One Design
One Server
One User Experience

Using Actuate BIRT Designer Professional

Contains information proprietary to:
Actuate Corporation, 2207 Bridgepointe Parkway, San Mateo, CA 94404

www.actuate.com
www.birt-exchange.com

Document No. 110812-2-745301 August 3, 2011

Contents

About *Using Actuate BIRT Designer Professional*

Using Actuate BIRT Designer Professional describes how to use Actuate BIRT Designer to create reports, and the Actuate BIRT option to configure and distribute BIRT reports.

Using Actuate BIRT Designer Professional describes the additional functionality available in Actuate BIRT Designer that is not available in the open-source BIRT Report Designer. Actuate provides this functionality as extra Eclipse features that integrate into the Eclipse BIRT report designer environment. For information about the functionality shared with the open-source BIRT Report Designer, see *BIRT: A Field Guide* and *Integrating and Extending BIRT*, both published by Addison-Wesley.

Using Actuate BIRT Designer Professional includes the following chapters:

- *About Using Actuate BIRT Designer Professional.* This chapter provides an overview of this guide.
- *Part 1. Retrieving data for reports.* This part explains how to connect to various data sources and retrieve data for use in reports.
- *Chapter 1. Accessing data.* This chapter lists all the types of data sources that Actuate BIRT Designer supports, and provides an overview of how reports access data.
- *Chapter 2. Creating data objects.* This chapter describes how to create data objects to provide data for dashboards and reports.
- *Chapter 3. Accessing data in a data object.* This chapter describes how to connect to and retrieve data from a data object.
- *Chapter 4. Accessing data in an information object.* This chapter describes how to connect to and retrieve data from an information object.
- *Chapter 5. Accessing data in a report document.* This chapter describes how to connect to and retrieve data from a report document.

- *Chapter 6. Accessing data in an e.report.* This chapter describes how to connect to and retrieve data from a report developed with Actuate eReport Designer Professional.
- *Chapter 7. Accessing data in a POJO.* This chapter describes how to connect to and retrieve data from a POJO.
- *Chapter 8. Combining data from multiple data sources.* This chapter describes how to combine data from different data sets.
- *Part 2. Designing reports.* This part describes the additional design functionality available in Actuate BIRT Designer.
- *Chapter 9. Formatting a report.* This chapter describes the additional report formatting options in Actuate BIRT Designer.
- *Chapter 10. Using Flash objects in a report.* This chapter describes the requirements and methods for adding Flash objects in a report.
- *Chapter 11. Using built-in Flash charts and gadgets.* This chapter describes how to create and format Flash charts and gadgets using the Flash chart and Flash gadget builders.
- *Chapter 12. Using the Flash object library.* This chapter describes how to add Flash objects from the InfoSoft Flash object library to a report.
- *Chapter 13. Writing expressions using EasyScript.* This chapter describes how to write expressions using EasyScript, which is an expression syntax similar to the syntax used in Excel formulas. The chapter also provides a reference to the EasyScript functions and operators.
- *Chapter 14. Specifying filter conditions at report run time.* This chapter describes how to create dynamic filters and report parameters, which provide users more control over what data they see in a report.
- *Chapter 15. Adding HTML buttons to a report.* This chapter describes how to use HTML buttons to run JavaScript code.
- *Chapter 16. Controlling user access to report pages and data.* This chapter describes how to use the page-level security and data security features in Actuate iServer to control user access to particular sections in a report and a particular set of data in a data object.
- *Chapter 17. Accessing iServer environment information.* This chapter describes how to write event handlers in a report to retrieve iServer environment information.
- *Part 3. Deploying reports and resources.* This part explains how to deploy reports and resources to an Actuate iServer encyclopedia.
- *Chapter 18. Deploying BIRT reports to Actuate BIRT iServer.* This chapter describes how to use the Actuate BIRT Report option to run and distribute BIRT reports in Actuate iServer.

- *Chapter 19. Configuring data source connections in Actuate BIRT iServer.* This chapter describes how to set up and use a data source configuration file using Actuate iServer.
- *Chapter 20. Configuring fonts in Actuate BIRT iServer.* This chapter describes how to set up and use custom fonts in Actuate BIRT reports using Actuate iServer.
- *Chapter 21. Working with BIRT encryption in Actuate BIRT iServer.* This chapter describes how to set up and use report encryption using Actuate iServer.
- *Chapter 22. Using custom emitters in Actuate BIRT iServer.* This chapter describes how to provide custom output formats for Actuate BIRT reports on Actuate iServer.
- *Part 4. Using Actuate BIRT APIs.* This part explains how to use classes in the com.actuate.birt.* public packages.
- *Chapter 23. Using the BIRT data object API.* This chapter describes how to work with BIRT data objects and report designs programmatically.

Part One

Retrieving data for reports

Chapter

1

Accessing data

This chapter contains the following topics:

- Supported data sources
- How a report accesses data

Supported data sources

Actuate BIRT Designer supports all the types of data sources that open-source BIRT Report Designer supports, and more. Table 1-1 lists the types of data sources that each product supports.

Table 1-1 Supported data source types

Data source type	Actuate BIRT Designer	BIRT Report Designer
Actuate data object	✓	
Actuate information object	✓	
ActuateOne for e.Reports	✓	
BIRT report document	✓	
Flat file	✓	✓
JDBC	✓	✓
JDBC connection for Query Builder	✓	
Plain Old Java Object (POJO)	✓	
Scripted	✓	✓
Static	✓	
Web service	✓	✓
XML document	✓	✓

Actuate data objects, Actuate information objects, and BIRT report documents are data files that report developers or data architects create with Actuate BIRT Designer. These files contain the information to connect to and retrieve data from enterprise data sources, such as databases and web applications.

ActuateOne for e.Reports is a data driver that supports the retrieval of data from reports developed using Actuate e.Report Designer Professional.

Like the JDBC data source, the JDBC connection for Query Builder data source supports the retrieval of data from JDBC databases. However, instead of typing a SQL query to select the data to retrieve, you use a graphical query builder to construct the SQL query.

A static data source is a set of data that you create in Actuate BIRT Designer. This type of data is useful when you need to create sample data quickly for testing purposes.

How a report accesses data

A report uses the same mechanism to access data from any of the sources listed in Table 1-1. First, you create a data source, which is a BIRT object that contains the information to connect to an underlying data source. Each type of data source requires different connection information. For a JDBC data source, for example, you specify the driver, URL, and user login to connect to a database. An XML data source requires the location of the XML file and, optionally, the location of the XML schema.

Next, you create a data set, which is a BIRT object that specifies and returns all the data that is available to a report. For a JDBC data source, for example, you write a SQL query or run a stored procedure to retrieve specific data from a database. For an XML data source, you use XPath expressions to specify the XML elements and attributes from which to retrieve data.

This book provides instructions for accessing data from data sources that only Actuate BIRT Designer supports. For information about accessing data from data sources that both BIRT products support, see *BIRT: A Field Guide*.

Chapter

2

Creating data objects

This chapter contains the following topics:

- About data objects
- Design considerations
- Building a data object
- Providing cached data
- Publishing a data object
- Managