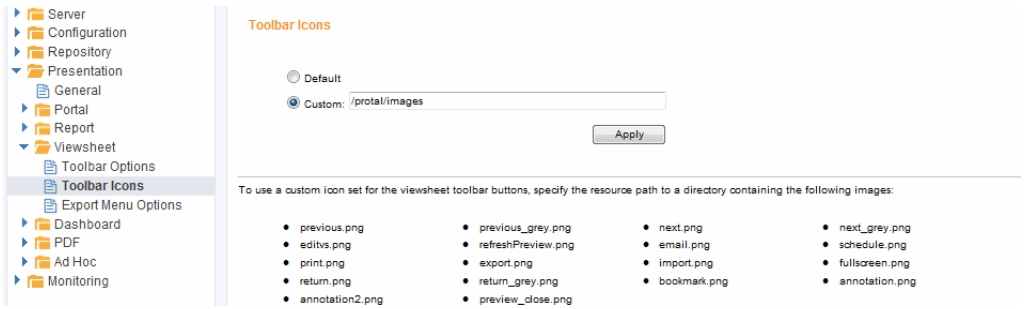
The ‘Help’ option enables or disables the display of the ‘Help’ button in the Chart Editor.

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 40. Viewsheet Toolbar Icons



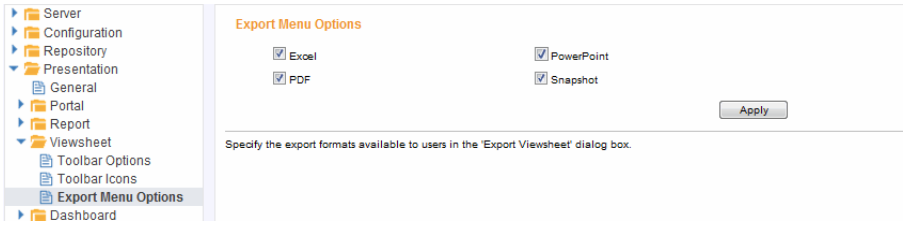
The required file names are listed on the ‘Toolbar Icons’ page, and the recommended image size is a 21-pixel square. If a custom image is not found for a particular button, the default button image is used.

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 41. 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.

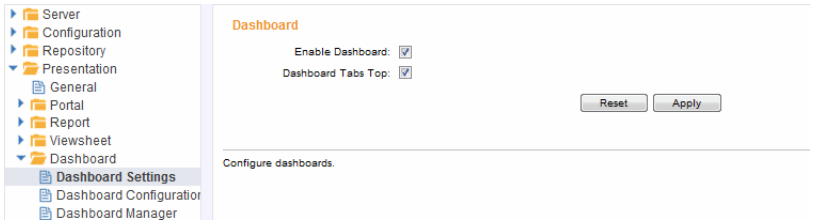
7.5.1 Enabling Dashboards

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

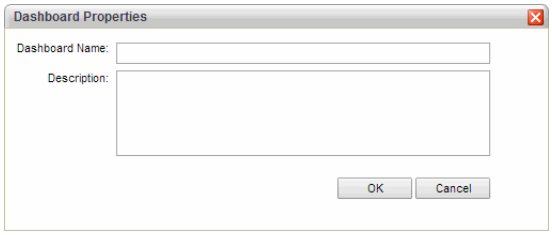
A dashboard consists of one or more Viewsheets. To create a new dashboard, follow these steps:

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.

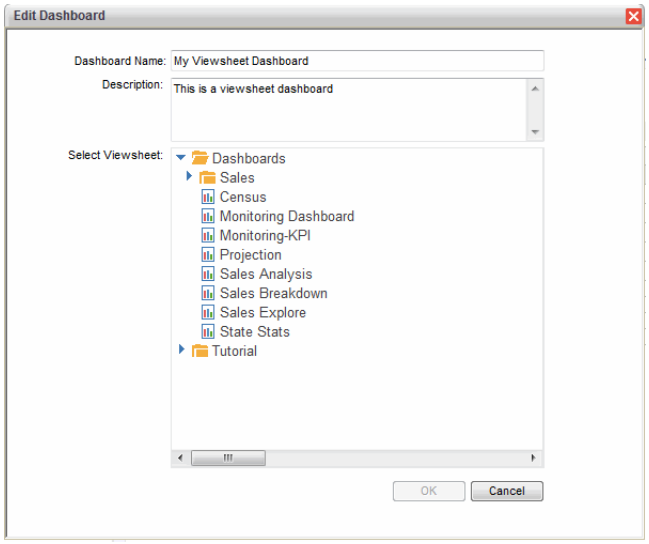
3. In the ‘Dashboard Name’ field, enter a name for the new dashboard. This name will be displayed on this dashboard’s tab in the User Portal.

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 Dashboard 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 Dashboard 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

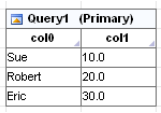
In the following example, you will create a Viewsheet that receives its data from an embedded Table in a Data Worksheet. You will then configure the Viewsheet to automatically update when the `DataChangeClient` is triggered.

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

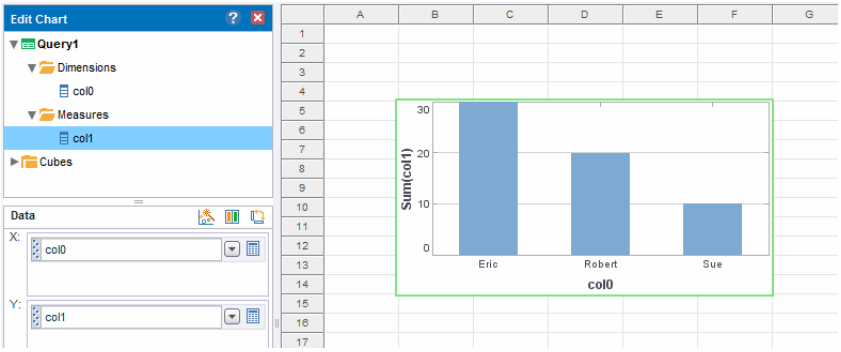
3. Right-click the 'col1' header, and select 'Column Type' from the context menu. This opens the 'Column Type' dialog box.

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*.)

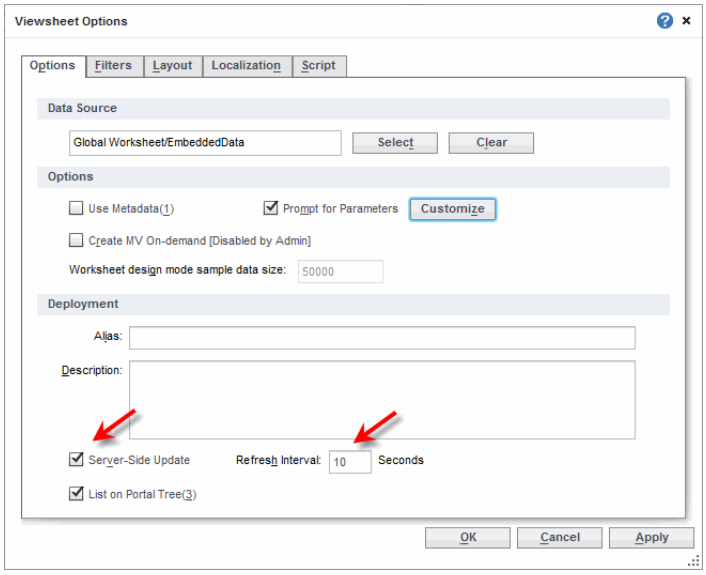
6. Create a new Viewsheet based on the 'EmbeddedData' Worksheet.

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.

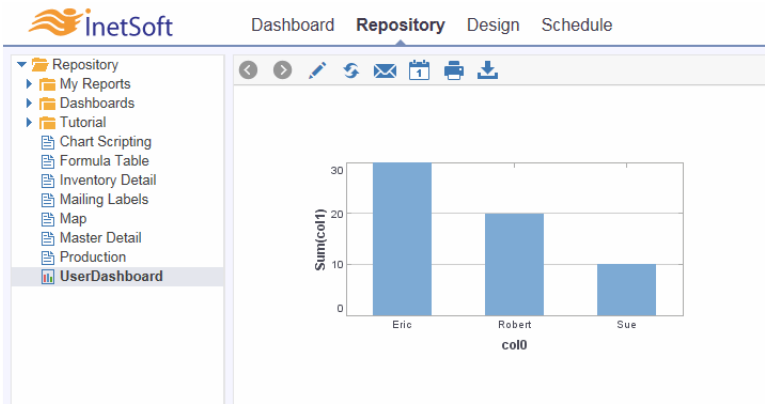
9. In the ‘Deployment’ panel at the bottom, enable the ‘Server-Side Update’ option, and set the ‘Refresh Interval’ to 10 seconds.



10. Click ‘OK’ to close the dialog box.

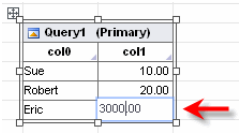
11. Save the Viewsheet with the name ‘UserDashboard’.

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.

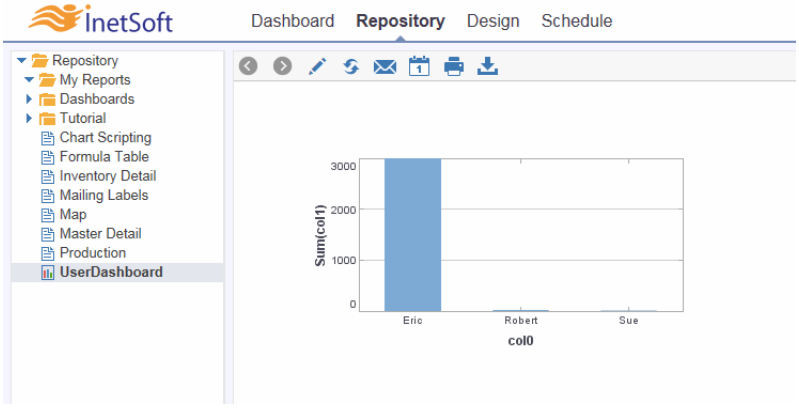


15. Return to the User Portal. (Do not reload the Viewsheet.) Note that the Viewsheet remains the same, even though the underlying data has changed.

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.



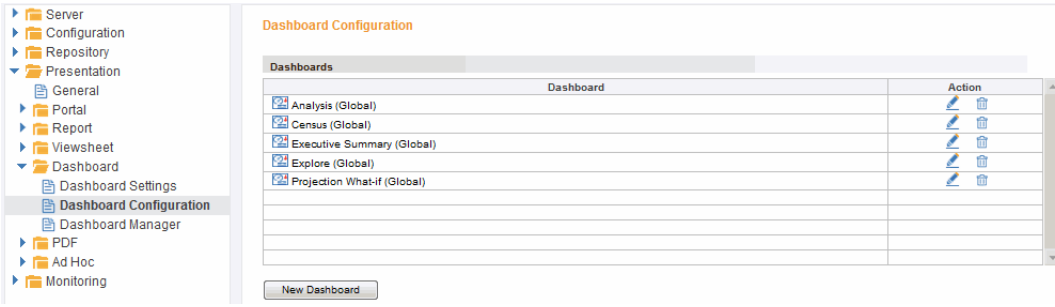
7.5.4 Editing or Deleting a Dashboard

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

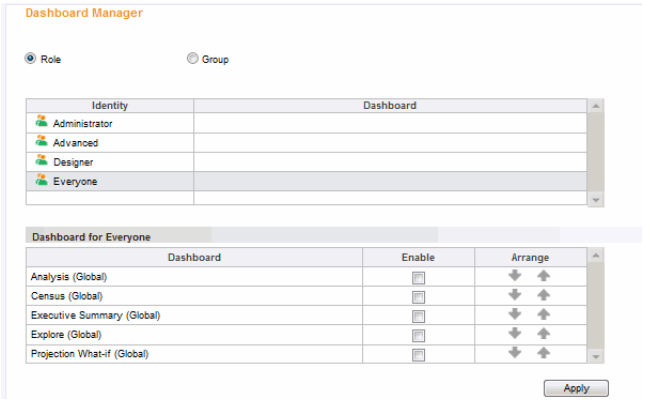
Before a user can access the dashboards that you have created in Enterprise Manager, you need to assign the user appropriate permissions based on the user's role or group. In general, you should first configure a security provider (see *Specifying a Security Provider*), otherwise, the only role available 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.

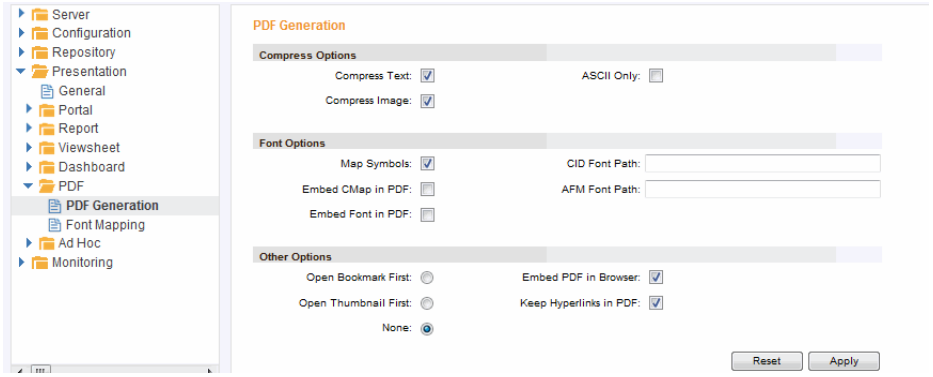
7.6 PDF Generation

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 *Advanced Report Scripting*, for more information.

7.8 Styling with CSS

You can deploy CSS files to set styles for dashboard components and for User Portal elements. The following sections explain how to create and upload these CSS definition files.

7.8.1 Creating a Component CSS Definition File

To prepare the CSS definition file for a dashboard or report, create a text file (UTF-8 format, without BOM) that contains the desired Class, ID, or Type definitions. Deploy the CSS file as described in *Uploading a CSS File*. Formatting specified in the CSS file will be applied to all elements that are assigned the given Class or ID or that have the specified Type.

The CSS style for a given property (e.g., `background-color`) is applied in increasing order of specificity:

1. **Global default (`*`)**
   See *CSS Global Default (*)* for syntax.

2. **CSS Type**
   See *CSS Type* and *CSS ChartPalette* for syntax.

3. **CSS Type[attribute=value]**
   See *CSS Type* for syntax.

4. **CSS Class**
   See *CSS Class* and *CSS TableStyle* for syntax.

5. **CSS ID**
   See *CSS ID* for syntax.

```
Formatting applied manually from the 'Format' dialog box always
overrides corresponding CSS style.
```

For example, a CSS property value assigned by CSS ID overrides the same property value set by CSS Type. See *CSS Component Properties* for details on which component properties (background color, line style, borders, etc.) can be assigned using CSS. See *CSS Combinations* for information on how to combine different CSS selectors.

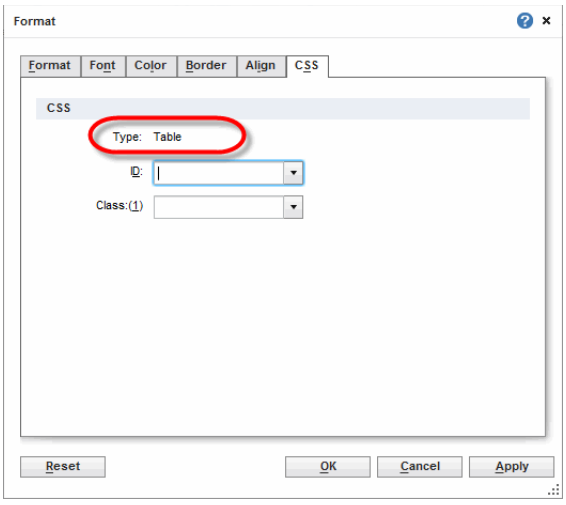
See Also
Example: Creating a CSS Definition File, for a sample CSS file.

CSS Type

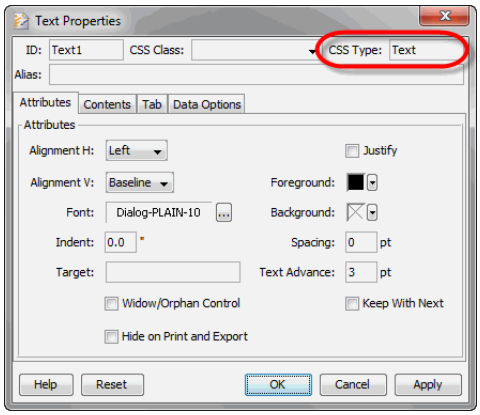
To specify a CSS Type definition, use the following syntax:

```
CheckBox {
   background-color: #FFEEBB;
   ...
   ...
}
```

The Type definition applies styles universally to all elements of a particular type (e.g., all CheckBoxes). For Viewsheet components, the 'Type' is displayed in the **CSS** tab of the element's 'Format' dialog box. See *CSS Tab* in *Advanced Dashboard Design* for a list of all available CSS component types.



For report components, the ‘CSS Type’ is displayed in the element’s ‘Properties’ dialog box. See *CSS Type Selectors* in *Advanced Report Design* for a list of all available CSS component types.



CSS TableStyle

To specify a CSS TableStyle definition, use the following syntax.

```
.className TableStyle[region='region'] {
   ...
   ...
}
```

For example, to define a class `grayHeader` that sets the background color of header rows to gray, use the following:

```
.grayHeader TableStyle[region=HeaderRow]{
   background-color: gray;
}
```

Use `TableStyle` without attributes to define global default formatting for tables:

```
.className TableStyle {
   ...
   ...
}
```

See *CSS TableStyle Properties* for details on which table regions can be styled using CSS.

CSS ChartPalette

To specify a CSS ChartPalette color definition for color index *n*, use the following syntax:

```
ChartPalette[index='n'] {
   color: azure;
   ...
   ...
}
```

ChartPalette specification can be combined with CSS Class to tailor the palette to charts with a particular class. See *CSS Combinations* for an example.

CSS Class

To specify a CSS Class definition, use the following syntax:

```
.className {
   background-color: yellow;
   ...
   ...
}
```

For example:

```
.Header3 {
   background-color: yellow;
   font-size: 24px;
   font-family: arial;
}
```

See *Styling a Dashboard with CSS* in *Advanced Dashboard Design* and *Styling a Report with CSS* in *Advanced Report Design* for information on how to assign a CSS Class or ID to a component.

CSS ID

To specify an CSS ID definition, use the following syntax:

```
#id {
   background-color: #FFEEBB;
   ...
   ...
}
```

For example:

```
#SalesTable {
   background-color: yellow;
   font-size: 10pt;
   font-family: arial;
}
```

See *Styling a Dashboard with CSS* in *Advanced Dashboard Design* and *Styling a Report with CSS* in *Advanced Report Design* for information on how to assign a CSS Class or ID to a component.

CSS Combinations

To restrict a Type style to a particular Class, use the CSS Type and Class in combination.

Example 1: To apply a blue color to table titles that have class 'SalesTable', use the following syntax:

```
TableTitle.SalesTable {
   color: blue;
}
```

Example 2: To apply a blue color to the third color in the chart palette for charts having class ‘SalesChart’, use the following syntax:

The initial ‘Chart’ qualifier can be omitted.

```
Chart.SalesChart ChartPalette[index="3"] {
   color: blue;
}
```

Example 3: To apply an alternating-row yellow TableStyle to tables with class ‘SalesTable’, use the following:

The initial ‘Table’ qualifier can be omitted.

```
Table.SalesTable TableStyle[region=Body][pattern=OddRow] {
   background-color: yellow;
}
```

Because the Viewsheet designer can only assign a Class or ID that is appropriate for the Type of the given component, you can use a single Class name or ID name to assign different styles to different component types. Consider the following CSS definition:

```
#Highlight {
   background-color: red;
}
Chart#Highlight {
   background-color: pink;
}
Table#Highlight {
   background-color: magenta;
}
```

In this case, when the Viewsheet designer selects the ‘Highlight’ ID for a Chart component, the CSS applies a pink background. When the Viewsheet designer selects the ‘Highlight’ ID for a Table component, the CSS applies a magenta background. When the Viewsheet designer selects the ‘Highlight’ ID for any other component type, the CSS applies a red background.

See *Styling a Dashboard with CSS* in *Advanced Dashboard Design* and *Styling a Report with CSS* in *Advanced Report Design* for information on how to assign a CSS Class or ID to a component.

CSS Global Default (*)

The global default (`*`) type defines default CSS properties. For example:

```
* {
  background-color: yellow;
  color: red;
  alpha: 0.5
  font-style: italic;
  font-weight: bold;
  font-size: 12px;
```

```
  font-family: arial;
  border-style: double;
  border-color: blue;
}
```

The values assigned to `*` are used to set properties for **all** CSS Types that do not have CSS explicitly assigned. For example, if a `TableTitle` style is not specifically defined, the properties for `TableTitle` are set by `*`, and are therefore **not** inherited from the `TableTitle` parent type, `Table`. Therefore, if you want to override a property assigned to `*` for a particular CSS Type, you must also supply overrides for any applicable child CSS Types as well.

See Also

CSS TableStyle Properties for information on how to define TableStyle CSS.

CSS Component Properties, for details on which component properties can be assigned styles.

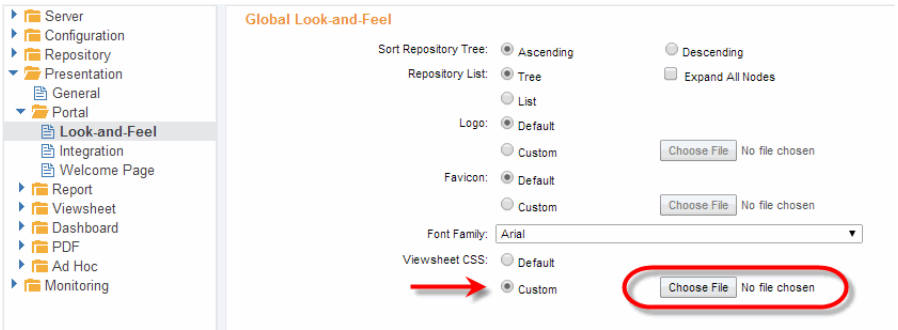
7.8.2 Uploading a CSS File

To upload an existing CSS file to the server, follow the steps below. (For information on how to create a CSS file, see *Creating a Component CSS Definition File*.)

1. Open the 'Presentation' > 'Portal' > 'Look-and-Feel' page in Enterprise Manager.

2. Select the 'Custom' option for 'Viewsheet CSS'.

3. Press the 'Choose File' button. This opens the standard file upload dialog box.



There can be only a single component CSS file at any time.

4. Locate the CSS file that you saved, and press 'Open'. The selected file is now displayed next to the 'Choose File' button.

5. Press 'Apply'. This uploads the specified CSS file to the server as *{SREE Home}/portal/format.css*.

When you upload a new CSS file, the new file replaces the existing *format.css* file. (The CSS definitions in the new and old files are **not** merged.)

7.8.3 CSS Component Properties

The table below lists the available properties and values for styling Viewsheet and report non-tabular components. (To apply CSS styles to Tables, Crosstabs, and Freehand Tables, see *CSS TableStyle Properties*.) For a list of component regions that can be styled using these properties, see *CSS Tab* in *Advanced Dashboard Design* or *CSS Type Selectors* in *Advanced Report Design*.

CSS properties in the table below can be applied using either Component *CSS Type* (e.g., "GaugeTickLabel"), *CSS Class*, *CSS ID*, or by using *CSS Global Default (*)*. Notes in the **Restrictions** column apply to both Component and Global Types unless otherwise indicated.

PROPERTY	AVAILABLE VALUES	RESTRICTIONS
text-align	'left', 'right', 'center'	**Dashboard**: Does not apply to Shape, Chart Y-axis title or labels, ChartPalette, Tabbed Interface, GaugeTickLabel, GaugeValueFill, Chart (component), Gauge (component). **Report**: Does not apply to Plot Area Value, Target Label, Section, Image, Painter, Image Button, Running Total, Newline, Space, Tab, Page Break, Conditional Page Break, Area Break, ChartPalette.
vertical-align	'top', 'middle', 'bottom'	**Dashboard**: For charts, applies only to Chart Y-axis titles. Does not apply to Shape, TextInput, Tabbed Interface, GaugeTickLabel, GaugeValueFill, Chart (component), Gauge (component). **Report**: Does not apply to Table, Crosstab, Plot Area Value, Target Label, Section, Image, Painter, Image Button, Running Total, Newline, Space, Tab, Page Break, Conditional Page Break, Area Break, ChartPalette.
word-wrap	'normal', 'break-word'	**Dashboard**: Does not apply to Shape, Chart, Image, Tabbed Interface, GaugeTickLabel, GaugeValueFill, Chart (component), Gauge (component). **Report**: Not available for any components.

PROPERTY	AVAILABLE VALUES	RESTRICTIONS
font-style	‘normal’, ‘italic’	**Dashboard**: Does not apply to Shape, Image, ChartPalette, GaugeValueFill, Gauge (component). **Report**: Does not apply to Section, Bean, Separator, Running Total, Newline, Space, Tab, Page Break, Conditional Page Break, Area Break, ChartPalette.
font-weight	‘normal’, ‘bold’	**Dashboard**: Does not apply to Shape, Image, ChartPalette, GaugeValueFill, Gauge (component). **Report**: Does not apply to Section, Bean, Separator, Running Total, Newline, Space, Tab, Page Break, Conditional Page Break, Area Break, ChartPalette.
font-size	‘xx-small’, ‘x-small’, ‘small’, ‘medium’, ‘large’, ‘x-large’, ‘xx-large’, or size in pixels	**Dashboard**: Does not apply to Shape, Image, ChartPalette, GaugeValueFill, Gauge (component). **Report**: Does not apply to Section, Bean, Separator, Running Total, Newline, Space, Tab, Page Break, Conditional Page Break, Area Break, ChartPalette.
font-family	Font name	**Dashboard**: Does not apply to Shape, Image, ChartPalette, GaugeValueFill, Gauge (component). **Report**: Does not apply to Section, Bean, Separator, Running Total, Newline, Space, Tab, Page Break, Conditional Page Break, Area Break, ChartPalette.
font	Shorthand: font-style font-weight font-size font-family	**Dashboard**: Does not apply to Shape, Image, ChartPalette, GaugeValueFill, Gauge (component). **Report**: Does not apply to Section, Bean, Separator, Running Total, Newline, Space, Tab, Page Break, Conditional Page Break, Area Break, ChartPalette.
text-decoration	‘underline’, ‘line-through’	**Dashboard**: Does not apply to Shape, Image, Gauge, GaugeTickLabel, GaugeValueLabel, GaugeValueFill, ChartPalette, Gauge (component). **Report**: Does not apply to Section, Bean, Separator, Running Total, Newline, Space, Tab, Page Break, Conditional Page Break, Area Break, ChartPalette.

PROPERTY	AVAILABLE VALUES	RESTRICTIONS
color	Color value (hex, rgb, color string)	**Dashboard**: Does not apply to Image, Gauge (component). **Report**: Does not apply to Section, Bean, Running Total, Newline, Space, Tab, Page Break, Conditional Page Break, Area Break, Image Button, Image, Painter.
background-color	Color value (hex, rgb, color string)	**Dashboard**: Does not apply to GaugeTickLabel, GaugeValueLabel, GaugeValueFill, Line, ChartPalette. **Report**: Does not apply to Target Label, Section, Bean, Separator, Running Total, Newline, Space, Tab, Page Break, Conditional Page Break, Area Break, Button, ChartPalette.
opacity	Float in range [0.0, 1.0]	**Dashboard**: Does not apply to ChartPalette, GaugeTickLabel, GaugeValueLabel, GaugeValueFill, Line. **Report**: Not available for any components.
border-style	Line style: 'none', 'dotted', 'dashed', 'solid', 'double' Single value applies to all borders, or Two values (top/ bottom left/right), or Three values (top right/left bottom), or Four values (top right bottom left)	**Dashboard**: Does not apply to Chart regions except for Chart (component). Does not apply to ChartPalette, Line, Rectangle, Oval, GaugeTickLabel, GaugeValueLabel, GaugeValueFill, ComboBox drop-down. **Report**: Does not apply to Chart, Section, Text, Image, Painter, TextField, TextArea, Button, Image Button, CheckBox, RadioButton, Choice, List, Date Combo, Bean, Separator, Bullet, TOC, Heading, Running Total, Newline, Space, Tab, Page Break, Conditional Page Break, Area Break, Button, ChartPalette. Use TableStyle CSS (*CSS TableStyle Properties*) at for component-level table formatting.
border-top-style border-right-style border-bottom-style border-left-style	Line style: 'none', 'dotted', 'dashed', 'solid', 'double'	See 'border-style'.
border-top-color border-right-color border-bottom-color border-left-color	Color value (hex, rgb, or color string)	See border-color.

PROPERTY	AVAILABLE VALUES	RESTRICTIONS
border-color	Color value (hex, rgb, or color string) Single value applies to all borders, or Two values (top/ bottom right/left), or Three values (top right/left bottom), or Four-values (top right bottom left)	See ‘border-style’.
border-top border-right border-bottom border-left	Shorthand: border-top-style border-top-color border-right-style border-right-color border-bottom-style border-bottom-color border-left-style border-left-color	See ‘border-style’ and border-color.
border	Shorthand: border-style border-color	See ‘border-style’ and border-color.

7.8.4 CSS TableStyle Properties

The table below lists table regions that can be formatted with a CSS TableStyle. See *CSS TableStyle* for more information about the TableStyle syntax.

If the CSS file does not include specific formatting for a table region, the TableStyle selector with the most similar attributes is used. For example, if `EvenCol` formatting is omitted, `Body` formatting is used instead. If no substitute is found, the default `TableStyle` formatting is used.

For `GroupHeader` and `GroupRows`, if there is no CSS specified for a particular grouping level, styling for the group header/footer row defaults to the

appropriate `OddRow` or `EvenRow` style. If these are not specified, styling defaults to `Body`.

TABLE REGION	DESCRIPTION
HeaderRow	Top row containing column headers. Syntax: `TableStyle[region=HeaderRow]`. **Restrictions**: Does not support `opacity` and `word-wrap` attributes.
HeaderCol	Left- most column containing row headers. Syntax: `TableStyle[region=HeaderCol]`. **Restrictions**: Available only for Crosstabs. Does not support `opacity` and `word-wrap` attributes.
TrailerRow	Bottom row containing grand total or other aggregation. Syntax: `TableStyle[region=TrailerRow]`. **Restrictions**: Does not support `opacity` and `word-wrap` attributes. Does not apply to Viewsheet Table.
TrailerCol	Right- most column containing grand total or other aggregation. Syntax: `TableStyle[region=TrailerCol]`. **Restrictions**: Available only for Crosstabs. Does not support `opacity` and `word-wrap` attributes.
Body	All table cells other than HeaderRow and HeaderCol. Syntax: `TableStyle[region=Body]`. **Restrictions**: Does not support `opacity` and `word-wrap` attributes.
Table	Line type and color for outer borders of table. Syntax: `TableStyle[region=Table]`.
GroupHeader	Group header of a table. Syntax: `TableStyle[region=Body][type=GroupHeader][level='n']`, where `n` is the integer group level. **Restrictions**: Does not support `opacity` and `word-wrap` attributes. Does not apply to Viewsheet Table.
GroupFooter	Group footer of a table. Syntax: `TableStyle[region=Body][type=GroupFooter][level='n']`, where `n` is the group level. **Restrictions**: Does not support `opacity` and `word-wrap` attributes. Does not apply to Viewsheet Table.
OddRow	Every other row, starting with row index 1.Syntax: `TableStyle[region=Body][pattern=OddRow]`. **Restrictions**: Does not support `opacity` and `word-wrap` attributes.
EvenRow	Every other row, starting with row index 2 Syntax: `TableStyle[region=Body][pattern=EvenRow]`. **Restrictions**: Does not support `opacity` and `word-wrap` attributes.
OddCol	Every other row, starting with column index 1. Syntax: `TableStyle[region=Body][pattern=OddCol]`. **Restrictions**: Does not support `opacity` and `word-wrap` attributes.

TABLE REGION	DESCRIPTION
EvenCol	Every other row, starting with column index 2. Syntax: `TableStyle[region=Body][pattern=EvenCol]`. **Restrictions**: Does not support `opacity` and `word-wrap` attributes.

7.8.5 Example: Creating a CSS Definition File

In this example, you will create a new CSS file.

1. Create a plain text file with the following contents.

```
/* Global format for all report/dashboard text */
/* See Example: Global Format (*) for details */
* {
   font-family: "Verdana";
}

/* Formatting for all Chart-type components */
/* See Formatting by Component and Subcomponent Type for
details */
Chart {
   background:Azure;
}

/* Formatting for all TableTitle sub-components */
/* See Formatting by Component and Subcomponent Type for
details */
TableTitle {
   color: green;
}

/* Formatting for all ChartAxisTitle sub-components */
/* See Formatting by Component and Subcomponent Type for
details */
ChartAxisTitle {
   background: yellow;
}

/* Formatting for all ChartAxisTitle X-axis sub-components
*/
/* See Formatting by Component and Subcomponent Type for
details */
ChartAxisTitle[axis=x] {
   background:DarkKhaki;
}

/* Formatting for all components having CSS Class "Header2"
*/
/* See Formatting by Component Class and ID for details */
.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;
}
```

```
/* Formatting for all components having CSS Class "AxisText"
*/
/* See Formatting by Component Class and ID for details */
.AxisText {
   font-size: 14pt;
   font-weight: bold;
}

/* Formatting for Table-type components having CSS ID
"SalesTable" */
/* See Formatting by Component Class and ID for details */
Table#SalesTable {
   background-color: #FFEEBB;
}

/* Formatting for odd rows of tables having CSS Class
"YellowRows" */
/* See Formatting by Component Class and ID for details */
.YellowRows TableStyle[region=Body][pattern=OddRow] {
   background-color: yellow;
}
```

2. Save the text file in UTF-8 format (without BOM) with the name *myStyle.css*.

3. Open the ‘Presentation’ > ‘Portal’ > ‘Look-and-Feel’ page in Enterprise Manager.

4. Select the ‘Custom’ option for ‘Viewsheet CSS’.

5. Press the ‘Choose File’ button. This opens the standard file upload dialog box.

There can be only a single CSS file at any time.

6. Locate the *myStyle.css* file that you saved, and press ‘Open’. The selected file is now displayed next to the ‘Choose File’ button.

7. Press ‘Apply’. This uploads the specified CSS file to the server as *{SREE Home}/portal/format.css*.

For an example of applying these CSS definitions to a dashboard see *Styling a Dashboard with CSS* in *Advanced Dashboard Design* or *Styling a Report with CSS* in *Advanced Report Design*.

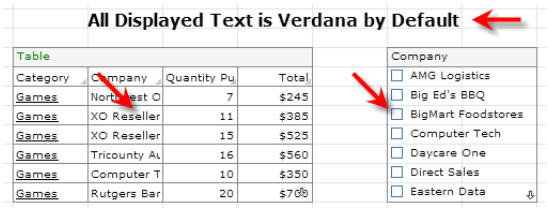
The definitions in this CSS file are explained in the next sections.

Example: Global Format (*)

The global (`*`) definition affects all components that do not have an explicit assignment for a given property (in this case `font-family`). For this example, any component that displays text (e.g., Text, Table, Selection List, etc.) will use the font Verdana unless otherwise specified by an explicit CSS definition for that component or sub-component.

```
/* Global font for all report/dashboard text */
* {
```

```
  font-family: "Verdana";
}
```

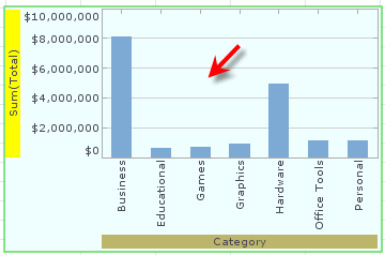


See *CSS Global Default (*)* for further information.

Formatting by Component and Subcomponent Type

The `Chart` definition affects only components of type Chart (in all reports and dashboards).

```
/* Formatting for all Chart-type components */
Chart {
   background:Azure;
}
```



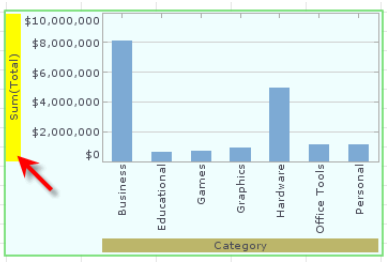
The `TableTitle` definition affects the title bar of all Table-type components (in all reports and dashboards).

```
/* Formatting for all TableTitle sub-components */
TableTitle {
   color: green;
}
```



The `ChartAxisTitle` definition affects the title text of all chart axes (in all reports and dashboards).

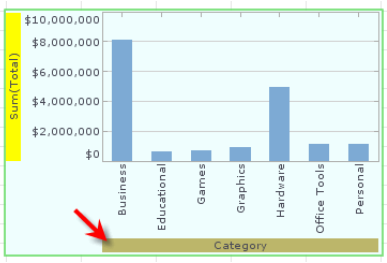
```
/* Formatting for all ChartAxisTitle sub-components */
ChartAxisTitle {
   background: yellow;
}
```



It can be overwritten by more specific axis formatting, such as the X-axis formatting below.

The `ChartAxisTitle[axis=x]` definition affects the title text of all chart X-axes (in all reports and dashboards).

```
/* Formatting for all ChartAxisTitle X-axis sub-components
*/
ChartAxisTitle[axis=x] {
   background:DarkKhaki;
}
```

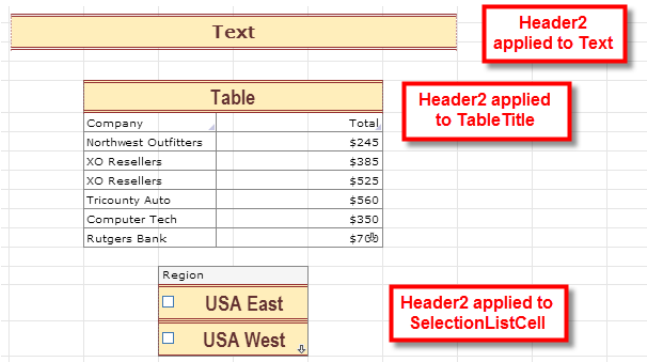


See *CSS Type* for further information.

Formatting by Component Class and ID

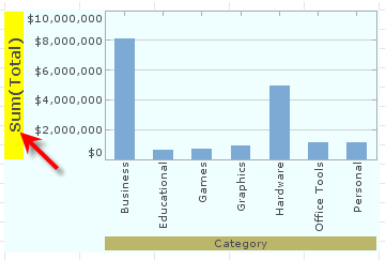
The `Header2` class definition affects only components that have CSS Class “Header2” assigned to them.

```
/* Formatting for all components having CSS Class "Header2"
*/
.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;
}
```



The `AxisText` class definition affects all components that have CSS Class "AxisText" assigned to them.

```
/* Formatting for all components having CSS Class "AxisText"
*/
.AxisText {
   font-size: 14pt;
   font-weight: bold;
}
```



The `Table#SalesTable` definition affects only components of type Table that have CSS ID "SalesTable" assigned to them.

```
/* Formatting for Table-type components having CSS ID
"SalesTable" */
Table#SalesTable {
   background-color: #FFEEBB;
}
```



The `.YellowRows TableStyle[region=Body][pattern=OddRow]` definition affects only components of type Table that have CSS Class "YellowRows" assigned to them. This styles the odd rows within the body of the table.

```
/* Formatting for odd rows of tables having CSS Class
"YellowRows" */
```

```
.YellowRows TableStyle[region=Body][pattern=OddRow] {
   background-color: yellow;
}
```

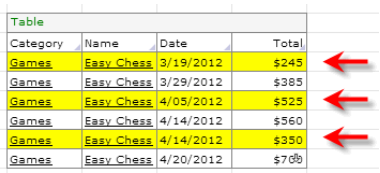


Table			
Category	Name	Date	Total
Games	Easy Chess	3/19/2012	$245
Games	Easy Chess	3/29/2012	$385
Games	Easy Chess	4/05/2012	$525
Games	Easy Chess	4/14/2012	$560
Games	Easy Chess	4/14/2012	$350
Games	Easy Chess	4/20/2012	$700

See *CSS TableStyle* for information about TableStyles.

For information on setting a component's CSS Class and ID, see *Styling a Dashboard with CSS* in *Advanced Dashboard Design* or *Styling a Report with CSS* in *Advanced Report Design*.

7.8.6 Creating Portal CSS Styles

You can customize elements of the User Portal using CSS styles. To do this, 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.

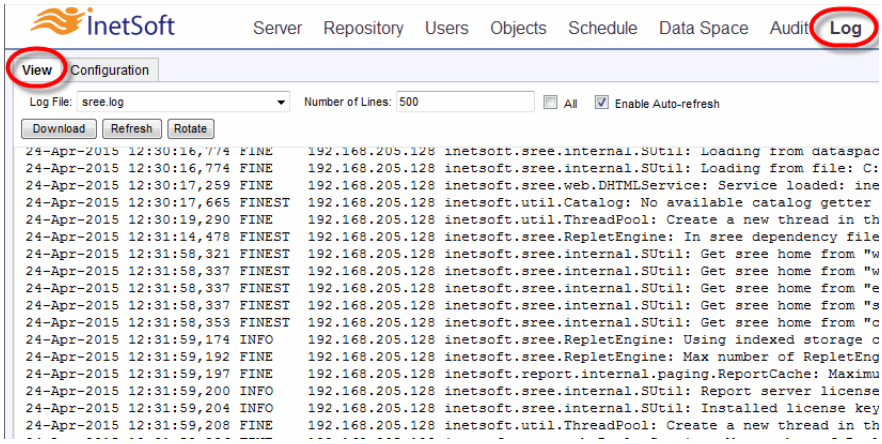
8 Logging

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:

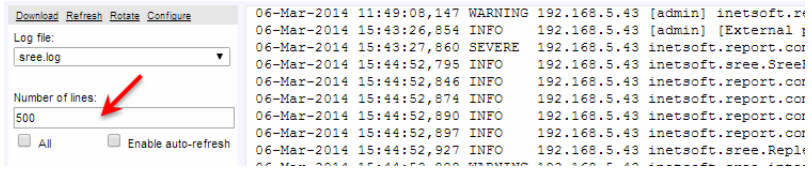
1. Select the main **Log** tab in Enterprise Manager, and select the **View** tab.



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.

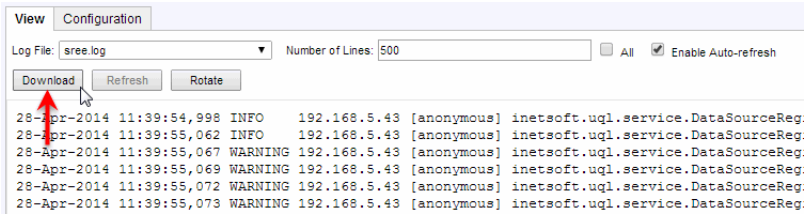
3. Select the 'All' option to display the entire contents of the log file (maximum 3000 lines by default), or enter the 'Number of Lines' from the end of the file to display.



`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.

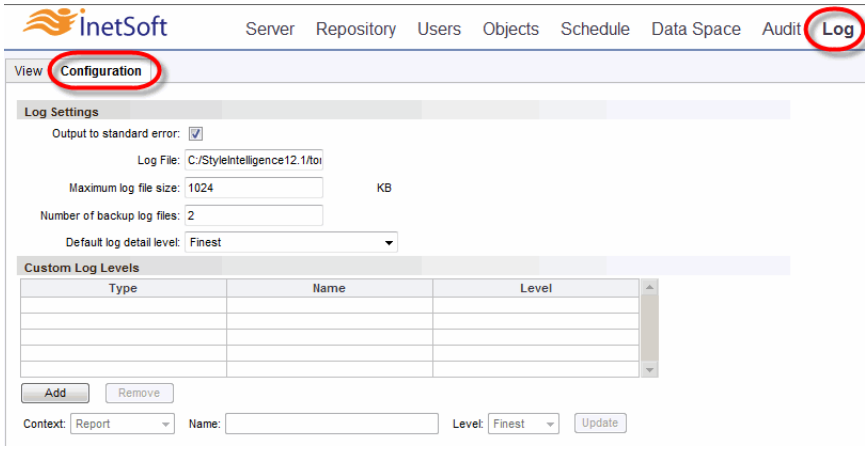
To configure the log settings, select the **Configure** tab. See *Configuring Logging* for more information.

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.



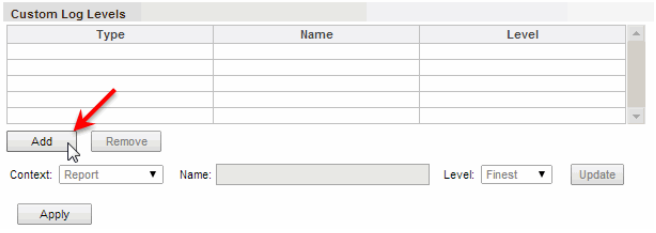
The following properties are available under the ‘Log Settings’ panel.

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.

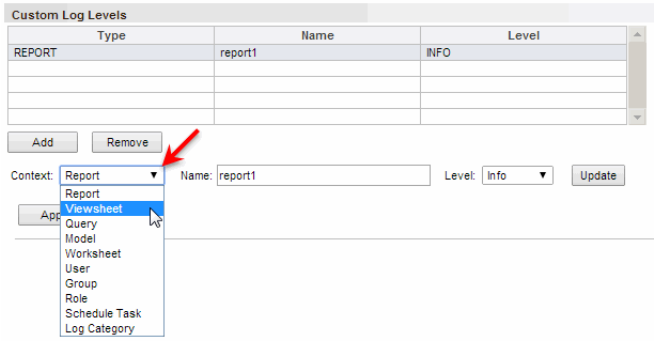
You can assign custom log levels to particular assets. For example, you can set a ‘Warning’ log level for the ‘Default Log Detail Level’ property, but still set a ‘Fine’ log level for a particular report.

To set a custom log level, follow the steps below:

1. Press the ‘Add’ button below the table. This adds a new log level.

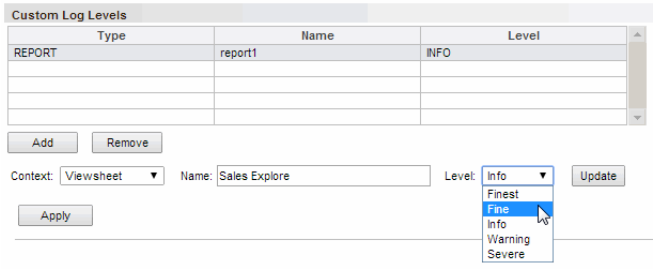


2. In the ‘Context’ menu, select the type of asset (e.g., report, Viewsheet, query, etc.) to which you want to assign a distinct log level.



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: Administration 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.

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.

Report alignment settings do not apply correctly in browser or in generated PDF.

Set property `font.metrics.source=Truetype` in *sree.properties*.

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.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.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.
cpu.affinity.mask	Controls which CPU cores the server process uses for execution. For example, cpu.affinity.mask=0,2,4 restricts the server to use only CPU 0, CPU 2, and CPU 4.

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 or *asset.dat.d* directory. 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 or *asset.dat.d* directory. 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 *Advanced 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*.
asset.max.idle	Sets the duration (in milliseconds) that a Viewsheet remains active in the Portal when no user interactions are detected. Default is 2 hours (7,200,000 milliseconds). This setting does not apply to Viewsheets opened in Visual Composer, which remain active indefinitely or until user session timeout. Viewsheets open in the Portal are automatically closed after three minutes if the web browser is closed.
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`).

flash.wmode	Controls z-index assignment of HTML layers for dashboards within IFrames. It can be assigned one of the following values: `window`, `opaque`, `transparent`. The `window` setting allows the browser to determine how the IFrame and dashboard are displayed on the page. The `opaque` setting presents the IFrame and dashboard with opaque background in their designated layer. The `transparent` setting presents the IFrame and dashboard with transparent background in their designated layer.
help.url	URL for the documentation. Default is “http://community.inetsoft.com/docs/{version}/userhelp”.
force.html5	When set to true, forces all Viewsheets opened in the User Portal to display using HTML5 rather than Flash. The default is false.

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.
ftp.knownhosts.path	Location of the known hosts file for SFTP connections. Default is *.ssh/known_hosts* in user’s home directory.

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.isHeaderFooterInPageSetup	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.
map.default.color	Specifies the default color for map shapes that contain data, and applies when no static color is selected in the 'Visual' panel. Color can be specified as a hex string (e.g., `#008000` or `008000`) or as a case-insensitive CSS color name (e.g., `LightCyan`).
map.empty.color	Specifies the default color for map shapes that do not contain data. Color can be specified as a hex string (e.g., `#008000` or `008000`) or as a case-insensitive CSS color name (e.g., `LightCyan`).
map.border.color	Specifies the border color between geographical regions on a map chart. Color can be specified as a hex string (e.g., `#008000` or `008000`) or as a case-insensitive CSS color name (e.g., `LightCyan`).

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.
sree.collator	Specifies a custom collator class for sorting. When set to `inetsoft.util.Collator_DEACCENT`, this removes all diacritical marks from letters prior to sorting strings (e.g., “ë” is sorted as “e”).
rmi.localhost.ip	Set to `true` to use localhost rather than the specific IP address for connection to RMI servers (Scheduling and Monitoring). Default is `false`.

APPENDIX C: Deploying the Application as a WAR File

The default Style Intelligence installation includes a Tomcat 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

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.

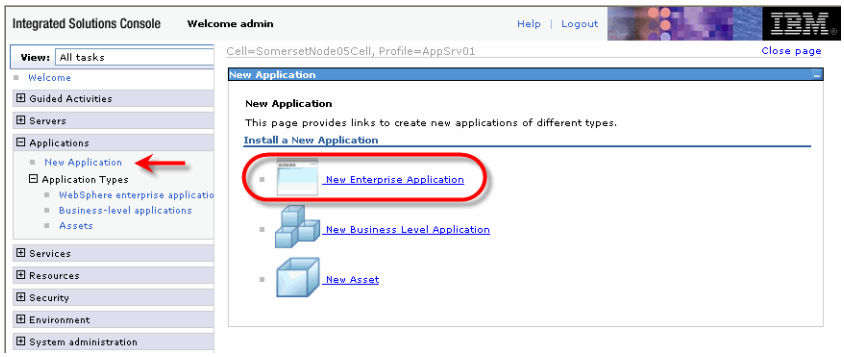
1. Install WebSphere by following the vendor's instructions.

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.

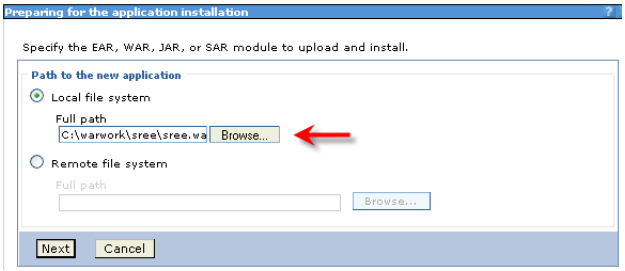
3. Run the WebSphere Administrative Console (default, `http://localhost:9090/admin`) and login.

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’.

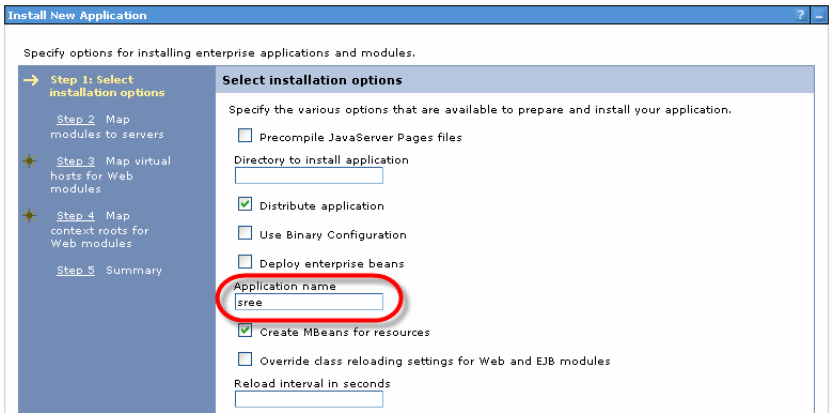


6. In the ‘Path to New Application’ panel, select the ‘Local File System’ option, and type the location of the WAR file in the ‘Full Path’ field (*C:\warwork\sree\sree.war* in the example below), or use the ‘Browse’ feature to locate the file. Click ‘Next’.



7. Under ‘How do you want to install the application?’, select ‘Fast Path’. Click ‘Next’.

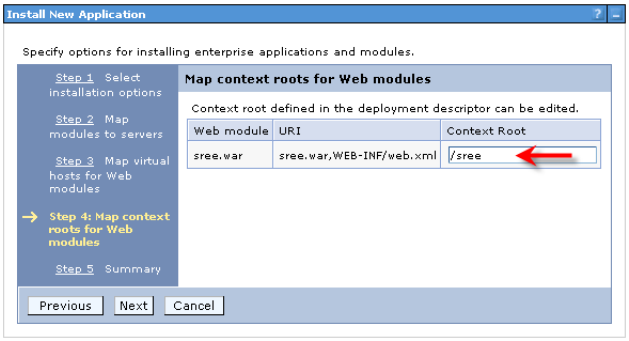
8. On the ‘Select installation options’ page, set the ‘Application name’ to match the WAR file name. (In the example, the ‘Application name’ is ‘`sree`’ because the WAR file is ‘`sree.war`’). Click ‘Next’.



9. On the 'Map modules to servers' page, select the '`sree.war`' module. Click 'Next'.

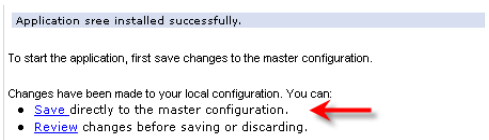
10. On the 'Map virtual hosts for Web modules' page, select the '`sree.war`' module. Click 'Next'.

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'.



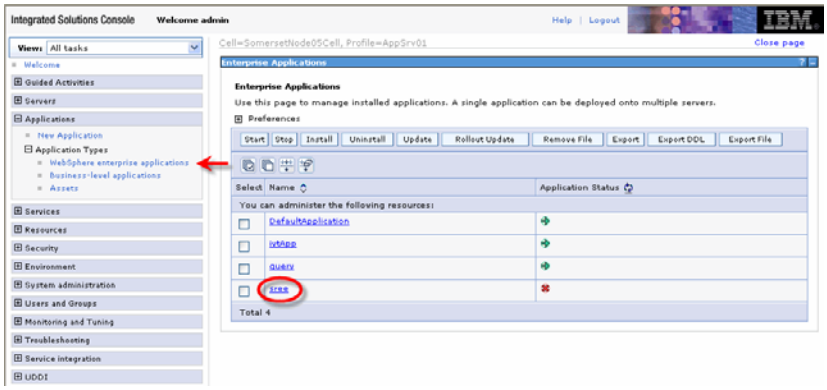
12. On the 'Summary' page, click 'Finish'. Wait until the application installs successfully.

13. Press 'Save directly to the master configuration'.

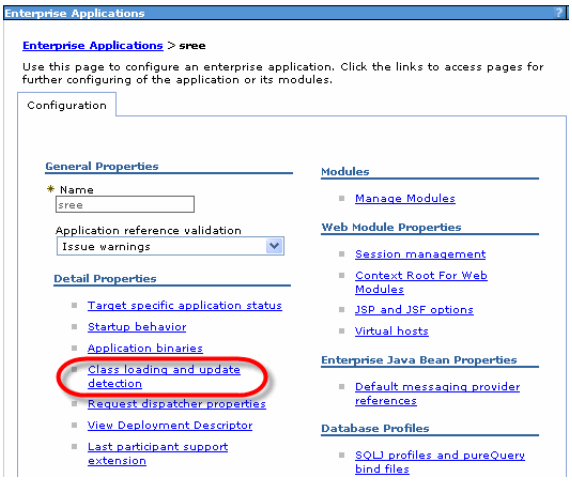


14. When the master repository is updated, expand the 'Application Types' node under 'Applications' (in the left panel), and click 'Websphere enterprise applications' link.

15. On the ‘Enterprise Applications’ page, click the link corresponding to the application that you just created (`sree`).



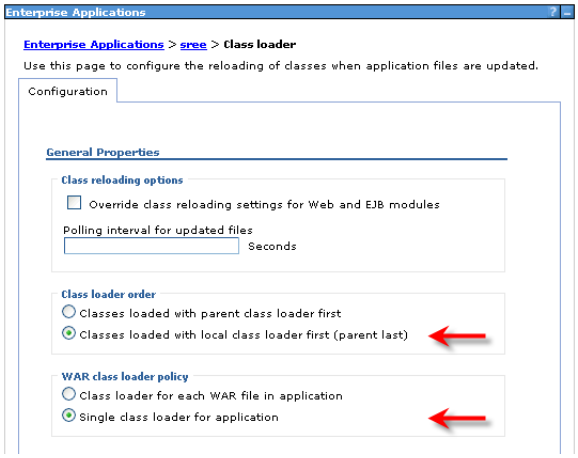
16. In the ‘Configuration’ panel, click the ‘Class loading and update detection’ link.



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
```

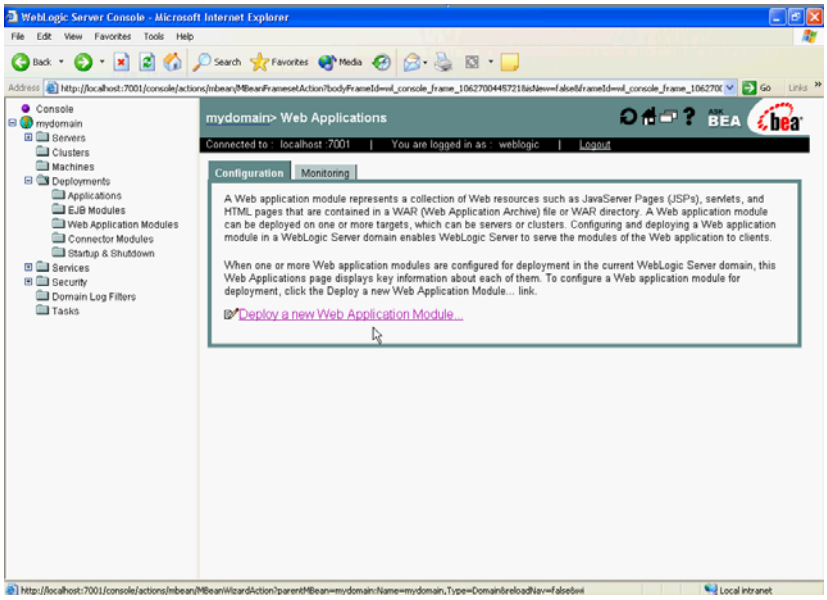
C.3 WebLogic 7.0 and Higher

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*.)

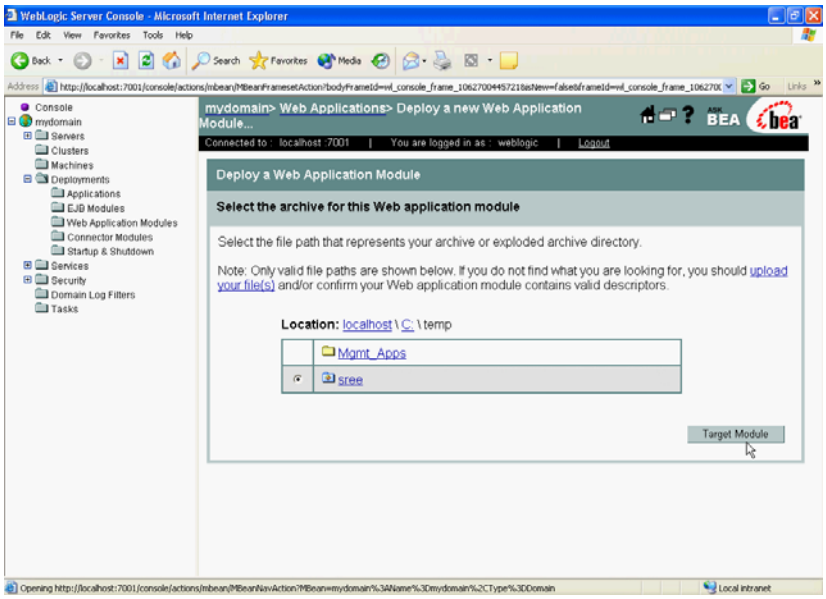
1. Start the default server and then the administrative console.

2. Expand the tree in the left pane and expand ‘Deployments’.

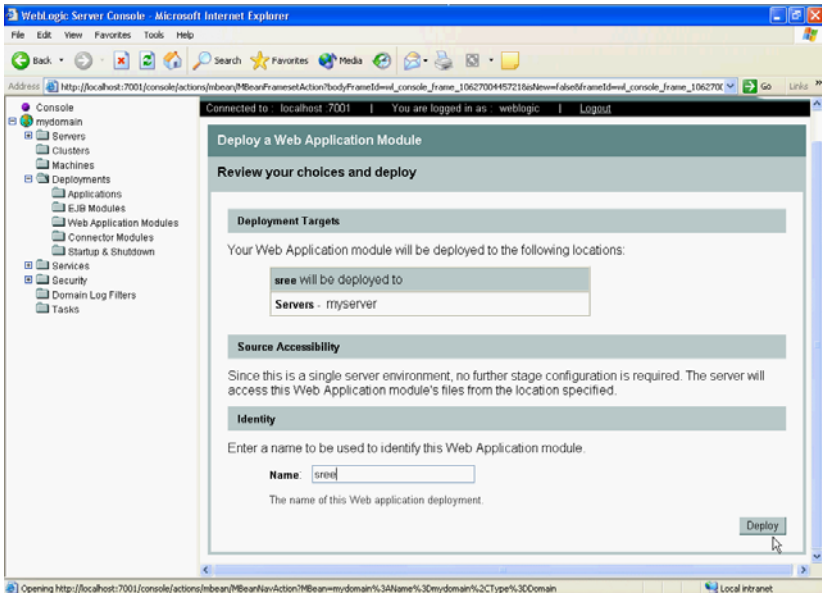
3. Click on ‘Web Application Modules’ and in the right pane click on ‘Deploy a new Web Application Module’.



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.

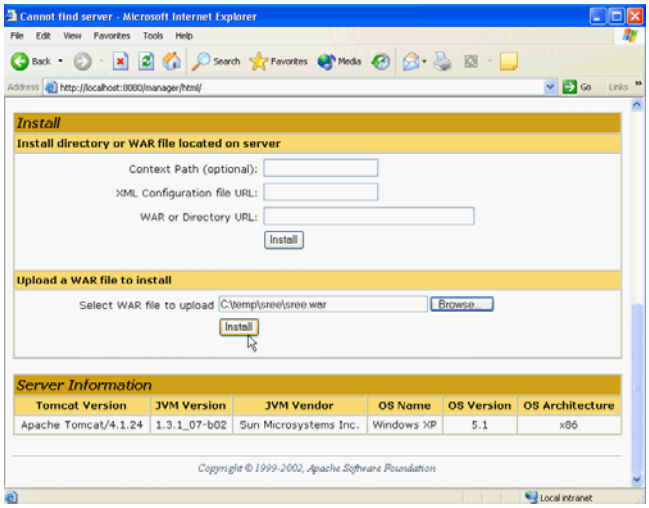
To deploy the Style Intelligence application to Tomcat, you can simply copy the entire *sree* directory from the installed version (without creating a WAR file) into the Tomcat server. Make sure that any paths specified in the *sree.proprties* file are correct for the new installation.

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">
   </dsm>
   ```

   This dsm task should modify the data source and use a WAR or JAR task to create a webapp. If `restore=true`, the data source is 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 now stored in the *stylereport.srl.d* directory, and Viewsheets, Worksheets, queries, and data sources are stored in the *asset.dat.d* directory. If these assets are stored in the single files *stylereport.srl* and *asset.dat* (as was the case in earlier versions), you can explode these files into the corresponding directory representation. The following sections explain how to do this.

E.1 Exploding asset.dat into Individual Components

You can explode the *asset.dat* 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

You can explode the *stylereport.srl* 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.

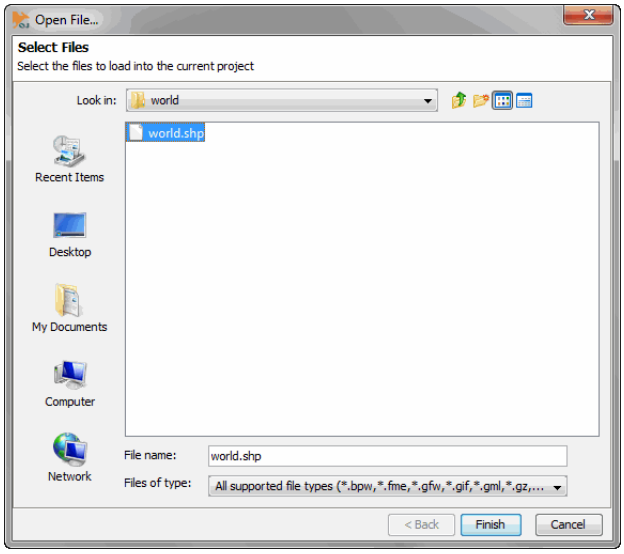
F.2 Exporting a Shape File as a Style Intelligence Map

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.

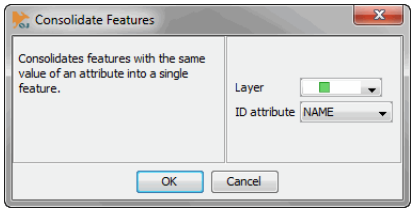
3. Choose ‘Open File’ from the OpenJUMP ‘File’ menu. Select the shapefile that you wish to export, and press ‘Finish’. (In this example, *world.shp* is a shapefile containing the political boundaries of all countries in the world.)



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:

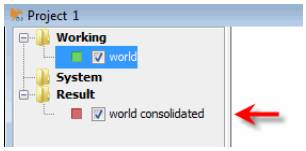
   a. Select ‘Consolidate Features’ from the ‘InetSoft’ menu. This opens the ‘Consolidate Features’ dialog box.



   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.



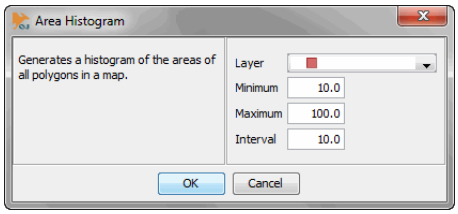
e. (Optional) To conserve memory, delete the original 'world' layer from the 'Working' node of the OpenJUMP project tree. To do this, right click on the 'world' entry, and choose 'Remove Selected Layers'.

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.

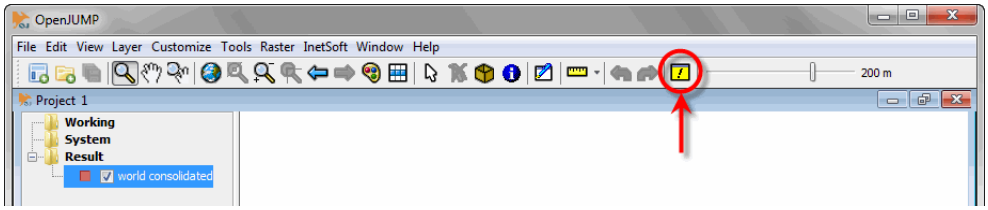
   b. Select the desired 'Layer', and set the 'Minimum', 'Maximum', and 'Interval' properties of the histogram.



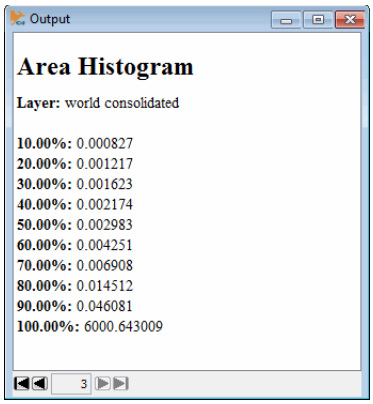
   For example, a minimum of 10%, maximum of 100%, and interval of 10% generates histogram values for the following percentages: 10, 20, 30, 40, 50, 60, 70, 80, 90, 100.

   c. Click 'OK' to generate the histogram.

   d. To view the histogram, press the 'Output Window' button in the OpenJUMP toolbar.

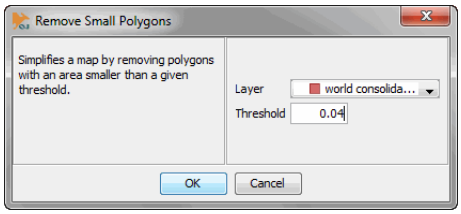


The histogram indicates the percentage of the polygons in the map which would be removed for a given area threshold. Use the histogram to identify the proper area threshold for removing polygons. For example, the histogram below indicates that by removing all polygons smaller than an area threshold of 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.

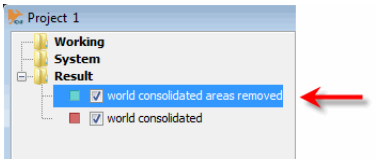
e. Select 'Remove small polygons' from the 'InetSoft' menu. This opens the 'Remove Small Polygons' dialog box.



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.

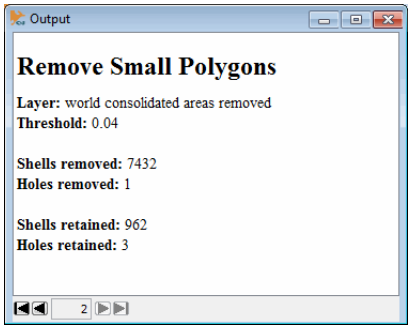
g. Press 'OK' to initiate the polygon removal. This creates a new map layer ("world consolidated areas removed" in the example).



h. (Optional) To remove the original layer ("world consolidated"), right-click the "world consolidated" node, and choose 'Remove Selected Layers' from the context menu.

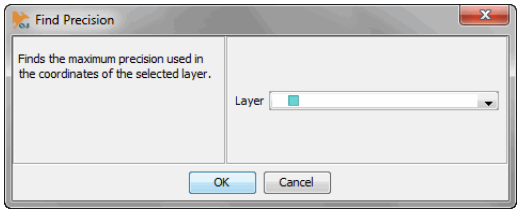
You may need to experiment with different area thresholds to determine which yields the best results.

i. To view the report for the polygon removal operation, press the 'Output Window' button in the OpenJUMP toolbar.



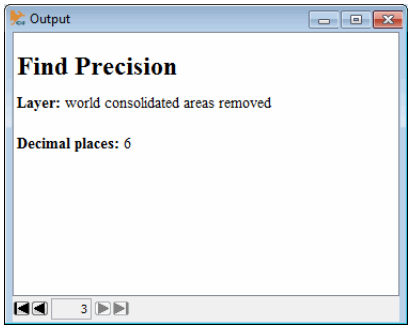
10. To help reduce the memory required for the map, determine the maximum precision to be used. Follow the steps below:

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.)

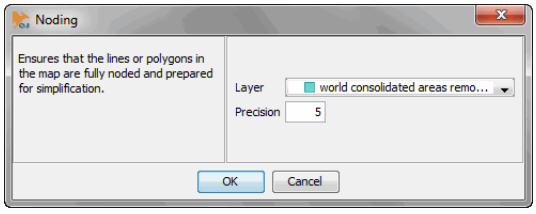


If the precision is unnecessarily high, you can reduce it in the next step. (A precision of six decimal places is more than adequate for most cases.)

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.

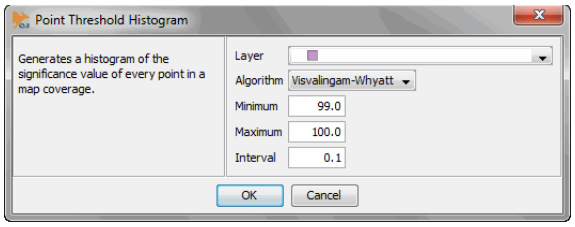
   d. (Optional) To remove the original layer ("world consolidated areas removed"), right-click the "world consolidated areas removed" node, and choose 'Remove Selected Layers' from the context menu.

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.

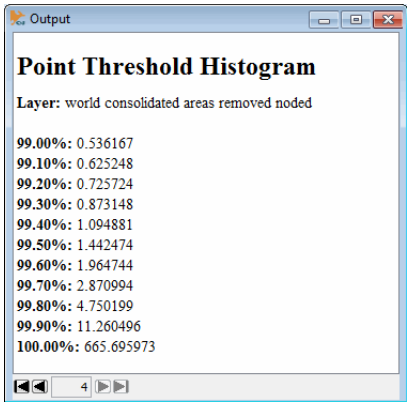
For example, a minimum of 99%, maximum of 100%, and interval of 0.1% generates histogram values for the following percentages: 99.0, 99.1, 99.2, 99.3, 99.4, 99.5, 99.6, 99.7, 99.8, 99.9, 100.



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.

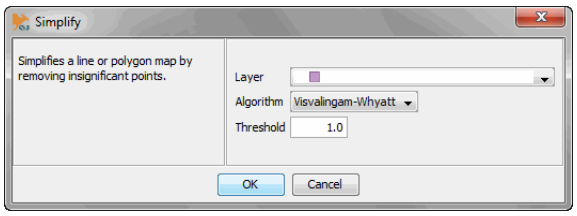
The histogram indicates the percentage of boundary points which would be removed for a given point "significance" threshold. Use the histogram to identify the proper point significance threshold for border simplification. For example, the histogram above indicates that by removing all points with significance 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.

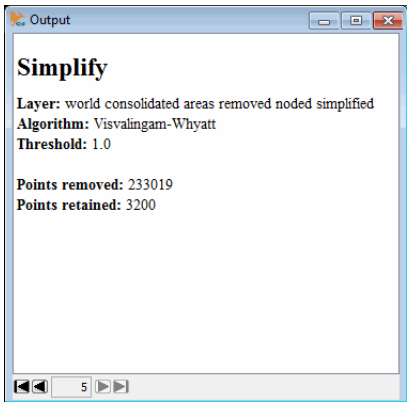
   c. Enter a ‘Threshold’ value. Boundary points with significance smaller than the threshold will be removed from the map. (In this case, a value of 1.0 will result in the removal of approximately 99% of the map’s points.)

   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.

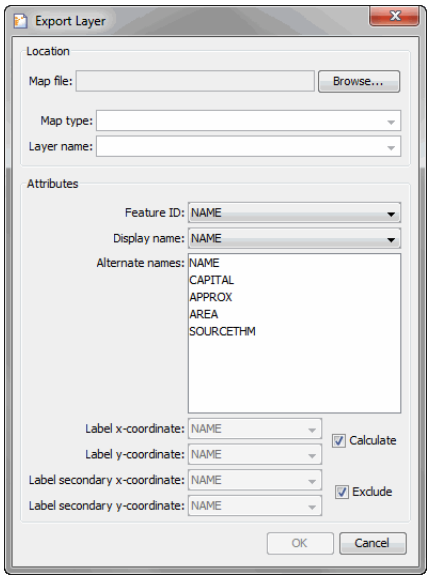
You may need to experiment with different point thresholds to determine which yields the best results.

   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”.

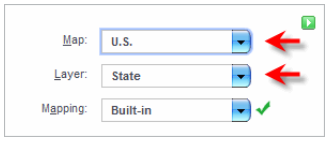
Select an existing layer to overwrite that layer.

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.

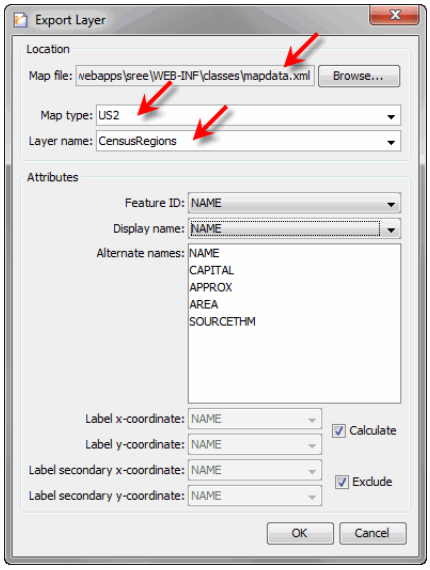
2. Proceed through the steps described in *Exporting a Shape File as a Style Intelligence Map* to process the shapefile appropriately.

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.

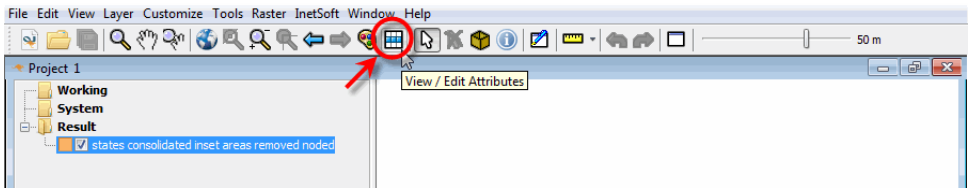
F.3.2 Adding a Layer Based on Merged Regions

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.

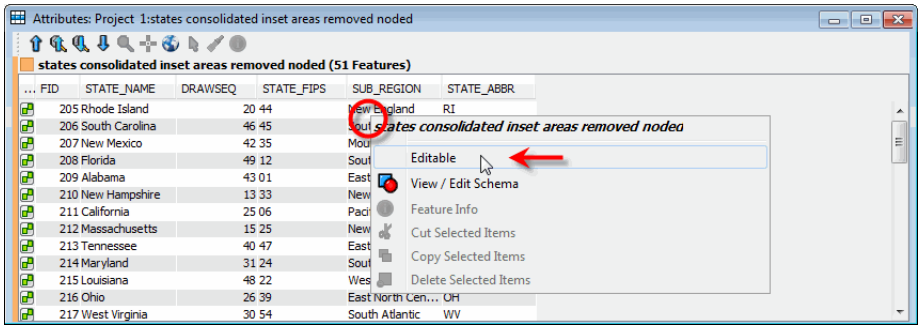
2. Perform any necessary simplification processing as described in *Exporting a Shape File as a Style Intelligence Map*.

3. Press the 'View/Edit Attributes' button in the OpenJUMP toolbar.



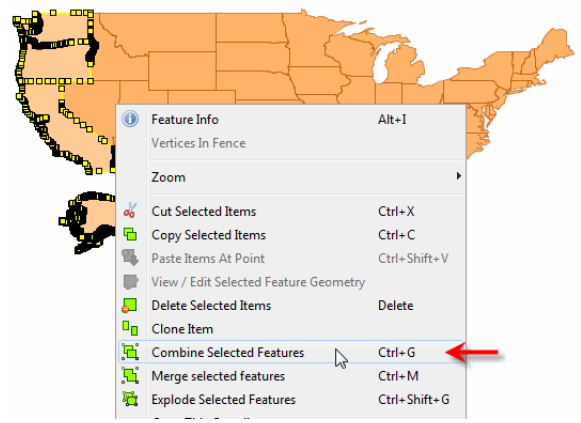
This displays the 'Attributes' table for the map.

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'.



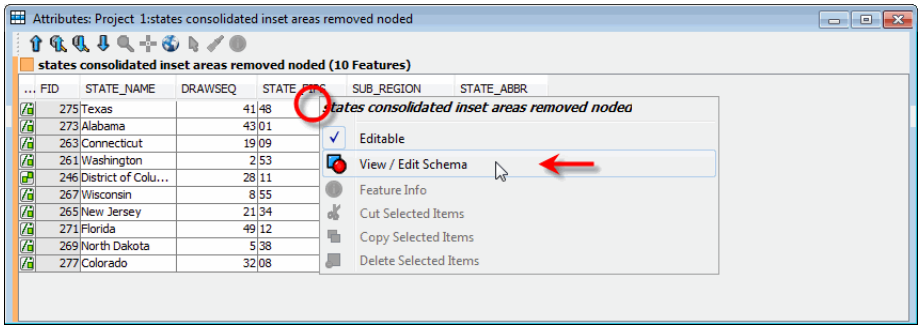
The selected states are now considered a single unit.

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:

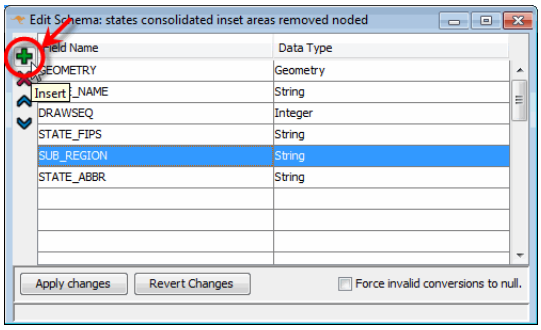
   a. Press the 'View/Edit Attributes' button in the OpenJUMP toolbar to display the 'Attributes' table.

   b. Right-click on the table, and select 'View/Edit Schema'.

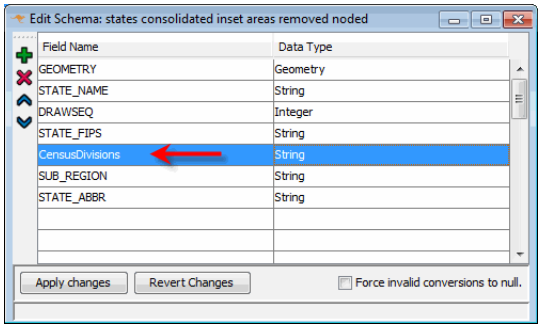


This opens the 'Edit Schema' dialog box.

c. Click in the table, and then press the 'Insert' (+) button to create a new column.

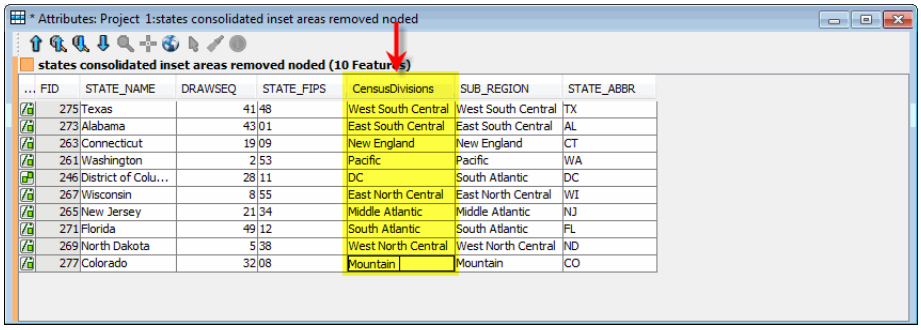


d. Enter the name and type of the new column (e.g., "CensusDivisions").



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.

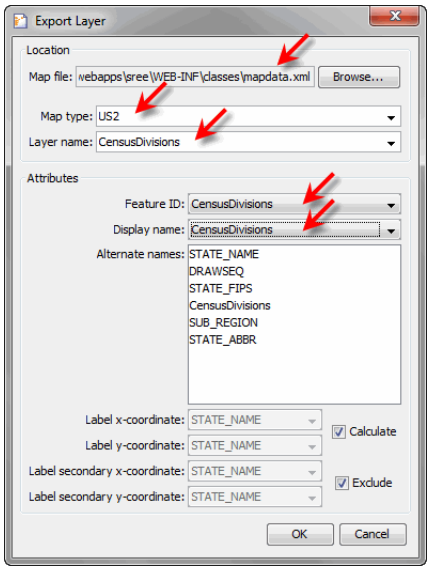
9. With the new column defined, proceed with the instructions in *Exporting a Shape File as a Style Intelligence Map*. When you select 'Export layer' from the 'InetSoft' menu, make the following settings:

   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*.

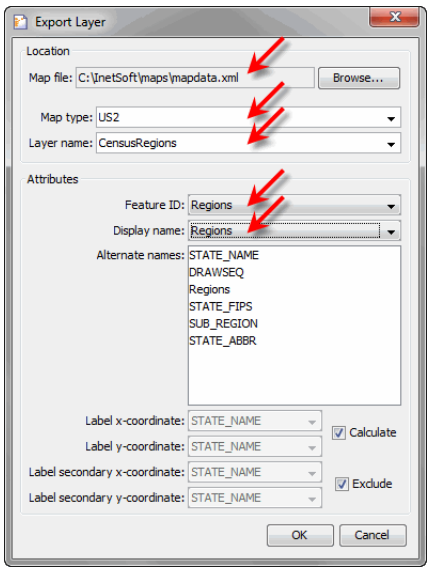
3. Select 'Export layer' from the 'InetSoft' menu, make the following settings:

   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.

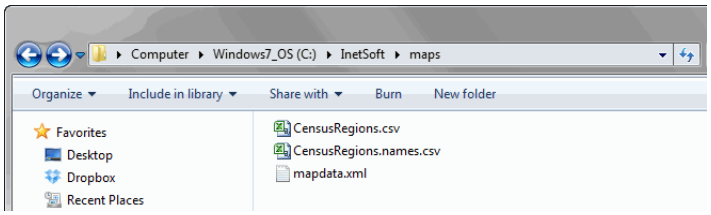
   b. For the 'Map type', enter a map name (e.g., 'US2').

   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:

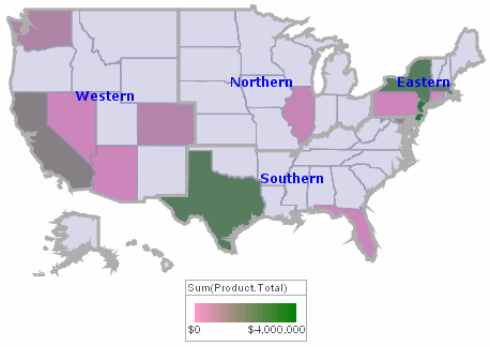
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<mapdata>
  <style name="label">
    <font font="Verdana-BOLD-12" color="#0000FF"/>
    <line width="3" dash="1"/>
    <fill color="#0000FF" alpha=".1"/>
  </style>
```

```
  <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.



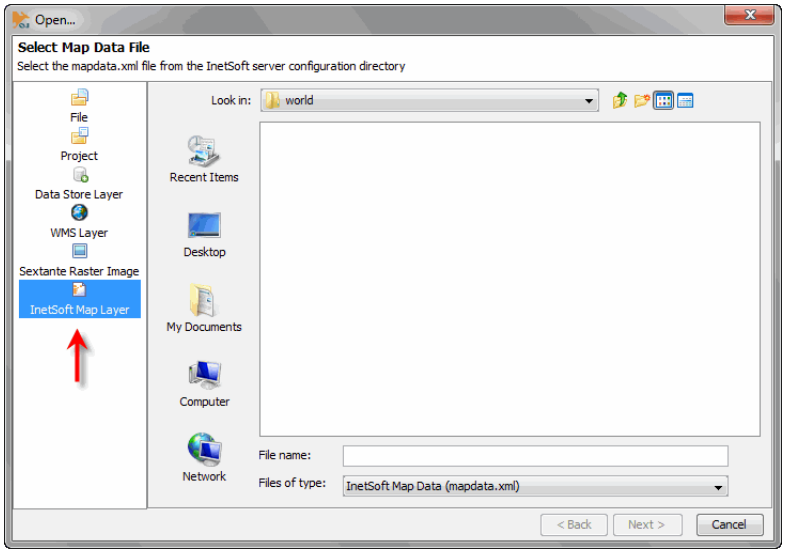
F.5 Editing a Custom Map

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*.

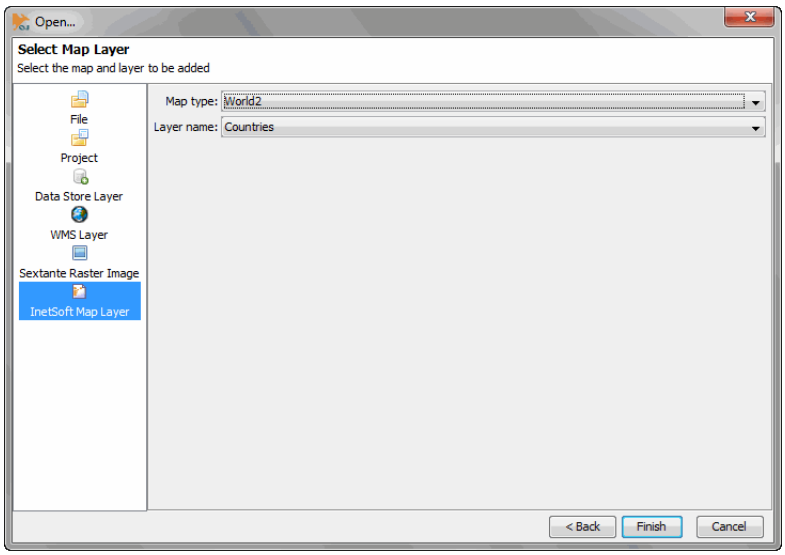
2. Launch OpenJUMP, and select ‘Open’ from the ‘File’ menu.

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.

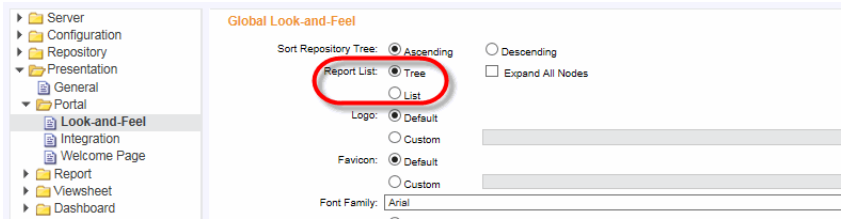
7. Edit the map as desired, and export the modified map. Follow the procedure described in *Exporting a Shape File as a Style Intelligence Map*.

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 Interface with an IFrame*
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 Interface with an IFrame

To embed a Style Intelligence module into your web application, 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.

See Also
Appendix C: *Servlet Repository Parameters*, for useful "op" codes.

1.1.1 Embedding the 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 **Repository** 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>
```

This parameter addresses certain scaling problems that may occur when the Viewsheet is embedded in a *.hta* file.

To load the Viewsheet in Visual Composer, see *Embedding the Visual Composer*. To control z-index assignment for the IFrame, use the

`flash.wmode` property. See *Web Properties and Safeguards* for further information.

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*.

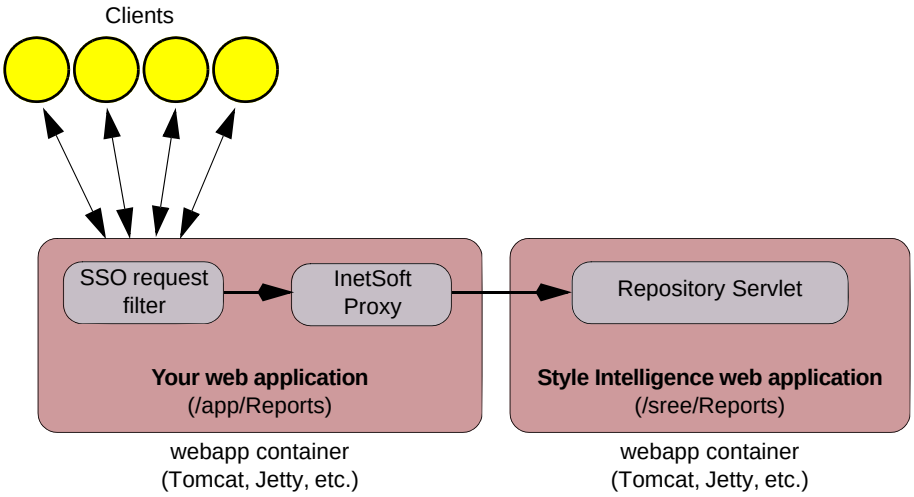
Request Filter mapped to the Style Intelligence Proxy Servlet

```
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*.

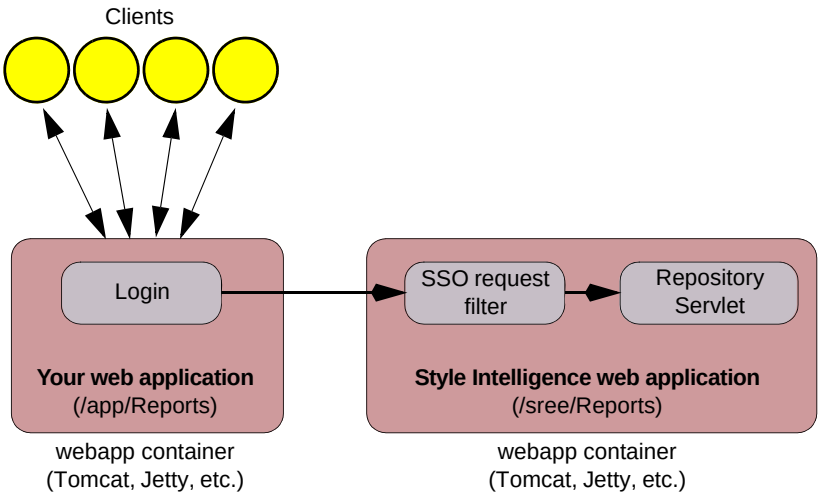
Next Steps: Create a request filter with mapping to the Proxy Servlet. See *Request Filter Example*.

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 {
    BufferedRequestWrapper bufferedRequest;
    if(request instanceof HttpServletRequest) {
      HttpServletRequest hrequest = (HttpServletRequest)
request;
      bufferedRequest = new
BufferedRequestWrapper(hrequest);
      HttpSession session = hrequest.getSession();
      SRPrincipal prin =  (SRPrincipal)session.getAttribute
   (RepletRepository.PRINCIPAL_COOKIE);
```

This simple credential-passing mechanism is intended for demonstration only. DO NOT USE THIS APPROACH IN YOUR PRODUCTION ENVIRONEMENT.

```
      // Extract user info based on your environment.
      // For demonstration purposes, we assume here that
      // it is passed as a URL parameter.

      String user = request.getParameter("UID");
      if(prin == null && user != null) {
        System.out.println("setting new user to:" + user +
"*********");
        prin = new SRPrincipal(user);
        session.setAttribute(RepletRepository.PRINCIPAL_COO
KIE, prin);
      }
      chain.doFilter(bufferedRequest, response);
    }
    else {
```

```
      chain.doFilter(request, response);
    }
  }

  public class BufferedRequestWrapper extends
  HttpServletRequestWrapper {
    ByteArrayInputStream bais;
    ByteArrayOutputStream baos;
    BufferedServletInputStream bsis;
    byte [] buffer;

    public BufferedRequestWrapper(HttpServletRequest req)
throws IOException {
      super(req);
      // Read InputStream and store its content in a buffer.
      InputStream is = req.getInputStream();
      baos = new ByteArrayOutputStream();
      byte buf[] = new byte[1024];
      int letti;
      while ((letti=is.read(buf))>0)
baos.write(buf,0,letti);
      buffer = baos.toByteArray();
    }

    public ServletInputStream getInputStream() {
      try {
        // Generate a new InputStream by stored buffer
        bais = new ByteArrayInputStream(buffer);
        // Instantiate a subclass of ServletInputStream
        // (Only ServletInputStream or subclasses are
accepted!)
        bsis = new BufferedServletInputStream(bais);
      }
      catch (Exception ex) {
        ex.printStackTrace();
      }
      finally {
        return bsis;
      }
    }
  }

  public class BufferedServletInputStream extends
ServletInputStream {
    ByteArrayInputStream bais;
    public BufferedServletInputStream(ByteArrayInputStream
bais) {
      this.bais = bais;
    }

    public int available() {
      return bais.available();
    }

    public int read() {
      return bais.read();
    }

    public int read(byte[] buf,int off,int len) {
      return bais.read(buf,off,len);
    }
  }
}
```

This example assumes that user information is passed via a request parameter called "UID". 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 per-

missions. 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 **Repository** 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

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

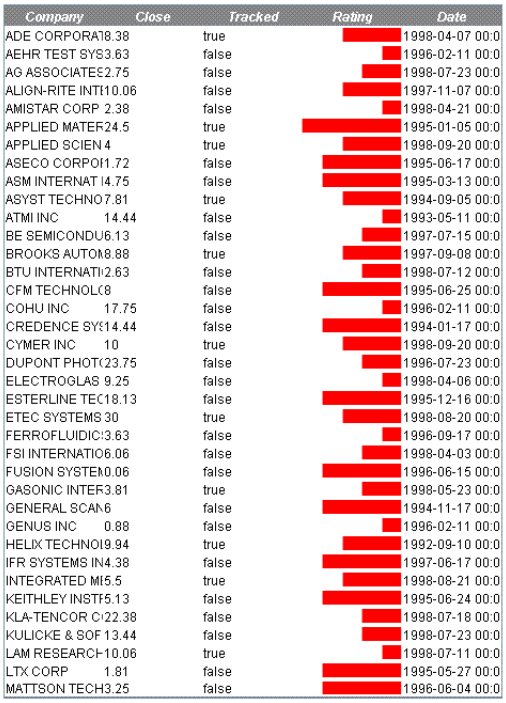
assigned to a table column to avoid using it on the wrong types of objects, if there is more than one type in the column.

```
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*, *asset.dat*, 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*, *asset.dat*, 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 Dashboard 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 *Advanced 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.

Advanced 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 setLocale()

The `setLocale()` property allows you to set the user locale. The syntax is as follows:

```
principal.setLocale(new
Locale("language_code","country_code","variant_name"));
```

The specified language code, country code, and variant name are used to load the localization mapping files with corresponding names:

```
SreeBundle_{language_code}_{country_code}_{variant_name}.pr
operties
srinter_{language_code}_{country_code}_{variant_name}.prope
rties
```

For more information about the structure and role of the *srinter* and *SreeBundle* files, see *Localization/Internationalization* in *Advanced Server Administration*.

The variant name allows you to specify a sub-locale or other arbitrary identifier. It is optional and available only within an SSO filter. (See

Configuring Single Sign-On (SSO) for more information on SSO implementation.)

The following example code sets the user locale to US English (without variant) and loads localization settings from mapping files *SreeBundle_en_US.properties* and *srinter_en_US.properties*.

```
HttpServletRequest hrequest = (HttpServletRequest) request;
HttpSession session = hrequest.getSession();
SRPrincipal principal =
(SRPrincipal)session.getAttribute(RepletRepository.PRINCIPA
L_COOKIE);
principal.setLocale("en","US");
```

The base file is not required.

When a variant is used to set the locale, localization mappings are loaded from both the optional base file (e.g., *srinter_en_US.properties*) as well as from the variant file (e.g., *srinter_en_US_NJ.properties*), with settings from the variant file overwriting any conflicting settings from the base file.

The following example code adds a variant to load the localization settings from variant mapping files *SreeBundle_en_US_NJ.properties* and *srinter_en_US_NJ.properties* as well as from base files *SreeBundle_en_US.properties* and *srinter_en_US.properties*.

```
HttpServletRequest hrequest = (HttpServletRequest) request;
HttpSession session = hrequest.getSession();
SRPrincipal principal =
(SRPrincipal)session.getAttribute(RepletRepository.PRINCIPA
L_COOKIE);
principal.setLocale("en","US","NJ");
```

5.2.3 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 *Advanced 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 Interface with an IFrame*.

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 Interface with an IFrame* 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 Interface with an IFrame* 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 Displaying a Viewsheet on InetSoft Mobile App

Use the following syntax to open a Viewsheet using the InetSoft Mobile app on Android or iOS:

```
inetsoftmobile://userid=<user
name>&password=<password>&viewsheet=<Viewsheet
path>&VSPARAM_param1=value1&VSPARAM_param2=value2&VSPARAM_p
aramN=valueN
```

To pass values into a parameter that accepts multiple values (for example, a parameter used in a "one of" condition), add a parameter-value pair corresponding to each value. In the following example, parameter `product` accepts a single value and parameter `employee` accepts multiple values:

```
inetsoftmobile://userid=admin&password=admin&viewsheet=/
Dashboard/Sales Explore&VSPARAM_product='Fast Go
Game'&VSPARAM_employee=Sue&VSPARAM_employee=Robert&VSPARAM_
employee=Eric
```

To open a user-scope Viewsheet, qualify the Viewsheet path with the user name, e.g,

```
inetsoftmobile://viewsheet=admin/MyFolder/MyViewsheet
```

Both form-based single-sign-on (SSO) and session-based single-sign-on are supported. See *Form-Based Single Sign-On* and *Session-Based Single Sign-On* for more details. For session-based SSO, use the `_sessionId_` attribute instead of `userid` and `password`.

```
inetsoftmobile://_sessionId_=<session
ID>&viewsheet=<Viewsheet
path>&VSPARAM_param1=value1&VSPARAM_param2=value2...
```

C.3 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.4 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.5 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.6 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 *Advanced Report Scripting*, for information about accessing parameters.

Parameterizing a Condition in *Advanced Report Design*, for information on parametrizing filters.

C.7 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.8 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.9 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.10 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.11 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.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.

Common Reference

The following appendices discuss general functions that are useful across multiple InetSoft 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 *Advanced 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 *Advanced 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 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.2 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.3 JavaScript Operators

The operators in JavaScript are almost identical to the operators in Java:

JS.3.1 Assignment 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.3.2 Comparison 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.4 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
```

DATE GROUPING LEVELS

Date grouping levels are used for Chart and Crosstab binding.

LEVEL	**DESCRIPTION**
YEAR_DATE_GROUP	Group by year
QUARTER_DATE_GROUP	Group by quarter
MONTH_DATE_GROUP	Group by month
WEEK_DATE_GROUP	Group by week
DAY_DATE_GROUP	Group by day
QUARTER_OF_YEAR_DATE_GROUP	Group by quarter of year
MONTH_OF_YEAR_DATE_GROUP	Group by month of year
WEEK_OF_YEAR_DATE_GROUP	Group by week of year
DAY_OF_MONTH_DATE_GROUP	Group by day of month
DAY_OF_WEEK_DATE_GROUP	Group by day of week
HOUR_OF_DAY_DATE_GROUP	Group by hour of day
HOUR_DATE_GROUP	Group by hour
MINUTE_DATE_GROUP	Group by minute

SECOND_DATE_GROUP	Group by second
NONE_DATE_GROUP	No date grouping

Type

```
integer
```

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 *Advanced Report Scripting*, for more information on setting report alignments.

Alignment Property, in *Advanced Dashboard Scripting*, for more information on setting Viewsheet alignments.

DATA TYPES

The following data type constants available.

TYPE	DESCRIPTION
BOOLEAN	true or false
BYTE	single byte number
CHAR	character
DATE	date with no time component
DOUBLE	double number
ENUM	user-defined string constants
FLOAT	float number

INTEGER	integer number
LONG	long integer number
SHORT	short integer number
STRING	text string
TIME	time
TIME_INSTANT	time and date

Type

`integer`

Example

```
RangeSlider1.rangeType = StyleConstant.DATE;
```

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;
```

SUMMARIZATION FORMULAS

The summarization formulas below are available for crosstab and chart scripting. For charts, they can also be accessed through the *Chart Constants* interface.

NONE_FORMULA	NTHMOSTFREQUENT_FORMULA
AVERAGE_FORMULA	NTHSMALLEST_FORMULA
CONCAT_FORMULA	POPULATIONSTANDARDDEVIATION_FORMULA
CORRELATION_FORMULA	POPULATIONVARIANCE_FORMULA
COUNT_FORMULA	PRODUCT_FORMULA
COVARIANCE_FORMULA	PTHPERCENTILE_FORMULA
DISTINCTCOUNT_FORMULA	STANDARDDEVIATION_FORMULA
MAX_FORMULA	SUMSQ_FORMULA

MEDIAN_FORMULA_FORMULA	SUMWT_FORMULA
MIN_FORMULA	SUM_FORMULA
MODE_FORMULA	VARIANCE_FORMULA
NTHLARGEST_FORMULA	WEIGHTEDAVERAGE_FORMULA

Type

```
integer
```

Example

```
style = StyleConstant.AVERAGE_FORMULA;
```

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;
```

JS.5 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.5.1 Variable 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.5.2 Function 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.5.3 The '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.5.6, *The 'switch' Statement*, to conveniently test a larger set of conditions.

JS.5.4 The '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.5.5 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.5.6 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.5.3, *The 'if' Statement*, to conveniently test a small set of conditions.

JS.5.7 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.6 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.SimpleDate-Format.

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, FORMAT, ROUND)

Formats a number according to the format string.

Parameters

```
number              The number to format
format              A format string following Java
DecimalFormat specification
round               Specifies how the number is to be
rounded (see below)
```

Example

```
formatNumber(3, '#,###.00','ROUND_HALF_UP');
```

Construct the number format string using the characters listed in the table below. The syntax is the same as Java's DecimalFormat.

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

The rounding options specified by the round parameter are the same as for the Java BigDecimal class.

ROUND SETTING	MEANING
ROUND_UP	Round away from zero
ROUND_DOWN	Round toward zero
ROUND_CEILING	Round toward +∞
ROUND_FLOOR	Round toward -∞
ROUND_HALF_UP	Round toward nearest neighbor unless both neighbors are equidistant, in which case round up
ROUND_HALF_DOWN	Round toward nearest neighbor unless both neighbors are equidistant, in which case round down
ROUND_HALF_EVEN	Round toward nearest neighbor unless both neighbors are equidistant, in which case round to even neighbor
ROUND_UNNECESSARY	Apply no rounding

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;
```

TOLIST(LIST [,OPTIONS])

Remove duplicate entries from input array to yield a list of unique elements with optional sorting and grouping.

Parameter

list	array to filter
options	a comma-separated Options String specifying property-value pairs below:

Options String

sort	Return sorted values {asc,desc,false}
sorton	Measure by which to sort, e.g., sum(Total), prior to filtering by maxrows
sorton2	Measure by which to sort, e.g., sum(Total), after filtering by maxrows
remainder	Label for 'Others' group in Top/Bottom-N filtering. Omit option to suppress 'Others'.
maxrows	Limits the number of rows returned.
distinct	Return only distinct values {true,false} The default is true.
date	Return values grouped by date {year/quarter/month/week/day/hour/minute/second/weekday/monthname/weekdayname}
rounddate	Same as 'date', but returns an array of dates

Example

```
Array1 = [2,3,1,2,3,3,2,2,1,0];
Array2 = toList(Array1,'sort=desc');  // returns [3,2,1,0]
```

See *toList* in the *Advanced Report Scripting* for more information about usage in reports. See *Creating a Freehand Table* in *Advanced Dashboard Design* for usage in dashboards.

JS.7 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'])
```

The following pages document some common string functions. Please consult a JavaScript reference for details about other string functions.

LENGTH

The length property of a string returns the number of characters in the string.

Example

```
var strLength = 'Hello'.length; // returns '5'
```

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)

Return a substring of a given string, where `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.

Example

```
var txt = 'Hello'.substring(1,3); // returns “el”
```

CHARAT(INDEX)

Return the character at the specified position in the string, indexed from 0.

Example

```
var txt = 'Hello'.charAt(4); // returns “o”
```

CHARCODEAT(INDEX)

Return the character code of the character at the specified position in the string.

Example

```
var txt = 'Hello'.charCodeAt(4); // returns “111”
```

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.

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.8 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.9 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.1, *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.10 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.11 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.12 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.12.1Date and Time Functions

The CALC object provides the date and time functions listed at right.

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 (February)
```

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

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.12.2Financial 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: The 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: The 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: The 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: The 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: The 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: The 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: The 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 (approximately 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: The 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 yield
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
```

CALC.INTRATE(SETTLEMENT, MATURITY, INVESTMENT, REDEMPTION, [BASIS])

Returns the interest rate for a fully invested security without compounding.

Parameters

settlement	Date object: The 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])

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`), where Macaulay duration is given by *CALC.duration(settlement, maturity, rate, yld, frequency, [basis])*.

Parameters

```
settlement    Date object: The 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 yield
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 paid 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 Effective 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: The 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 yield
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: The 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: The 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 yield

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 (approximately $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: The 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: The 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: The 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.

```
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.12.3Logical 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_false Expression 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.12.4 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 exponent 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 variance of items 11: Population 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.12.5Statistical 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 Boolean 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
```

The result should be comparable to that obtained with *CALC.pearson(array1, array2)*.

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 and non-empty elements in `array`.

Parameters

array Array of numeric or non-numeric values

Example

```
CALC.counta(['asd',null,3,2,,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 value of the exponential distribution for specified value of random variable `x` and rate parameter `lambda`. Value `x` typically represents the time interval between events in a Poisson process.

Parameters

x	Time interval for which to compute probability.
lambda	Rate parameter (number of events per unit time)
cumulative	Boolean: true: computes cumulative distribution function false: computes probability density function

Example

Probability that the interval between successive events is at most 0.2 seconds for a Poisson process with a constant rate of 10 events per second.

```
CALC.expondist(0.2, 10, true);
// returns 0.8646647167633873 (approximately 86.5%)
```

CALC.FISHER(X)

Returns the Fisher z-transformation at numeric value `x`. This transformation produces a function that is normally distributed rather than skewed, and can be used to perform hypothesis testing on the sample correlation coefficient.

Parameters

x	Value to transform

Example

```
CALC.fisher(0.75);
// returns 0.9729550745276566
```

CALC.FISHERINV(Z)

Returns the inverse of the Fisher transformation.

Parameters

z	Z-value for which to compute the inverse transformation.

Example

```
CALC.fisherinv(1);
// returns 0.7615941559557649
```

CALC.FORECAST(X, KNOWN_Y_VALUES, KNOWN_X_VALUES)

Predicts the future value of a function based on past values using linear regression.

Parameters

x	Value at which to compute function
known_y_values	Array of dependent (function) values corresponding to known_x_values
known_x_values	Array of independent values corresponding to known_y_values

Example

For a function with the following (*x*,*y*) pairs, {(20,6),(28,7),(31,9),(38,15),(40,21)}, compute the function value at *x*=30.

```
CALC.forecast(30, [6,7,9,15,21],[20,28,31,38,40]);
// returns 10.607253086419755
```

CALC.FREQUENCY(DATA_ARRAY, BINS_ARRAY)

Returns an array representing the frequencies with which values in `data_array` occur within the ranges specified by `bins_array`. For example, if `bins_array` specifies four values, `[value1, value2, value3, value4]`, this groups the values in `data_array` into the following five ranges:

- `x <= value1`
- `value1 < x <= value2`
- `value2 < x <= value3`
- `value3 < x <= value4`
- `x > value4`

The returned array contains the count of the values in each of these five bins.

Parameters

data_array Array of data to group and count
bins_array Array of thresholds defining the groups

Example

Compute the frequencies of values {79, 85, 78, 85, 50, 81, 95, 88, 97} in the ranges {$x \le 70$, $70 < x \le 79$, $79 < x \le 89$, $x > 89$).

```
var freq =
CALC.frequency([79,85,78,85,50,81,95,88,97],[70,79,89]);
for (var i=0;i<freq.length;i++) {
  log('Count in bin ' + i + ': ' + freq[i])
}

// Returns:
// Count in bin 0: 1
// Count in bin 1: 2
// Count in bin 2: 4
// Count in bin 3: 2
```

CALC.GEOMEAN(ARRAY)

Returns the geometric mean of values in `array`.

Parameters

array Array of positive values for which to compute geometric mean

Example

```
CALC.geomean([4,5,8,7,11,4,3]);
// returns 5.476986969656962
```

CALC.HARMEAN(ARRAY)

Returns the harmonic mean of values in `array`. The harmonic mean is the reciprocal of the arithmetic mean of reciprocals.

Parameters

array	Array of positive values for which to compute harmonic mean

Example

```
CALC.harmean([4,5,8,7,11,4,3]);
// returns 5.028375962061728
```

CALC.HYPGEOMDIST(SAMPLE_S, NUMBER_SAMPLE, POPULATION_S, NUMBER_POPULATION)

Returns the value of the hypergeometric distribution, the probability of `sample_s` successes in a sample of size `number_sample` (drawn without replacement), where the population of size `number_population` has `population_s` successes.

Parameters

sample_s	Number of successes in sample
number_sample	Sample size
population_s	Number of successes in population
number_population	Population size

Example

Probability of 1 success in a sample of size 4 (drawn without replacement), where the population of size 20 has 8 successes:

```
CALC.hypgeomdist(1, 4, 8, 20);
// returns 0.3632610939112487 (approximately 36.3%)
```

CALC.INTERCEPT(ARRAY_Y, ARRAY_X)

Calculates the point at which the best-fit regression line for the values in `array_x` and `array_y` intersects the *y*-axis.

Parameters

array_y	Array of y values
array_x	Array of x values

Example

```
CALC.intercept([8,11,14,17],[1,2,3,4]);
// returns 5.0
```

CALC.KURT(ARRAY)

Returns the sample excess kurtosis of a data set, which indicates the peakedness or flatness of the data relative to the normal distribution. Positive kurtosis indicates relatively peaked, while negative kurtosis indicates relatively flat.

Parameters

array	Array of values

Example

```
CALC.kurt([4,5,8,7,8,4,5,3,7]);
// returns -1.6851311953352766 (relatively flat)
```

CALC.LARGE(ARRAY, K)

Returns the *k*th largest value in the `array`. Duplicate values are included in the ranking. For example, ranking set {4,5,8,7,8,4,5,3,7} from greatest to least yields {8,8,7,7,5,5,4,4,3}.

Parameters

array	Array of values
k	Rank of the value to obtain (k=1 is largest)

Example

The third-largest value in set {4,5,8,7,8,4,5,3,7}:

```
CALC.large([4,5,8,7,8,4,5,3,7],3);
// returns 7.0
```

CALC.MAX(ARRAY)

Returns the largest value in `array`.

Parameters

array	Array of numerical values

Example

Largest value in set {4,5,8,7,8,4,5,3,7}:

```
CALC.max([4,5,8,7,8,4,5,3,7]);
// returns 8.0
```

CALC.MAXA(ARRAY)

Returns the largest value in `array`, where Booleans are considered 1 if `true` and 0 if `false`, and string values are considered 0.

Parameters

array	Array of values (numeric, Boolean, or string)

Example

Largest value in set {4,5,8,8,4,true,3,yesterday}:

```
CALC.maxa([4,5,8,8,4,true,3,'yesterday']);
// returns 8.0
```

CALC.MEDIAN(ARRAY)

Returns the median of the elements in `array`.

Parameters

`array` `Array of numeric values`

Example

```
CALC.median([1,2,3,4,5,6,7]);
// returns 4.0
```

CALC.MIN(ARRAY)

Returns the smallest value in `array`.

Parameters

`array` `Array of numerical values`

Example

Smallest value in set {4,5,8,7,8,4,5,3,7}:

```
CALC.min([4,5,8,7,8,4,5,3,7]);
// returns 3.0
```

CALC.MINA(ARRAY)

Returns the smallest value in `array`, where Booleans are considered 1 if `true` and 0 if `false`, and string values are considered 0.

Parameters

`array` `Array of values (numeric, Boolean, or string)`

Example

Smallest value in set {4,5,8,8,4,true,3,yesterday}:

```
CALC.mina([4,5,8,8,4,true,3,'yesterday']);
// returns 0.0 (value of string 'yesterday')
```

CALC.MODE(ARRAY)

Returns the most frequently occurring value in `array`. If there are several elements that occur with the maximum frequency, the smallest of these is returned.

Parameters

`array` `Array of values`

Example

```
CALC.mode([1,1,2,3,4,5,2,6,7]);
// 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 in a series of independent Bernoulli trials, when the constant probability of a success is `probability_s`.

Parameters

number_s Number of successes in trials
number_f Number of failures preceding the number_s-th success
probability_s Probability of success on each trial

Example

Probability that there will be 10 failures preceding the fifth success in a series of trials having 25% chance of success on any given trial:

```
CALC.negbinomdist(10, 5, 0.25);
// returns 0.05504866037517786 (approximately 5.5%)
```

CALC.PEARSON(ARRAY1, ARRAY2)

Returns the Pearson product moment 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.pearson([3,2,4,5,6],[9,7,12,15,17]);
// returns 0.9970544855015815
```

The result should be comparable to that obtained with *CALC.correl(array1, array2)*.

CALC.PERCENTILE(ARRAY, K)

Returns the *k*-th percentile of values in `array`. The *k*-th percentile is the value below which fraction *k* of the data are found. For example, k=0.5 represents the value below which half (50%) of the data are found, also known as the median.

Parameters

array	Array of real values
k	Desired percentile

Example

```
CALC.percentile([9,7,5,3,1], 0.9);
// returns 8.2 (90 percent of values fall below 8.2)

CALC.percentile([9,7,5,3,1], 0.5);
// returns 5.0 (50 percent of values fall below 5, the
median)
```

CALC.PERCENTRANK(ARRAY, X, [SIGNIFICANCE])

Returns the rank of a value `x` within the data provided in `array`. The rank is given as a percentage indicating the percentage of values in `array` that fall below value `x`.

Parameters

array	Array of real values
x	Value to test
significance	Number of significant digits in the returned percentage value

Example

Where does the value 2 rank within the set {13,12,11,8,4,3,2,1,1,1}, to three decimal places.

```
CALC.percentrank([13,12,11,8,4,3,2,1,1,1],2,3);
// returns 0.333 (approximately 33.3% of values are less
than 2)
```

CALC.PERMUT(NUMBER, NUMBER_CHOSEN)

Returns the number of possible permutations in which `number_chosen` objects can be selected from `number` of objects, where order is significant.

Parameters

number	Number of objects in complete set
number_chosen	Number of objects in selected subsets

Example

The number of ways in which 2 objects can be selected from a set of 3, where order is significant.

```
CALC.permut(3,2);
// returns 6.0
```

CALC.POISSON(X, LAMBDA, CUMULATIVE)

Returns the value of the Poisson distribution for specified value of random variable `x` and rate parameter `lambda`. Value `x` typically represents the number of events that occur over a period of time in an independently

distributed random process, where `lambda` represents the expected (average) number of events over such a period.

Parameters

x	Number of events for which to compute probability
lambda	Average number of events
cumulative	Boolean: true: computes cumulative distribution function false: computes probability density function

Example

Probability that 2 or fewer events occur in a random process during the period of time when 5 events would occur on average.

```
CALC.poisson(2, 5, true);
// returns 0.12465201948308113 (approximately 12.5%)
```

CALC.PROB(ARRAY_X, ARRAYPROB, LOWER_LIMIT, [UPPER_LIMIT])

Returns the probability that a value falls between `lower_limit` and `upper_limit` (inclusive) based on the discrete probabilities in `arrayProb`. The probability is computed by adding together the probabilities in `arrayProb` corresponding to the values in `array_x` that fall in the specified range.

Parameters

array_x	Array of values corresponding to probabilities in arrayProb
arrayProb	Array of probabilities corresponding to values in array_x. Sum of probabilities must be 1 or less.
lower_limit	Lower boundary of the range
upper_limit	Upper boundary of the range. If omitted, equality with lower limit is tested.

If `upper_limit` is not supplied, the function returns the probability that values in `array_x` are equal to `lower_limit`.

Example

Given values {0,1,2,3} with associated probabilities {0.2,0.3,0.1,0.4}, find the probability that a value lies between 1 and 3 (inclusive).

```
CALC.prob([0,1,2,3], [0.2,0.3,0.1,0.4], 1, 3);
// returns 0.8 (the summed probability for values 1,2,3)
```

CALC.QUARTILE(ARRAY, QUART)

Returns the specified quartile for the values in `array`.

Parameters

array	Array of real values
quart	Desired quartile:

```
                0 for minimum value
                1 for first quartile (25th percentile)
                2 for second quartile (50th percentile,
median)
                3 for third quartile (75th percentile)
                4 for maximum value
```

Example

First quartile (25th percentile) for dataset {1,2,4,7,8,9,10,12}:

```
CALC.quartile([1,2,4,7,8,9,10,12], 1);
// returns 3.5
```

CALC.RANK(NUMBER, ARRAY, ORDER)

Returns the rank of `number` within the values in `array`. The rank is the position of the value in the list after sorting in ascending or descending order.

Parameters

```
number          The value for which to find rank. This value
must occur within array.
array           Array of real values. Non-numeric values are
ignored.
order           1: Find rank in ascending-sorted array
                0: Find rank in descending-sorted array
```

Example

```
CALC.rank(7,[1,2,4,7,8,9,10,12],1);
// returns 4 (value '7' is fourth from bottom)

CALC.rank(7,[1,2,4,7,8,9,10,12],0);
// returns 5 (value '7' is fifth from top)
```

CALC.RSQ(ARRAY1, ARRAY2)

Returns the square of the Pearson product moment correlation coefficient (*r*-squared) for the values in `array1` and `array2`. The squared correlation coefficient indicates the degree to which variance in `array2` can be accounted for by variance in `array1` (or vice versa). An *r*-squared value near 0 indicates that the values in `array1` and `array2` bear little linear relationship. An *r*-squared value near 1 indicates that the values in `array1` and `array2` bear a strong linear relationship, such that the variance in `array2` can be largely explained by variance in `array1` using a linear model.

Parameters

```
array1          Array of real values
array2          Array of real values
```

Example

```
CALC.rsq([3,2,4,5,6],[9,7,12,15,17]);
// returns 0.9941176470588233
```

CALC.SKEW(ARRAY)

Returns the skewness of a data set, which indicates the asymmetry of a distribution around its mean. Positive skewness indicates that the right-side tail of the distribution is longer than the left, while negative skewness indicates that the left-side tail of the distribution is longer than the right.

Parameters

array	Array of values

Example

```
CALC.skew([3,4,5,2,3,4,5,6,4,7]);
// returns 0.3595430714067974
```

CALC.SLOPE(ARRAY_Y, ARRAY_X)

Calculates the slope of the best-fit regression line through the values in `array_x` and `array_y`.

Parameters

array_y	Array of y values
array_x	Array of x values

Example

```
CALC.slope([8,11,14,17],[1,2,3,4]);
// returns 3.0
```

CALC.SMALL(ARRAY, K)

Returns the `k`-th smallest value in `array`.

Parameters

array	Array of real values
k	Position (from smallest) in array of data to return

Example

```
CALC.small([1,2,4,7,8,9,10,12],3);
// returns 4.0 (value '4' is third from smallest)
```

CALC.STANDARDIZE(X, MEAN, STANDARD_DEV)

Returns the standard score (*z*-score) for value `x` based on the distribution characterized by population `mean` and `standard_dev`. The standardized value is `(x-mean)/standard_dev`, and indicates the deviation of `x` from the mean value in terms of the standard deviation.

Parameters

x	Value to standardize
mean	Mean of the distribution
standard_dev	Standard deviation of the distribution

Example

Standard score of value 42, where the population mean is 40 and standard deviation is 1.5.

```
CALC.standardize(42, 40, 1.5);
// returns 1.3333333333333333 (42 is approximately 1.33
standard deviations above the mean)
```

CALC.STDEV(ARRAY)

Returns the sample standard deviation for the values in `array`. Standard deviation is the square root of the variance and provides a measure of how widely the data are spread about the mean.

Parameters

array Array of real values. Text and Boolean values are ignored.

Example

```
CALC.stdev([1345,1301,1368,1322,1310,1370,1318,1350,1303,12
99]);
// returns 27.46391571984349
```

CALC.STDEVA(ARRAY)

Returns the sample standard deviation for the values in `array`. Standard deviation is the square root of the variance and provides a measure of how widely the data are spread about the mean. Boolean values in `array` are considered to be 1 if `true` and 0 if `false`.

Parameters

array Array of real or Boolean values. Boolean 'true' evaluates to 1; 'false' evaluates to 0. String values are ignored.

Example

```
CALC.stdeva([true,
false,1345,1301,1368,1322,1310,1370,1318,1350,1303,1299]);
// returns 517.5588021050011
```

CALC.STDEVP(ARRAY)

Returns the *population* standard deviation for the values in `array`. Standard deviation is the square root of the variance and provides a measure of how widely the data are spread about the mean.

Parameters

array Array of real values. Text and Boolean values are ignored.

Example

```
CALC.stdevp([1345,1301,1368,1322,1310,1370,1318,1350,1303,1
299]);
```

```
// returns 26.054558142482477
```

CALC.STDEVPA(ARRAY)

Returns the *population* standard deviation for the values in `array`. Standard deviation is the square root of the variance and provides a measure of how widely the data are spread about the mean. Boolean values in `array` are considered to be 1 if `true` and 0 if `false`.

Parameters

array Array of real or Boolean values. Boolean 'true' evaluates to 1; 'false' evaluates to 0. String values are ignored.

Example

```
CALC.stdevpa([true,
false,1345,1301,1368,1322,1310,1370,1318,1350,1303,1299]);
// returns 495.5248269932261
```

CALC.STEYX(ARRAY_Y, ARRAY_X)

Returns the standard error of the estimate for a regression model based on independent values in `array_x` and dependent (function) values in `array_y`. The standard error represents the average distance of values in `array_y` from the regression line.

Parameters

array_y Array of y (dependent) values corresponding to values in array_x

array_x Array of x (independent) values corresponding to values in array_y

Example

```
CALC.steyx([2,3,9,1,8,7,5], [6,5,11,7,5,4,4]);
// returns 3.305718950210041
```

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. This provides a measure of the mean that is not influenced by outliers.

Parameters

array Array of values to trim and average

percent Fraction of data points to exclude

Example

Mean of data set {4,5,6,7,2,3,4,5,1,2,3} after removing 20% of the most extreme values (10% from each tail).

```
CALC.trimmean([4,5,6,7,2,3,4,5,1,2,3], 0.2);
// returns 3.7777777777777777
```

CALC.VARN(ARRAY)

Returns the sample variance for the values in `array`. Variance is the square of the standard deviation and provides a measure of how widely the data are spread about the mean.

Parameters

array Array of real values. Text and Boolean values are ignored.

Example

```
CALC.varn([1345,1301,1368,1322,1310,1370,1318,1350,1303,1299]);
// returns 754.2666666666665
```

CALC.VARA(ARRAY)

Returns the sample variance for the values in `array`. Variance is the square of the standard deviation and provides a measure of how widely the data are spread about the mean. Boolean values in `array` are considered to be 1 if `true` and 0 if `false`.

Parameters

array Array of real or Boolean values. Boolean 'true' evaluates to 1; 'false' evaluates to 0. String values are ignored.

Example

```
CALC.vara([true,false,1345,1301,1368,1322,1310,1370,1318,1350,1303,1299]);
// returns 267867.11363636365
```

CALC.VARP(ARRAY)

Returns the *population* variance for the values in `array`. Variance is the square of the standard deviation and provides a measure of how widely the data are spread about the mean.

Parameters

array Array of real values. Text and Boolean values are ignored.

Example

```
CALC.varp([1345,1301,1368,1322,1310,1370,1318,1350,1303,1299]);
// returns 678.8399999999999
```

JS.12.6Text Functions

This section presents the text functions provided by the CALC object.

CALC.CHARACTER(NUMBER)

Returns the unicode character specified by a decimal value.

Parameters

number Positive integer specifying a character

Example

```
CALC.character(77);
// returns M
```

CALC.CODE(STRING)

Returns the numeric character code corresponding to the first character in `string`.

Parameters

string A string whose first character should be evaluated

Example

```
CALC.code('jsd');
// returns 106 (code for 'j')
```

CALC.CONCATENATE(ARRAY)

Returns the strings in `array` joined into a single string.

Parameters

array An array of strings

Example

```
CALC.concatenate(['madam','i','am','adam']);
// returns madamiamadam
```

If the number of strings to concatenate is relatively small, you can also use the JavaScript concatenation operator. For example:

```
'madam' + 'i' + 'am' + 'adam';
// returns madamiamadam
```

CALC.DOLLAR(NUMBER, DECIMALS)

Converts a numerical value to text format and applies a currency symbol. Decimals is the number of digits to the right of the decimal point.

Parameters

number The value to convert to currency
decimals Number of digits to right of decimal point in the result

Example

```
CALC.dollar(12345.789, 2);
```

```
// returns $12,345.79
```

CALC.EXACT(STRING1, STRING2)

Returns `true` if `string1` is identical to `string2`, and returns `false` otherwise.

Parameters

string1	A string to compare
string2	A string to compare

Example

```
CALC.exact('hello','Hello');
// returns false (due to case mismatch)
```

You can also use the JavaScript identity operator for this purpose. For example:

```
'hello' === 'Hello';
// returns false (due to case mismatch)
```

CALC.FIND(STRING1, STRING2, SEARCH_INDEX)

Returns the location of the first occurrence of `string1` within `string2`, starting the search from the position specified by `search_index`.

Parameters

string1	A string to find
string2	The string in which to search
search_index	Location at which to start search (first character has index 1)

Example

Searching within the string “asdqwertasd” starting with the sixth character (“e”), at what index does the substring “asd” appear?

```
CALC.find('asd','asdqwertasd', 6);
// returns 9
```

CALC.FIXED(NUMBER, DECIMALS, NO_COMMAS)

Returns a number rounded to the specified number of `decimals` and formatted as text using a period and commas.

Parameters

number	A numerical value to round and format as text
decimals	The number of digits to right of decimal point
no_commas	Boolean: true to suppress commas in result false to allow commas in result

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 side of `string`.

Parameters

`string` A string from which to extract characters
`num_chars` Size of substring to extract, starting from left-most character

Example

```
CALC.left('abcdefghij', 4);
// returns abcd
```

To extract a substring with arbitrary starting and ending indexes, use *substring(indexA, indexB)* or *CALC.mid(String, start_num, num_chars)*.

CALC.LEN(STRING)

Returns the number of characters in a text string, which is generally the same as `string.length`.

Parameters

`string` A string in which to count characters

Example

```
CALC.len('abc def ghij');
// returns 12
```

CALC.LOWER(STRING)

Converts all uppercase characters in `string` to lowercase characters. This is generally the same as `string.toLowerCase()`.

Parameters

`string` A string to convert to lowercase

Example

```
CALC.lower('abCdefgHIJ');
// returns abcdefghij
```

CALC.MID(STRING, START_NUM, NUM_CHARS)

Returns `num_chars` consecutive characters from `string`, starting with the location given by `start_num`.

Parameters

`string` A string from which to extract characters
`start_num` Starting index of extracted string. (Index of first character is 1.)
`num_chars` Number of consecutive characters to extract, starting from start_num

Example

```
CALC.mid('abcdefghij', 2, 4);
// returns bcde
```

To extract a substring with arbitrary starting and ending indexes, use *substring(indexA, indexB)*.

CALC.PROPER(STRING)

Capitalizes the first letter of each word in `string`.

Parameters

string	A string to convert to initial capitals

Example

```
CALC.proper('my name is alex');
// returns My Name Is Alex
```

CALC.REPLACE(OLD_STRING, START_NUM, NUM_CHARS, NEW_STRING)

Replaces a segment of `old_string` with the entirety of `new_string`. The segment replaced in `old_string` begins with index `start_num` and continues for `num_chars` characters.

Parameters

old_string	A string in which to replace characters
start_num	Starting index in old_string for replacement. (Index of first character is 1.)
num_chars	Number of characters in old_string to replace
new_string	String containing replacement text

Example

Replace the five characters “fghij” in string “abcdefghijkl” with the new string ‘*’.

```
CALC.replace('abcdefghijkl',6,5,'*');
// returns abcde*kl
```

This function replaces text that occurs at a specific location in `old_string`. Use *CALC.substitute(old_string, rep_string , new_string, instance_num)* to replace an occurrence of a particular substring in `old_string`.

CALC.REPT(STRING, NUMBER_TIMES)

Repeats `string` a given number of times (concatenating the string to itself).

Parameters

string	String to repeat
number_times	Integer number of times to repeat string

Example

```
CALC.rept('abcd',5);
```

```
// returns abcdabcdabcdabcdabcd
```

CALC.RIGHT(STRING, NUM_CHARS)

Returns `num_char` number of characters from the right side of `string`.

Parameters

string A string from which to extract characters
num_chars Size of substring to extract, starting from right-most character

Example

```
CALC.right('abcdefghij', 4);
// returns ghij
```

To extract a substring with arbitrary starting and ending indexes, use *substring(indexA, indexB)* or *CALC.mid(String, start_num, num_chars)*.

CALC.SEARCH(FIND_STRING, SEARCH_STRING, START_NUM)

Locates `find_string` within `search_string` and returns the index of the location where `find_string` begins. The function returns `-1` if the string is not found.

Parameters

find_string The substring to locate within search_string
search_string The string in which to search
start_num Index in search_string at which to begin searching

Example

```
CALC.search('e','statements', 6);
// returns 7
```

CALC.SUBSTITUTE(OLD_STRING, REP_STRING , NEW_STRING, INSTANCE_NUM)

Replaces an instance of `rep_string` within `old_string` using replacement text provided by `new_string`.

Parameters

old_string A string containing one or more instances of rep_string
rep_string The string to replace within old_string
new_string String containing replacement text
instance_num Occurrence of rep_string instance to replace

Example

Replace the second instance of “/” in string “asd/pqr/xyz” with “-”.

```
CALC.substitute('asd/pqr/xyz','/','-', 2);
// returns asd/pqr-xyz
```

This function replaces specific text within `old_string`. Use *CALC.replace(old_string, start_num, num_chars, new_string)* to replace text that occurs at a specific location in `old_string`.

CALC.T(VALUE)

Returns the string or date supplied by `value`. If `value` is numerical or Boolean, the function returns an empty string.

Parameters

`value` `String, date, numeric, or Boolean value. Strings are returned unchanged. Dates are converted to string format. Numeric and Boolean values return an empty string.`

Example

```
CALC.t('asd');
// returns asd
```

Example

```
CALC.t(true)
CALC.t(56);
// returns ''
```

CALC.TEXT(VALUE, FORMAT)

Converts a numerical value to a string based on the specified format.

Parameters

`value` `Numerical value to convert to text`
`format` `A valid number format. See below.`

Example

```
CALC.text(2800, '$0.00');
// returns $2800.00
```

Construct the number format string using the characters in the table below. The syntax is the same as Java's DecimalFormat.

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

CALC.TRIM(STRING)

Removes all white space characters from `string` except for single spaces after words or punctuation.

Parameters

`string` A string from which to remove extra white space

Example

```
CALC.trim('   Mary had a   little     lamb.')
// returns Mary had a little lamb.
```

CALC.UPPER(STRING)

Converts all lowercase characters in `string` to uppercase characters. This is generally the same as `string.toUpperCase()`.

Parameters

`string` A string to convert to uppercase

Example

```
CALC.upper('abCdefgHIJ');
// returns ABCDEFGHIJ
```

CALC.VALUE(STRING)

Converts a formatted number string to the equivalent numerical value.

Parameters

`string` A string to convert

Example

```
CALC.value('$2800.00');
// returns 2800.0
```

JS.13 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);
```

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.
Advanced 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 *onInit* 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 *Advanced Report Scripting*, for information on the report handler.
onRefresh Handler, in *Advanced Dashboard Scripting*, for information on the Viewsheet handler.
Element Script, in *Advanced 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 in ‘Orders’ > ‘Sample Queries’ 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:

   a. For a report, right-click an empty area of the report and select ‘Script’ from the context menu to open the *onLoad Handler*.

   b. For a Viewsheet, press the ‘Options’ button in the toolbar to open the ‘Viewsheet Options’ dialog box, then select the **Script** tab to open the *onRefresh Handler*.

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);
```

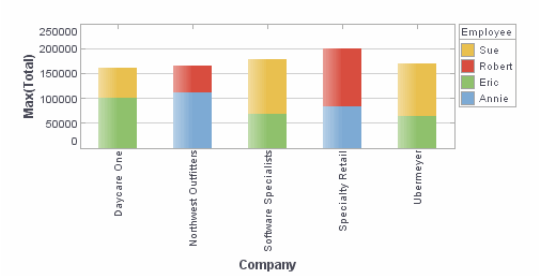
9. Close the Editor, an preview the report or Viewsheet.

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 , *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 in 'Orders' > 'Sample Queries' 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.

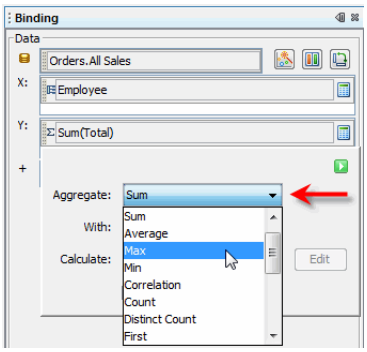
4. For a report, in the **Data** panel, expand the 'Orders' data source, the 'Sample Queries' folder, and the 'All Sales' query. For a Viewsheet, in the **Data Source** panel, expand the 'All Sales' data block.

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.



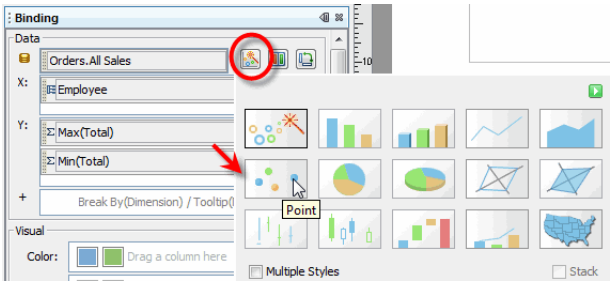
   c. From the 'All Sales' source, drag the 'Total' field (a second time) to the 'Y' region.

   d. Click the 'Edit Measure' button next to the second 'Total' field. Set the 'Aggregate' to 'Min', and click the green 'Apply' button.

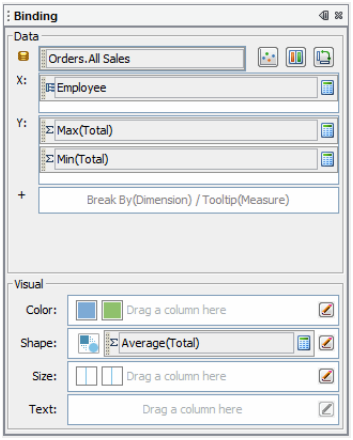
   e. From the 'All Sales' source, drag the 'Total' field (a third time) to the 'Shape' region of the 'Visual' area.

   f. Click the 'Edit Measure' button next to the third 'Total' field. Set the 'Aggregate' to 'Average', and click the green 'Apply' button.

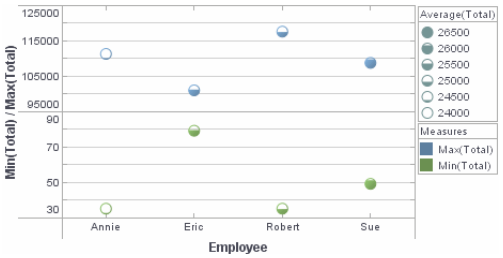
7. Click the 'Select Chart Style' button. Double-click to select the Point Style chart.



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
```

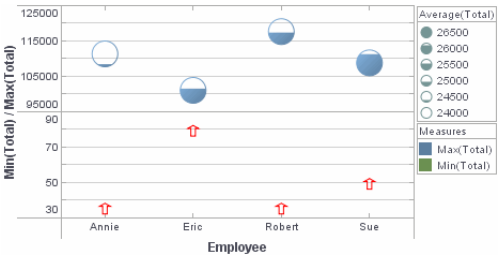
12. Assign the visual frames to the appropriate data elements using the element’s *GraphElement.setShapeFrame(frame)*, *GraphElement.setColorFrame(frame)*, and *GraphElement.setSizeFrame(frame)* properties.

```
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.

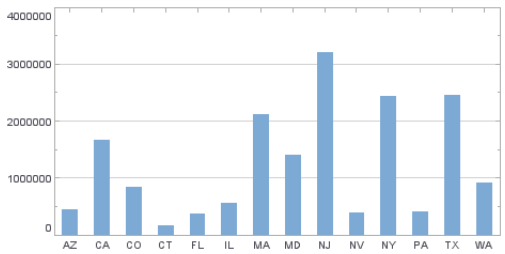
6. **Add the created elements to the Chart object**. Pass the *GraphElement* object to the Chart's *`EGraph.addElement(elem)`* method. This adds the "bar" elements to the existing Chart object.

   ```
   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*. For details about the syntax of `runQuery()`, see *Running a Query from Script* in *Advanced Dashboard Scripting* or *Running a Query from Script* in *Advanced Report Scripting*.

If you want to access a data block in the Data Worksheet on which the current Viewsheet is based, you can use the simpler `data` syntax shown below:

```
Chart1.data = Sales; // “Sales” is the data block name
graph = new EGraph();
var elem = new IntervalElement("State", "Sales");
graph.addElement(elem);
```

If the data block name contains spaces, use the syntax `viewsheet['Data Block Name']`. See *Referencing Data in the Worksheet Data Block* in

Advanced Dashboard Scripting for more information about accessing Worksheet data in Viewsheet script.

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 *Advanced 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]];
```

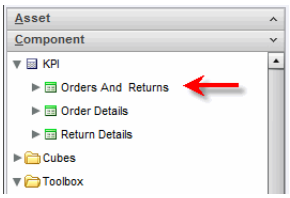
You can see an example of this approach in *Modifying a Chart Data Binding*.

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*.

If you want to construct the chart using Chart API functions that operate on the *EGraph* object (rather than using `bindingInfo` properties to construct

the chart), use the chart's `data` property instead of the `query` property. For example:

```
Chart1.data = Sales; // "Sales" is the data block name
graph = new EGraph();
var elem = new IntervalElement("State", "Sales");
graph.addElement(elem);
```

If the data block name contains spaces, use the syntax `viewsheet['Data Block Name']`. See *Referencing Data in the Worksheet Data Block* in *Advanced Dashboard Scripting* for more information about accessing Worksheet data in Viewsheet script.

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.

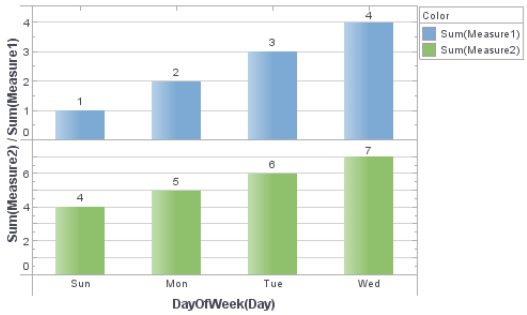
```
Chart1.data = [["State","Quantity"],["NJ",200],["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
graph.addElement(elem);
```

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);

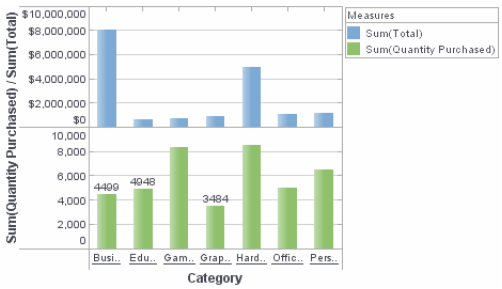
    // Set the label position and alignment
    form.setValues([Xvalue,Yvalue]);
    form.setAlignmentX(Chart.CENTER_ALIGNMENT);

    // 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*.

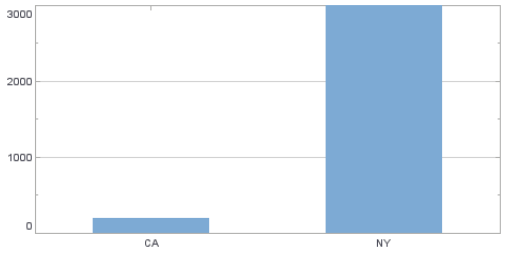
A *Scale* object determines how abstract data values are mapped to physical chart representations such as position, color, shape, etc. The *EGraph* object and the *VisualFrame* object both map data values to physical representations, so both of these objects require a *Scale*.

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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");
   ```

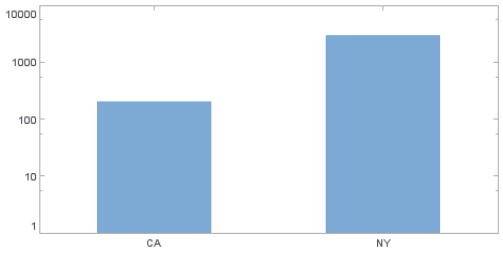
2. Assign the new scale to the Y-axis of the chart object using the `setScale()` method.

   ```
   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:

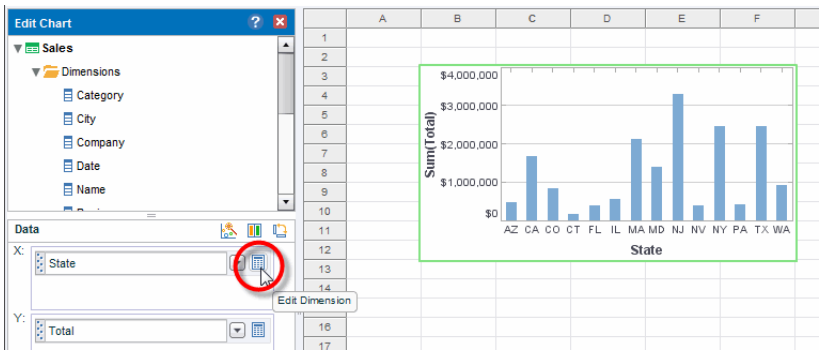
1. Create a new Viewsheet based on the ‘Sales’ > ‘Sales Explore’ Worksheet.

2. Add a Chart component to the Viewsheet, and click the center of the chart to open the Chart Editor.

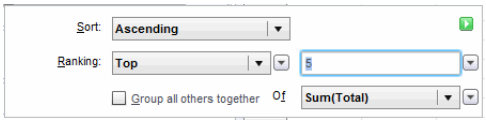
3. From the ‘Dimensions’ node of the ‘Data Source’ panel, drag the ‘State’ field to the ‘X’ region of the ‘Data’ panel.

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:

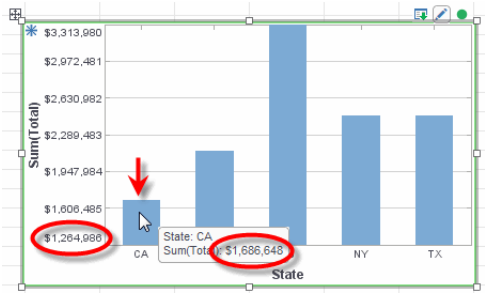
```
var dMin = 10000000; // Default minimum

// Get a handle to the chart's Y-axis Scale:
var yScale = graph.getCoordinate().getYScale();

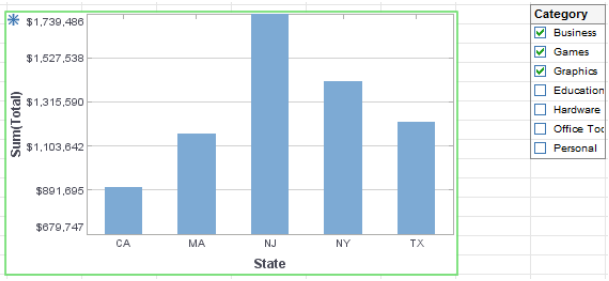
// Find the minimum Y-value in the chart data:
for (var i=0; i<dataset.getRowCount(); i++) {
  yVal = dataset.getData('Sum(Total)',i);
  if(yVal < dMin) {
    dMin = yVal;
  }
}

// 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.

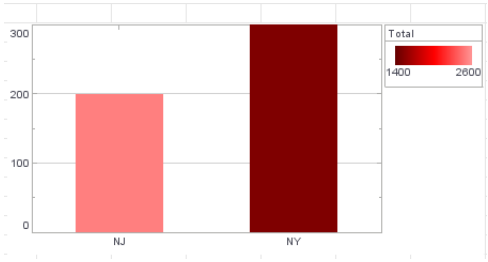
CT.7.2Changing Scaling for a VisualFrame

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

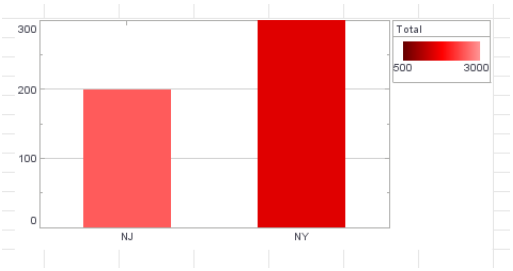
```
var scale = new LinearScale("Total");
scale.setFields("Total");
```

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);
```

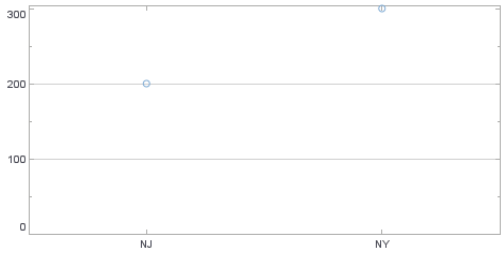
CT.8 Changing the Appearance of Chart Elements

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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).

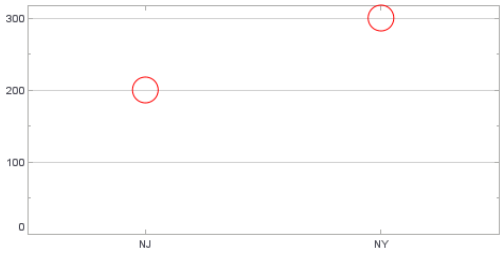
   ```
   var cframe = new StaticColorFrame();
   cframe.setColor(java.awt.Color(0xff0000)); // red
   ```

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.

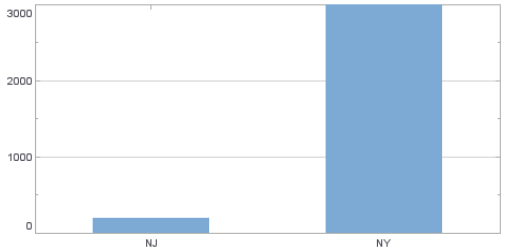
CT.9 Changing Axis Properties

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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*.

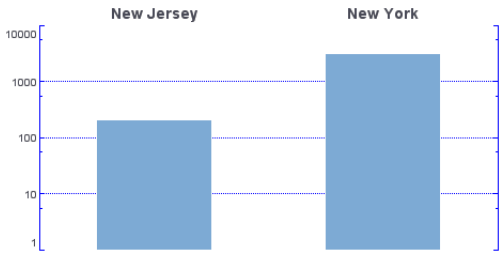
```
var tspec = new TextSpec();
tspec.setFont(java.awt.Font('Dialog',
              java.awt.Font.BOLD, 14));
xspec.setTextSpec(tspec);
xspec.setAxisStyle(AxisSpec.AXIS_SINGLE2);
```

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);
```

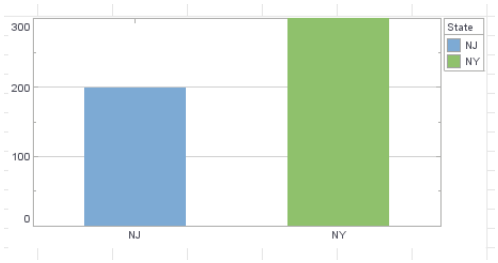
CT.10Changing Legend Properties

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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*.

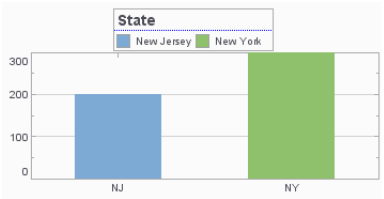
   ```
   var tspec = new TextSpec();
   tspec.setFont(java.awt.Font('Dialog',
                 java.awt.Font.BOLD, 14));
   legend.setTitleTextSpec(tspec);
   legend.setTitle('State');
   ```

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 in the 'Orders' > 'Sample Queries' folder. (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, the 'Sample Queries' folder, 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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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];

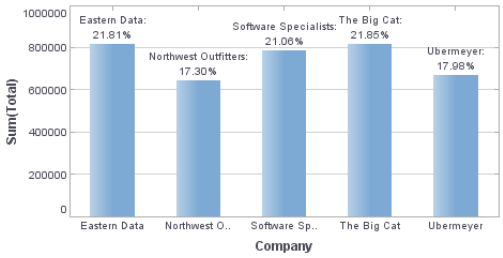
  // Compute the fractional value of corresponding 'Total':
  var barFraction = table[i][1]/sumTotal;

  // 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++) {

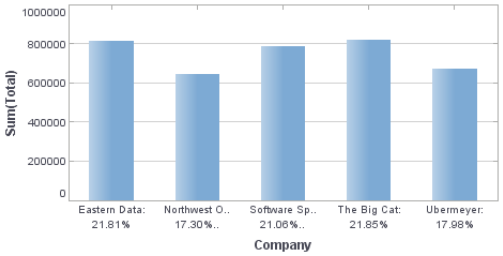
  // Get the original label (company):
  var oldLabel = table[i][0];

  // 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:

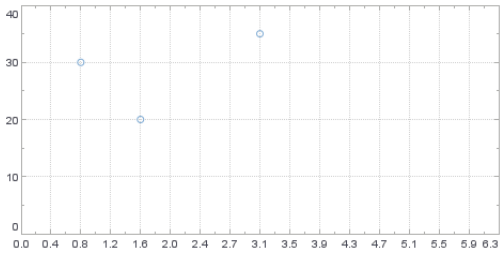
Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced Dashboard Scripting* for more information.

```
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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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);
```

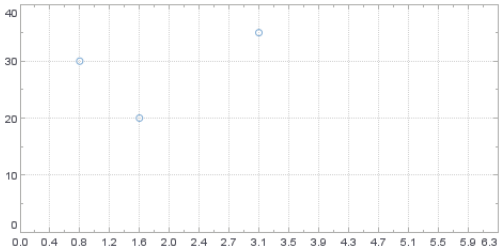
The resulting chart is the same as in the previous case.

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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);

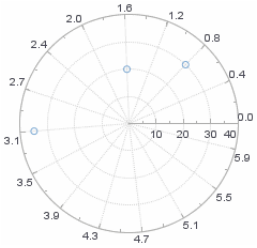
xscale.setMin(0);
xscale.setMax(1.95*Math.PI);
xscale.setIncrement(Math.PI/8);
var xaxis = new AxisSpec();
var tspec = new TextSpec();
tspec.setFormat(new java.text.DecimalFormat("0.0"));
xaxis.setTextSpec(tspec);
xaxis.setGridStyle(Chart.DOT_LINE);
xscale.setAxisSpec(xaxis);
```

```
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*.

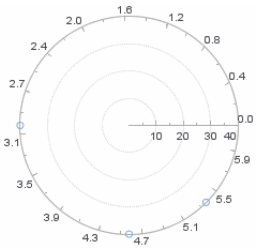
The example below is the same as that in *Converting Rectangular to Polar Coordinates*, except that here only the angle dimension is mapped:

```
Note: Script that modifies 'graph' should be placed at the element
level. See Editing Script in Advanced Report Scripting and Adding
Component Script in Advanced 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);
```



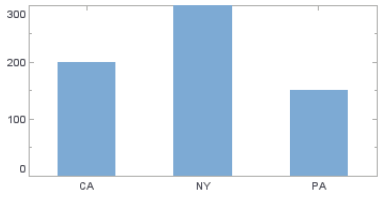
The result is that all points have the same magnitude, with variation only along the angle dimension. To reverse a mapping, use the *Coordinate* object's `transpose()` method.

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced Dashboard Scripting* for more information.

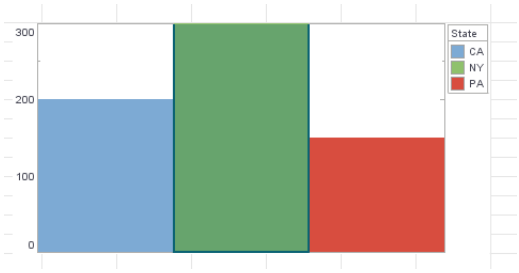
```
dataset = [["State", "Revenue"], ["CA",
200],["NY",300],["PA",150]];
graph = new EGraph();
var elem = new IntervalElement("State","Revenue");
var xscale = new CategoricalScale("State")
var yscale = new LinearScale("Revenue");
var rect = new RectCoord(xscale, yscale);
graph.setCoordinate(rect);
graph.addElement(elem);
```



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());
```

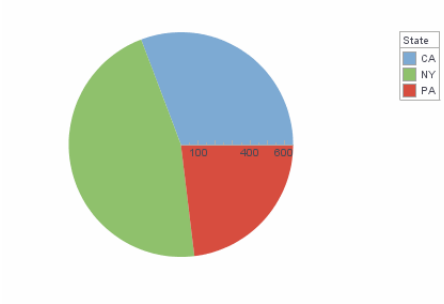


3. Create polar coordinates from the existing rectangular coordinates. By default, this maps the non-null coordinate (Y-axis) to the polar coordinate's magnitude dimension. Specify that it should be mapped to the angle dimension instead.

```
var polar = new PolarCoord(rect);
polar.setType(PolarCoord.THETA);
```

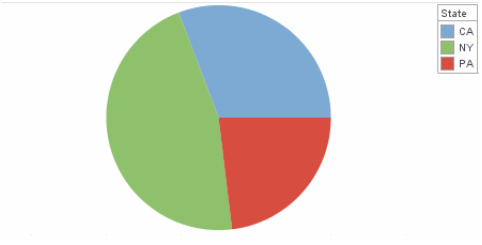
4. Use the Chart's `setCoordinate()` method to apply the new polar coordinates.

```
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);
```

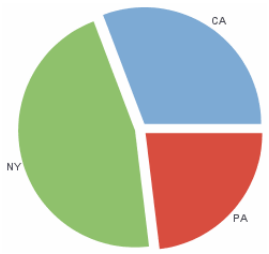


6. Add the state names to the individual slices and hide the legend. To do this, create a new *TextFrame* object based on the "State" field, and assign it to the *GraphElement*. To hide the legend, create a new *LegendSpec* object, and assign it to the *ColorFrame*.

```
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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

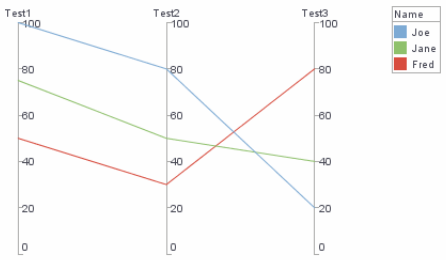
```
var frame = new CategoricalColorFrame("Name");
elem.setColorFrame(frame);
```

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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').

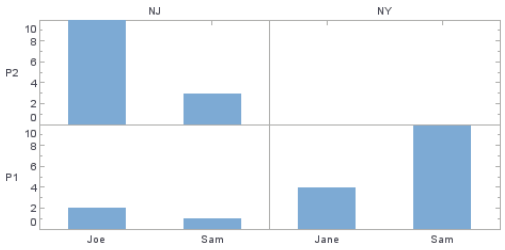
```
var inner = new RectCoord(name, priority);
var outer = new RectCoord(state, product);
```

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)
```

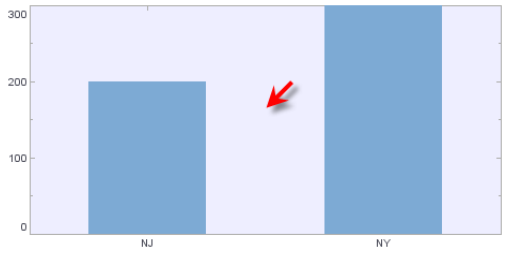
CT.12.5Setting a Coordinate Background

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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

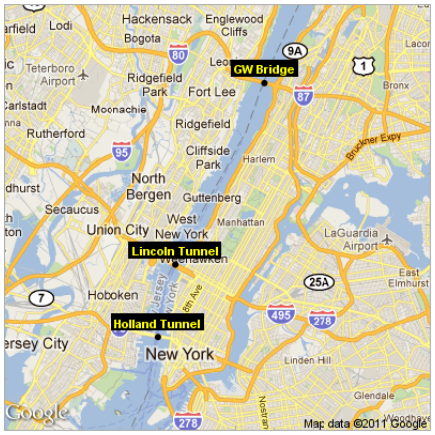
// Set chart scales to match image coordinates:
var latscale = new LinearScale("Latitude");
var lonscale = new LinearScale("Longitude");
```

```
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 *Advanced 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.

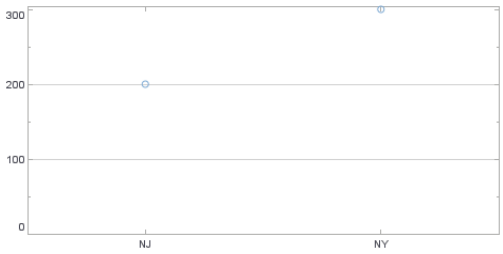
CT.13Representing Data with Shape, Color, Size

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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:

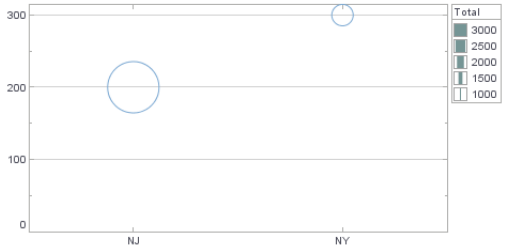
   ```
   frame.setScale(scale);
   ```

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.

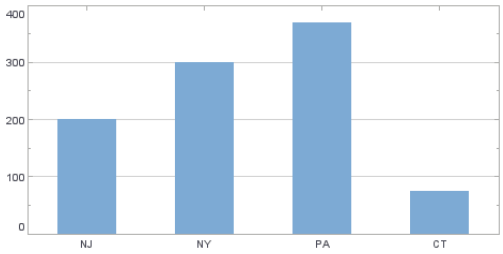
CT.14Representing Multiple Measures

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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);
```



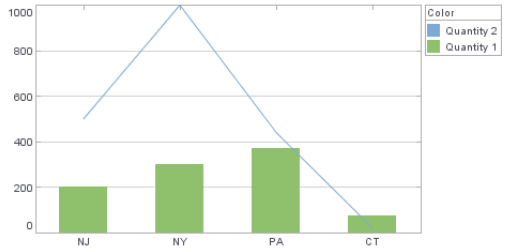
This creates a basic bar chart displaying the dimensions 'State' and 'Quantity 1'. To add the measure 'Quantity 2' to the chart as line element, follow the steps below:

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.

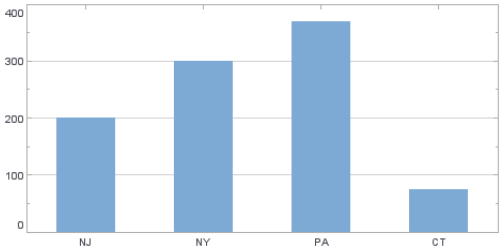
CT.15Adding Decorative Elements

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 Editing Script in Advanced Report Scripting and Adding
Component Script in Advanced 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.

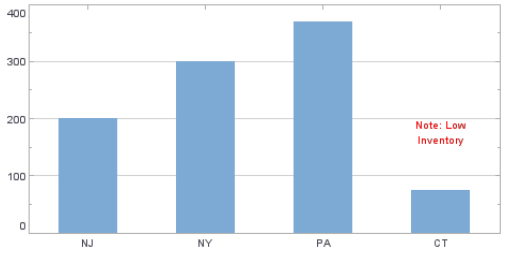
   ```
   var labelform = new LabelForm();
   labelform.setValues(['CT', 150]);
   ```

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.

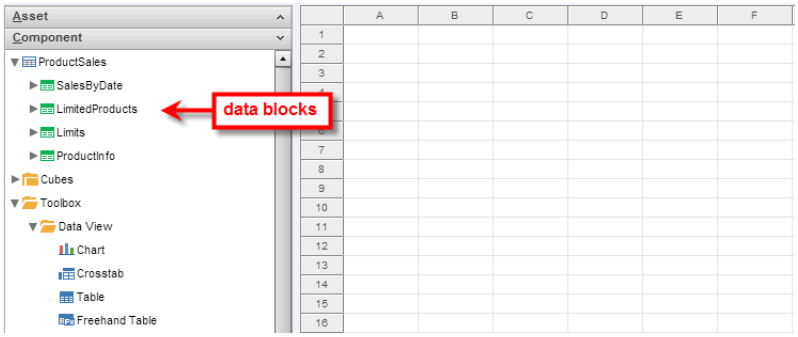
Modifying a Chart Element using API Functions, for examples of modifying the contents of a chart.

CR.1.1 General Properties

This section presents general Chart properties.

QUERY

Specifies the data block to be used by the Chart component. The available data blocks appear at the top of the **Component** panel in Visual Composer.



Type

`string`

Example

```
query = 'SalesByDate';
```

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']
```

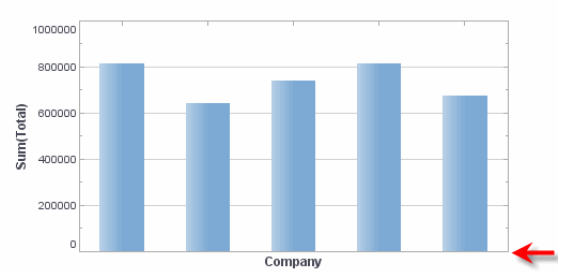
Example
```
Chart1.axis['Company'].lineColor = [0,255,255];
Chart1.axis['Sum(Total)'].lineColor = [0,255,255];
```

AXIS.LABELVISIBLE

Sets the visibility of the dimension or measure axis labels.

Type
Boolean

Example
```
Chart1.axis['Company'].labelVisible = false;
```

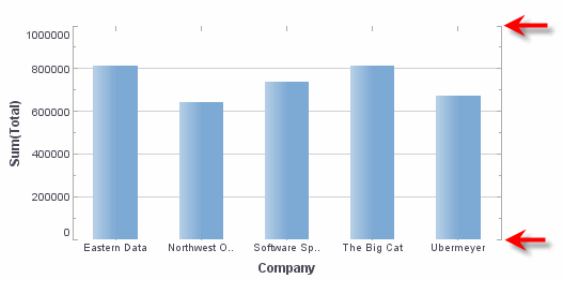


AXIS.LINEVISIBLE

Sets the visibility of the dimension or measure axis line.

Type
Boolean

Example
```
Chart1.axis['Company'].lineVisible = false;
```

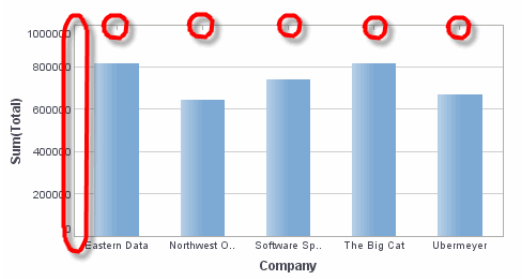


AXIS.TICKSVISIBLE

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

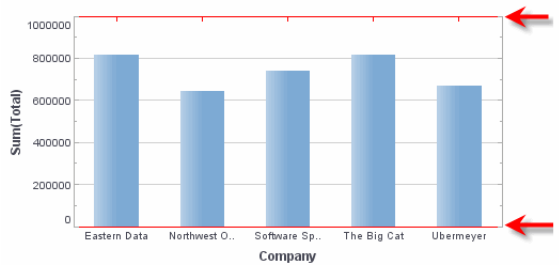
Sets the color of the dimension or measure axis line.

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

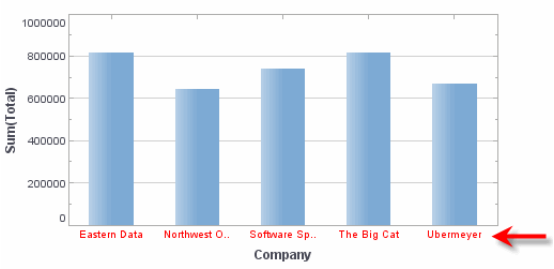
Sets the color of the dimension or measure axis labels.

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.

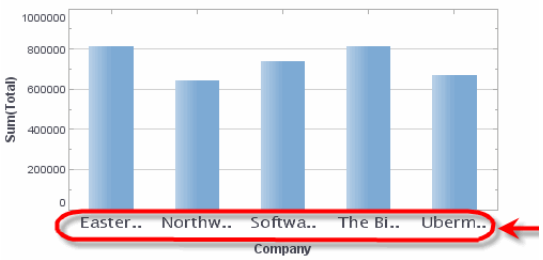
Sets the font of the dimension or measure axis labels.

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.

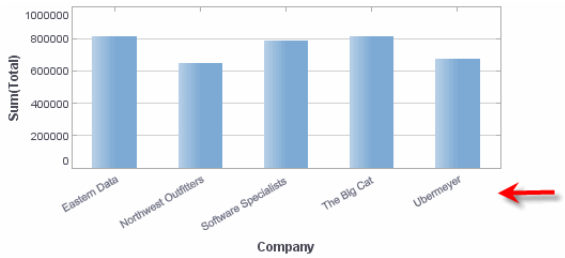
Sets the rotation of the dimension or measure axis labels.

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

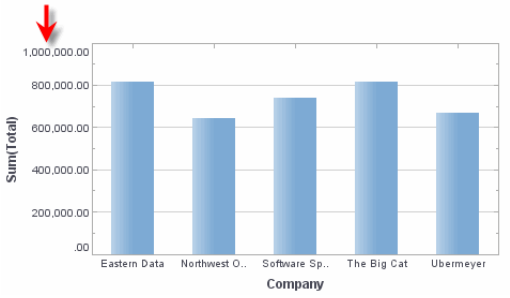
For the Date, Decimal, and Message formats, optionally specify the format and mask as items in an array. See *Format Types* in *Advanced Report Design* and *Format Tab* in *Advanced Dashboard Design* for more information about the masks.

Type

```
Constant (format) or Array ([format, mask])
```

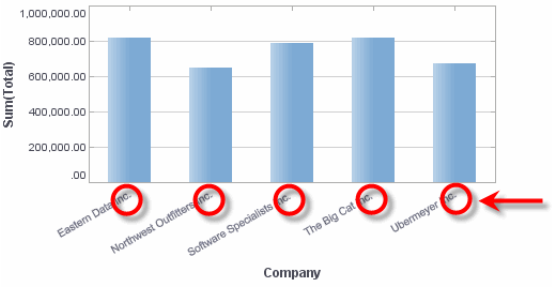
Example (Number)

```
Chart1.axis['Sum(Total)'].format =
[Chart.DECIMAL_FORMAT,"#,###.00"];
```



Example (Text)

```
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

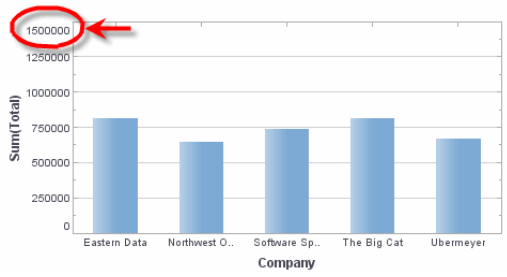
Sets the maximum value for a measure axis.

Type

number

Example

```
Chart1.axis['Sum(Total)'].maximum = 1500000;
```



AXIS.MINIMUM

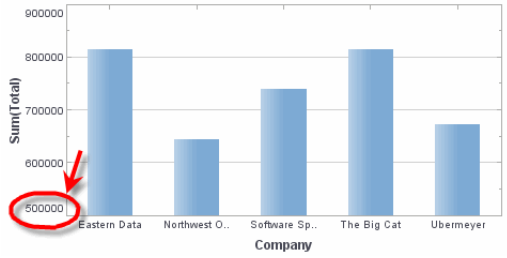
Sets the minimum value for a measure axis.

Type

number

Example

```
Chart1.axis['Sum(Total)'].minimum = 500000
```

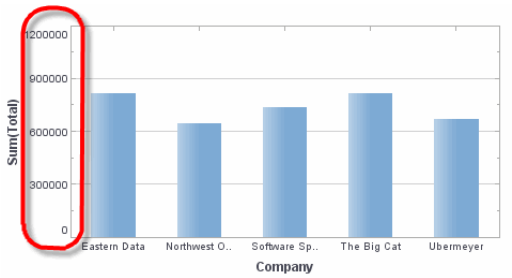


AXIS.INCREMENT

Sets the numerical increment for measure axis labels.

Type
`number`

Example
```
Chart1.axis['Sum(Total)'].increment = 300000
```

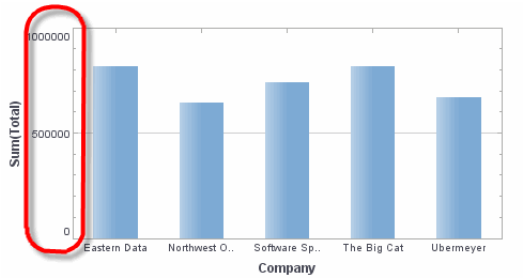


AXIS.MINORINCREMENT

Sets the numerical increment for measure axis tick marks.

Type
`number`

Example
```
Chart1.axis['Sum(Total)'].increment = 500000
Chart1.axis['Sum(Total)'].minorIncrement = 100000
```

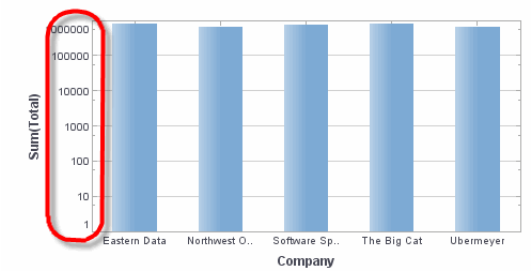


AXIS.LOGARITHMIC

Sets logarithmic scaling for a measure axis.

Type
`Boolean`

Example
```
Chart1.axis['Sum(Total)'].logarithmic = true
```



AXIS.REVERSED

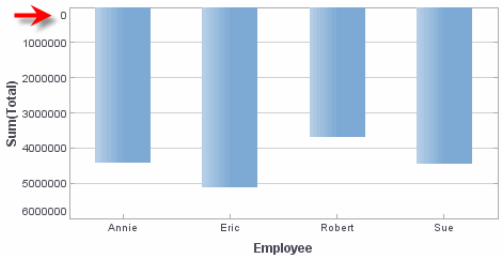
Reverses the direction of the axis scale so that higher numbers appear at the bottom or left.

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.

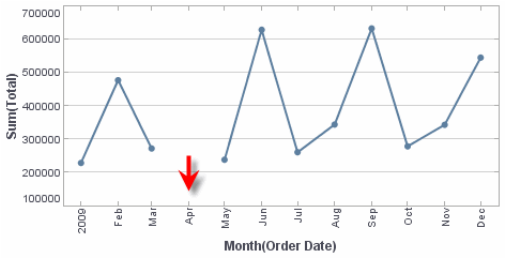
Example 1

```
Chart1.fillTimeGap = false
```



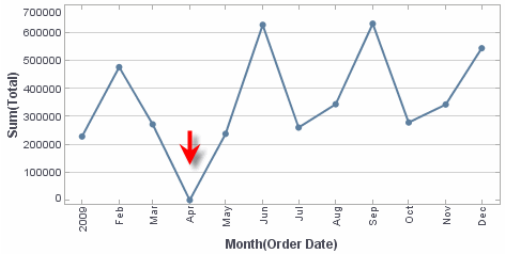
Example 2

```
Chart1.fillTimeGap = true
Chart1.fillZero = false
```



Example 3

```
Chart1.fillTimeGap = true
Chart1.fillZero = true
```



FILLZERO

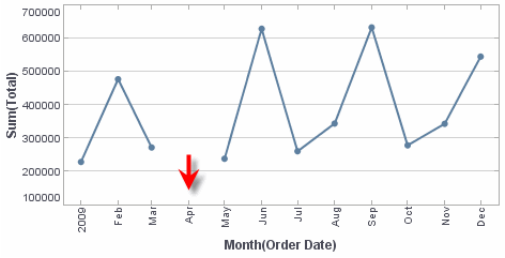
When you set *fillTimeGap* to `true` or set the corresponding property in the dialog box, the `fillZero` property determines whether gaps are represented as ‘Null’, which leaves an empty space at the location, or as ‘Zero’, which inserts a numerical value of 0 at the location.

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.

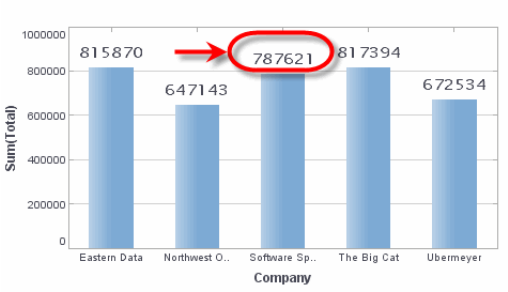
Sets the font for data values displayed in the chart plot area.

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

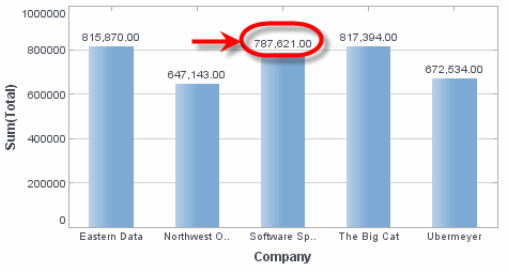
For the Date, Decimal, and Message formats, optionally specify the format and mask as items in an array. See *Format Types* in *Advanced Report Design* and *Format Tab* in *Advanced Dashboard Design* for more information about the masks.

Type

```
Constant (format) or Array ([format, mask])
```

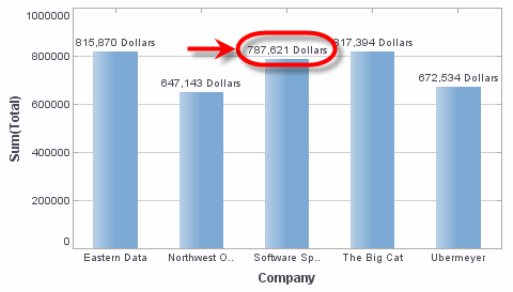
Example (Number)

```
Chart1.valueFormat = [Chart.DECIMAL_FORMAT,"#,###.00"];
```



Example (Text)

```
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

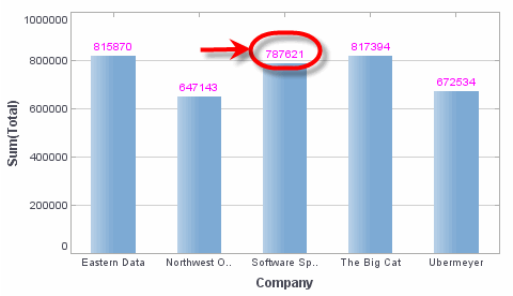
Sets the color for data values displayed in the chart plot area.

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

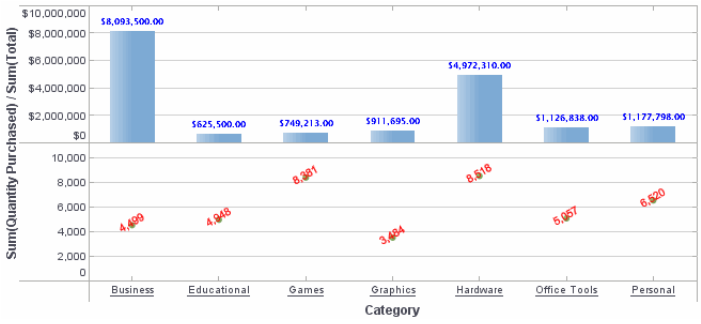
valueFormats, to set the color of a value on a multi-style chart.

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

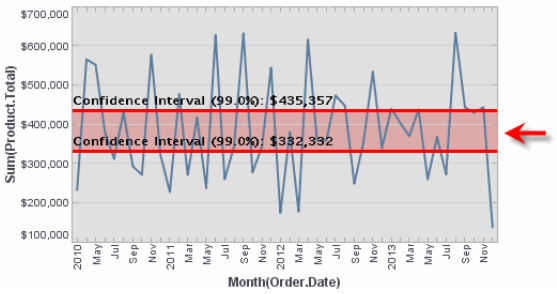
measure Name of measure on which to draw confidence intervals
colors Color (see *valueColor*) or array of colors to fill area of confidence intervals
values Confidence level as a percentage (e.g., '99' for 99% confidence interval)
options Name-value pairs of options (see below, e.g., {fillAbove:0xff0000, 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})

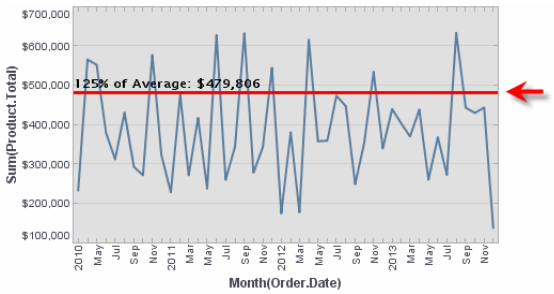
Options

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)

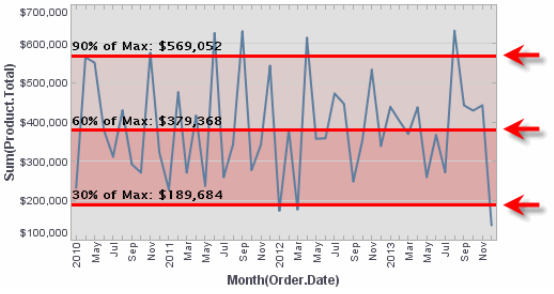
```
var options =
{fillAbove:[225,225,225],fillBelow:[225,225,225],label:['{1
}:
```

```
{0,number,$#,##0}'],lineColor:'red',lineStyle:Chart.THICK_LINE,labelFont:'Lucida Sans Unicode-BOLD-11',labelColor:'black',percentageAggregate:'Average'}
addPercentageTarget('Sum(Product.Total)','null',125,options)
```



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_LINE,labelFont:'Lucida Sans Unicode-BOLD-11',labelColor:'black',percentageAggregate:'Maximum'}
addPercentageTarget('Sum(Product.Total)',[0xDDAAAA,0xDDCCCC],[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

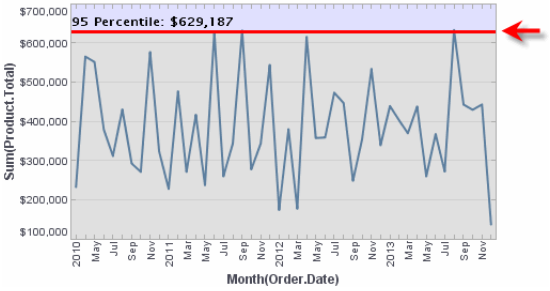
options Name-value pairs of options (see below, e.g., {fillAbove:0xff0000, fillBelow:0x00ff00})

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

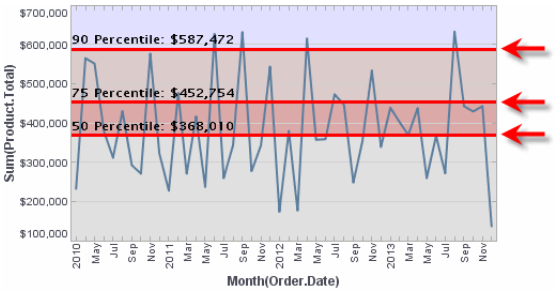
Example (Single Percentile Line)

```
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})

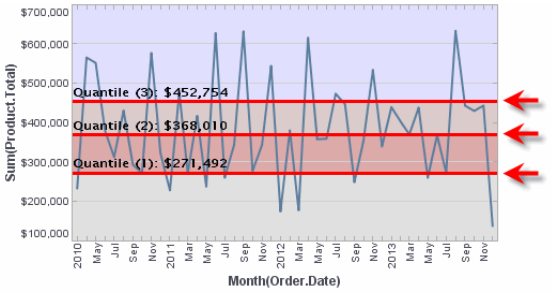
Options

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

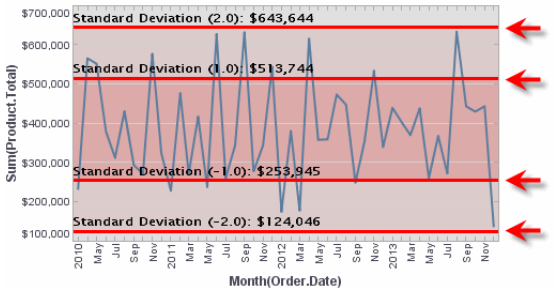
measure Name of measure on which to draw standard deviation lines
colors Color (see *valueColor*) or array of colors to fill areas between standard deviation lines
values Array of multipliers for which to draw standard deviation lines
options Name-value pairs of options (see below, e.g., {fillAbove:0xff0000, 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})

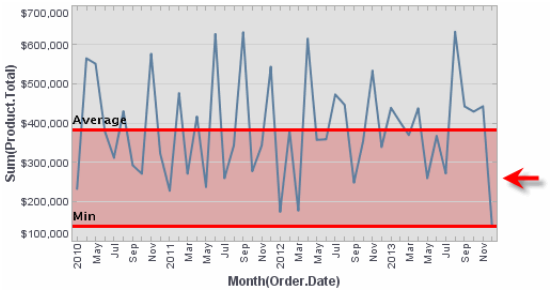
Options

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

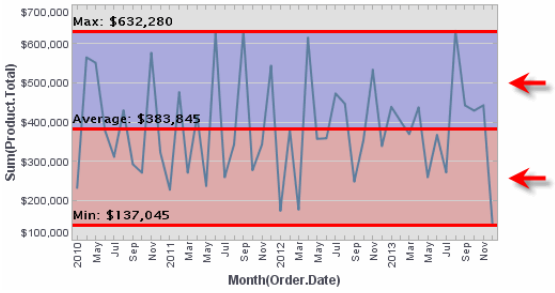
Example (One Target Band)

```
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

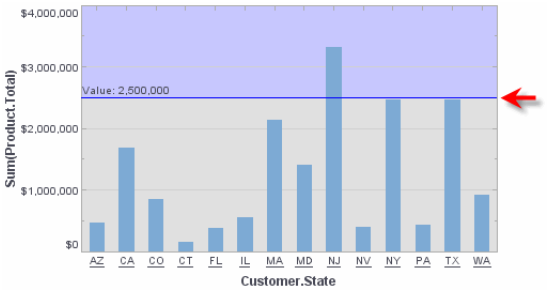
measure Name of measure on which to draw target line
colors Color (see *valueColor*) or array of colors to fill area between target lines
values Array of values and/or valid aggregation methods ('max','min','avg','med') at which to draw the lines
options Name-value pairs of options (see below, e.g., {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

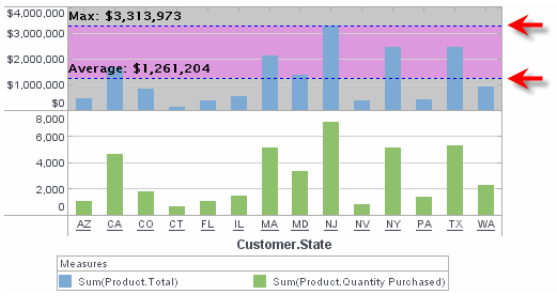
Example (One Target Line)

```
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

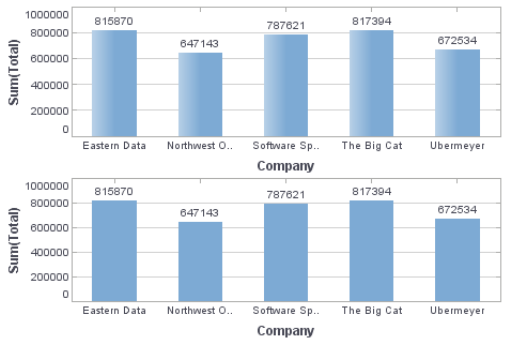
Enables a 3d visual effect for chart elements.

Type

`Boolean`

Example

```
Chart1.applyEffect = true
Chart2.applyEffect = false
```



PLOTALPHA

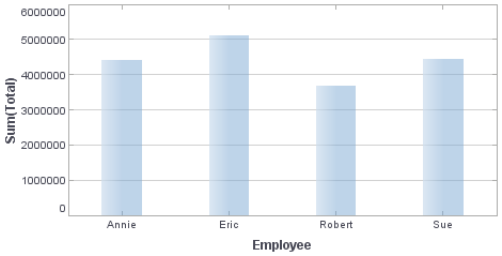
Sets the transparency of chart elements (bars, lines, points, etc.). A value of 0 indicates complete transparency (i.e., chart elements not visible), and a value of 1 indicates complete opacity.

Type

`Number` `Number in range [0,1]`

Example

```
Chart1.plotAlpha = .5;
```



PIEEXPLODED

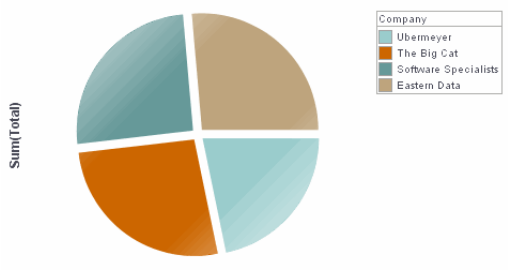
Separates the slices of a pie chart by a small gap.

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.

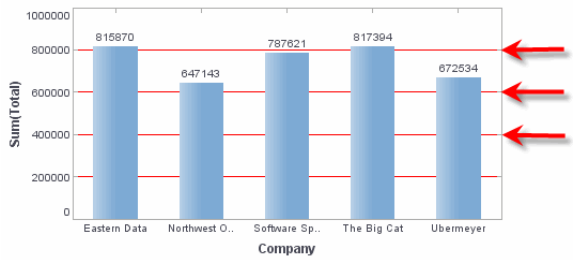
Type

```
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

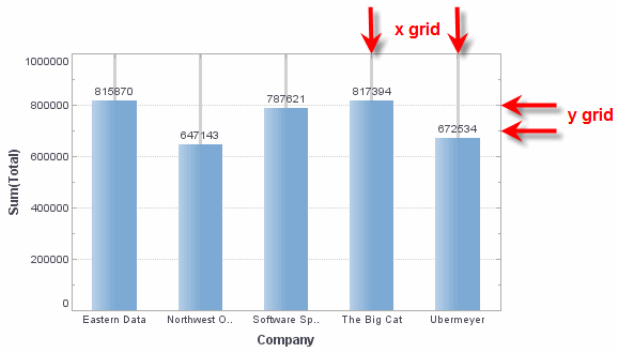
Sets the X-axis grid style. A grid line is drawn at each axis tick.

Type

Chart Constants constant
(see *Line Styles*)

Example

```
Chart1.yGridStyle=Chart.DOT_LINE
Chart1.xGridStyle=Chart.THICK_LINE
```



YGRIDSTYLE

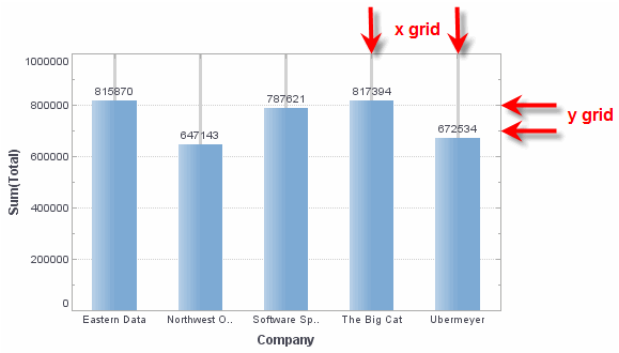
Sets the Y-axis grid style. A grid line is drawn at each axis tick.

Type

Chart Constants constant
(see *Line Styles*)

Example

```
Chart1.yGridStyle=Chart.DOT_LINE
Chart1.xGridStyle=Chart.THICK_LINE
```



XBANDCOLOR

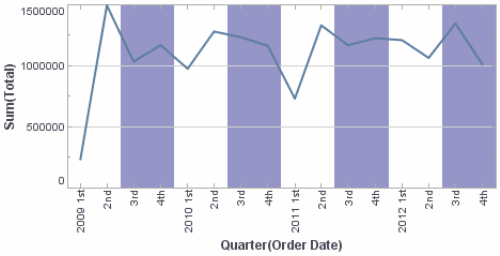
Displays bands of the specified color on a categorical or timescale X-axis.

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

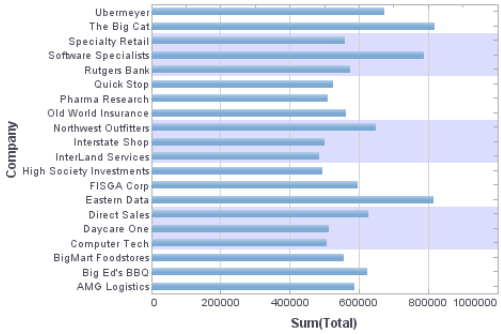
Displays bands of the specified color on a categorical or timescale Y-axis.

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

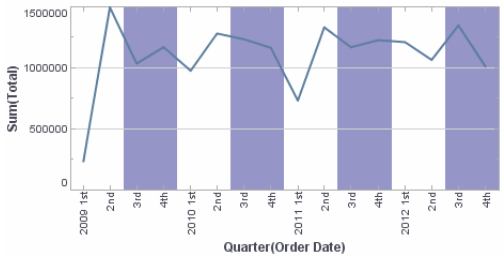
Specifies the width of the bands on a categorical or timescale X-axis. The band size is an integer indicating the number of category groups (for a categorical axis) or the number of date groups (for a timescale axis) to be spanned by each band.

Type

```
integer              number of groups spanned by band
```

Example

```
Chart1.xBandSize = 2;
```



See Also

xBandColor, to set the band color.

YBANDSIZE

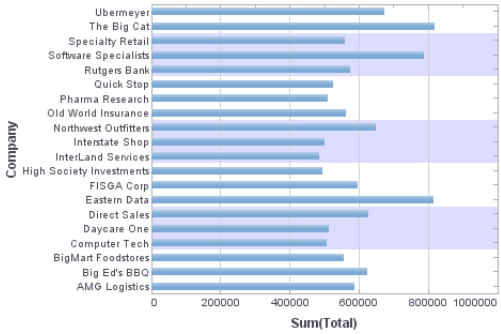
Specifies the width of the bands on a categorical or timescale Y-axis. The band size is an integer indicating the number of category groups (for a categorical axis) or the number of date groups (for a timescale axis) to be spanned by each band.

Type

```
integer              number of groups spanned by band
```

Example

```
Chart1.yBandSize = 3;
```



See Also

yBandColor, to set the band color.

QUADRANTCOLOR

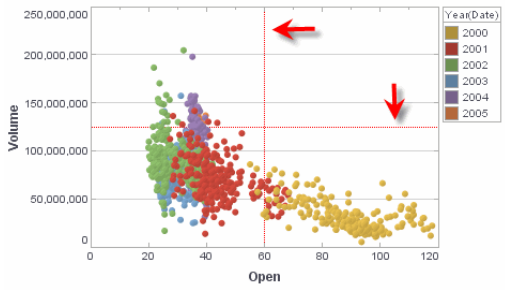
Sets the color of the quadrant lines.

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.

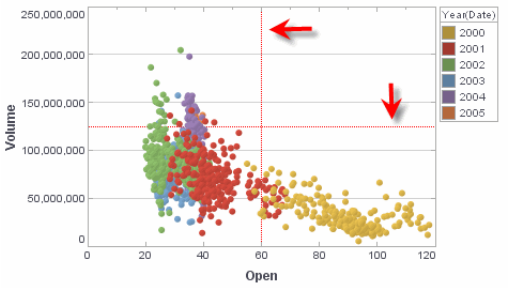
Type

```
Chart Constants constant
(see Line Styles)
```

Example

```
Chart1.quadrantColor=[255,0,0];
```

```
Chart1.quadrantStyle=Chart.DOT_LINE;
```



DIAGONALCOLOR

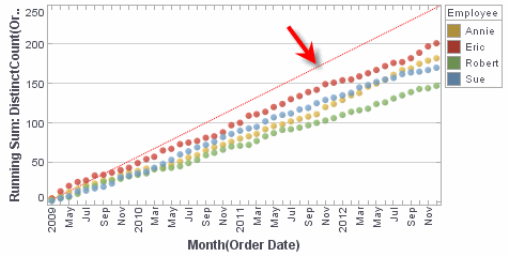
Sets the color of the diagonal line.

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

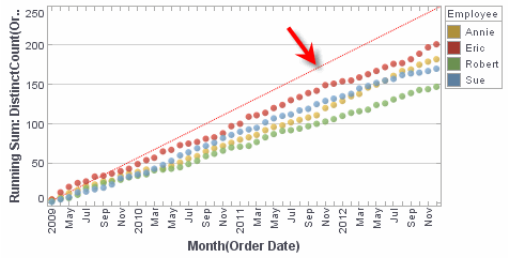
Sets the style of the diagonal line.

Type

Chart Constants constant
(see *Line Styles*)

Example

```
Chart1.diagonalColor=[255,0,0];
Chart1.diagonalStyle=Chart.DOT_LINE;
```



LEGENDPOSITION

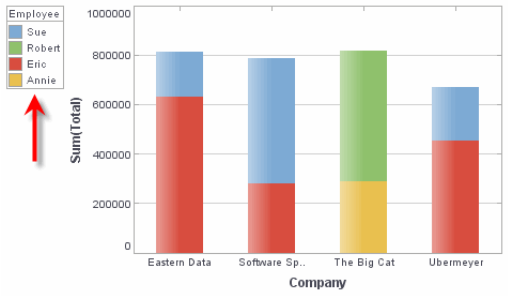
Sets the legend position relative to the chart. Set the property to `Chart.NONE` to hide the legend.

Type

Chart Constants constant
(see *Legend Positions*)

Example

```
Chart1.legendPosition = Chart.LEFT;
```



LEGENDBORDER

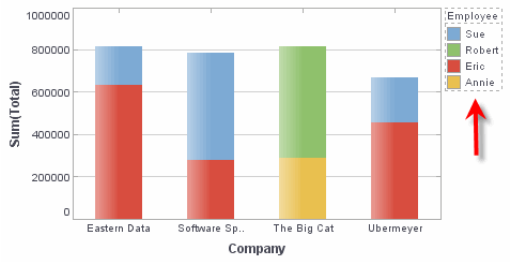
Sets the legend border line style. Set the property to `Chart.NONE` to hide the legend border.

Type

Chart Constants constant
(see *Line Styles*)

Example

```
Chart1.legendBorder = Chart.DASH_LINE;
```



LEGENDBORDERCOLOR

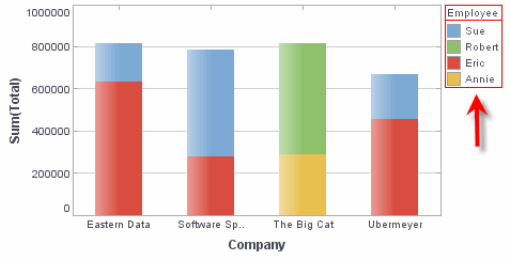
Sets the color of the legend border.

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.

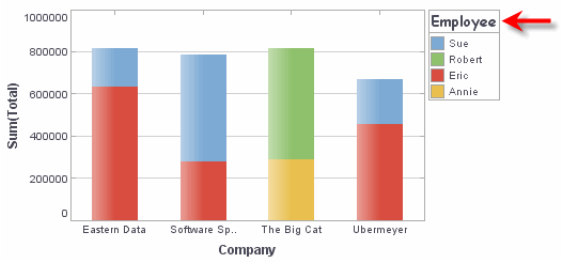
Sets the font for the legend title.

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

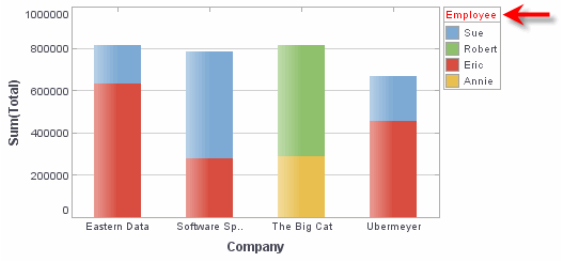
Sets the color of the legend title text.

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.

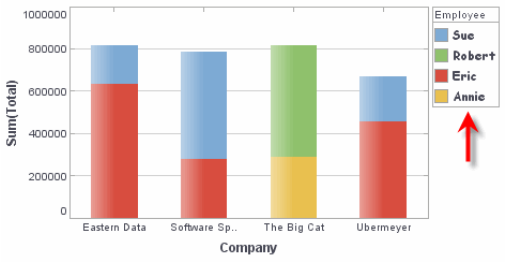
Sets the font for labels within a color-key legend.

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.

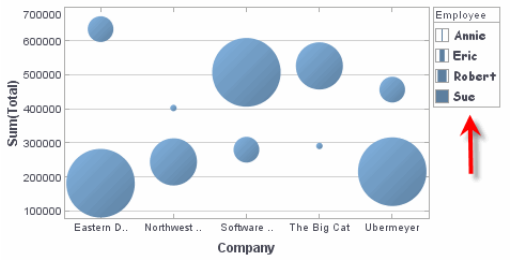
Sets the font for labels within a size-key legend.

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

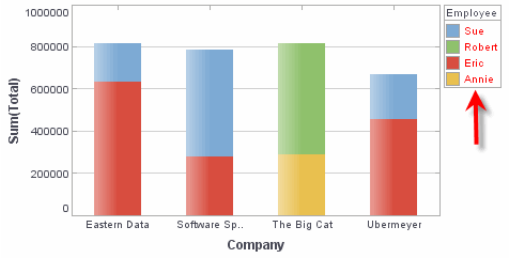
Sets the color for labels within a color-key legend.

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

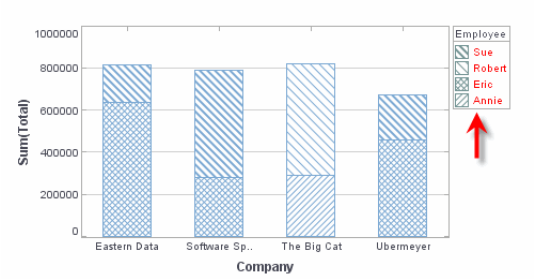
Sets the color for labels within a shape-key legend.

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

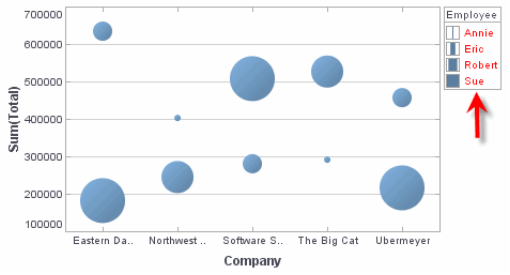
Sets the color for labels within a size-key legend.

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

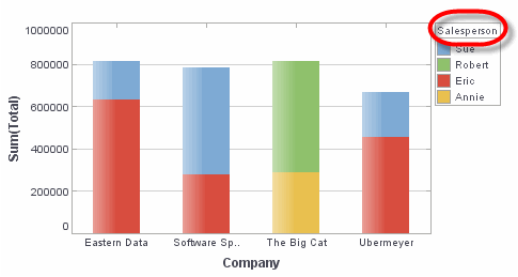
Sets the legend title for a color-key legend.

Type

String

Example

```
Chart1.colorLegend.title = 'Salesperson';
```



SHAPELEGEND.TITLE

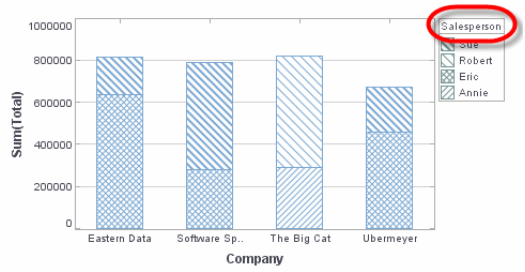
Sets the legend title for a shape-key legend.

Type

`String`

Example

```
Chart1.shapeLegend.title = 'Salesperson';
```



SIZELEGEND.TITLE

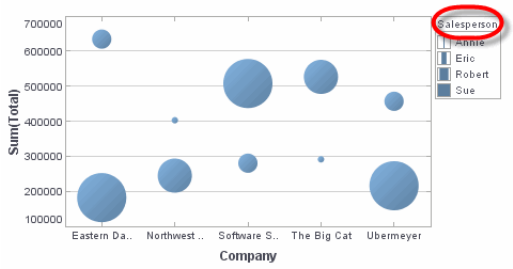
Sets the legend title for a size-key legend.

Type

`String`

Example

```
Chart1.sizeLegend.title = 'Salesperson';
```



COLORLEGEND.TITLEVISIBLE

Sets the visibility of the legend title bar for a color-key legend.

Type

`Boolean`

Example

```
Chart1.colorLegend.titleVisible = false;
```



SHAPELEGEND.TITLEVISIBLE

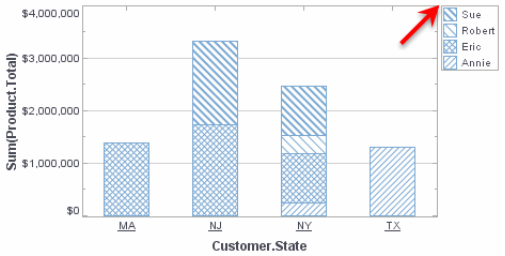
Sets the visibility of the legend title bar for a shape-key legend.

Type

`Boolean`

Example

```
Chart1.shapeLegend.titleVisible = false;
```



SIZELEGEND.TITLEVISIBLE

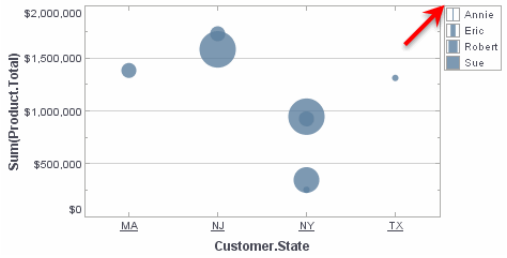
Sets the visibility of the legend title bar for a size-key legend.

Type

`Boolean`

Example

```
Chart1.sizeLegend.titleVisible = false;
```



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

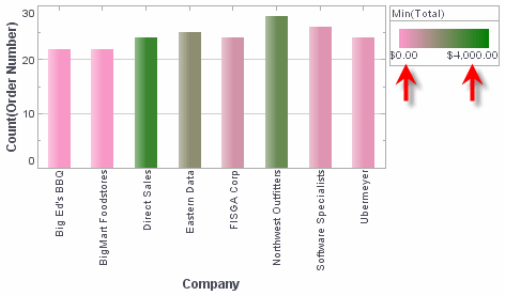
For the Date, Decimal, and Message formats, optionally specify the format and mask as items in an array. See *Format Types* in *Advanced Report Design* and *Format Tab* in *Advanced Dashboard Design* for more information about the masks.

Type

```
Constant (format) or Array ([format, mask])
```

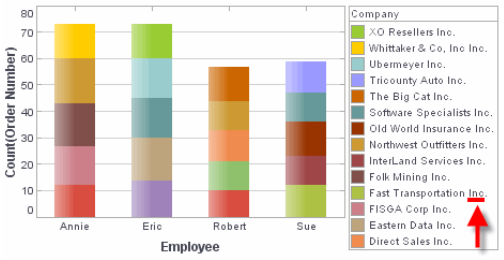
Example (Number Format)

```
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

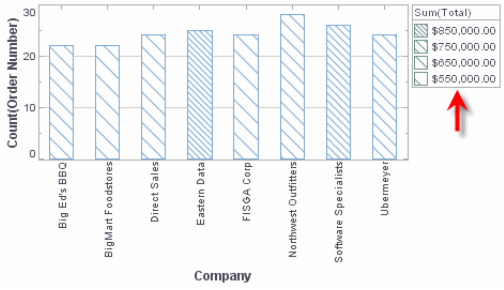
For the Date, Decimal, and Message formats, optionally specify the format and mask as items in an array. See *Format Types* in *Advanced Report Design* and *Format Tab* in *Advanced Dashboard Design* for more information about the masks.

Type

```
Constant (format) or Array ([format, mask])
```

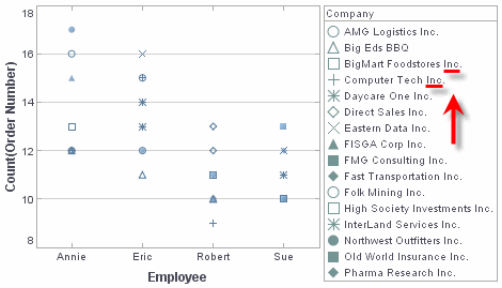
Example (Number Format)

```
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

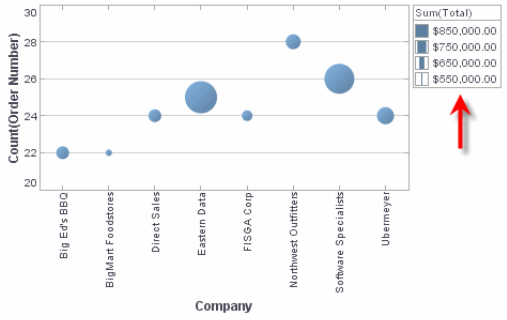
For the Date, Decimal, and Message formats, optionally specify the format and mask as items in an array. See *Format Types* in *Advanced Report Design* and *Format Tab* in *Advanced Dashboard Design* for more information about the masks.

Type

```
Constant (format) or Array ([format, mask])
```

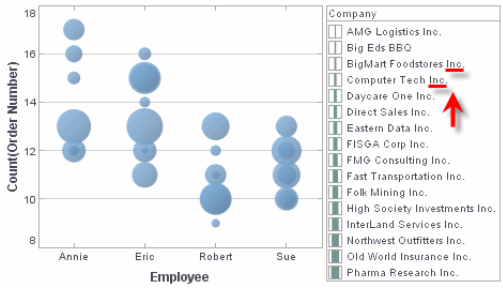
Example (Number Format)

```
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)

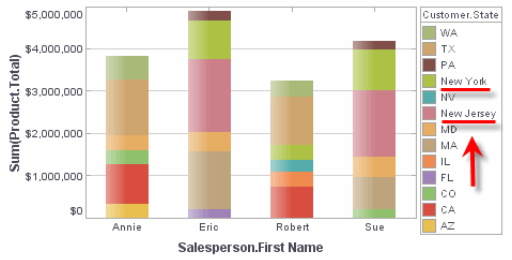
Replace the specified label in the color legend with an alias.

Parameters

value	Value to be replaced
alias	Replacement string

Example

```
Chart1.setLabelAliasOfColorLegend('NJ','New Jersey');
Chart1.setLabelAliasOfColorLegend('NY','New York');
```



SETLABELALIASOFSHAPELEGEND(VALUE, ALIAS)

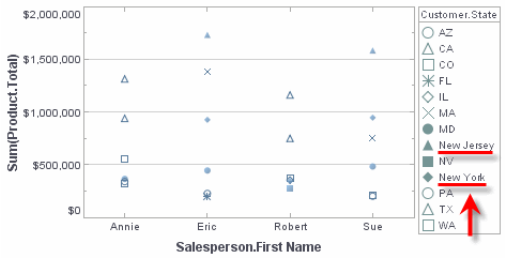
Replace the specified label in the shape legend with an alias.

Parameters

value	Value to be replaced
alias	Replacement string

Example

```
Chart1.setLabelAliasOfShapeLegend('NJ','New Jersey');
Chart1.setLabelAliasOfShapeLegend('NY','New York');
```



SETLABELALIASOFSIZELEGEND(VALUE, ALIAS)

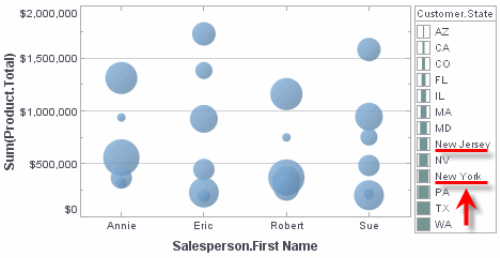
Replace the specified label in the size legend with an alias.

Parameters

value	Value to be replaced
alias	Replacement string

Example

```
Chart1.setLabelAliasOfSizeLegend('NJ','New Jersey');
Chart1.setLabelAliasOfSizeLegend('NY','New York');
```

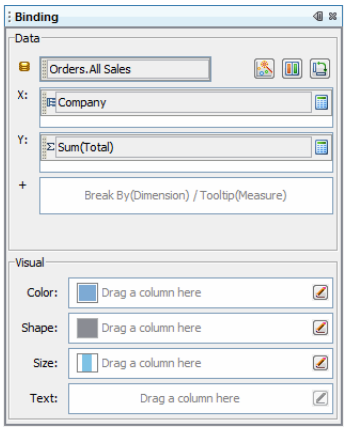


CR.1.6 Chart bindingInfo Property

```
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

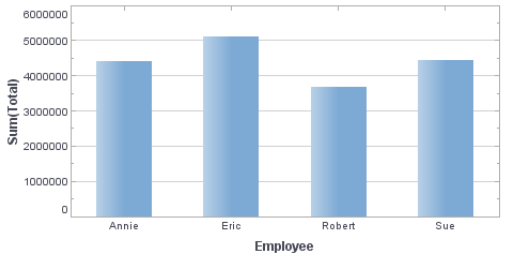
Specifies the query fields to be displayed on the X-axis.

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

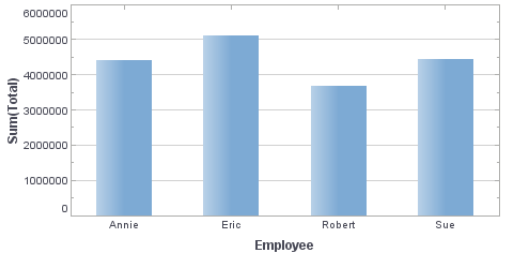
Specifies the query fields to be displayed on the Y-axis.

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

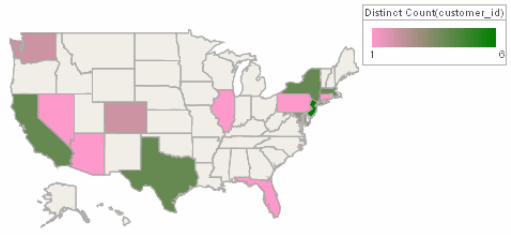
Specifies the query fields to be used as geographic fields. These fields will be bound to the 'Geographic' region of the Chart Editor.

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.

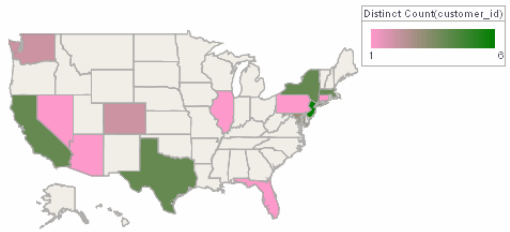
Parameters

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
```

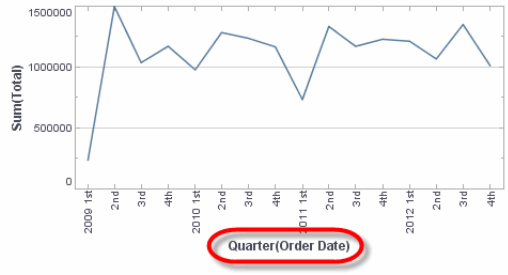
The distinction between grouping levels such as `Chart.MONTH_INTERVAL` and `Chart.MONTH_OF_YEAR_PART` is as follows:

- `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)

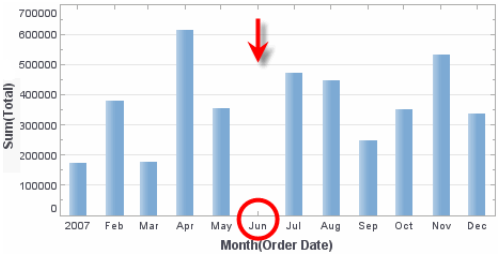
Specifies that gaps in Date data should be retained. For example, if the data in the chart is grouped by month, and there is no data for the month of June, the 'timeSeries' property ensures that the month of June is still retained in the chart.

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,[SORTCOL])

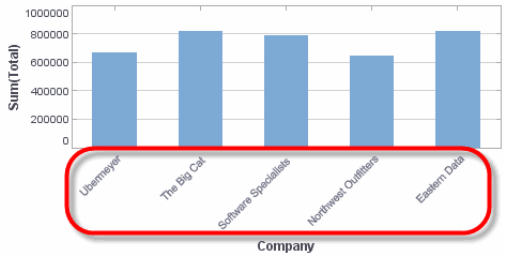
Specifies the sorting order for a specified dimension. To sort a dimension by measure value, set `order` to `SORT_VALUE_ASC` or `SORT_VALUE_DESC`, and use the `sortCol` string to supply the column and aggregation by which to sort.

Parameter

col	Name of field on which to apply sorting
order	The sort order, a *Chart Constants* constant: Chart.SORT_NONE Chart.SORT_ASC // ascending Chart.SORT_DESC // descending Chart.SORT_VALUE_ASC // ascending by value Chart.SORT_VALUE_DESC // ascending by value
sortCol	String: Formula for sorting: 'Sum(column_name)' 'Average(column_name)' 'Max(column_name)' 'Min(column_name)' 'Count(column_name)' 'Distinct Count(column_name)' 'First(column_name1,column_name2)' 'Last(column_name1,column_name2)' 'Correlation(column_name1,column_name2)' 'Covariance(column_name1,column_name2)' 'Variance(column_name)' 'Median(column_name)' 'Mode(column_name)' 'Std Deviation(column_name)' 'Variance(Pop)(column_name)' 'Std Deviation(Pop)(column_name)' 'Weighted Average(column_name1,column_name2)'

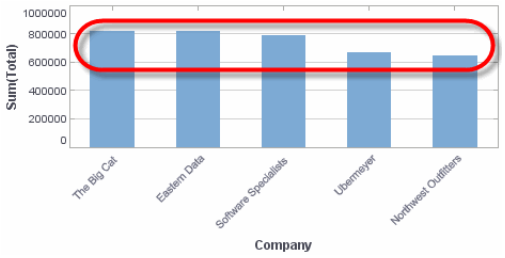
Example

```
Chart1.bindingInfo.setColumnOrder('Company',Chart.SORT_DESC
)
```



Example

```
Chart1.bindingInfo.setColumnOrder('Company',Chart.SORT_VALU
E_DESC,'Sum(Total)');
```



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)

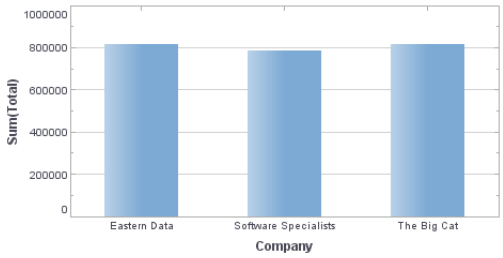
Filters the top or bottom N groups for a specified field based on an aggregate measure.

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

Returns

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)

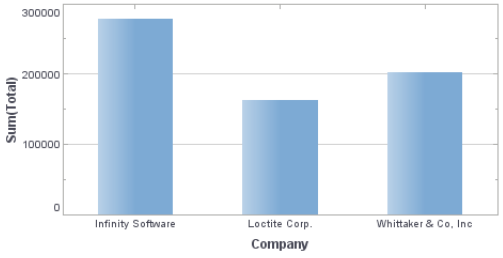
Specifies whether filter should extract the top-N or the bottom-N groups based on an aggregate.

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)

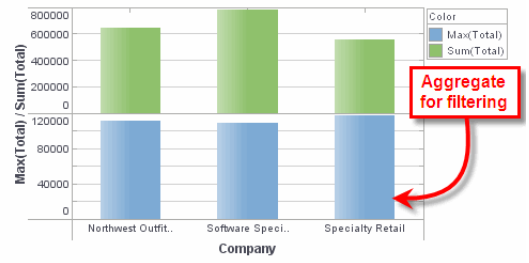
Specifies the aggregate to be used for Top-N/Bottom-N filtering on a given field.

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
```

Returns

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*).

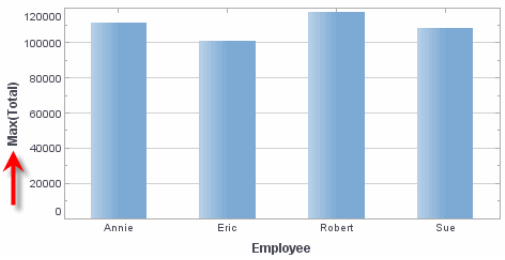
Parameter

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
```

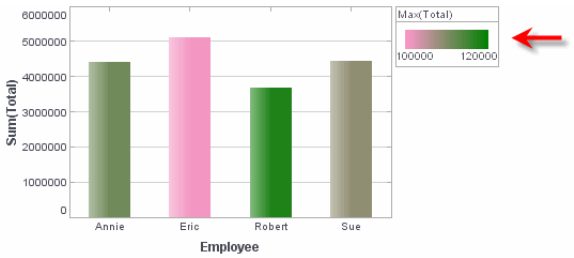
Example (axis-bound field)

```
// 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
```

Returns

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)

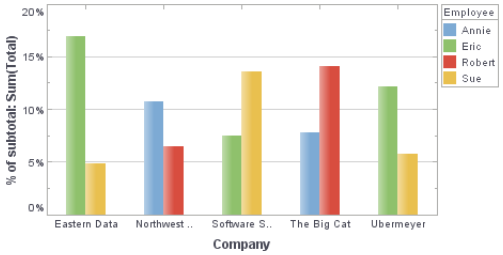
Specifies the basis for computing percentages, grand total or group total.

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

Returns

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)

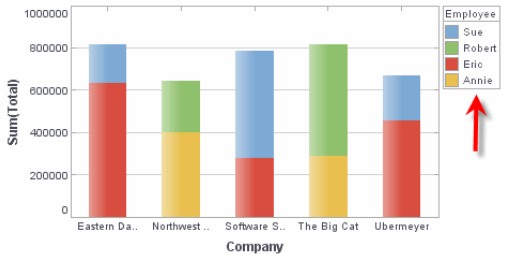
Bind the specified field to a *ColorFrame* (color coding).

Parameter

col Field to be used for color-coding
type *Chart Constants* constant: STRING, DATE, or NUMBER

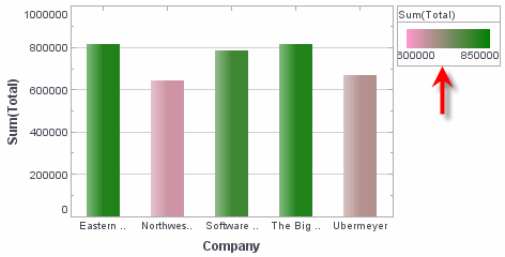
Example (Dimension field)

```
// 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:

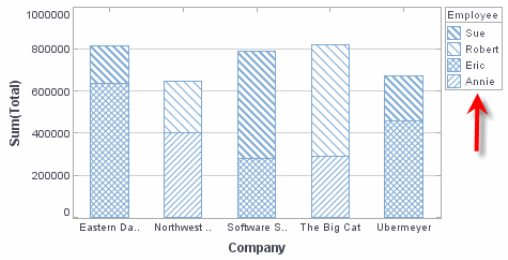
- Bar-type charts apply a *TextureFrame*, which draws different fill patterns. (For dimension binding, the default is the *CategoricalTextureFrame*. For measure binding, the default is *LeftTiltTextureFrame*.)
- Line-type charts apply a *LineFrame*, which draws different line styles. (For dimension binding, 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

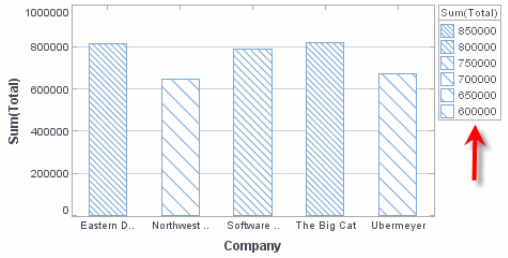
Example (Dimension field)

```
// 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)

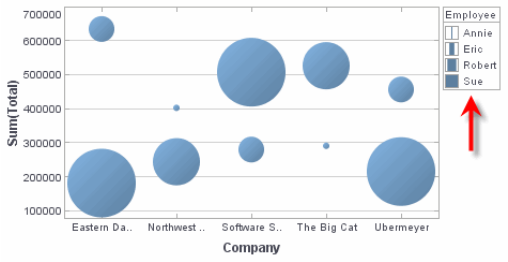
Bind the specified field to a *SizeFrame* (size coding).

Parameter

`col` Field to be used for size-coding
`type` *`Chart Constants`* constant: STRING, DATE, or NUMBER

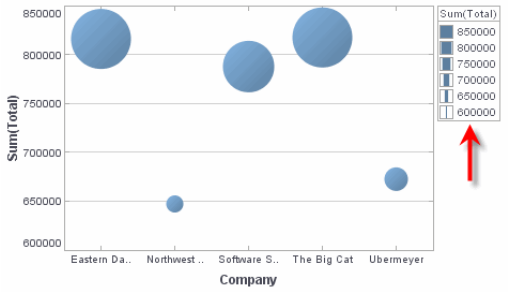
Example (Dimension field)

```
// 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)

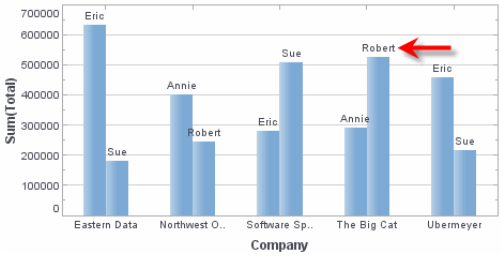
Bind the specified field to a *TextFrame* (text coding).

Parameter

col	Field to be used for text-coding
type	*Chart Constants* constant: STRING, DATE, or NUMBER

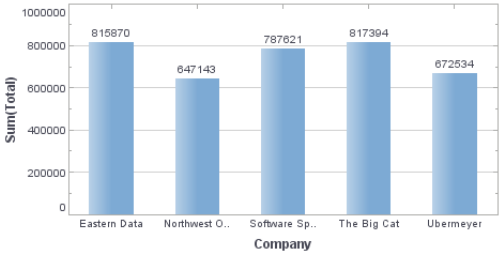
Example (Dimension field)

```
// 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 (under 'Data Source' > 'Other Data Sources') as the data source.

2. Add a new Chart element to the report or Viewsheet.

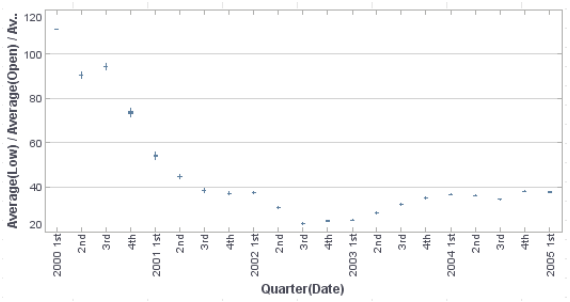
3. Enter the following script in the onLoad or onRefresh handler of the report or Viewsheet.

```
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];

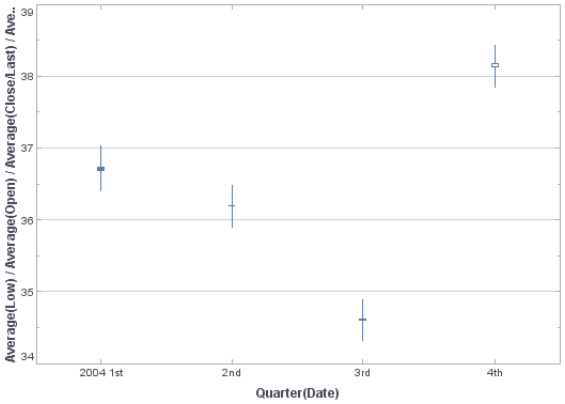
Chart1.bindingInfo.xFields=[['Date',Chart.DATE]];
Chart1.bindingInfo.setGroupOrder('Date',Chart.QUARTER_INTER
VAL);
Chart1.bindingInfo.setCandleBindingField(data_high);
Chart1.bindingInfo.setCandleBindingField(data_low);
Chart1.bindingInfo.setCandleBindingField(data_close);
Chart1.bindingInfo.setCandleBindingField(data_open);
```

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 (under ‘Data Source’ > ‘Other Data Sources’) 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.

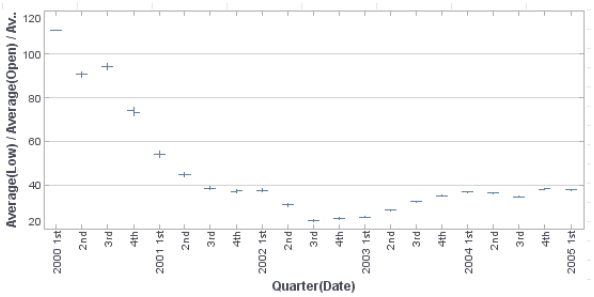
```
Chart1.query='Stock Prices';

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];

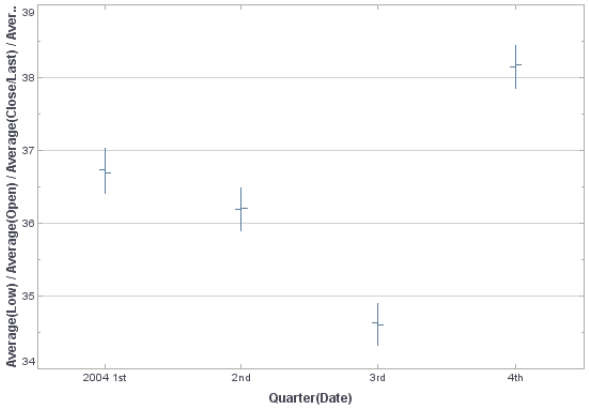
Chart1.bindingInfo.xFields=[['Date',Chart.DATE]];
Chart1.bindingInfo.setGroupOrder('Date',Chart.QUARTER_INTER
VAL);
Chart1.bindingInfo.setStockBindingField(data_high);
Chart1.bindingInfo.setStockBindingField(data_low);
Chart1.bindingInfo.setStockBindingField(data_close);
Chart1.bindingInfo.setStockBindingField(data_open);
```

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

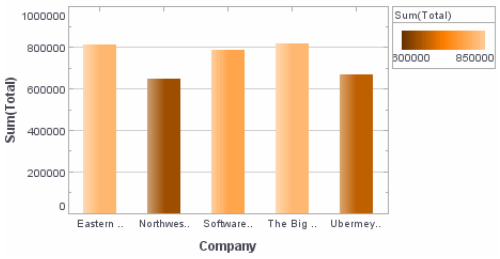
Specifies the aesthetic color treatment for the chart elements. A *ColorFrame* can color-code chart elements by value, or simply specify a static color scheme. The default for dimensions is *CategoricalColorFrame*, and the default for measures is *GradientColorFrame*.

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

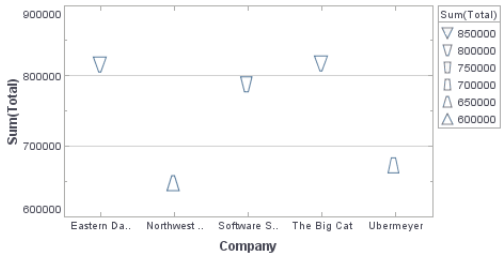
Specifies the aesthetic treatment for shape-coded data on point-type charts. A *ShapeFrame* can shape-code chart elements by value, or simply specify a static shape style.

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

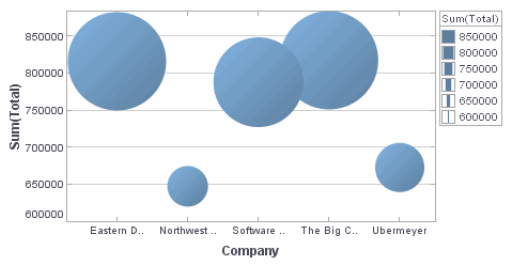
Specifies the size of graphical elements. A *SizeFrame* can size-code chart elements by value, or simply specify a static size. The default for dimensions is *CategoricalSizeFrame*. The default for measures is *LinearSizeFrame*.

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

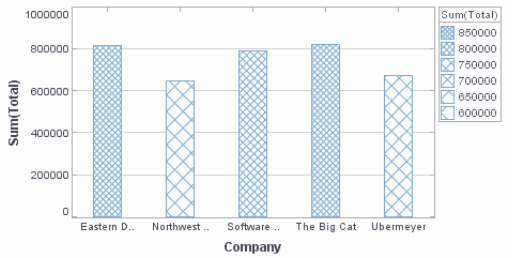
Specifies the aesthetic treatment (fill pattern) for the shape-coded data on a bar-type chart. A *TextureFrame* can texture-code chart elements by value, or simply specify a static texture style.

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

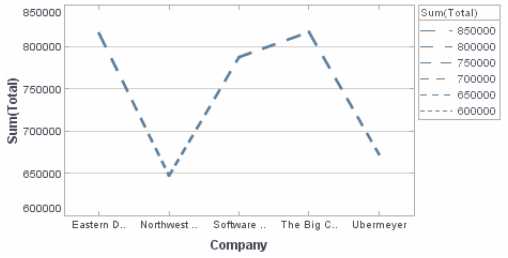
Specifies the aesthetic treatment (line style) for the shape-coded data on a line-type chart. A *LineFrame* can texture-code chart elements by value, or simply specify a static texture style.

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

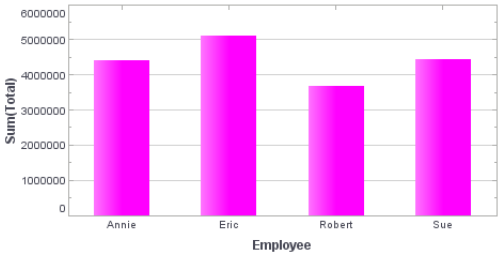
Specifies a *StaticColorFrame* (fixed color) for a particular measure.

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

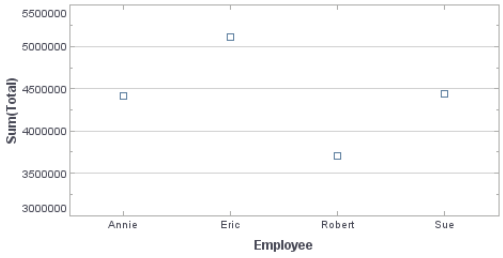
Specifies a *StaticShapeFrame* (fixed shape) for a particular measure on point-type charts.

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

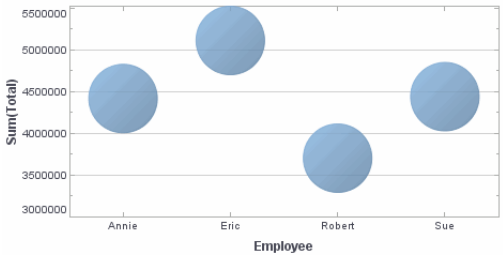
Specifies a *StaticSizeFrame* (fixed size) for all measures.

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

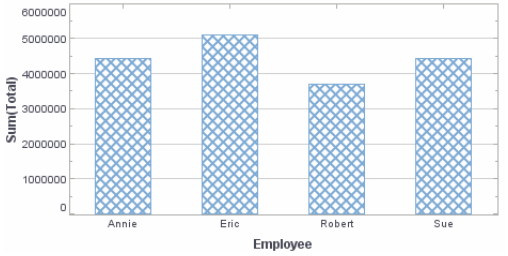
Specifies a *StaticTextureFrame* (fixed pattern) for a particular measure on bar-type charts.

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

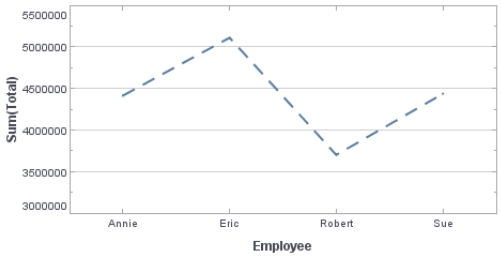
Specifies a *StaticLineFrame* (fixed line style) for a particular measure on line-type charts.

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

Example

The latitude and longitude for code 'USA0113' are defined in inetsoft/graph/geo/data/us-cities.csv.

```
// 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).

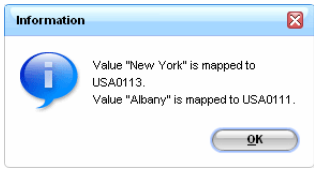
Parameter

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

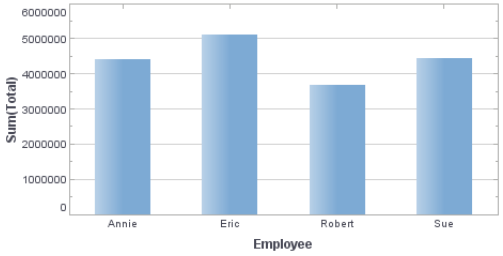
Creating a Chart Using API Functions, for an example using runQuery().
DataSet, for information on setting the chart dataset.
Table, for information on accessing the dataset along with column headers.

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 in 'Orders' > 'Sample Queries'.



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 *Advanced 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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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)

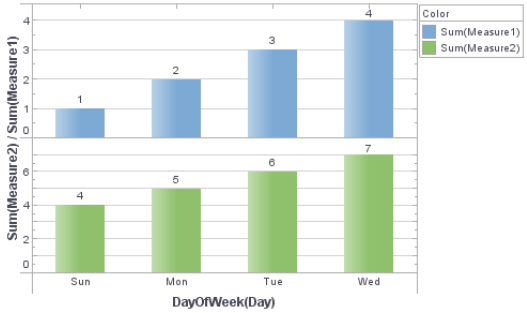
Returns the value in the *DataSet* object specified by the column and row indices. The first column (index 0) contains the X-axis labels.

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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).

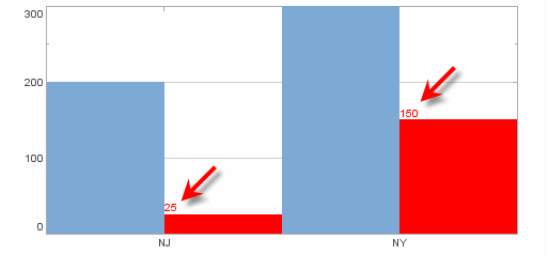
Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced Dashboard Scripting* for more information.

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*.

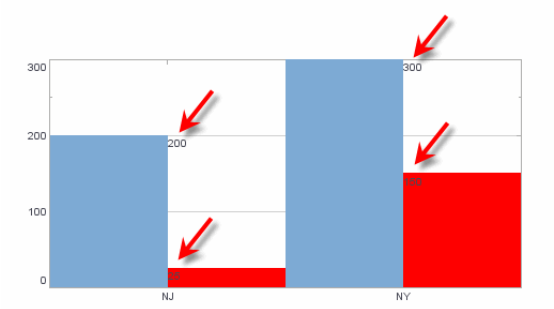
Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced Dashboard Scripting* for more information.

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

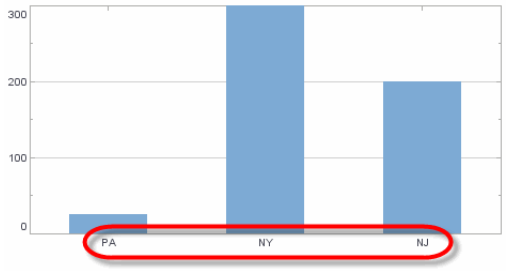
dim	String giving name of the dimension to sort
arr	Array of strings with desired label order: ['label1','label2','label3',...]

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

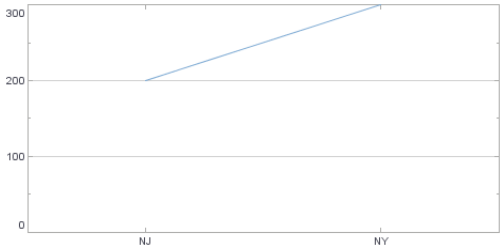
Parameter

elem a *GraphElement* object

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

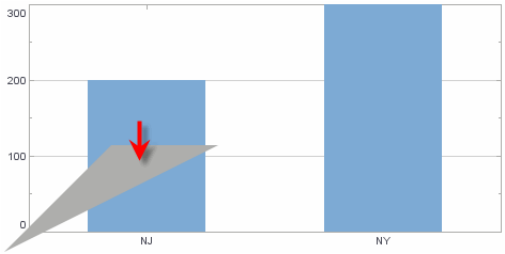
Parameter

form	a *GraphForm* object

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced Dashboard Scripting* for more information.

Example (Report)

Bind a bar-type chart to the sample 'All Sales' query in 'Orders' > 'Sample Queries', 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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced Dashboard Scripting* for more information.

Example (Report)

Bind a bar-type chart to the sample 'All Sales' query in 'Orders' > 'Sample Queries', 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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced Dashboard Scripting* for more information.

Example (Report)

Bind a bar-type chart to the sample 'All Sales' query in 'Orders' > 'Sample Queries', 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.

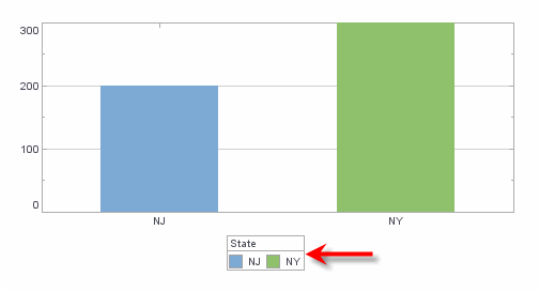
Parameter

```
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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

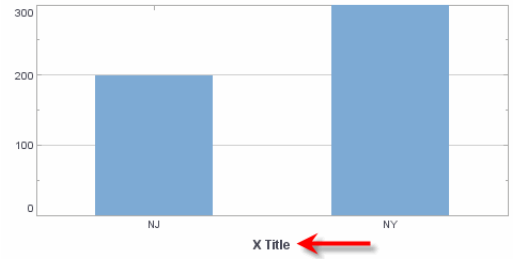
Parameter

spec a *TitleSpec* object

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

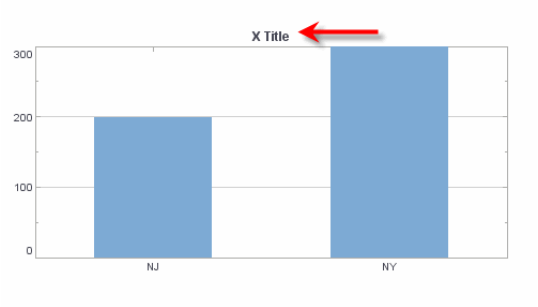
Parameter

spec a *TitleSpec* object

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

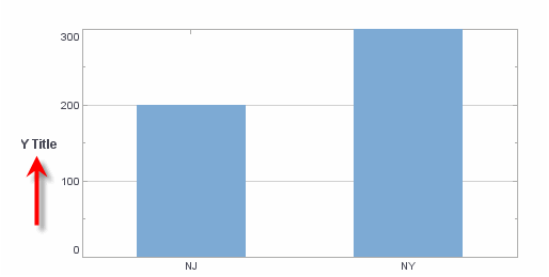
Parameter

spec a *TitleSpec* object

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

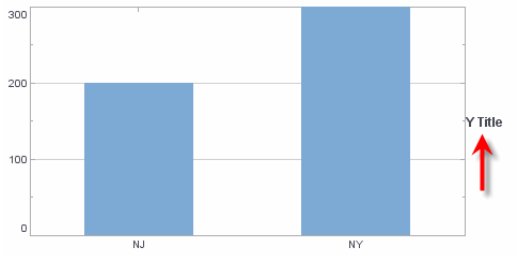
Type

spec a *TitleSpec* object

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

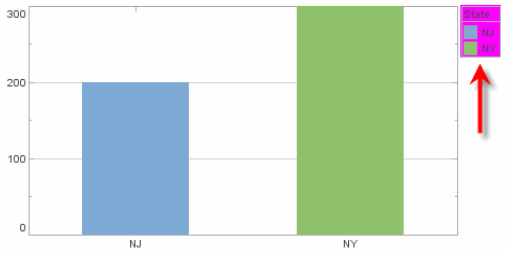
Parameter

value a `java.awt.Color` object

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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

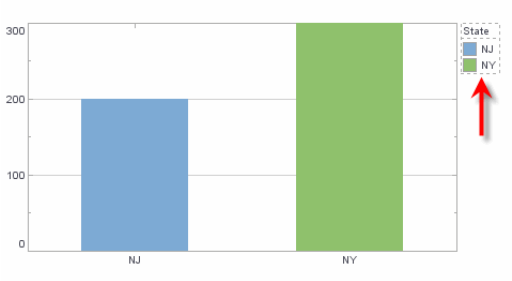
value a *GLine* object

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding*

Component Script in *Advanced 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.

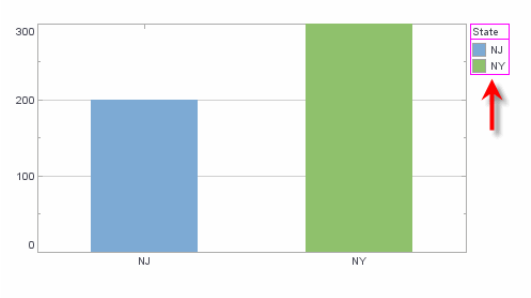
Parameter

value a `java.awt.Color` object

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

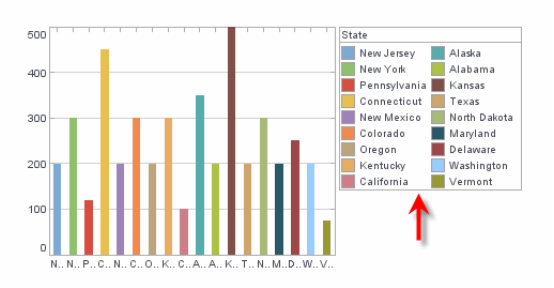
Type

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.)

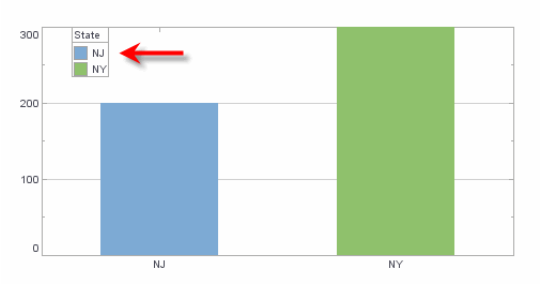
Type

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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`.

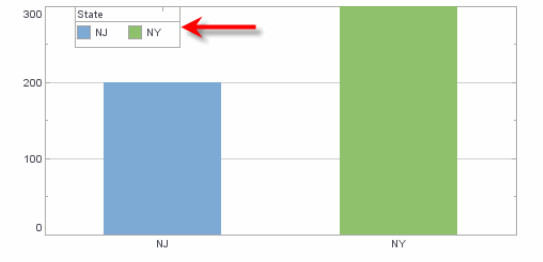
Parameter

value a `java.awt.Dimension` object

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

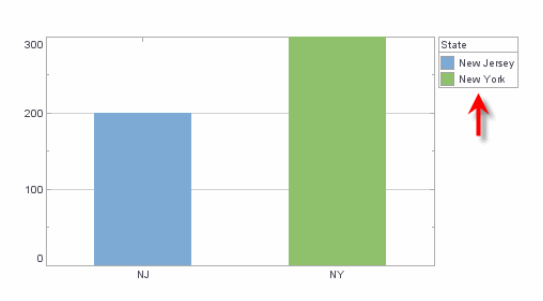
Parameter

frame a *TextFrame* object

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

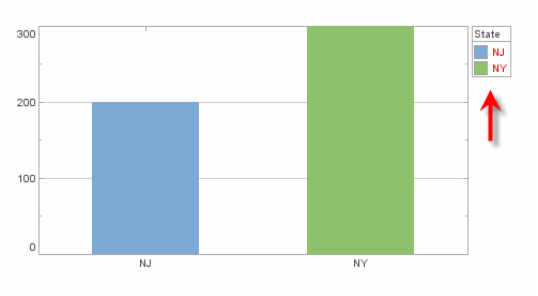
Parameter

spec a *TextSpec* object

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

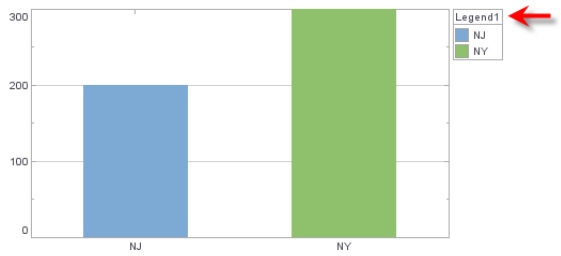
Parameter

value a String containing the title text

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

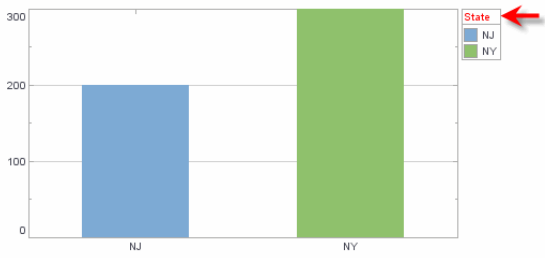
Parameter

spec a *TextSpec* object

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

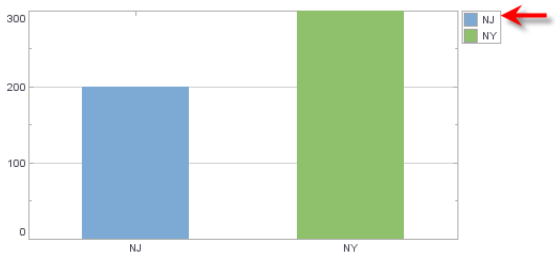
Parameter

boolean	true: show title false: hide title

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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

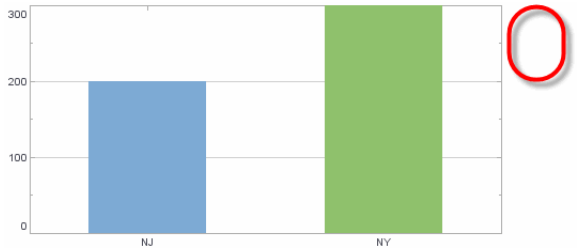
boolean	true: show legend false: hide legend

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

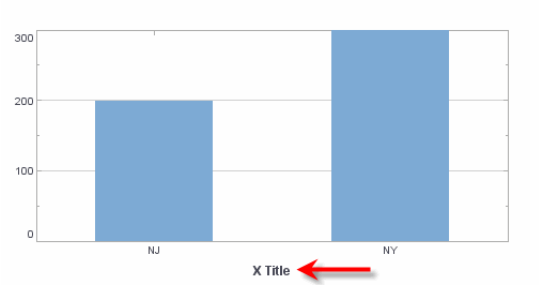
Parameter

value a String containing the label

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

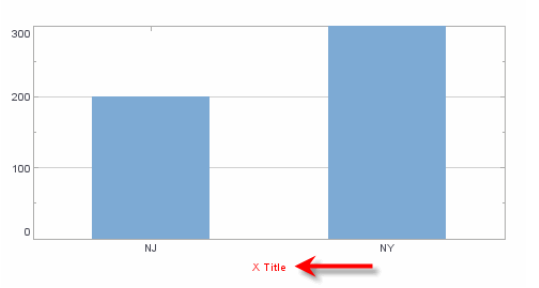
Parameter

spec a *TextSpec* object

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

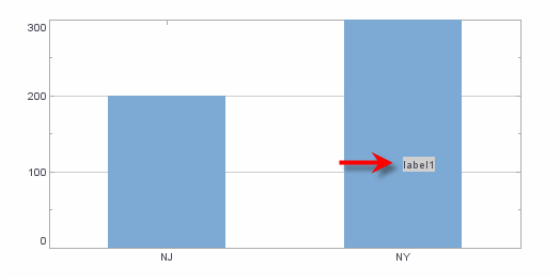
Parameter

value a `java.awt.Color` object

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

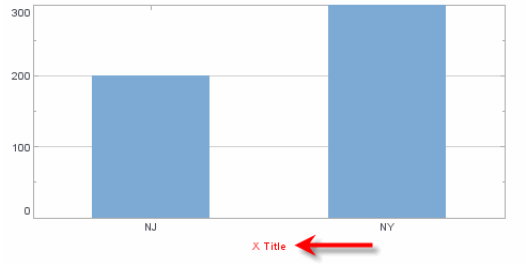
Parameter

value a `java.awt.Color` object

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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'`.

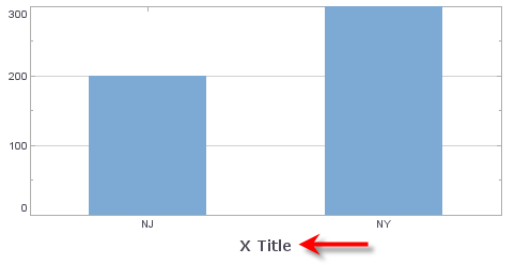
Parameter

value a `java.awt.Font` object

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

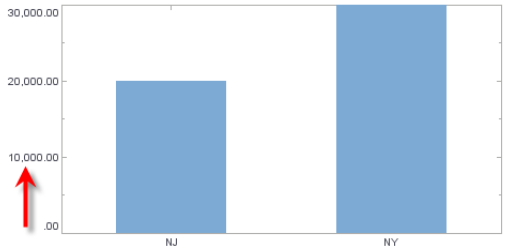
Parameter

format	a `java.text.Format` object

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

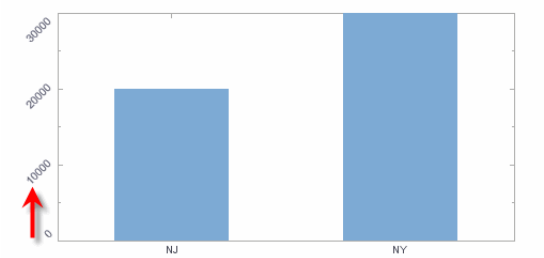
Parameter

value a number of degrees

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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

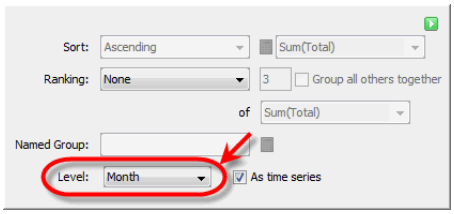
Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced Dashboard Scripting* for more information.

Example (Report or Viewsheet)

For a report, add a blank chart and bind the chart to the sample 'All Sales' query in 'Orders' > 'Sample Queries'. For a Viewsheet, create a new Viewsheet based on the sample 'All Sales' query, and add a blank chart. The follow the steps below:

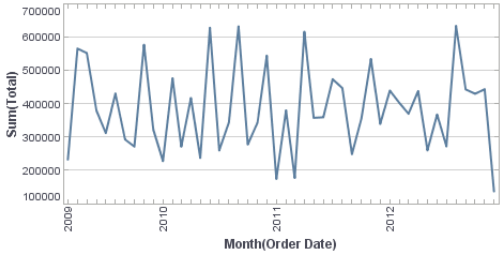
1. Open the Chart Editor for the chart, and drag 'Order Date' to the X-axis and 'Total' to the Y-axis.

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

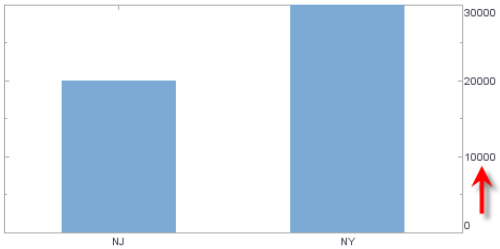
value	AxisSpec.AXIS_SINGLE	(axes on left/bottom)
	AxisSpec.AXIS_SINGLE2	(axis on top/right)

```
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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

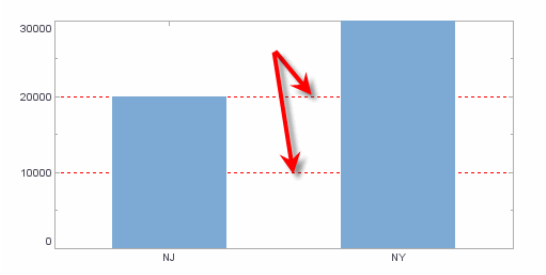
Parameter

value	a `java.awt.Color` object

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

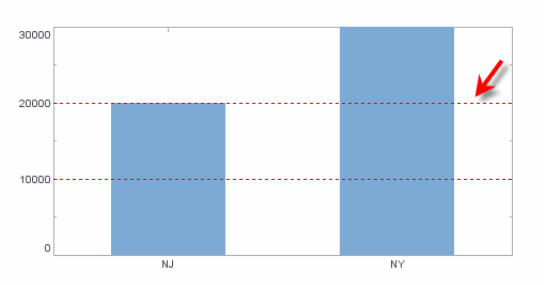
Parameter

boolean	true: grid overlays elements false: elements overlay grid

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

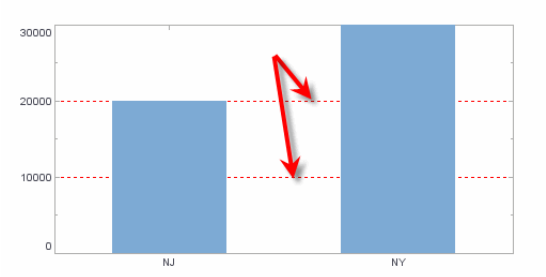
Parameter

value a *GLine* value

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

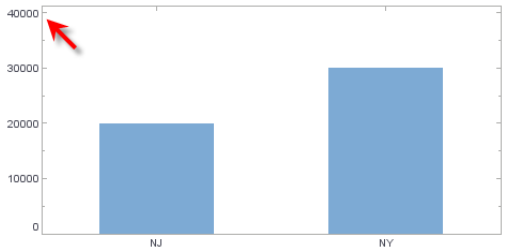
Parameter

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

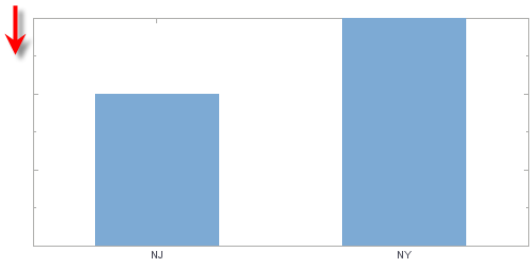
Parameter

boolean	true if visible false if not visible

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

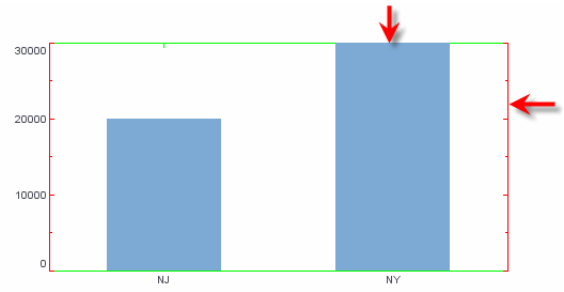
Parameter

value a `java.awt.Color` object

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

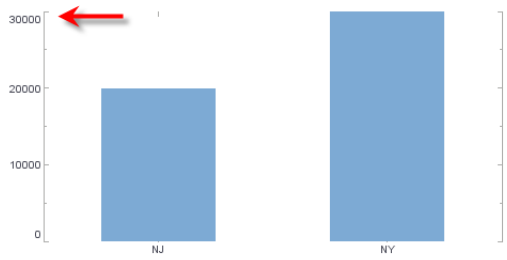
Parameter

boolean	true if visible false if not visible

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

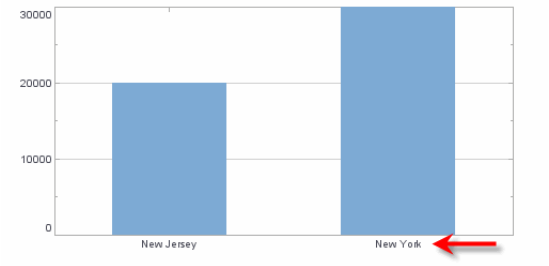
Parameter

frame a *TextFrame* object

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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

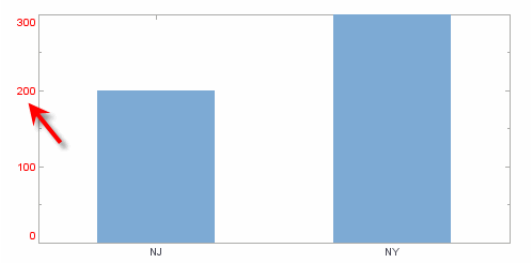
spec a *TextSpec* object

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

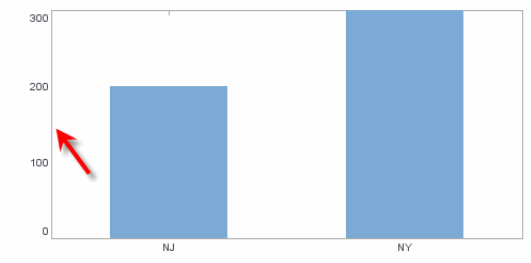
Parameter

boolean	true if visible false if not visible

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

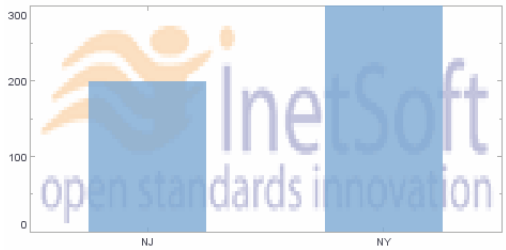
Parameter

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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).

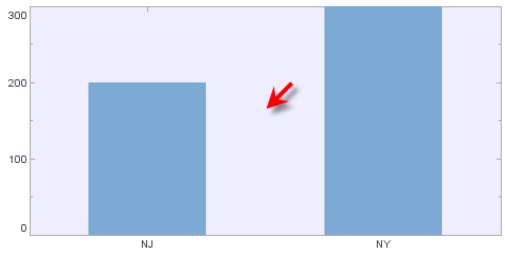
Parameter

value a `java.awt.Color` object

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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");
```

Note: The gray background is set from the 'Chart Properties' dialog box (Style Studio) or the 'Format' dialog box (Viewsheet).

```
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.SETXMAX(VALUE)

Specifies the X-axis value at which to place the right edge of the background image.

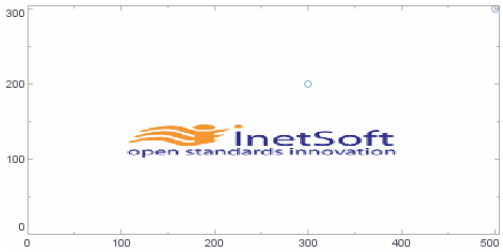
Parameter

value a number

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

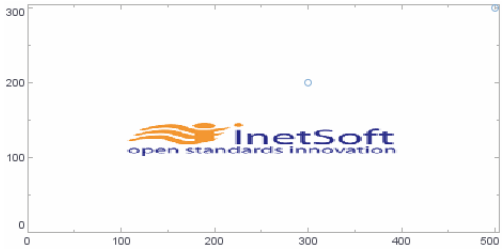
Parameter

value a number

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

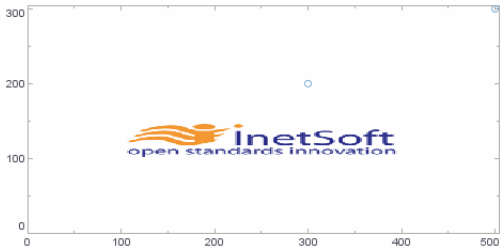
Parameter

value a number

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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

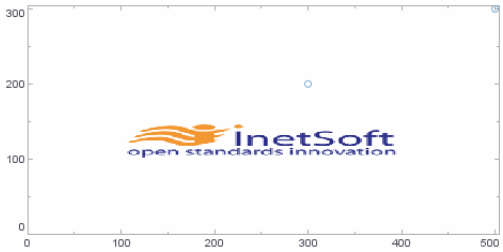
value a number

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

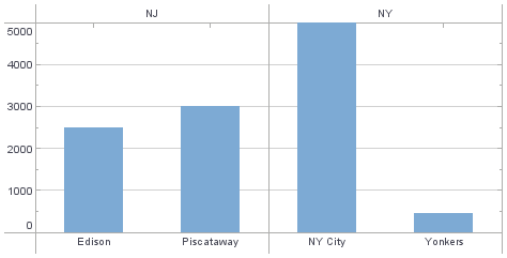
Parameter

field String containing name of dimension

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

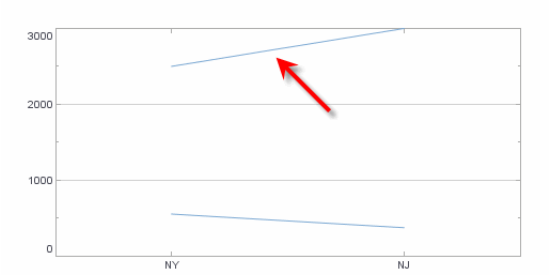
Parameter

field String containing name of variable

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

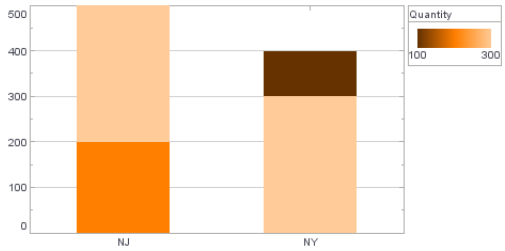
Parameter

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

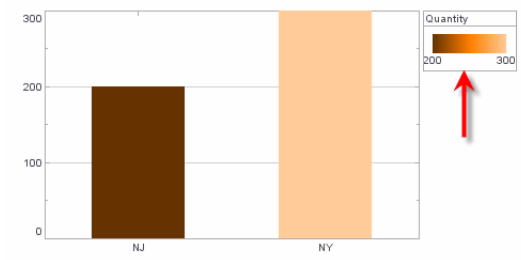
Parameter

frame a *ColorFrame* object

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

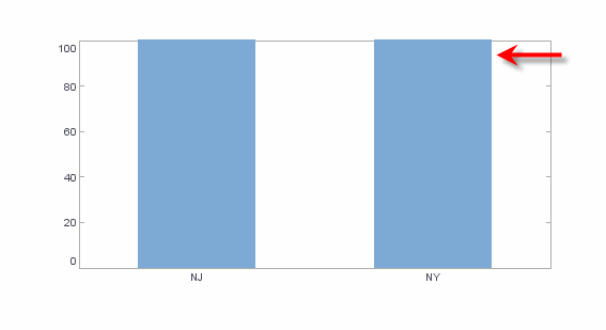
Parameter

boolean	true: resize chart false: do not resize chart (crop elements)

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

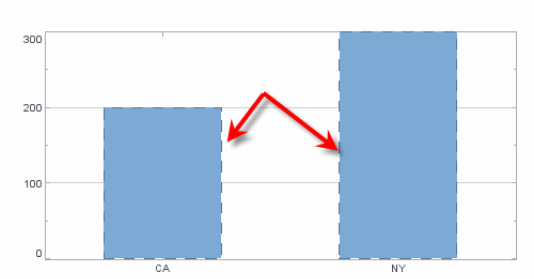
Parameter

frame a *LineFrame* object

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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).

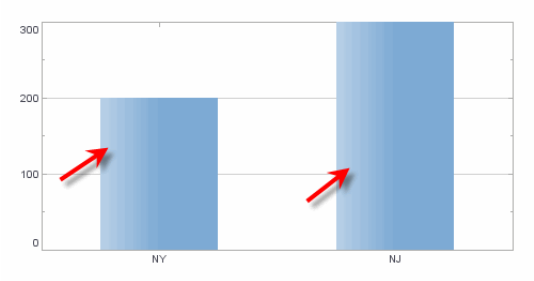
Parameters

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

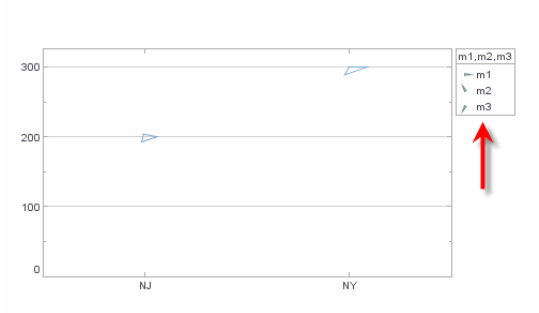
Parameter

frame	a *ShapeFrame* object

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

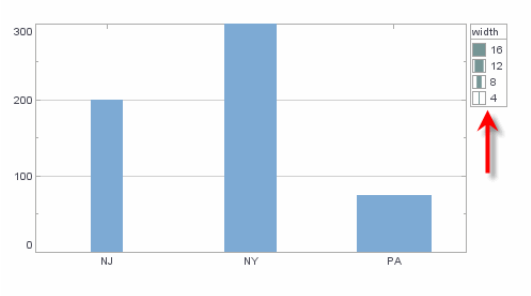
Parameter

frame a *SizeFrame* object

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

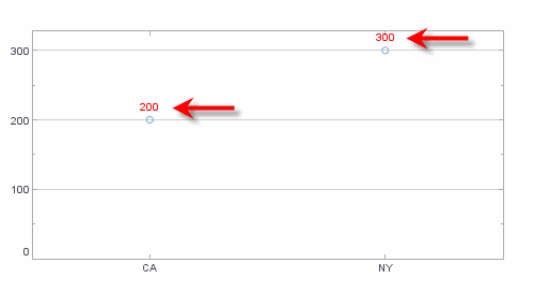
Parameter

spec a *TextSpec* object

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

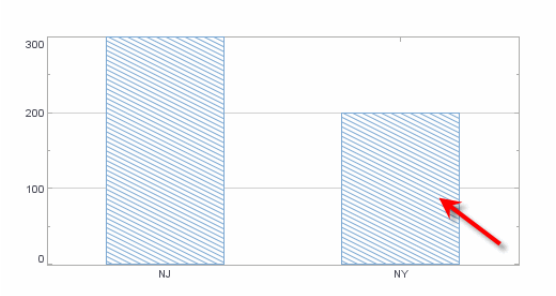
Parameter

frame a *TextureFrame* object

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

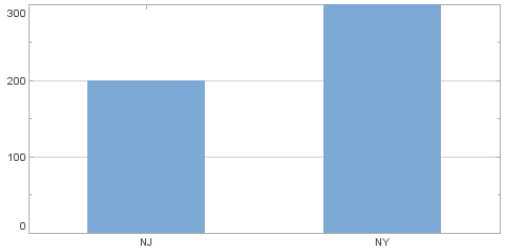
```
var elem = new IntervalElement("State", "Quantity");
```

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

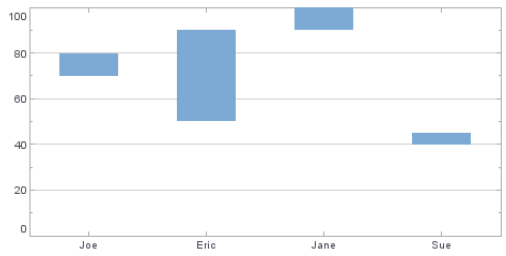
Parameters

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

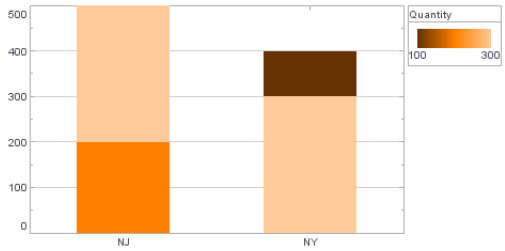
Parameters

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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).

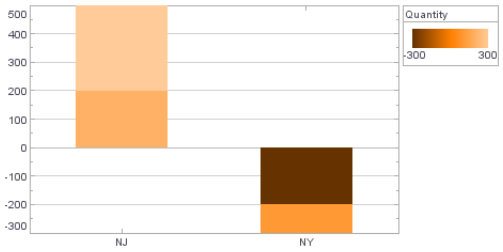
Parameters

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

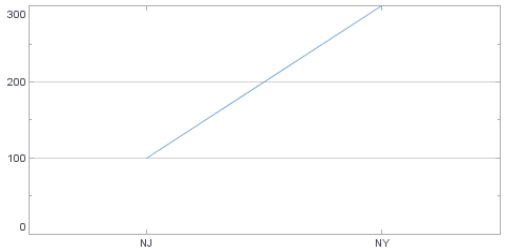
```
var elem = new LineElement("State", "Quantity");
```

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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).

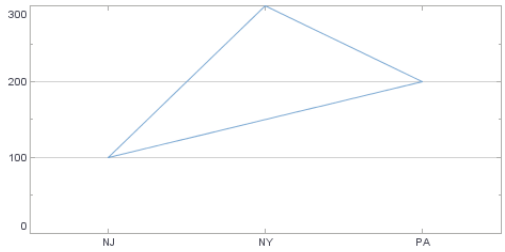
Type

boolean true: close the figure
false: do not close

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

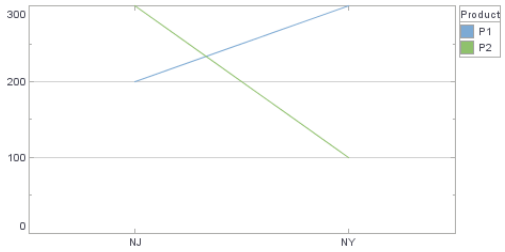
Parameters

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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).

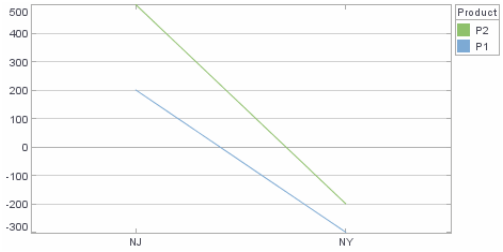
Parameters

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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).

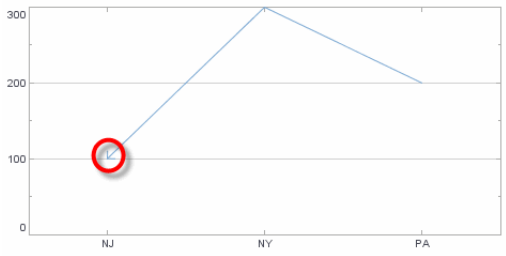
Parameters

boolean	true: draw arrow false: do not draw arrow

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

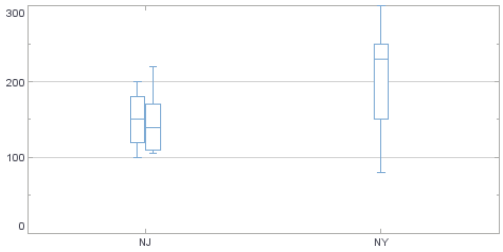
```
var elem = new SchemaElement("State", "Quantity");
```

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

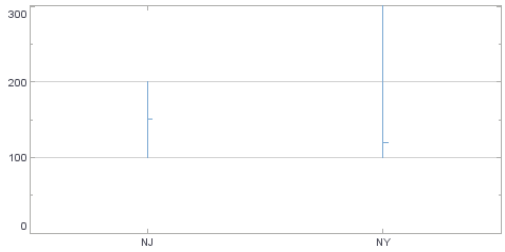
Parameter

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

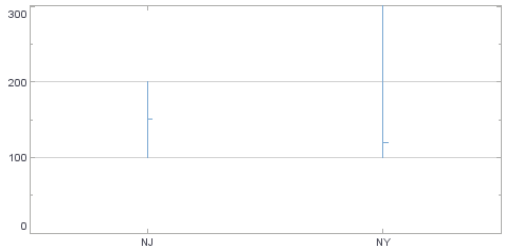
Parameter

col	list of columns for schema binding

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

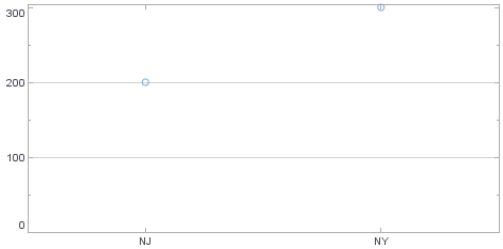
```
var elem = new PointElement("State", "Quantity");
```

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

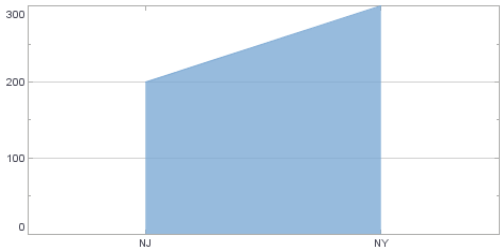
```
var elem = new AreaElement("State", "Quantity");
```

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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

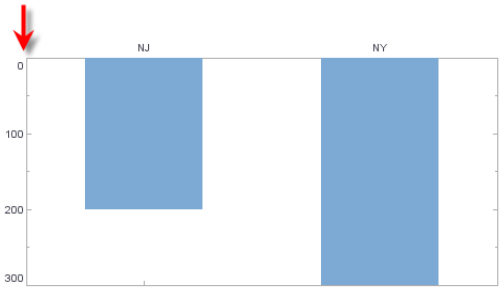
vert	true: reflect about horizontal axis false: reflect about vertical axis

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

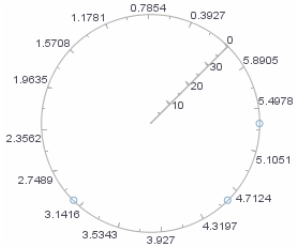
Parameter

value an Angle in degrees

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced Dashboard Scripting* for more information.

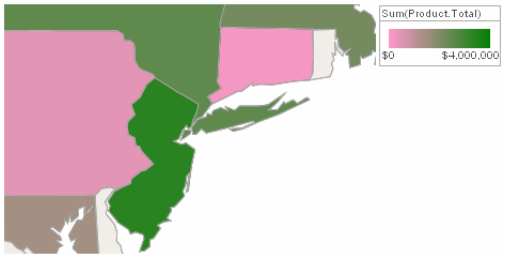
Example (Report or Viewsheet)

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

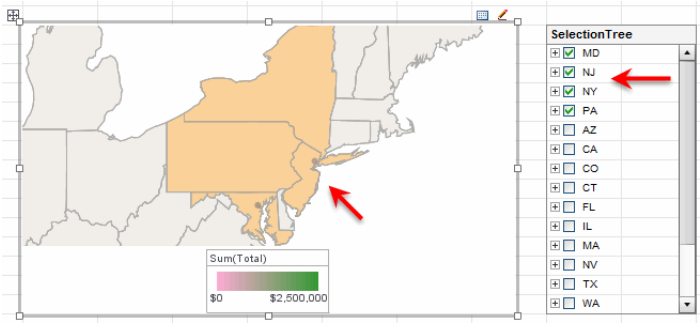
boolean (default)	`true: fit entire map in chart borders`
	`false: fit only data regions in chart borders`

`Note: Script that modifies 'graph' should be placed at the element level. See` *`Editing Script`* `in` *`Advanced Report Scripting`* `and` *`Adding Component Script`* `in` *`Advanced 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:

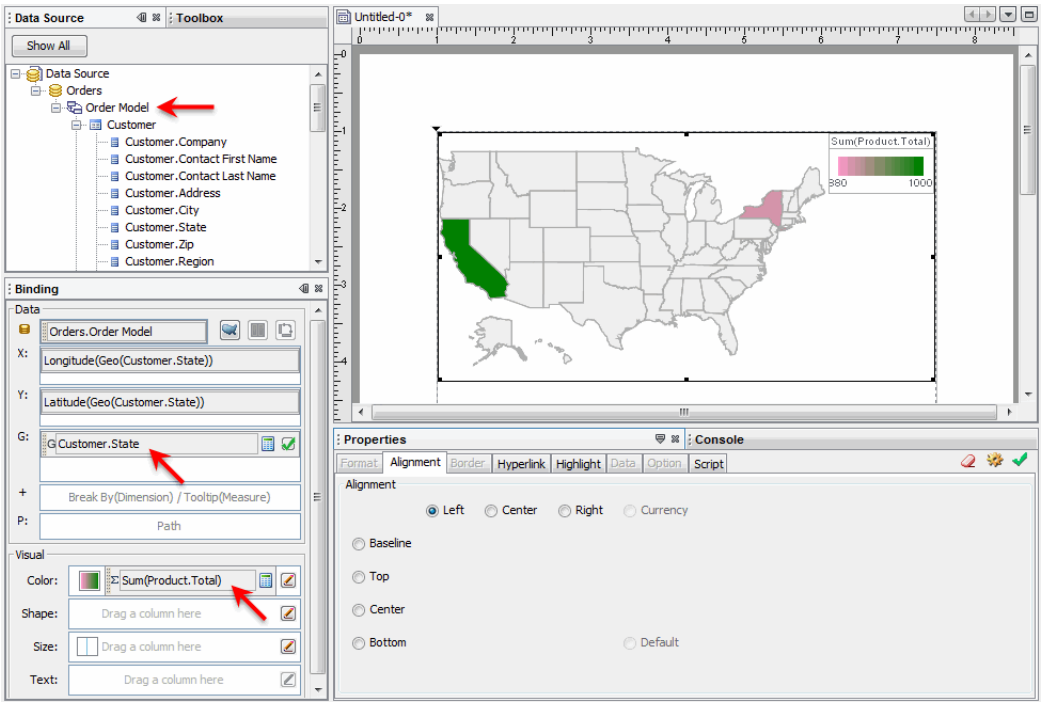
```
var zoomed = SelectionTree1.selectedObjects.length > 0;
graph.getCoordinate().setFullMap(!zoomed);
graph.getElement(0).setInPlot(!zoomed);
```



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.

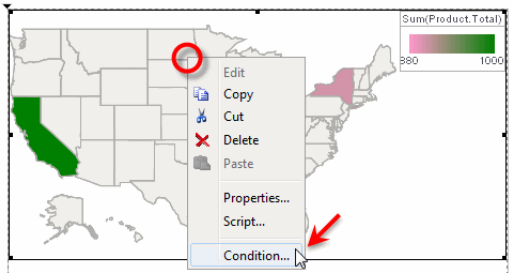
Example (Report)

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.)

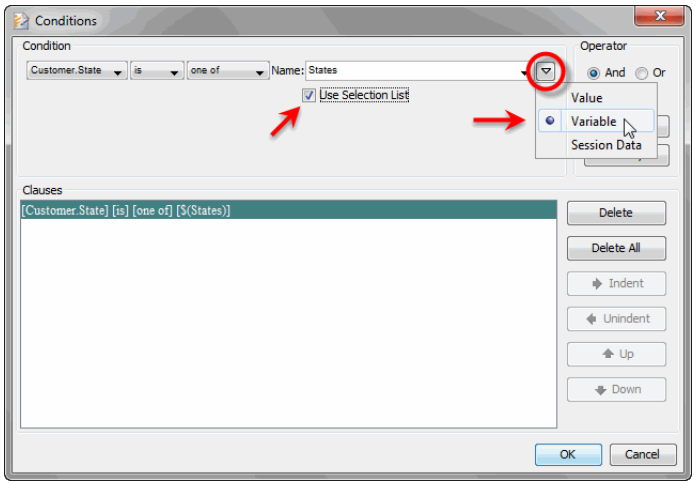


Add a parameterized condition to the map. To do this, follow the steps below:

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'.

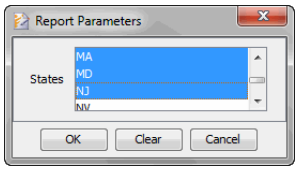
4. Right-click on the map and select 'Script' from the context menu to open the Script Editor. Add the following script:

```
graph.getCoordinate().setFullMap(false);
graph.getElement(0).setInPlot(false);
```

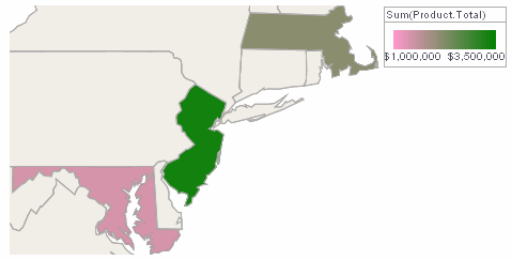
The `setInPlot(false)` function allows the chart to display only the specified portion of the map. Otherwise, the chart will force the entire map to display regardless of the `setFullMap` setting. See *GraphElement.setInPlot(boolean)* for more information.

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.



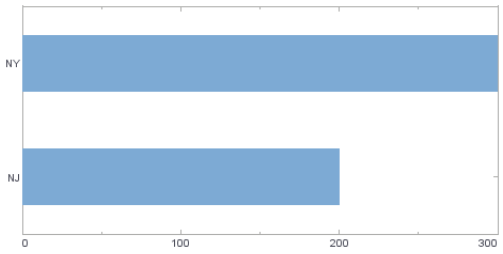
Coordinate.TRANSPOSE()

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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:

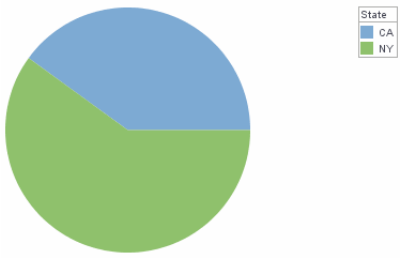
```
var polar = new PolarCoord(rect);
```

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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

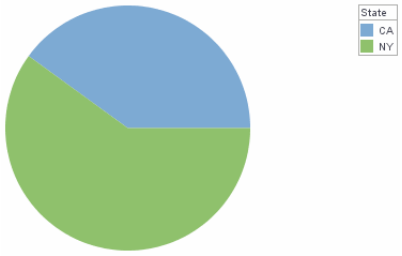
value	PolarCoord.THETA	(use only angle)
	PolarCoord.THETA_RHO	(use angle and radius)
	PolarCoord.RHO	(use only radius)

PolarCoord.PLUS (angle & radius, with hole)

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

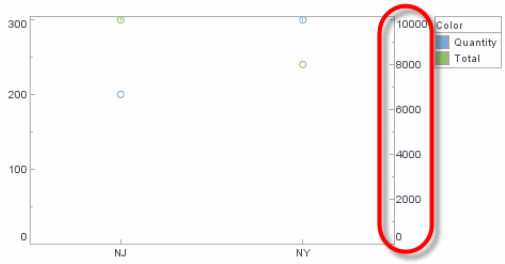
Type

scale a *Scale* object

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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:

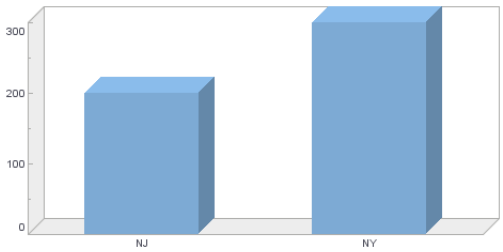
```
var rect = new Rect25Coord(xscale,yscale);
```

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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:

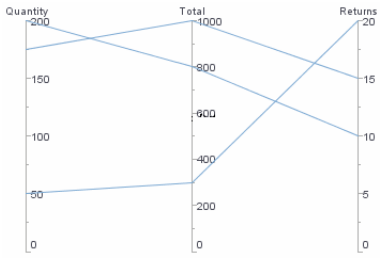
```
var rect = new ParallelCoord(scale1,scale2,...);
```

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

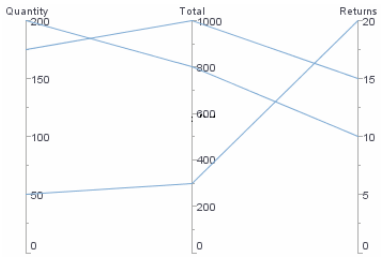
Parameter

scales Array of *Scale* objects

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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:

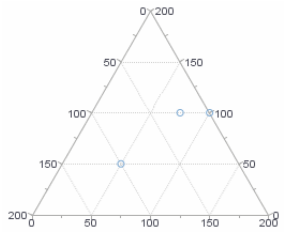
```
var coord = new TriCoord(qscale);
```

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

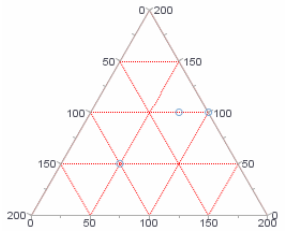
Parameter

scale a *Scale* object

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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:

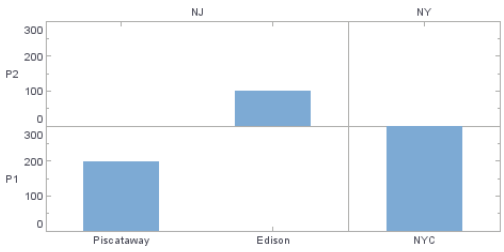
```
var rect = new FacetCoord(outerCoord,innerCoord);
```

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

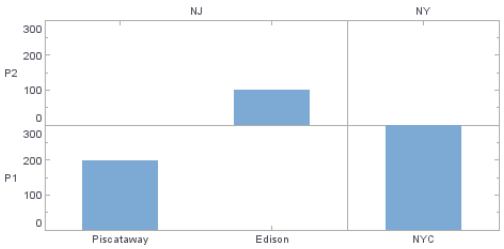
Parameter

coord Array of *RectCoord* objects

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

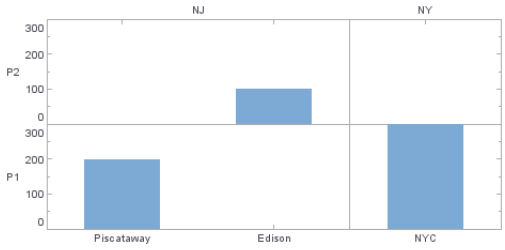
Parameter

coord	a *RectCoord* object

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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

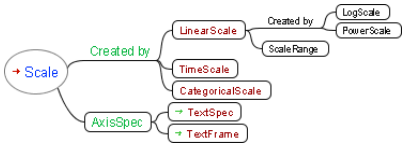
boolean	true: stack vertically (default) false: stack horizontally

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

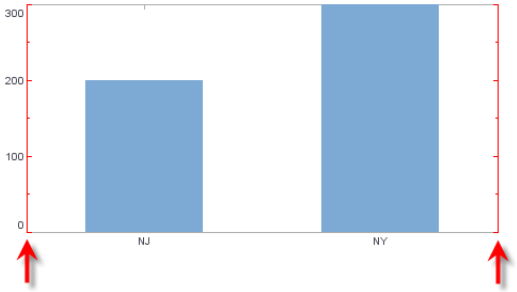
Parameter

spec *AxisSpec*

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

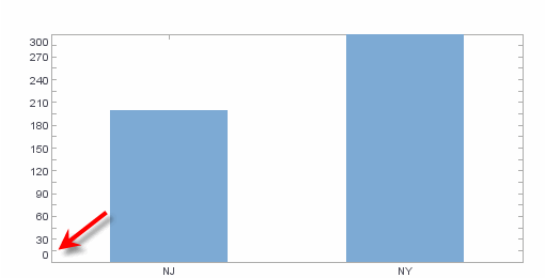
Parameter

value	Number specifying interval between values

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

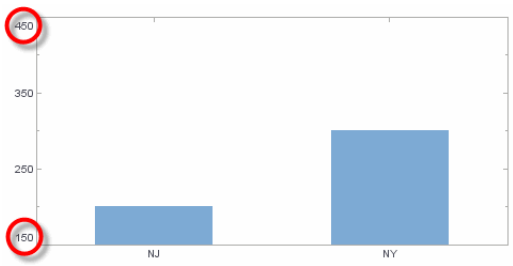
Parameter

value	Number specifying the maximum scale value

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

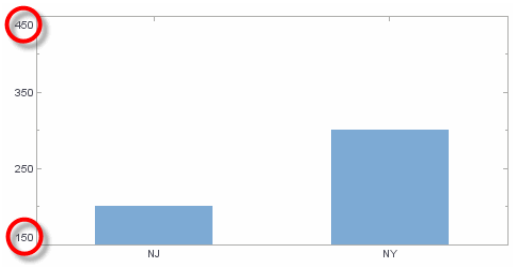
Parameter

value	Number specifying the minimum scale value

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

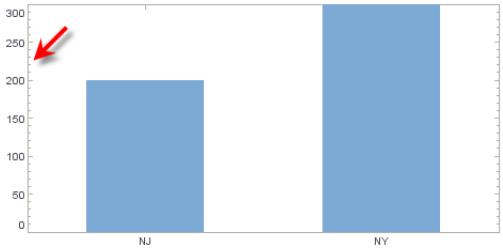
Parameter

value	Number specifying interval between ticks

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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

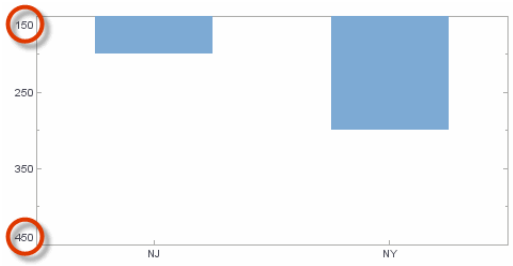
boolean	true: value increases from top to bottom false: value increases from bottom to top

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

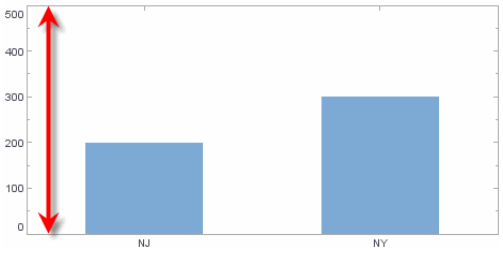
Parameter

range	a *ScaleRange* object

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

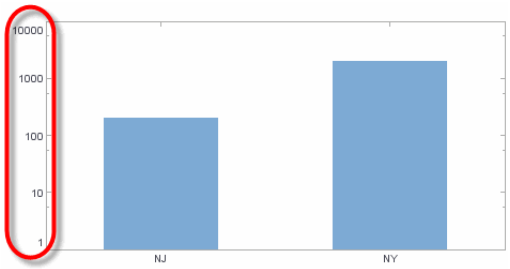
```
var qscale = new LogScale('Last Year', 'This Year');
```

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

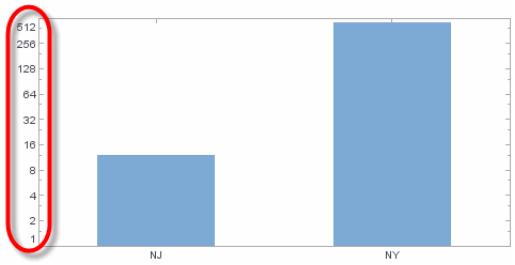
Parameter

value	a number specifying the logarithm base

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

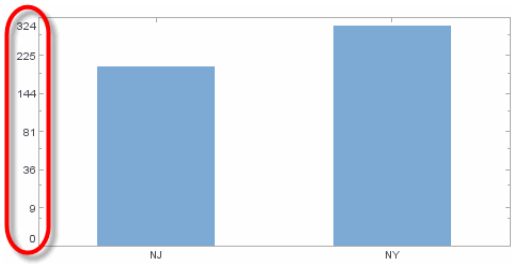
```
var qscale = new PowerScale('Last Year', 'This Year');
```

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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`.

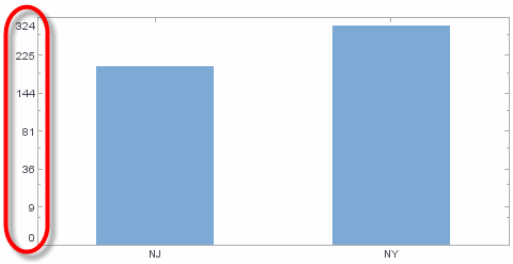
Parameter

value a number specifying the scaling exponent

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

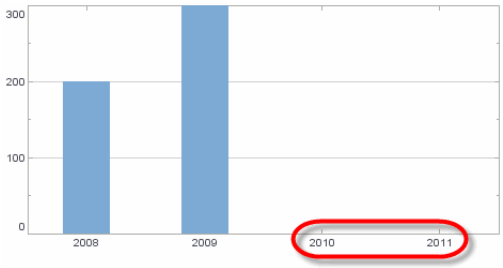
```
var qscale = new TimeScale('Date');
```

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

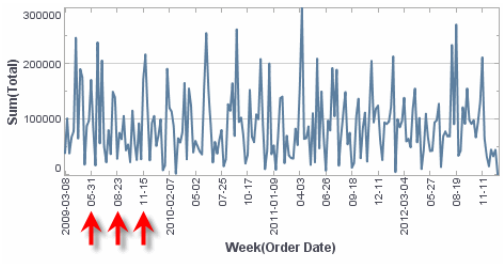
Parameter

`value` `an integer`

`Note: Script that modifies 'graph' should be placed at the element level. See Editing Script in Advanced Report Scripting and Adding Component Script in Advanced 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.

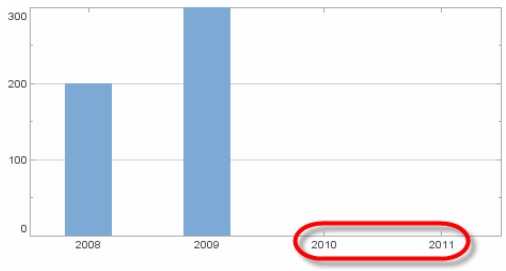
Parameter

value a Date object

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

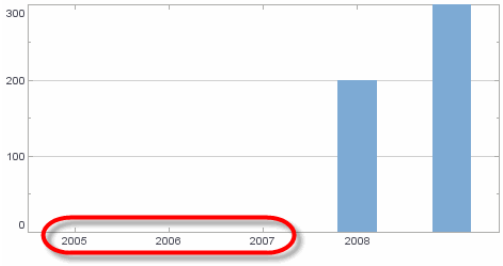
Parameter

value a Date object

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.)

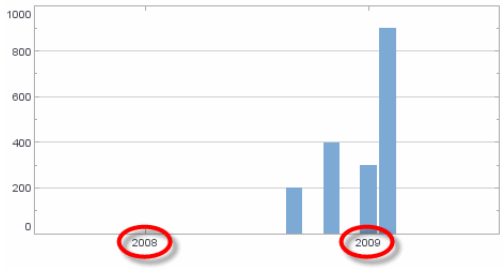
Parameter

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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).

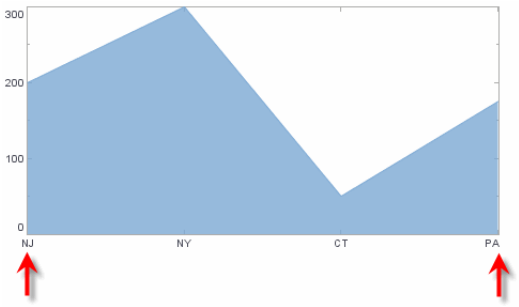
Parameter

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

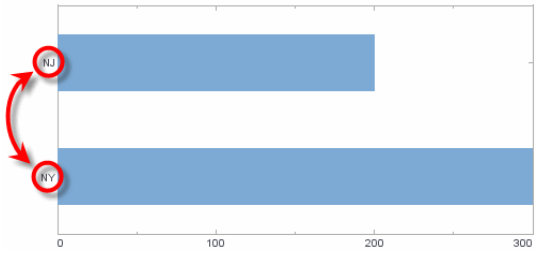
Parameter

value Array of String

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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).

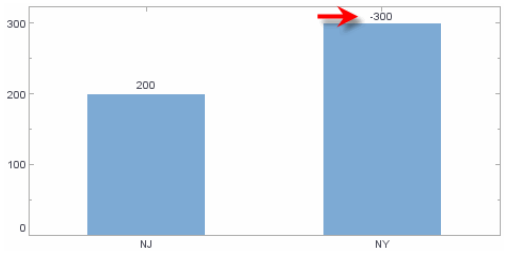
Parameter

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

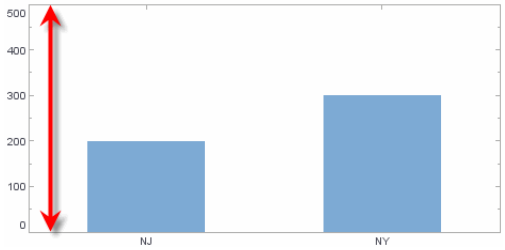
```
var range = new StackRange();
```

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

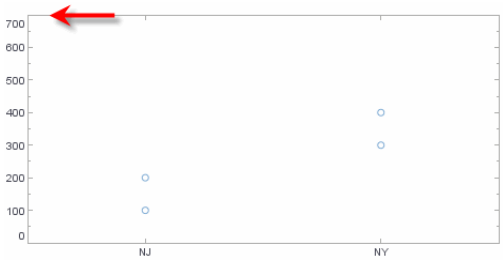
Parameter

value the field name, a String

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

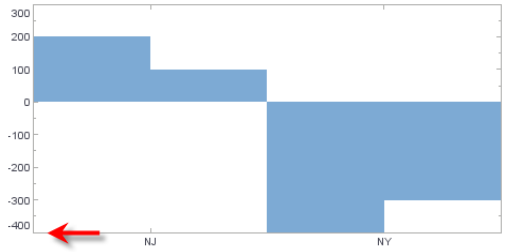
Parameter

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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

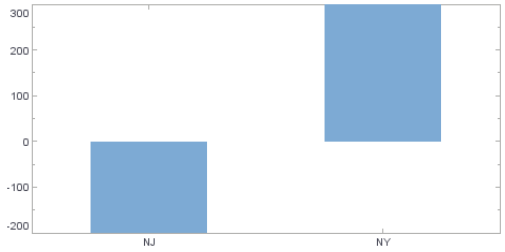
The LinearRange object computes the range by using the minimum and maximum data values. To create a LinearRange object, call the LinearRange constructor.

```
var range = new LinearRange();
```

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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

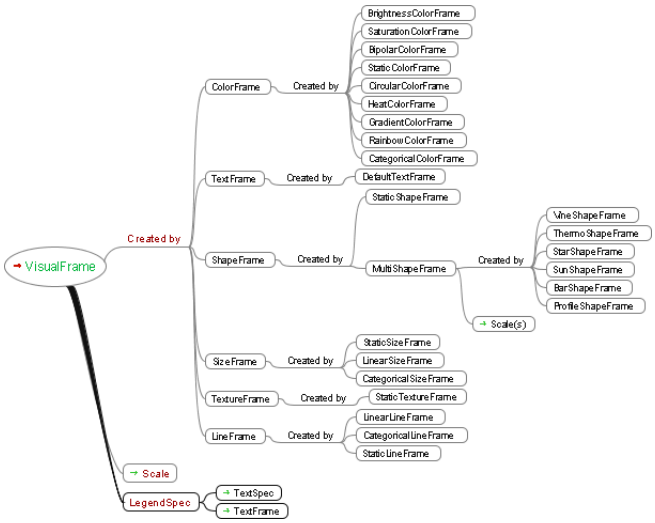
This section discusses the *VisualFrame* objects that can be added to chart elements to introduce visual style. *VisualFrame* objects allow you to represent additional data dimensions by using the physical attributes of chart elements, or to apply a fixed (static) visual style.

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

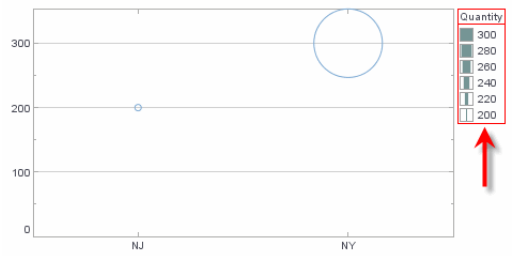
Parameter

spec a *LegendSpec* object

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

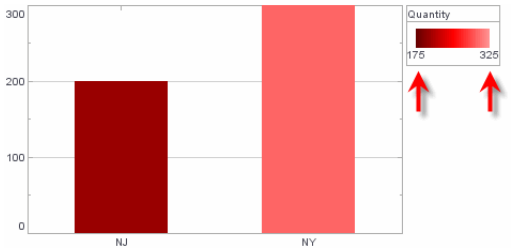
Parameter

scale a *Scale* object

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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

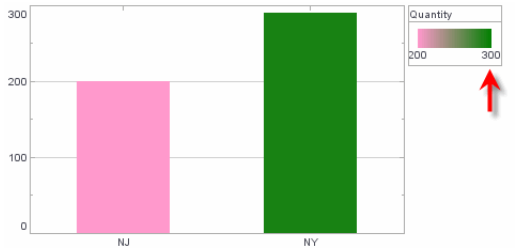
value	`Scale.TICKS` `Scale.ZERO`

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

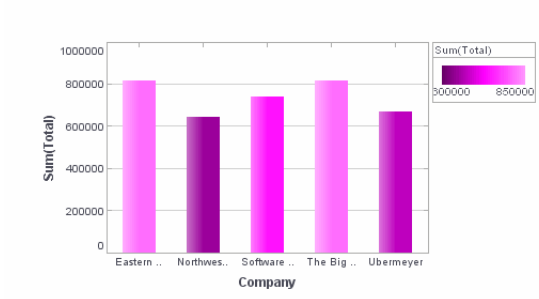
CR.7.2 ColorFrame

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)

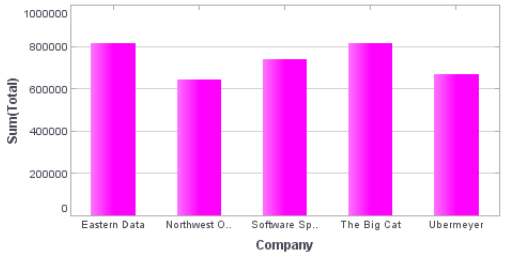
```
Chart1.bindingInfo.setColorField("Total",Chart.NUMBER);
Chart1.bindingInfo.colorFrame = new BrightnessColorFrame;
Chart1.bindingInfo.colorFrame.color = 0xFF00FF;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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}
```

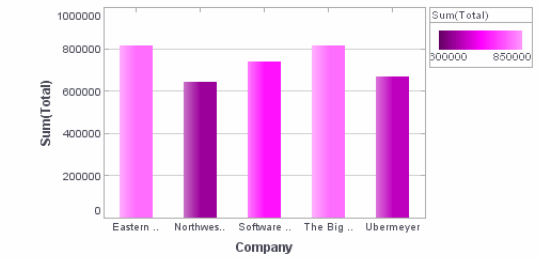
Value (Function Syntax)

value a java.awt.Color object

Note: Report script that modifies 'bindingInfo' should be placed in the onLoad handler.

Example (Report)

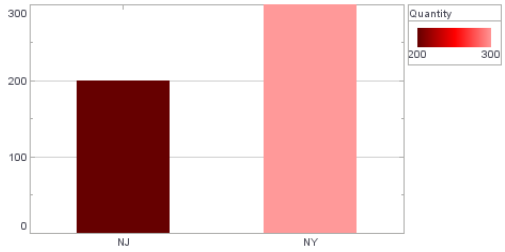
```
Chart1.bindingInfo.setColorField("Total",Chart.NUMBER);
Chart1.bindingInfo.colorFrame = new BrightnessColorFrame;
Chart1.bindingInfo.colorFrame.color = 0xFF00FF;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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)

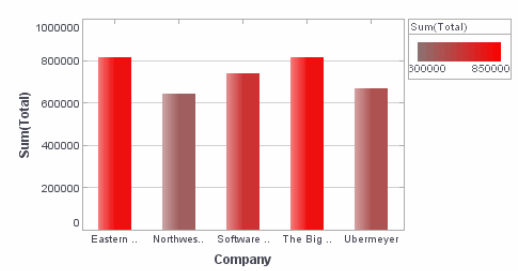
`value` a `java.awt.Color` object

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 in 'Orders' > 'Sample Queries', with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the *onLoad Handler*.

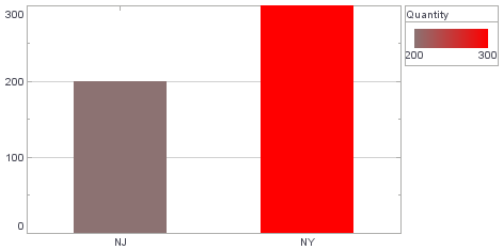
```
Chart1.bindingInfo.setColorField("Total",Chart.NUMBER);
Chart1.bindingInfo.colorFrame = new SaturationColorFrame;
Chart1.bindingInfo.colorFrame.color = 0xFF0000;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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');
```

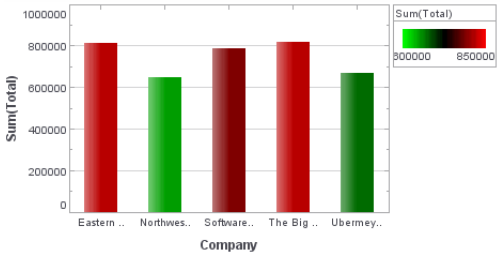
You can pass a field name to the constructor (e.g., "Quantity"), or specify this later using the inherited *VisualFrame.setField(field)* property.

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 in 'Orders' > 'Sample Queries', with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the *onLoad Handler*.

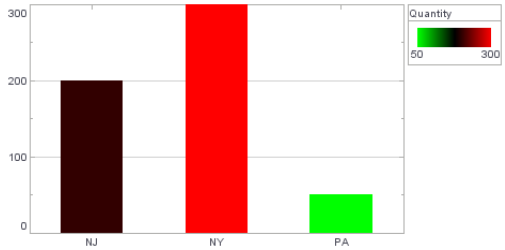
```
Chart1.bindingInfo.setColorField("Total",Chart.NUMBER);
Chart1.bindingInfo.colorFrame = new BipolarColorFrame;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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)

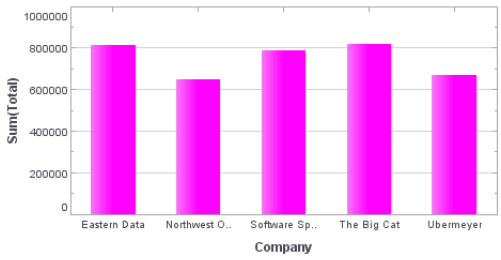
value a java.awt.Color object

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 in 'Orders' > 'Sample Queries', with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the *onLoad Handler*.

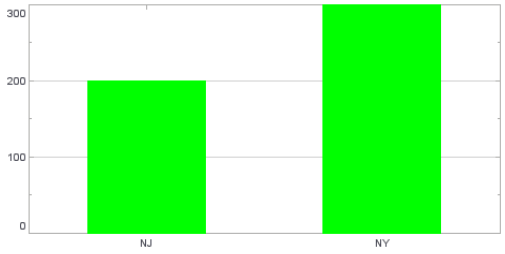
```
Chart1.bindingInfo.setColorField("Company",Chart.STRING);
Chart1.bindingInfo.colorFrame = new StaticColorFrame;
Chart1.bindingInfo.colorFrame.color = 0xFF00FF;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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:

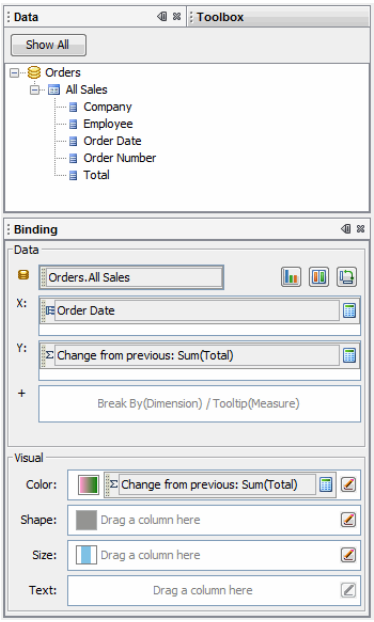
1. Create a new report, and add a Chart component to the report.
2. Click on the chart to open the Chart Editor.
3. In the **Data** tab of the Chart Editor, expand the 'All Sales' query in 'Orders' > 'Sample Queries'.
4. Drag the 'Order Date' field from the **Data** tab to the 'X' region.
5. Press the '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'.

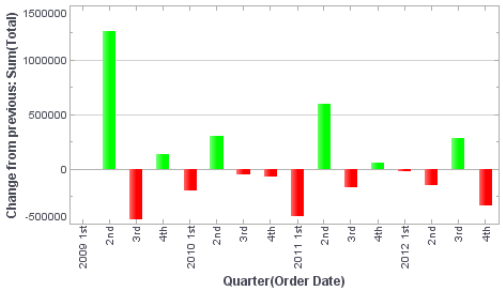


10. Add the following script in the onLoad handler.

```
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.

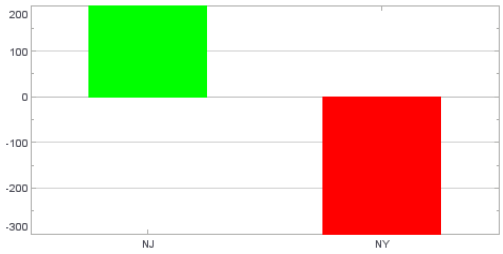
11. Preview the report, and observe that negative-valued bars are displayed in the specified static color.



Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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');
```

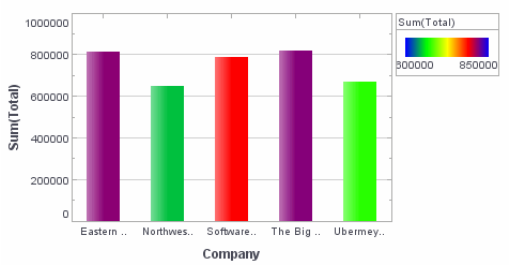
You can pass the name of a field (e.g., 'Quantity') to the constructor, or specify this later using the inherited *VisualFrame.setField(field)* property.

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 in 'Orders' > 'Sample Queries', with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

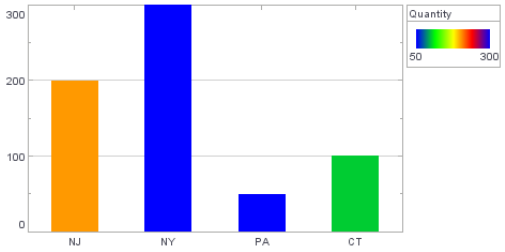
```
Chart1.bindingInfo.setColorField("Total",Chart.NUMBER);
Chart1.bindingInfo.colorFrame = new CircularColorFrame;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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();
```

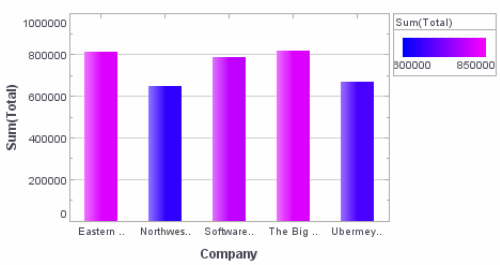
You can pass the name of a field (e.g., 'Quantity') to the constructor, or specify this later using the inherited *VisualFrame.setField(field)* property.

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 in 'Orders' > 'Sample Queries', with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

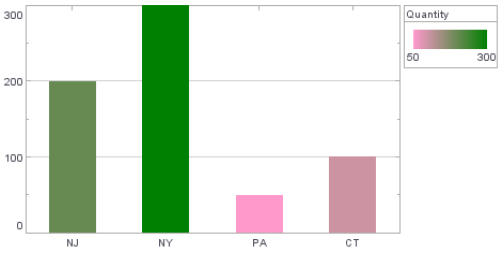
```
Chart1.bindingInfo.setColorField("Total",Chart.NUMBER);
Chart1.bindingInfo.colorFrame = new GradientColorFrame;
Chart1.bindingInfo.colorFrame.fromColor = 0x0000FF;
Chart1.bindingInfo.colorFrame.toColor = 0xFF00FF;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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)

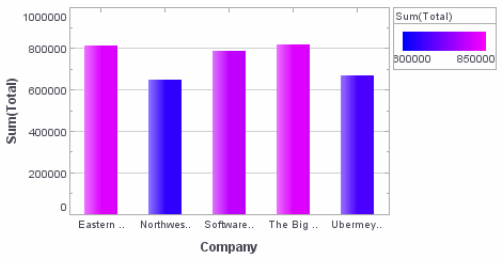
value	a java.awt.Color object

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 in 'Orders' > 'Sample Queries', 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;
```

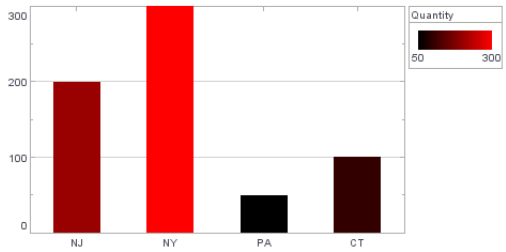


Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced Dashboard Scripting* for more information.

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)

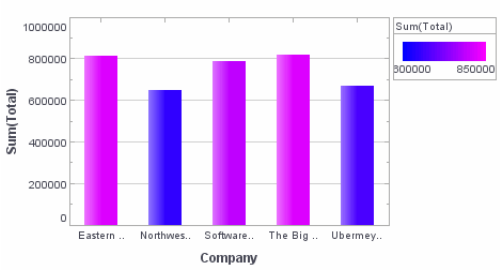
value a java.awt.Color object

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 in ‘Orders’ > ‘Sample Queries’, with ‘Company’ (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

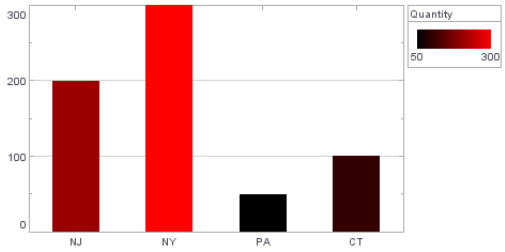
```
Chart1.bindingInfo.setColorField("Total",Chart.NUMBER);
Chart1.bindingInfo.colorFrame = new GradientColorFrame;
Chart1.bindingInfo.colorFrame.fromColor = 0x0000FF;
Chart1.bindingInfo.colorFrame.toColor = 0xFF00FF;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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');
```

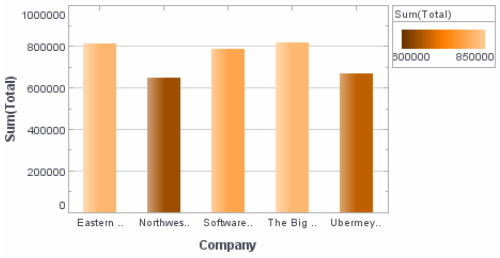
You can pass the name of a field (e.g., 'Quantity') to the constructor, or specify this later using the inherited *VisualFrame.setField(field)* property.

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 in 'Orders' > 'Sample Queries', with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

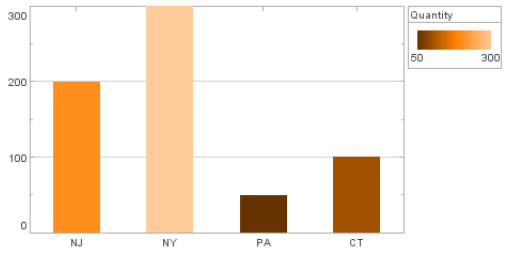
```
Chart1.bindingInfo.setColorField("Total",Chart.NUMBER);
Chart1.bindingInfo.colorFrame = new HeatColorFrame();
```



Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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');
```

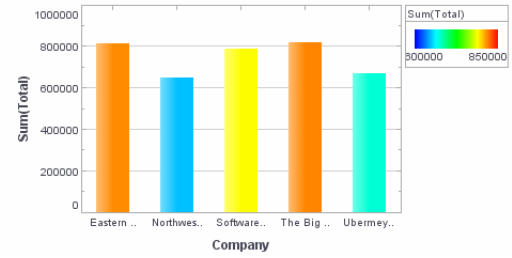
You can pass the name of a field (e.g., 'Quantity') to the constructor, or specify this later using the inherited *VisualFrame.setField(field)* property.

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 in 'Orders' > 'Sample Queries', with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

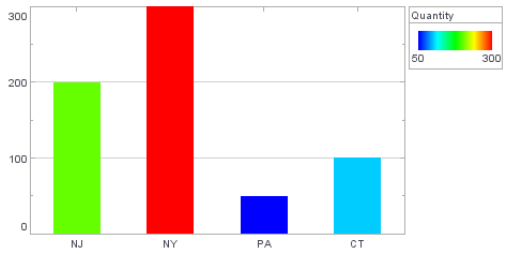
```
Chart1.bindingInfo.setColorField("Total",Chart.NUMBER);
Chart1.bindingInfo.colorFrame = new RainbowColorFrame;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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]);
```

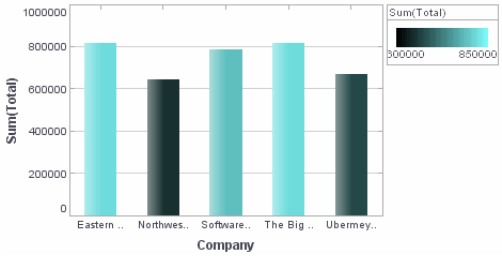
You can pass the name of a field (e.g., 'Quantity') to the constructor, or specify this later using the inherited *VisualFrame.setField(field)* property.

```
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 in 'Orders' > 'Sample Queries', with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

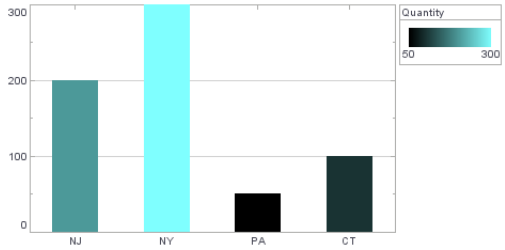
```
Chart1.bindingInfo.setColorField("Total",Chart.NUMBER);
Chart1.bindingInfo.colorFrame = new
RGBCubeColorFrame([[0,0,0],[.5,1,1]]);
```



Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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');
```

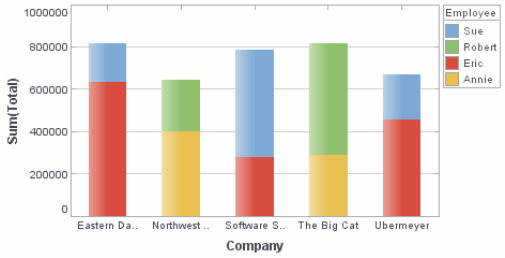
You can pass the name of a field (e.g., 'State') to the constructor, or specify this later using the inherited *VisualFrame.setField(field)* property.

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 in 'Orders' > 'Sample Queries', with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

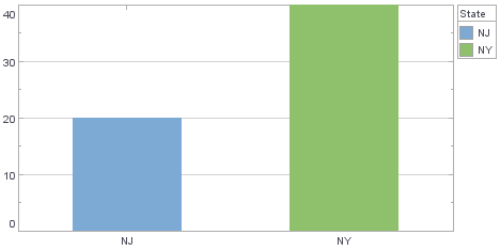
```
Chart1.bindingInfo.setColorField("Employee",Chart.STRING);
Chart1.bindingInfo.colorFrame = new CategoricalColorFrame;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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

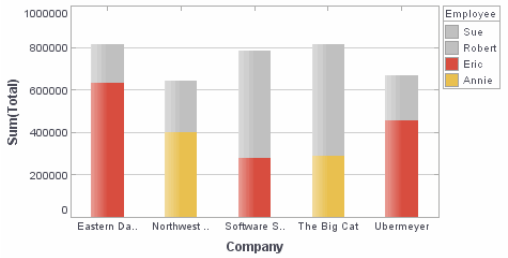
val a data value

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 in 'Orders' > 'Sample Queries', with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

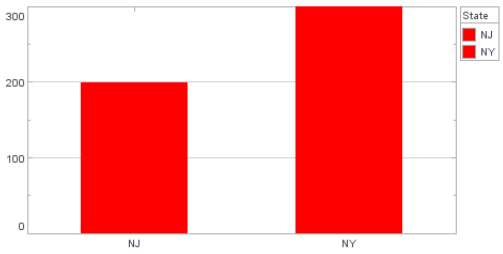
```
Chart1.bindingInfo.setColorField("Employee",Chart.STRING);
Chart1.bindingInfo.colorFrame = new CategoricalColorFrame;
Chart1.bindingInfo.colorFrame.setColor("Robert",java.awt.Co
lor.lightGray);
var robColor =
Chart1.bindingInfo.colorFrame.getColor("Robert")
Chart1.bindingInfo.colorFrame.setColor("Sue",robColor);
```



Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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

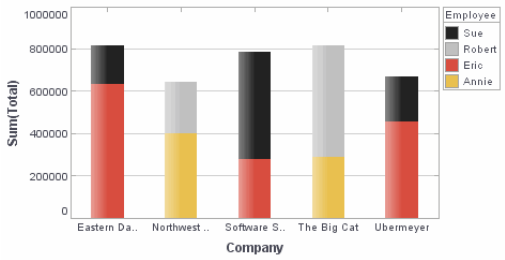
val	data value
color	java.awt.Color to represent data value

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 in 'Orders' > 'Sample Queries', with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

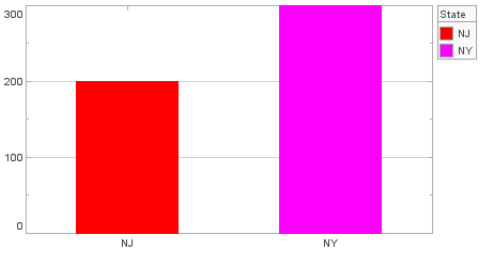
```
Chart1.bindingInfo.setColorField("Employee",Chart.STRING);
Chart1.bindingInfo.colorFrame = new CategoricalColorFrame;
Chart1.bindingInfo.colorFrame.setColor("Robert",java.awt.C
olor.lightGray);
Chart1.bindingInfo.colorFrame.setColor("Sue",java.awt.Colo
r(0x222222));
```



Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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

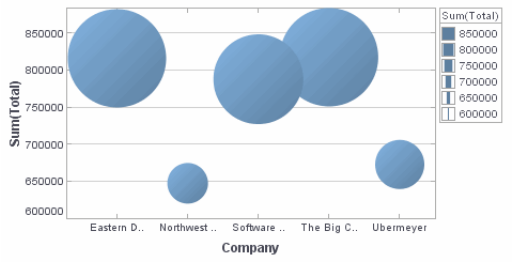
The SizeFrame object contains the size scale for visual chart objects. You can use a SizeFrame object to represent data dimensions with size (size coding), or to apply a fixed (static) size.

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 in ‘Orders’ > ‘Sample Queries’, 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;
```

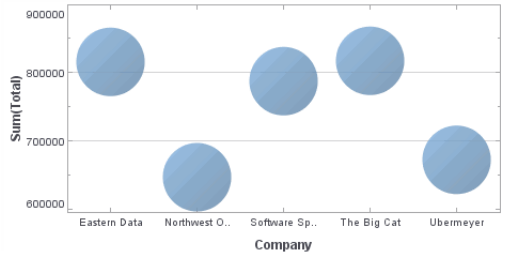


Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced Dashboard Scripting* for more information.

Example (Report or Viewsheet)

Bind a point-type chart to the sample 'All Sales' query in 'Orders' > 'Sample Queries', 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

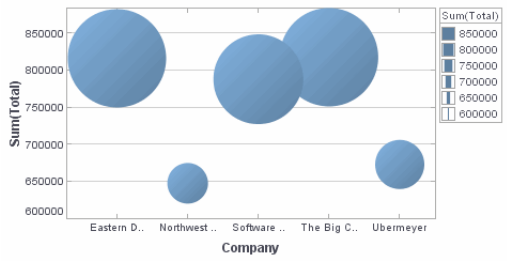
value	Number of pixels, or relative size

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 in 'Orders' > 'Sample Queries', 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;
```

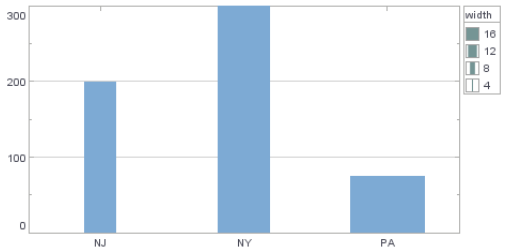


Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced Dashboard Scripting* for more information.

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

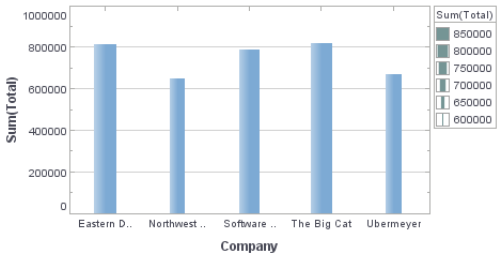
value Number of pixels, or relative size

```
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 in ‘Orders’ > ‘Sample Queries’, with ‘Company’ (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

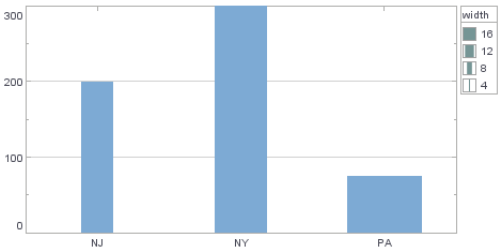
```
Chart1.bindingInfo.setSizeField("Total",Chart.NUMBER);
Chart1.bindingInfo.sizeFrame = new LinearSizeFrame;
Chart1.bindingInfo.sizeFrame.smallest = 10;
Chart1.bindingInfo.sizeFrame.largest = 50;
Chart1.bindingInfo.sizeFrame.max = 100;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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

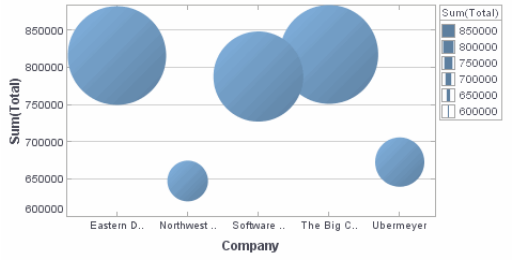
value Number of pixels, or relative size

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 in 'Orders' > 'Sample Queries', with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

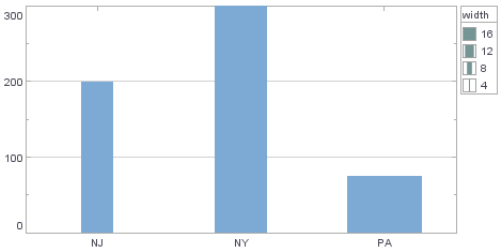
```
Chart1.bindingInfo.setSizeField("Total",Chart.NUMBER);
Chart1.bindingInfo.sizeFrame = new LinearSizeFrame;
Chart1.bindingInfo.sizeFrame.smallest = 10;
Chart1.bindingInfo.sizeFrame.largest = 50;
Chart1.bindingInfo.sizeFrame.max = 100;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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);
```

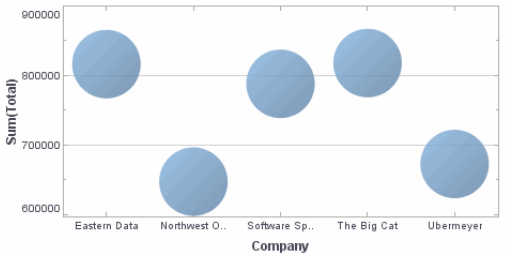
You can pass a numerical size (e.g., 10) to the constructor, or specify this later using the *StaticSizeFrame.setSize(value)* property.

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 in 'Orders' > 'Sample Queries', with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

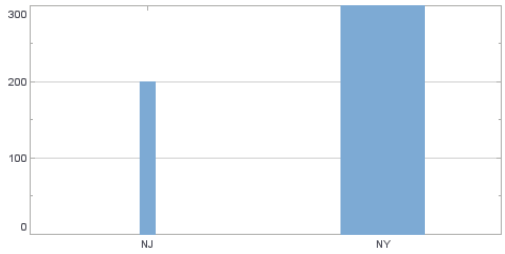
```
Chart1.bindingInfo.setSizeField("Company",Chart.STRING);
Chart1.bindingInfo.sizeFrame = new StaticSizeFrame;
Chart1.bindingInfo.sizeFrame.setSize(30);
```



Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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

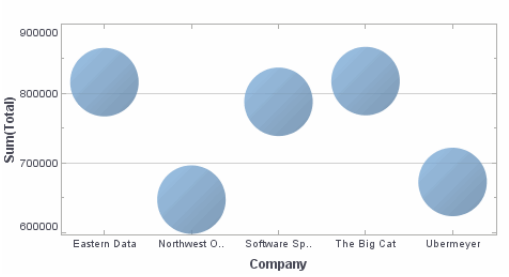
`value` `Number of pixels, or relative size`

```
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 in ‘Orders’ > ‘Sample Queries’, with ‘Company’ (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

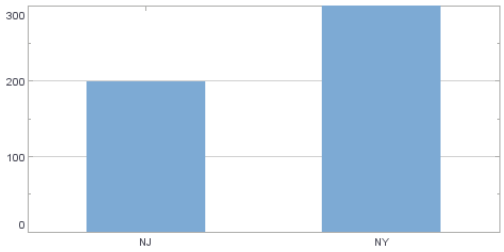
```
Chart1.bindingInfo.setSizeField("Company",Chart.STRING);
Chart1.bindingInfo.sizeFrame = new StaticSizeFrame;
Chart1.bindingInfo.sizeFrame.size = 30;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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');
```

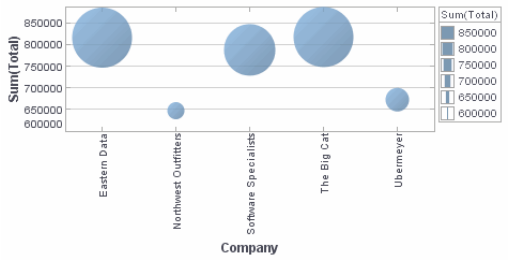
You can pass the name of a field (e.g., 'Quantity') to the constructor, or specify this later using the inherited *VisualFrame.setField(field)* property.

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 in 'Orders' > 'Sample Queries', with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

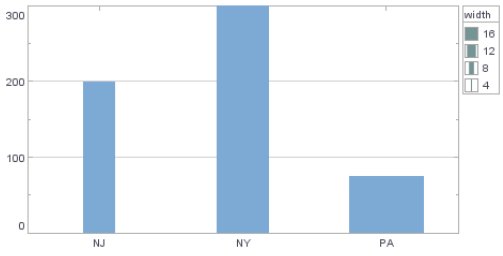
```
Chart1.bindingInfo.setSizeField("Total",Chart.NUMBER)
Chart1.bindingInfo.sizeFrame = new LinearSizeFrame;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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');
```

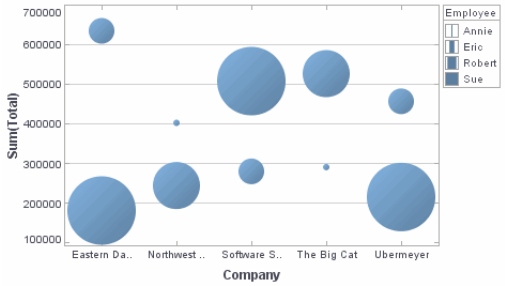
You can pass the name of a field (e.g., 'State') to the constructor, or specify this later using the inherited *VisualFrame.setField(field)* property.

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 in 'Orders' > 'Sample Queries', with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

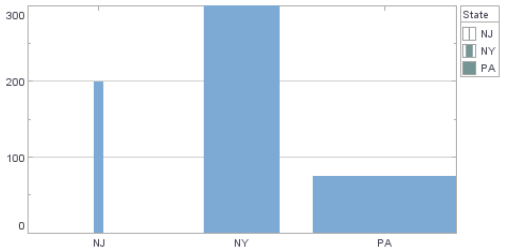
```
Chart1.bindingInfo.setSizeField("Employee",Chart.STRING)
Chart1.bindingInfo.sizeFrame = new CategoricalSizeFrame;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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

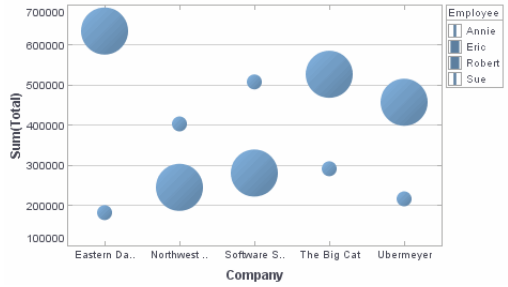
val	a data value
size	size at which to represent data value

```
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 in 'Orders' > 'Sample Queries', with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

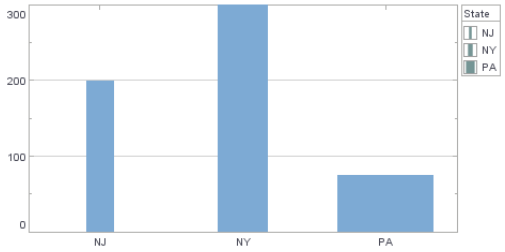
```
Chart1.bindingInfo.setSizeField("Employee",Chart.STRING)
Chart1.bindingInfo.sizeFrame = new CategoricalSizeFrame;
Chart1.bindingInfo.sizeFrame.setSize('Robert',20);
Chart1.bindingInfo.sizeFrame.setSize('Eric',20)
Chart1.bindingInfo.sizeFrame.setSize('Sue',5);
Chart1.bindingInfo.sizeFrame.setSize('Annie',5);
```



Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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

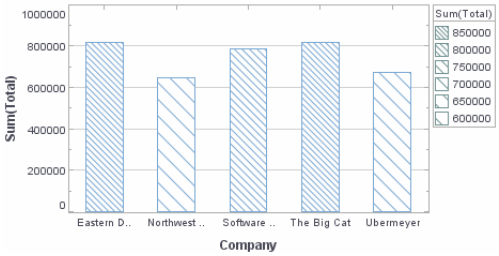
The TextureFrame object contains the texture for visual chart objects. You can use a TextureFrame object to represent data dimensions with texture (texture coding), or to apply a fixed (static) texture pattern.

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 in ‘Orders’ > ‘Sample Queries’, 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;
```

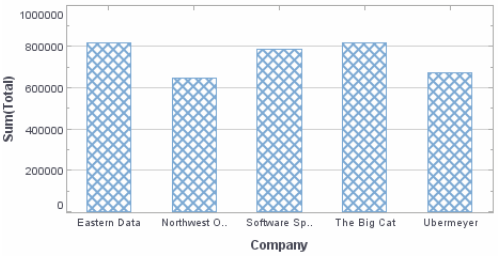


Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced Dashboard Scripting* for more information.

Example (Report)

Bind a point-type chart to the sample 'All Sales' query in 'Orders' > 'Sample Queries', 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);
```

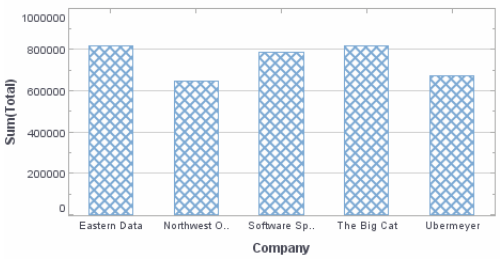
or specify it later using the *StaticTextureFrame.setTexture(value)* property.

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced Dashboard Scripting* for more information.

Example (Report)

Bind a point-type chart to the sample 'All Sales' query in 'Orders' > 'Sample Queries', with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis.

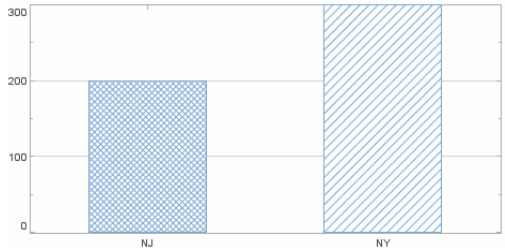
```
var frame = new StaticTextureFrame();
frame.setTexture(GTexture.PATTERN_5);
var elem = graph.getElement(0); // bar elements
elem.setTextureFrame(frame);
```



Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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

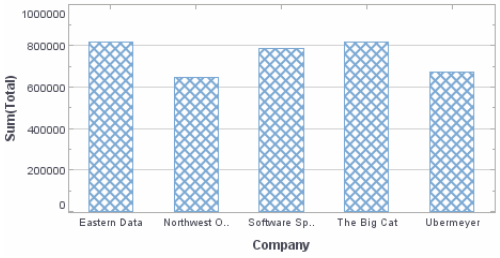
value a `GTexture` object

```
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 in ‘Orders’ > ‘Sample Queries’, with ‘Company’ (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

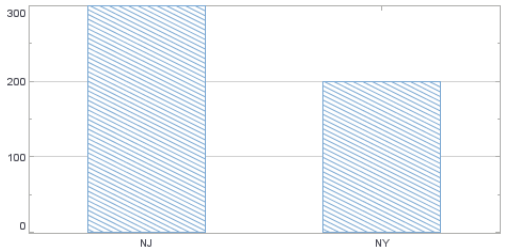
```
Chart1.bindingInfo.setShapeField("Company",Chart.STRING);
Chart1.bindingInfo.textureFrame = new StaticTextureFrame();
Chart1.bindingInfo.textureFrame.texture =
GTexture.PATTERN_5;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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');
```

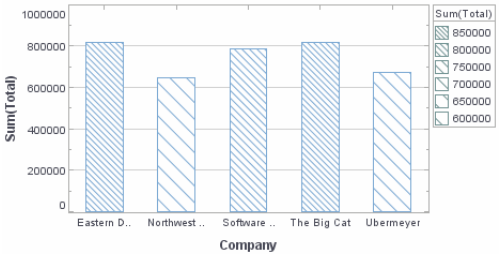
You can pass the name of a field (e.g., 'Total') to the constructor, or specify this later using the inherited *VisualFrame.setField(field)* property.

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 in 'Orders' > 'Sample Queries', with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

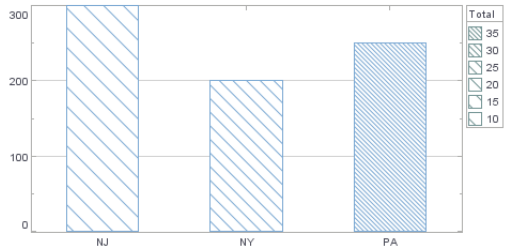
```
Chart1.bindingInfo.setShapeField("Total",Chart.NUMBER);
Chart1.bindingInfo.textureFrame = new LeftTiltTextureFrame;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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');
```

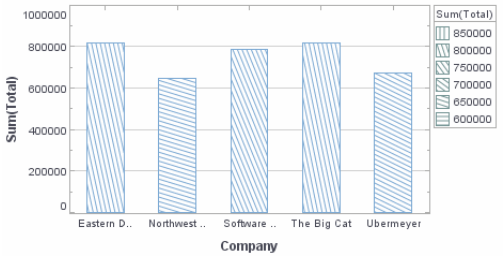
You can pass the name of a field (e.g., 'Total') to the constructor, or specify this later using the inherited *VisualFrame.setField(field)* property.

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 in 'Orders' > 'Sample Queries', with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

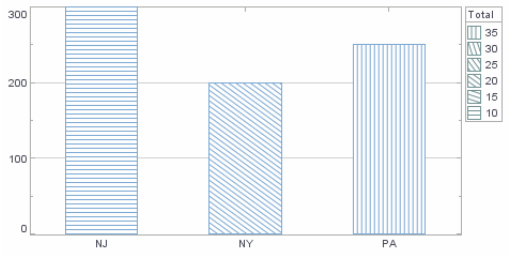
```
Chart1.bindingInfo.setShapeField("Total",Chart.NUMBER);
Chart1.bindingInfo.textureFrame = new
OrientationTextureFrame;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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');
```

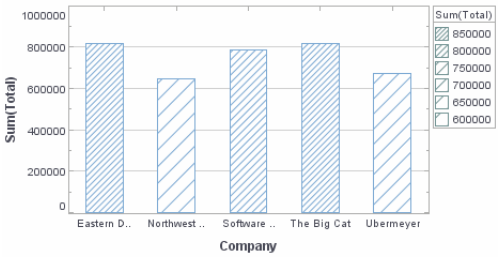
You can pass the name of a field (e.g., 'Total') to the constructor, or specify this later using the inherited *VisualFrame.setField(field)* property.

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 in 'Orders' > 'Sample Queries', with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

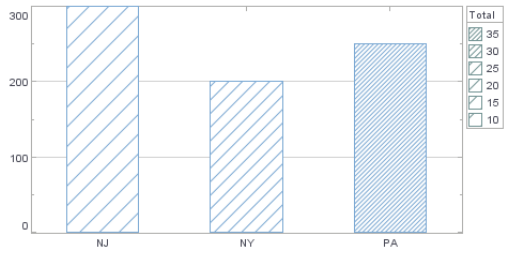
```
Chart1.bindingInfo.setShapeField("Total",Chart.NUMBER);
Chart1.bindingInfo.textureFrame = new
RightTiltTextureFrame;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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

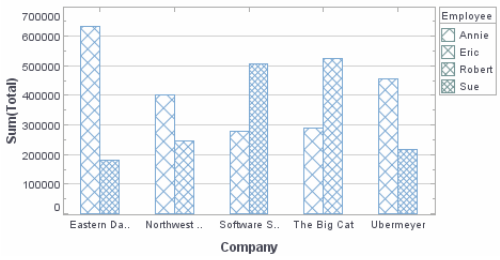
```
inetsoft.graph.aesthetic.GridTextureFrame
```

```
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 in 'Orders' > 'Sample Queries', with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

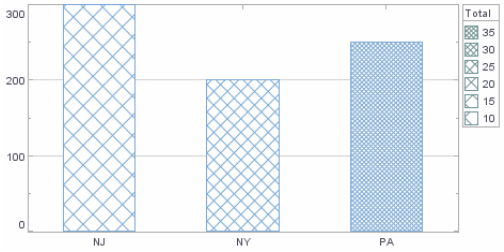
```
Chart1.bindingInfo.setShapeField("Employee",Chart.STRING);
Chart1.bindingInfo.textureFrame = new GridTextureFrame();
```



Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

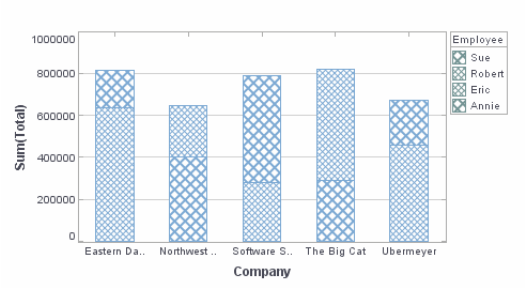
Note: Report script that modifies 'bindingInfo' should be placed in the onLoad handler.

Example (Report)

Bind a point-type chart to the sample 'All Sales' query in 'Orders' > 'Sample Queries', 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);
```

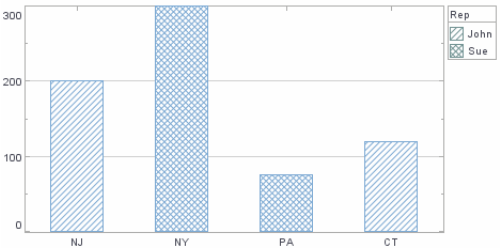
```
Chart1.bindingInfo.textureFrame = new
CategoricalTextureFrame;
Chart1.bindingInfo.textureFrame.setTexture('Robert',GTextur
e.PATTERN_1);
Chart1.bindingInfo.textureFrame.setTexture('Eric',GTexture.
PATTERN_1);
Chart1.bindingInfo.textureFrame.setTexture('Sue',GTexture.P
ATTERN_5);
Chart1.bindingInfo.textureFrame.setTexture('Annie',GTexture
.PATTERN_5);
```



Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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

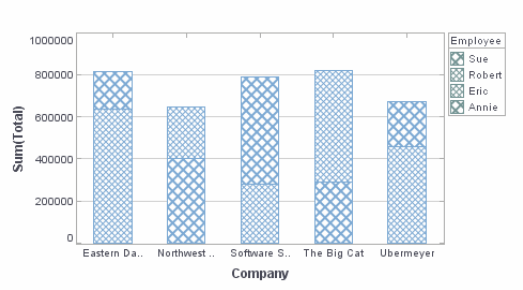
val	a data value
shape	a *GTexture* object

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 in 'Orders' > 'Sample Queries', with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

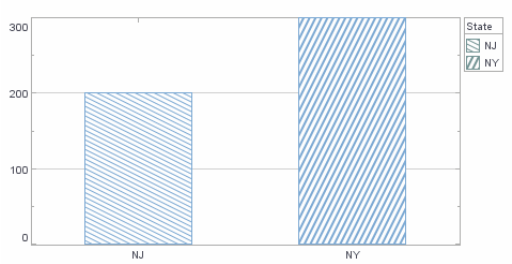
```
Chart1.bindingInfo.setShapeField("Employee",Chart.STRING);
Chart1.bindingInfo.textureFrame = new
CategoricalTextureFrame;
Chart1.bindingInfo.textureFrame.setTexture('Robert',GTextu
re.PATTERN_1);
Chart1.bindingInfo.textureFrame.setTexture('Eric',GTexture
.PATTERN_1);
Chart1.bindingInfo.textureFrame.setTexture('Sue',GTexture.
PATTERN_5);
Chart1.bindingInfo.textureFrame.setTexture('Annie',GTextur
e.PATTERN_5);
```



Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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

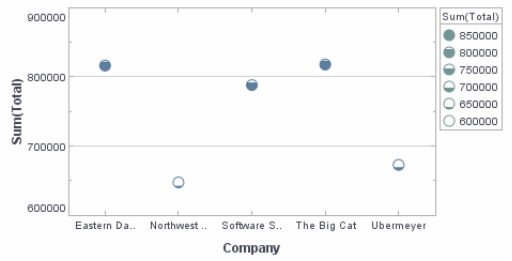
A ShapeFrame object contains the shape style for visual chart objects. You can use a ShapeFrame object to represent data dimensions with shape (shape coding), or to apply a fixed (static) shape style.

```
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 in ‘Orders’ > ‘Sample Queries’, 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;
```

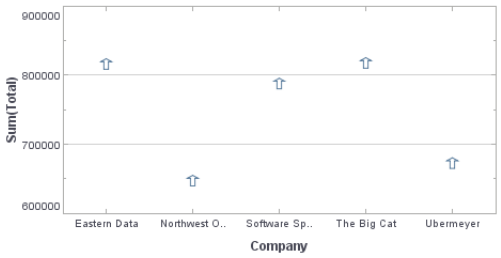



```
Note: Script that modifies 'graph' should be placed at the element
level. See Editing Script in Advanced Report Scripting and Adding
Component Script in Advanced Dashboard Scripting for more
information.
```

Example (Report or Viewsheet)

Bind a point-type chart to the sample ‘All Sales’ query in ‘Orders’ > ‘Sample Queries’, 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');
```

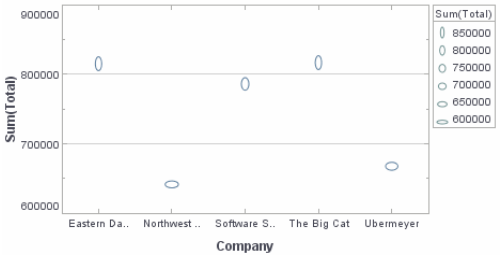
You can pass the name of a field (e.g., 'Total') to the constructor, or specify this later using the inherited *VisualFrame.setField(field)* property.

```
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 in 'Orders' > 'Sample Queries', with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

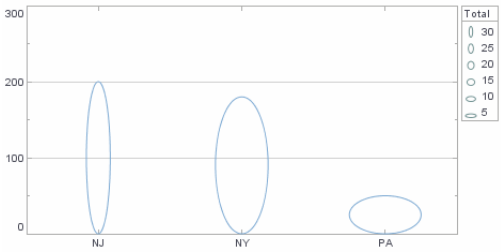
```
Chart1.bindingInfo.setShapeField("Total",Chart.NUMBER);
Chart1.bindingInfo.shapeFrame = new OvalShapeFrame();
```



Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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

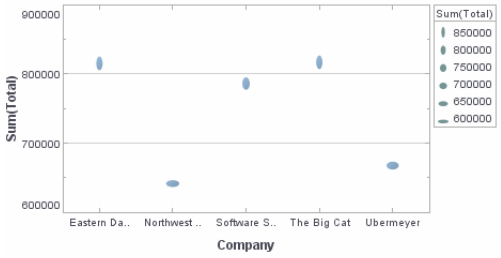
boolean	true: fill oval false: do not fill oval

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 in 'Orders' > 'Sample Queries', with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

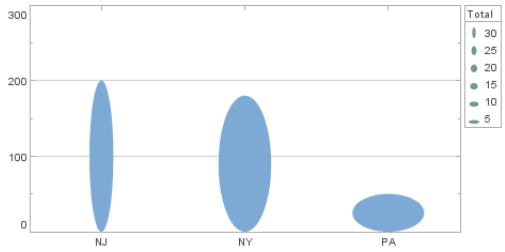
```
Chart1.bindingInfo.setShapeField("Total",Chart.NUMBER);
Chart1.bindingInfo.shapeFrame = new OvalShapeFrame;
Chart1.bindingInfo.shapeFrame.fill = true;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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');
```

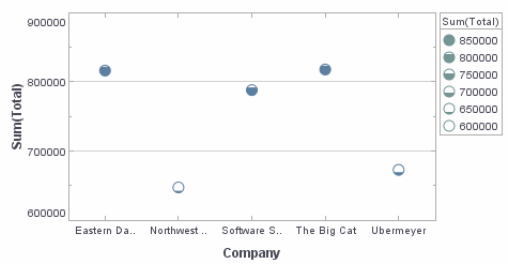
You can pass the name of a field (e.g., 'Total') to the constructor, or specify this later using the inherited *VisualFrame.setField(field)* property.

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 in 'Orders' > 'Sample Queries', with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

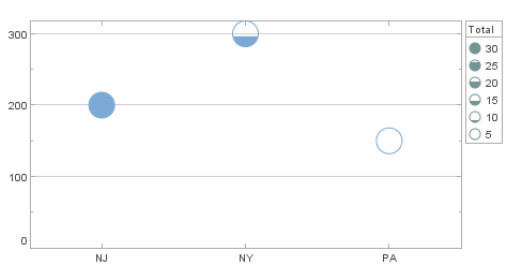
```
Chart1.bindingInfo.setShapeField("Total",Chart.NUMBER);
Chart1.bindingInfo.shapeFrame = new FillShapeFrame;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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');
```

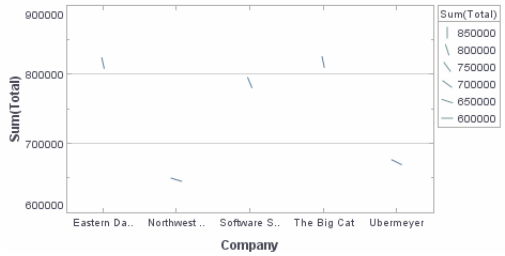
You can pass the name of a field (e.g., 'Total') to the constructor, or specify this later using the inherited *VisualFrame.setField(field)* property.

```
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 in 'Orders' > 'Sample Queries', 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 Editing Script in Advanced Report Scripting and Adding
```

Component Script in *Advanced 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');
```

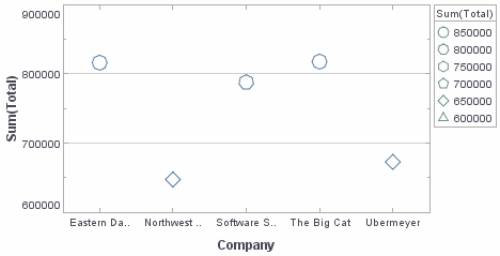
You can pass the name of a field (e.g., 'Total') to the constructor, or specify this later using the inherited *VisualFrame.setField(field)* property.

```
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 in 'Orders' > 'Sample Queries', with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

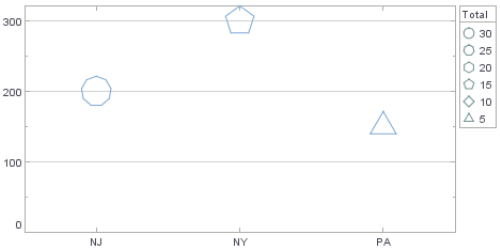
```
Chart1.bindingInfo.setShapeField("Total",Chart.NUMBER);
Chart1.bindingInfo.shapeFrame = new PolygonShapeFrame;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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

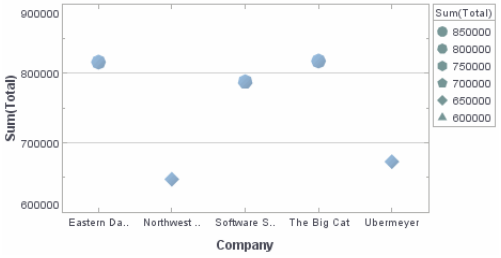
boolean	true: fill oval false: do not fill oval

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 in 'Orders' > 'Sample Queries', with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

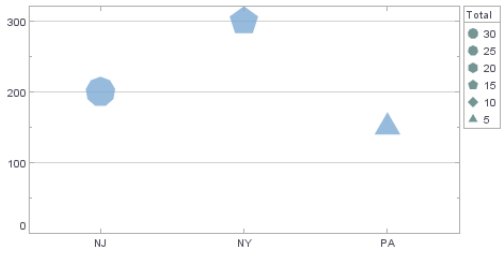
```
Chart1.bindingInfo.setShapeField("Total",Chart.NUMBER);
Chart1.bindingInfo.shapeFrame = new PolygonShapeFrame;
Chart1.bindingInfo.shapeFrame.fill = true;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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');
```

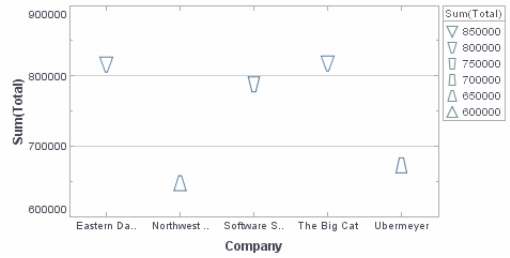
You can pass the name of a field (e.g., 'Total') to the constructor, or specify this later using the inherited *VisualFrame.setField(field)* property.

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 in 'Orders' > 'Sample Queries', 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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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

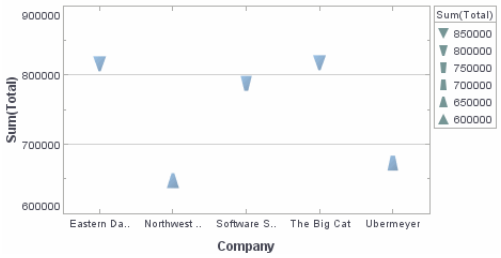
boolean `true: fill triangle`
`false: do not fill triangle`

`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 in ‘Orders’ > ‘Sample Queries’, 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;
```

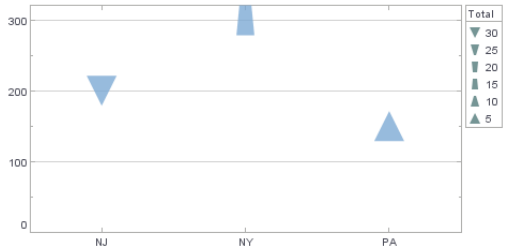


`Note: Script that modifies 'graph' should be placed at the element level. See` *`Editing Script`* `in` *`Advanced Report Scripting`* `and` *`Adding`*

Component Script in *Advanced 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.

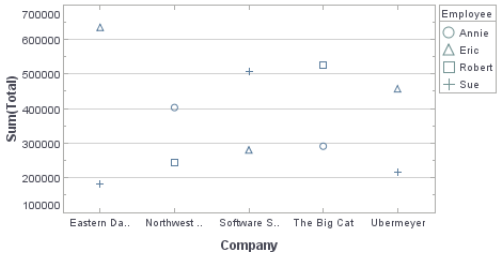
```
Note: Report script that modifies 'bindingInfo' should be placed in
the onLoad handler.
```

Example (Report)

Bind a point-type chart to the sample 'All Sales' query in 'Orders' > 'Sample Queries', with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

Example

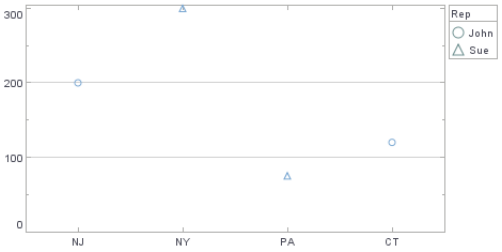
```
Chart1.bindingInfo.setShapeField("Employee",Chart.STRING);
Chart1.bindingInfo.shapeFrame = new CategoricalShapeFrame;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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)

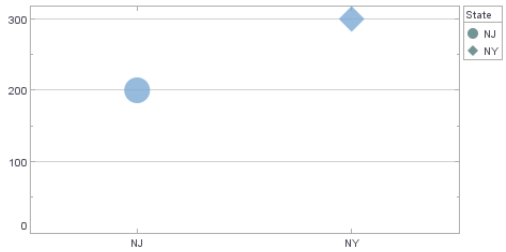
Bind a point-type chart to the sample 'All Sales' query in 'Orders' > 'Sample Queries', with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

```
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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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,

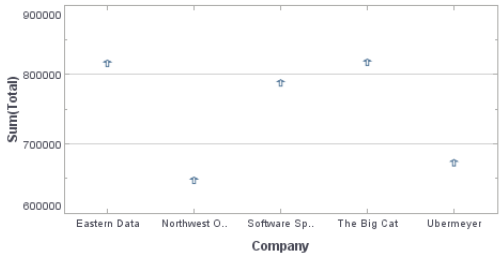
or specify this later using the *StaticShapeFrame.setShape(shape)* or inherited *VisualFrame.setField(field)* property.

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 in 'Orders' > 'Sample Queries', with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

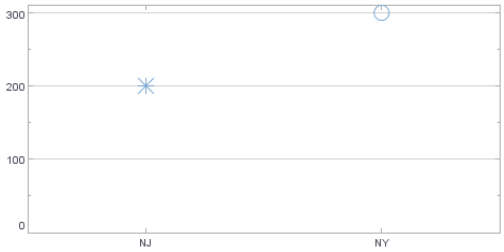
```
Chart1.bindingInfo.setShapeField("Company",Chart.STRING);
Chart1.bindingInfo.shapeFrame = new StaticShapeFrame;
Chart1.bindingInfo.shapeFrame.setShape(GShape.ARROWBAR);
```



Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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

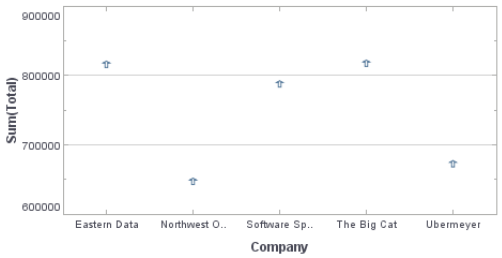
shape *GShape*, *GShape.ImageShape*, or *SVGShape* object

`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 in ‘Orders’ > ‘Sample Queries’, with ‘Company’ (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

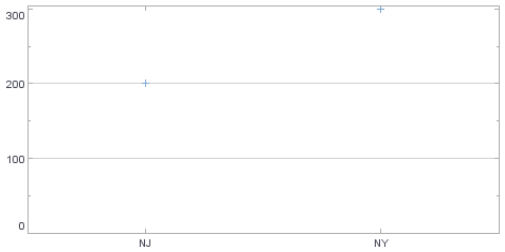
```
Chart1.bindingInfo.setShapeField("Company",Chart.STRING);
Chart1.bindingInfo.shapeFrame = new StaticShapeFrame;
Chart1.bindingInfo.shapeFrame.setShape(GShape.ARROWBAR);
```



Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

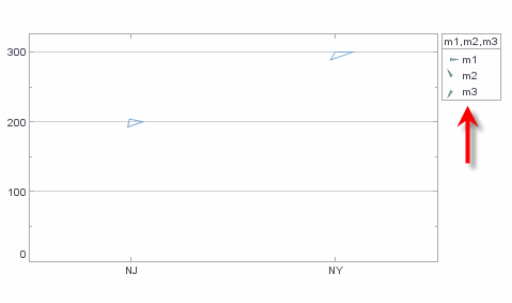
Parameter

arr	Array of Strings

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

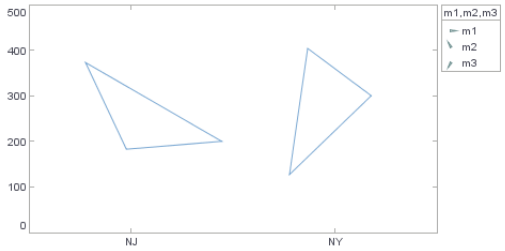
Parameter

arr	Array of *Scale*

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.)

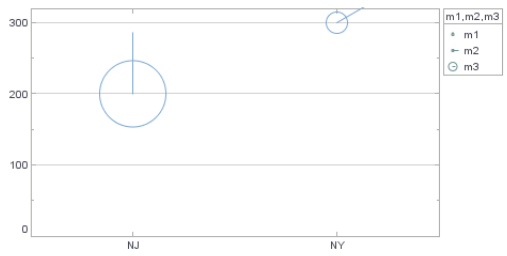
Parameter

`value` `the angle in degrees`

`Note: Script that modifies 'graph' should be placed at the element level. See` *`Editing Script`* `in` *`Advanced Report Scripting`* `and` *`Adding Component Script`* `in` *`Advanced 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.)

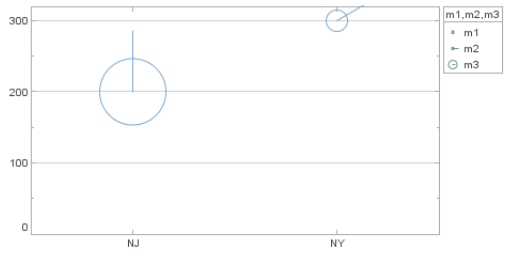
Parameter

value	the angle in degrees

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

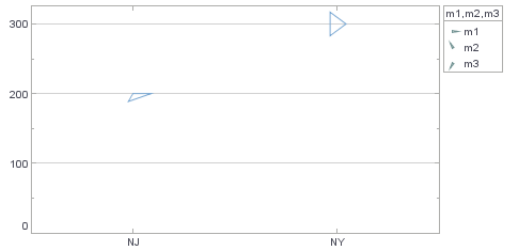
```
var frame = new StarShapeFrame("m1","m2","m3");
```

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

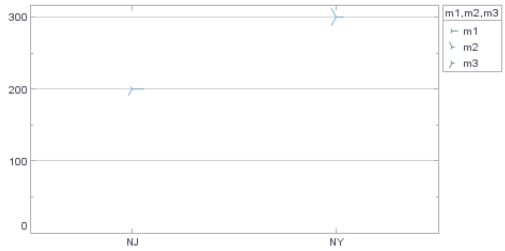
```
var frame = new SunShapeFrame("m1","m2","m3");
```

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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");
```

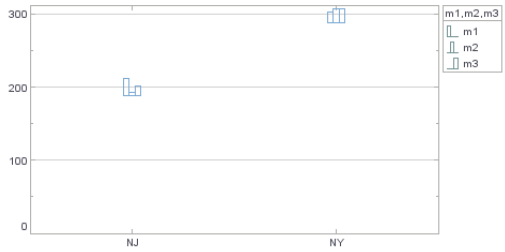
You can pass a set of field names (e.g., ‘m1’, ‘m2’, ‘m3’) to the constructor, or specify this later using the inherited *MultiShapeFrame.setFields(arr)* property. The dimensions are mapped to the bar heights from left to right.

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding*

Component Script in *Advanced 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.

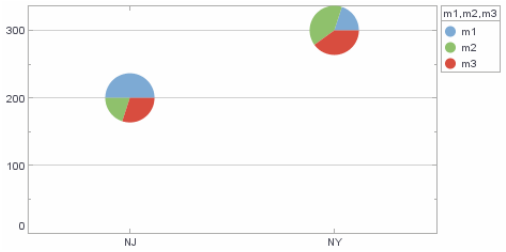
```
var frame = new PieShapeFrame("m1","m2","m3");
```

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

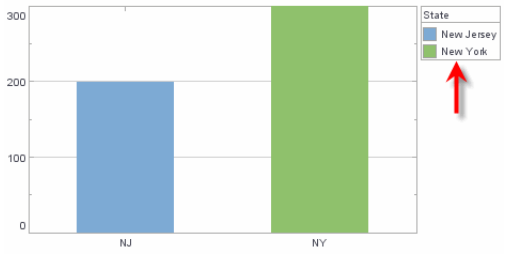
Parameters

value	the data value to be replaced
text	the replacement string

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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');
```

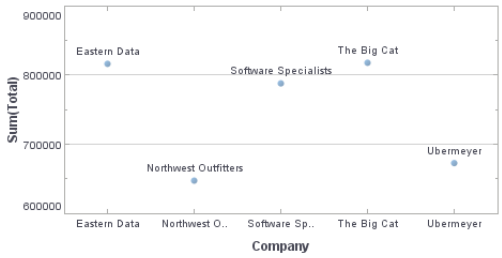
You can pass a field name (e.g., 'Quantity') to the constructor, or specify this later using the inherited *VisualFrame.setField(field)* property. To stylize or reposition text created using a DefaultTextFrame, use the *GraphElement.setTextSpec(spec)* and *GraphElement.setLabelPlacement(value)* options.

```
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 in 'Orders' > 'Sample Queries', with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

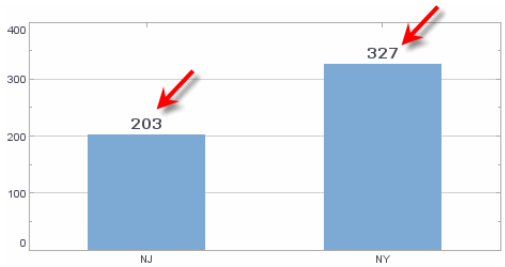
```
Chart1.bindingInfo.setTextField("Company",Chart.STRING)
Chart1.bindingInfo.textFrame = new DefaultTextFrame;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced Dashboard Scripting* for more information.

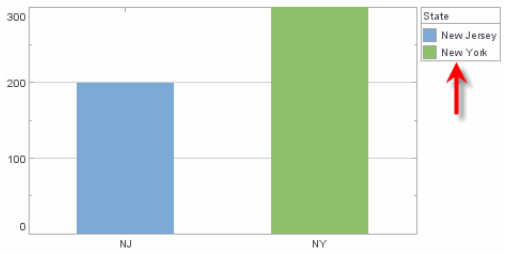
Example (Report or Viewsheet)

```
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

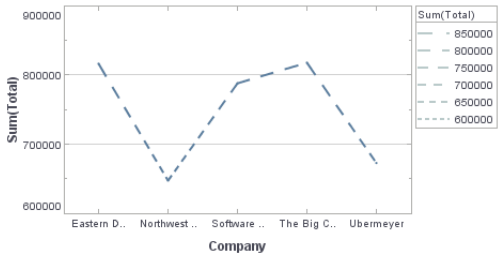
The LineFrame object contains the line design for visual chart objects. LineFrame objects allow you to represent additional data dimensions by using the physical attributes of chart elements, or to apply a fixed (static) line style.

`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 in 'Orders' > 'Sample Queries', with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

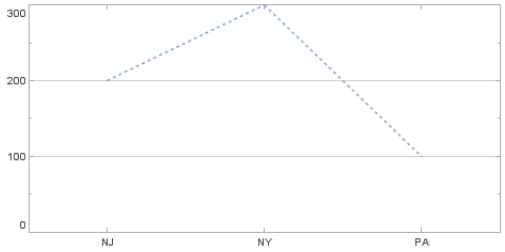
```
Chart1.bindingInfo.setShapeField("Total",Chart.NUMBER);
Chart1.bindingInfo.lineFrame = new LinearLineFrame;
```



`Note: Script that modifies 'graph' should be placed at the element level. See` *`Editing Script`* `in` *`Advanced Report Scripting`* `and` *`Adding Component Script`* `in` *`Advanced 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);
```

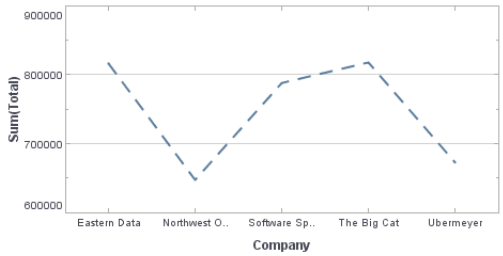
You can pass a *GLine* object or a field name (e.g., 'Line') to the constructor, or specify this later using the *StaticLineFrame.setLine(value)* or inherited *VisualFrame.setField(field)* property.

```
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 in 'Orders' > 'Sample Queries', 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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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

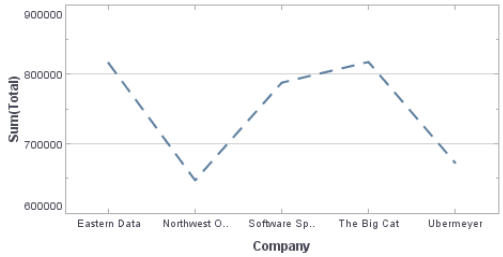
value *GLine* or
`Chart.NONE  // Empty border`

`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 in 'Orders' > 'Sample Queries', 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` *Editing Script* `in` *Advanced Report Scripting* `and` *Adding Component Script* `in` *Advanced 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');
```

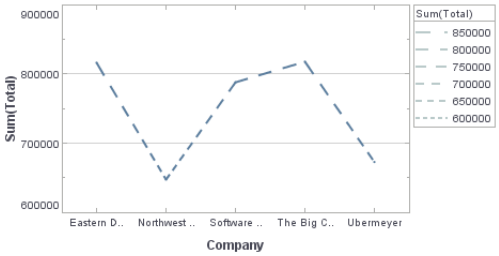
You can pass the name of a field (e.g., 'Quantity') to the constructor, or specify this later using the inherited *VisualFrame.setField(field)* property.

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 in 'Orders' > 'Sample Queries', with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

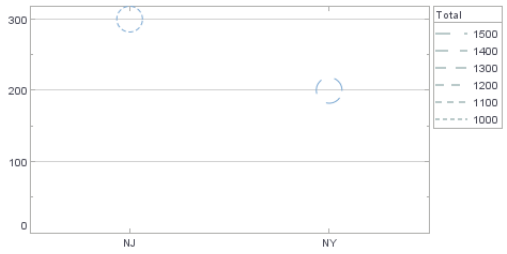
```
Chart1.bindingInfo.setShapeField("Total",Chart.NUMBER);
Chart1.bindingInfo.lineFrame = new LinearLineFrame;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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');
```

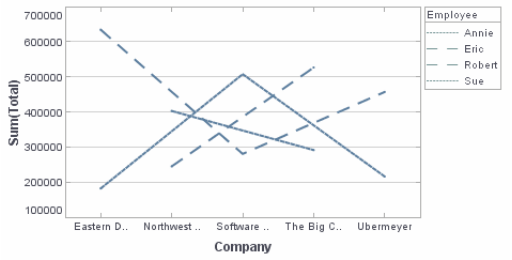
You can pass the name of a field (e.g., 'Quantity') to the constructor, or specify this later using the inherited *VisualFrame.setField(field)* property.

```
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 in 'Orders' > 'Sample Queries', 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);
```



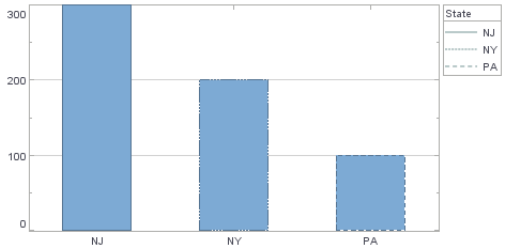
Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced Dashboard Scripting* for more information.

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

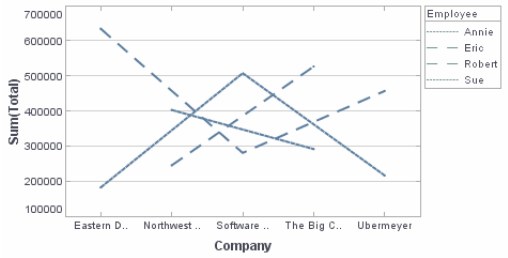
val	a data value
line	a *GLine* value

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 in 'Orders' > 'Sample Queries', with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

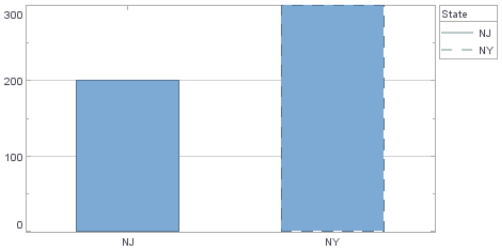
```
Chart1.bindingInfo.setShapeField("Employee",Chart.STRING);
Chart1.bindingInfo.lineFrame = new CategoricalLineFrame;
Chart1.bindingInfo.lineFrame.setLine('Robert',
GLine.LARGE_DASH);
Chart1.bindingInfo.lineFrame.setLine('Eric',
GLine.LARGE_DASH);
Chart1.bindingInfo.lineFrame.setLine('Sue',
GLine.DOT_LINE);
Chart1.bindingInfo.lineFrame.setLine('Annie',
GLine.DOT_LINE);
```



Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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

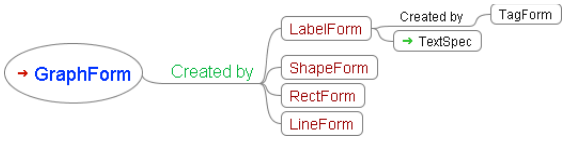
This section describes objects used to add arbitrary text, shapes, and lines to a chart.

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.

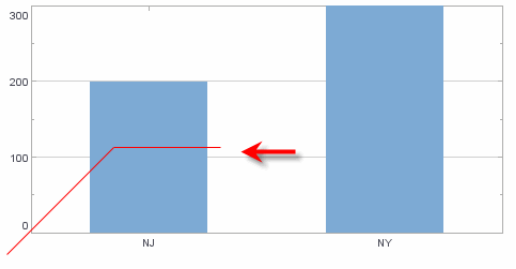
Type

```
value        Number
```

```
Note: Script that modifies 'graph' should be placed at the element
level. See Editing Script in Advanced Report Scripting and Adding
Component Script in Advanced 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.

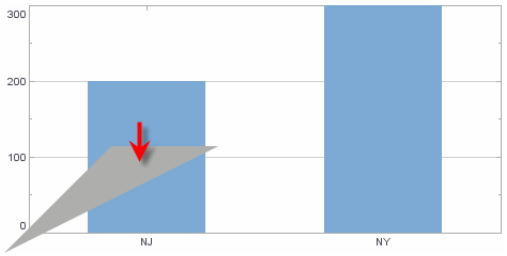
Type

boolean true: fill the shape
false: do not fill the shape

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

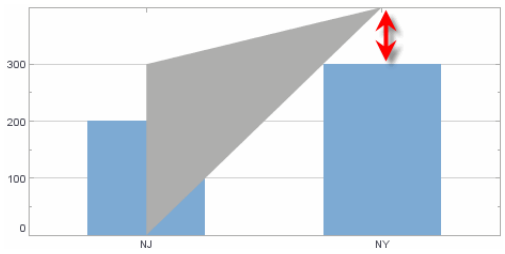
Type

boolean	true: resize chart false: do not resize chart (crop forms)

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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

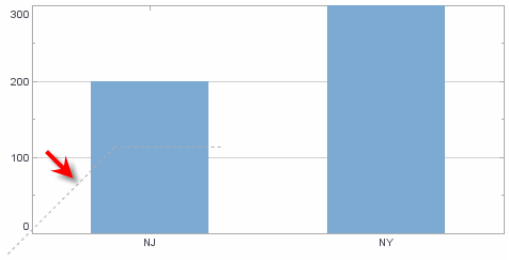
value	a *GLine* constant

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding*

Component Script in *Advanced 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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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

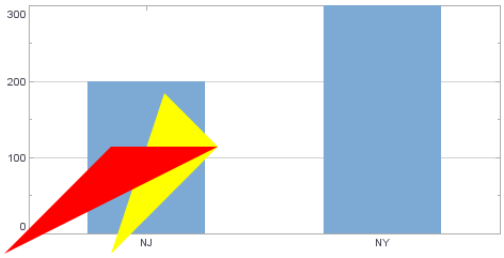
value	the index, a positive integer

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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).

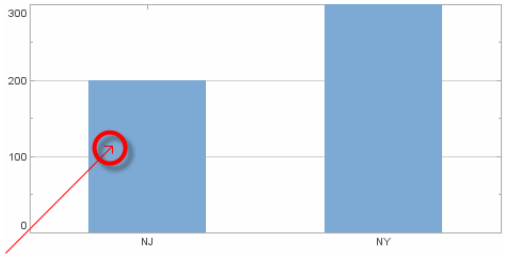
Type

boolean	true: draw arrow false: no arrow

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced Dashboard Scripting* for more information.

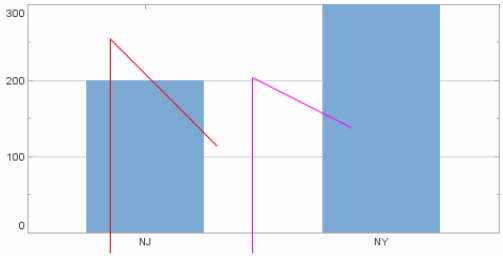
Example (Report or Viewsheet)

```
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).

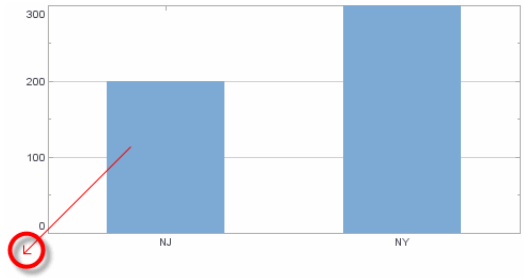
Type

boolean	true: draw arrow false: no arrow

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

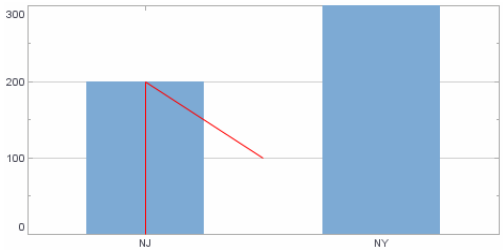
Type

value	An [X,Y] pair

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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').

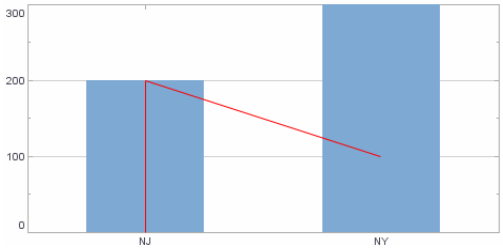
Type

value an [X,Y] pair

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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

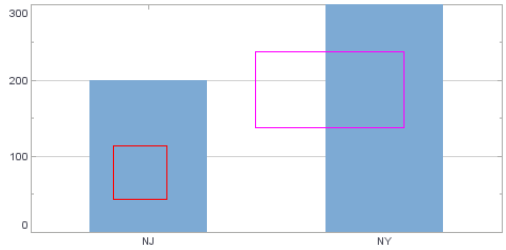
Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced Dashboard Scripting* for more information.

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.

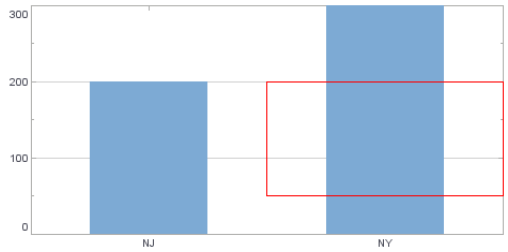
Type

value An [X,Y] pair

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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

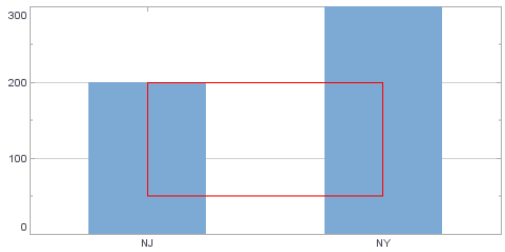
value an [X,Y] pair

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding*

Component Script in *Advanced 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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced Dashboard Scripting* for more information.

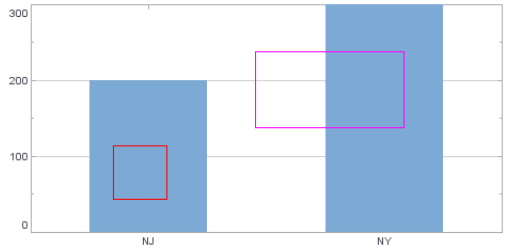
Example (Report or Viewsheet)

```
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.

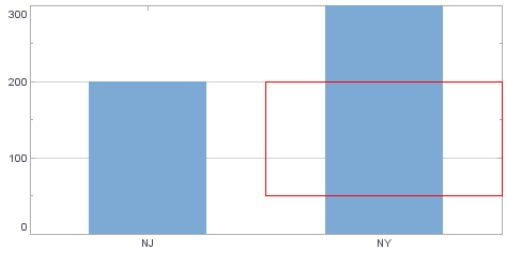
Type

value An [X,Y] pair

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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').

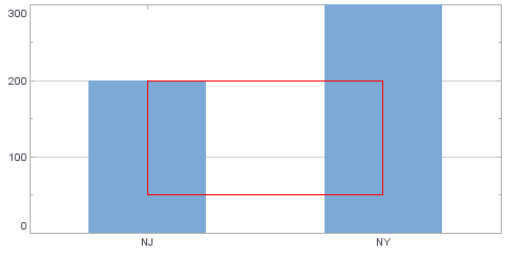
Type

value	an [X,Y] pair

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

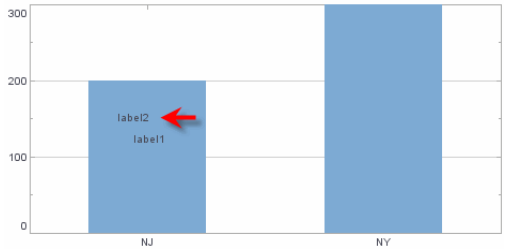
Values

```
Chart.LEFT_ALIGNMENT
Chart.CENTER_ALIGNMENT
Chart.RIGHT_ALIGNMENT
```

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

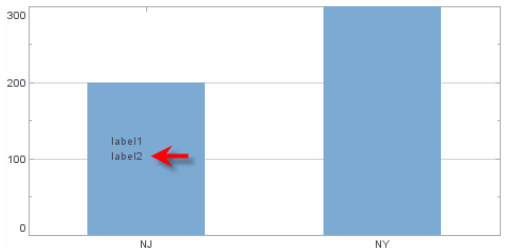
Values

```
Chart.TOP_ALIGNMENT
Chart.MIDDLE_ALIGNMENT
Chart.BOTTOM_ALIGNMENT
```

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

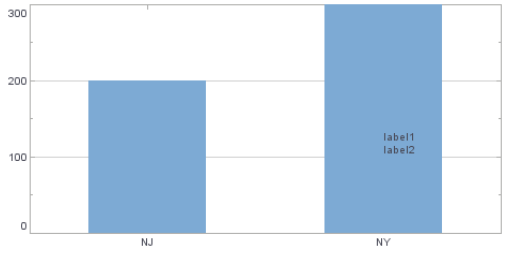
Values

```
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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

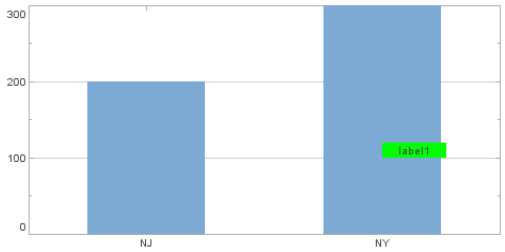
Type

value	a java.awt.Insets object

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.)

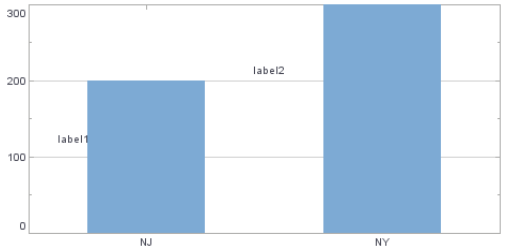
Type

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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

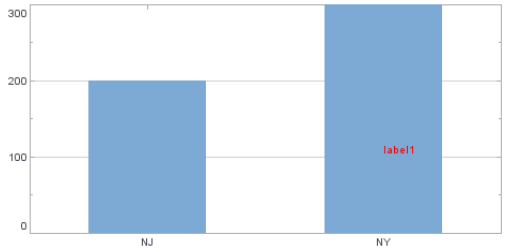
Type

spec a *TextSpec* object

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

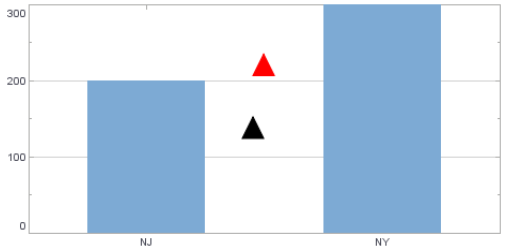
Values

```
Chart.LEFT_ALIGNMENT
Chart.CENTER_ALIGNMENT
Chart.RIGHT_ALIGNMENT
```

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

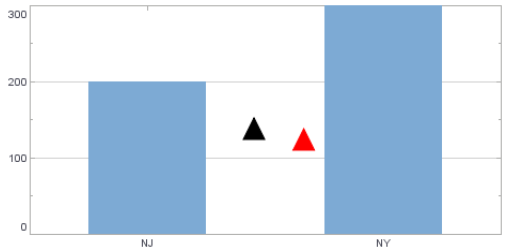
Values

```
Chart.TOP_ALIGNMENT
Chart.MIDDLE_ALIGNMENT
Chart.BOTTOM_ALIGNMENT
```

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced Dashboard Scripting* for more information.

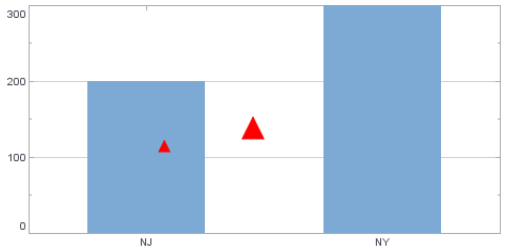
Example (Report or Viewsheet)

```
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.

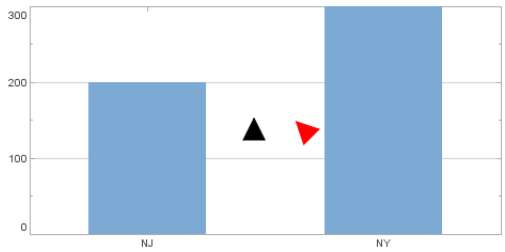
Type

Number

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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*.

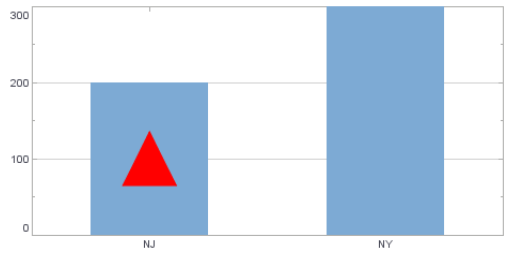
Type

GShape
SVGShape
GShape.ImageShape

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced Dashboard Scripting* for more information.

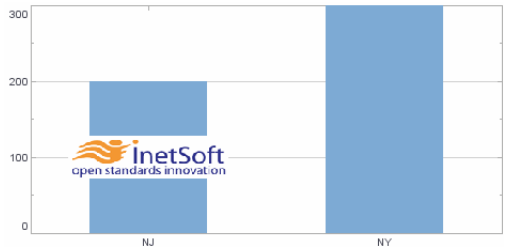
Example (Report or Viewsheet)

```
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 *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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.

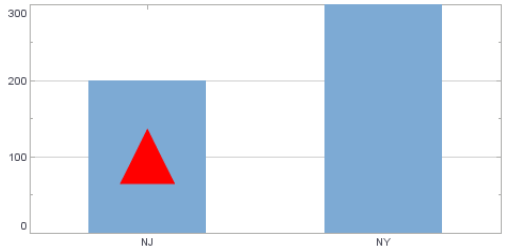
Type

value An [X,Y] pair

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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

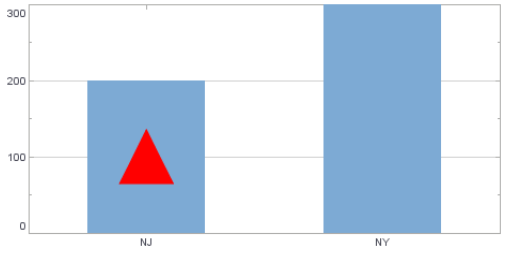
value an [X,Y] pair

Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced 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

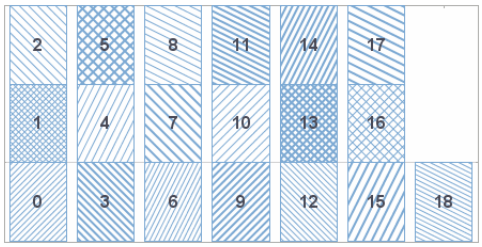
This section describes objects which provide useful constants.

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.

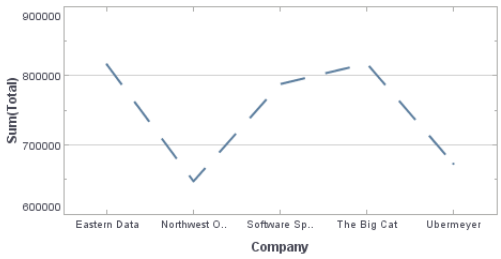
```
var line = new GLine(dashsize, width);
```

```
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 in 'Orders' > 'Sample Queries', 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");
```

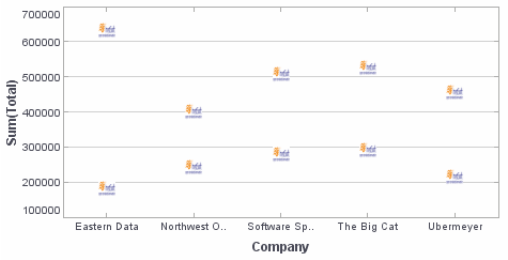
or specify this later using the *GShape.ImageShape.image* property.

```
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 in ‘Orders’ > ‘Sample Queries’, with ‘Company’ (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

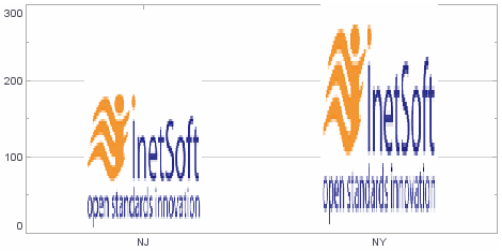
```
Chart1.bindingInfo.setShapeField('Employee',Chart.STRING)
var logo = getImage("http://www.inetsoft.com/images/home/
logo.gif");
var shape = new GShape.ImageShape();
shape.image = logo;
var frame = new StaticShapeFrame(shape);
Chart1.bindingInfo.shapeFrame = frame;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced Dashboard Scripting* for more information.

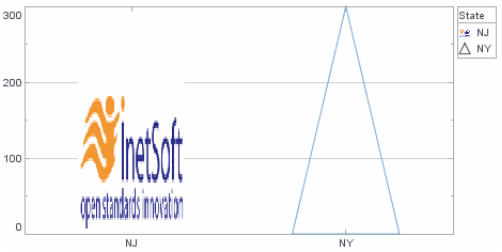
Example (Report or Viewsheet)

```
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

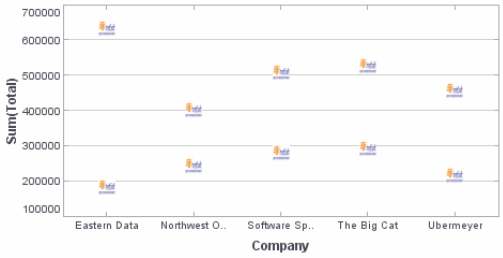
`Image object, see getImage()`

`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 in 'Orders' > 'Sample Queries', 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;
```

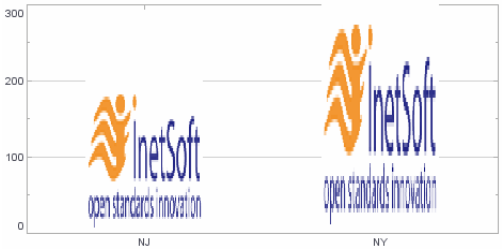


`Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding Component Script* in *Advanced Dashboard Scripting* for more information.`

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

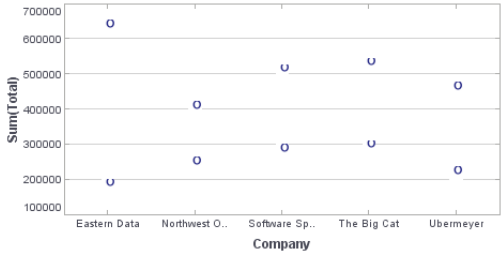
boolean	true: keep original size, and tile to fit false: stretch to fit (default)

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 in 'Orders' > 'Sample Queries', 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;
```

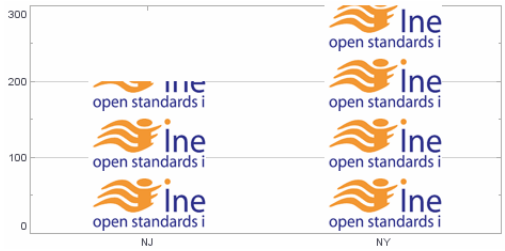


Note: Script that modifies 'graph' should be placed at the element level. See *Editing Script* in *Advanced Report Scripting* and *Adding*

Component Script in *Advanced 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

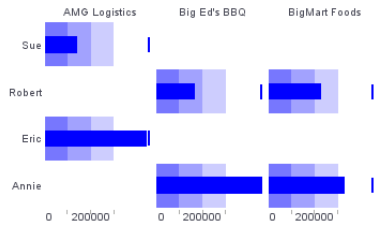
Consider the Worksheet Data Block called, ‘Query1’, shown below. (This Data Block is a mirror of the ‘All Sales’ query in ‘Orders’ > ‘Sample Queries’, with grouping on the ‘Employee’ and ‘Company’ fields, aggregation on the ‘Total’ field, and expression columns for ‘Target’, ‘range1’, ‘range2’, and ‘range3’.)

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.

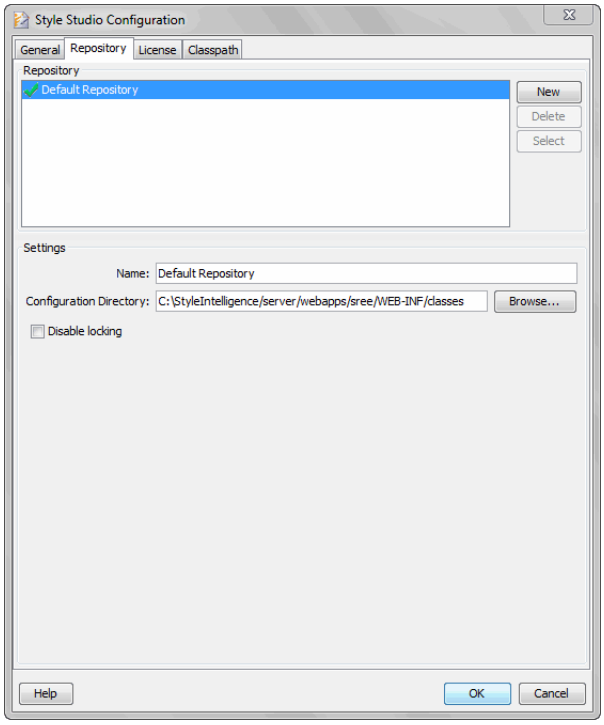
APPENDIX ST: Using Style Studio

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 and directories:

These files are accessed both by Style Studio and any Style Scope server that uses the same repository.

- **repository.xml** file: 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.d** directory: The asset registry, containing global queries, Data Worksheets, database connections, data models, virtual private models (VPMs), and Viewsheets created on the server.
- **stylereport.srl.d** directory: The library, containing reusable dashboard components (table styles, scripts, etc.).

```
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:

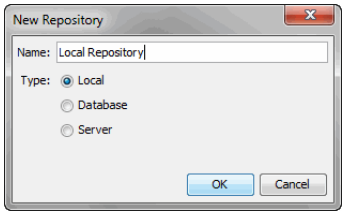
1. From the Style Studio ‘File’ menu, select ‘Configure’. This opens the ‘Style Studio Configuration’ dialog box.

2. Select the **Repository** tab.

3. On the **Repository** tab, press the ‘New’ button. This opens the ‘New Repository’ dialog box.

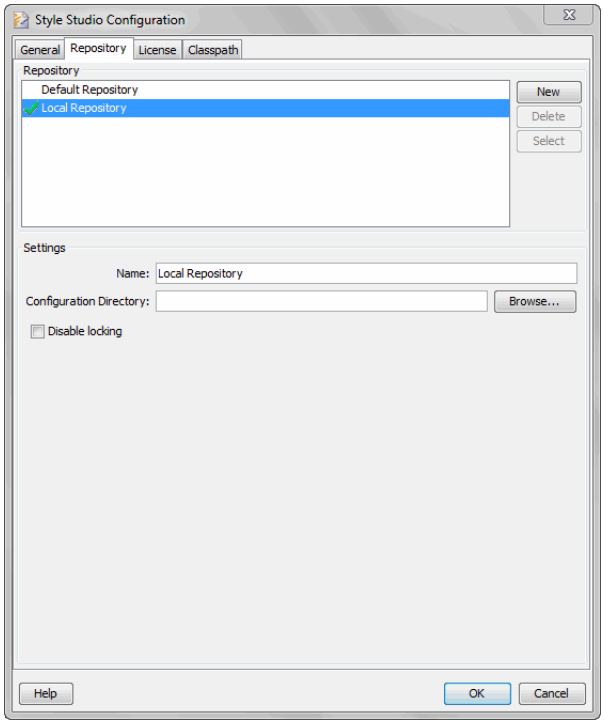
4. In the ‘Name’ field, enter the label under which the repository should be listed.

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:

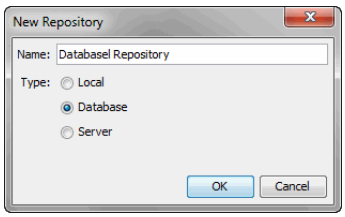
1. From the Style Studio ‘File’ menu, select ‘Configure’. This opens the ‘Style Studio Configuration’ dialog box.

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.

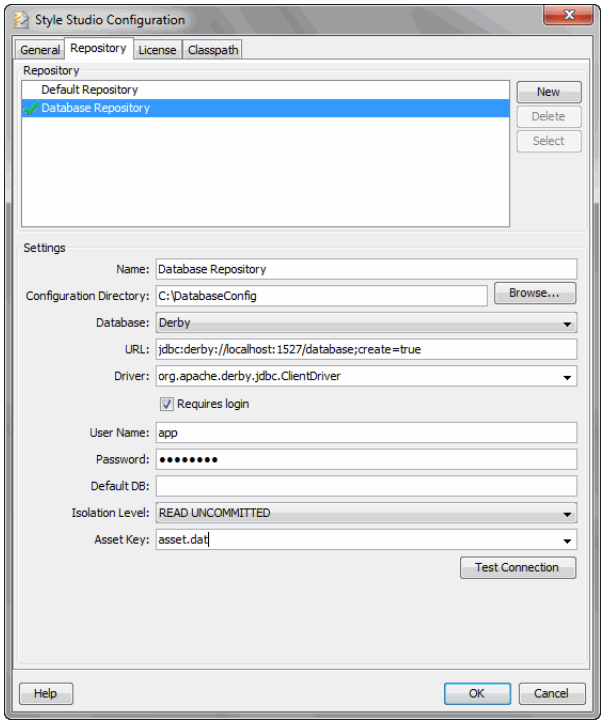
7. In the ‘Database’ menu, select the type of database to use for storage of the repository files. The options are ‘SQLServer’, ‘Oracle’, ‘Derby’, ‘DB2’, PostgreSQL, MySQL. (To use the sample audit database of the default installation, select ‘Derby’.)

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:

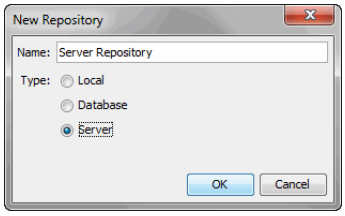
1. From the Style Studio ‘File’ menu, select ‘Configure’. This opens the ‘Style Studio Configuration’ dialog box.

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.

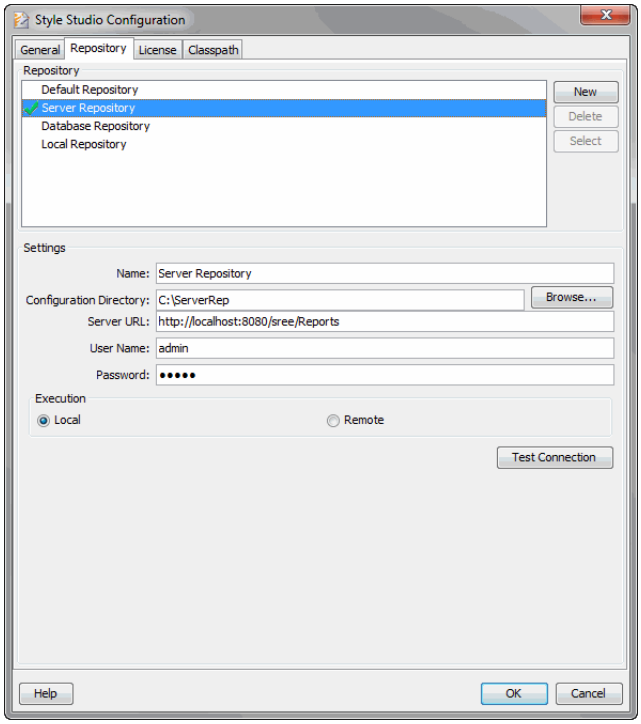
This adds a new server repository to the listing in the **Repository** tab, and automatically sets this new repository to be the working repository. (See *Selecting the Working Repository* for information on manually setting the repository.)

```
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.) When 'Remote' execution is selected, the default system web browser is used to display report previews.

   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:

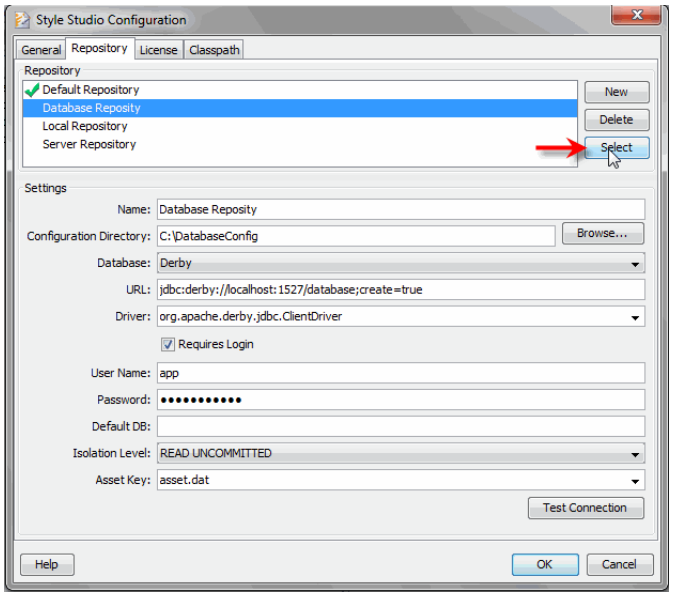
1. From the Style Studio 'File' menu, select 'Configure'. This opens the 'Style Studio Configuration' dialog box.

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*.

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 files (e.g., in *asset.dat.d*, *stylereport.srl.d*), 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

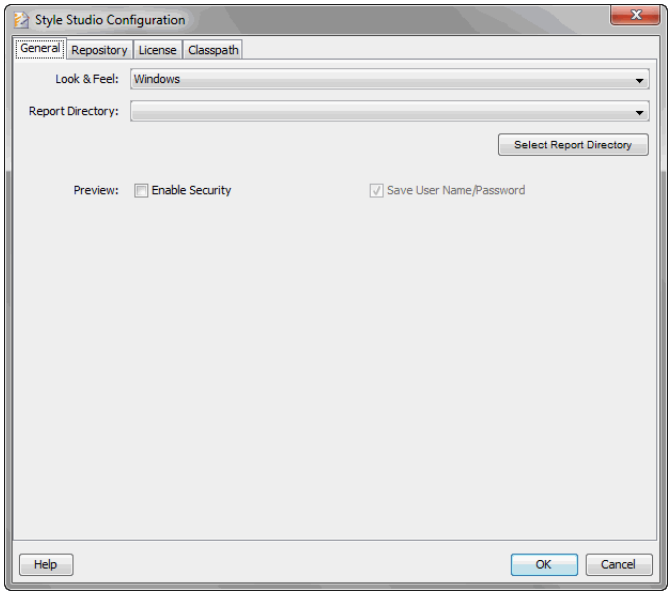
When you first install Style Scope, the repository is set to 'Default Repository', which is a 'Local' repository populated with a number of example data sources, queries, and other assets. The location of this default local repository is *server\webapps\sree\WEB-INF\classes*, which is also the default 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 (*asset.dat.d*, *stylereport.srl.d*, 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.

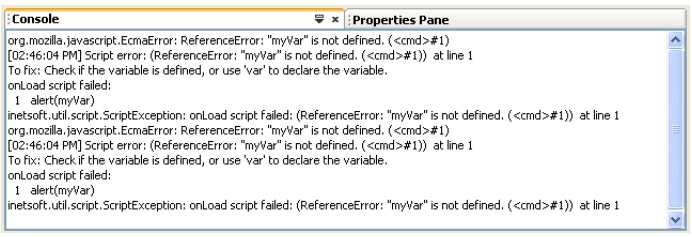
4. Press ‘Add Directory’ to add a file system directory containing resources to the classpath.

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.

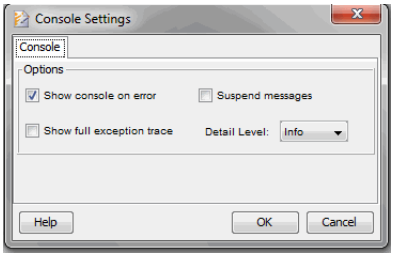
ST.2 Using the Console

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

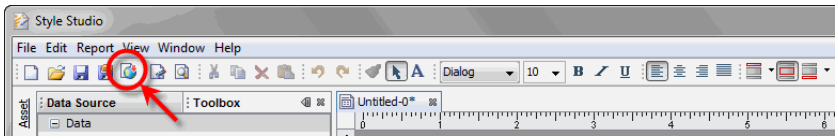
'Server', verify that the indicated database or server is ready to accept connections. See *Configuring a Repository* for more information.

ST.3.1 Deploying an Asset to a Remote Repository

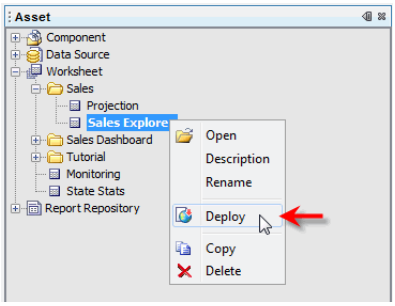
To save an asset to the *working* repository, simply press the 'Save' button in the toolbar.

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.

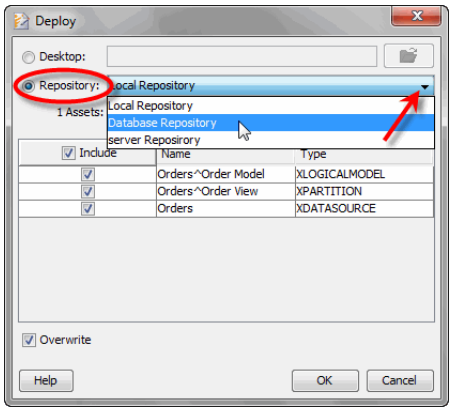


   Alternatively, expand the Style Studio **Asset** panel, and right-click the asset that you wish to deploy. Select 'Deploy' from the context menu to open the 'Deploy' dialog box.



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

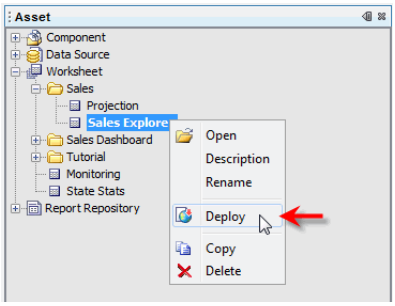
You can deploy an asset to a JAR file on the local file system. This allows you to easily transfer the bundled asset to a different environment at a later time.

To deploy an asset to a JAR file, follow the steps below:

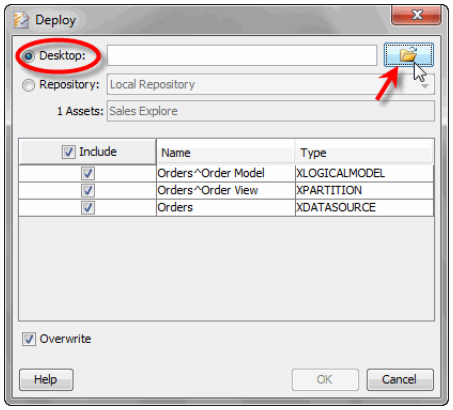
1. Open the asset in Style Studio, and press the ‘Deploy’ button in the Style Studio toolbar. This opens the ‘Deploy’ dialog box.



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.

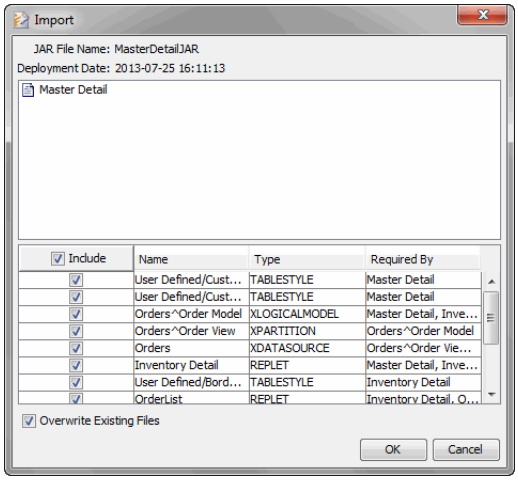
ST.3.3 Importing Assets from a JAR File

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 (Optional)* 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 and Mashing Up 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 **Repository** tab.	Can be viewed under the Portal **Repository** 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 and Mashing Up 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*.

Copyright

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, ©1996-2016 InetSoft Technology Corp. All rights reserved. All other products, names, or brands are the trademarks of their respective holders.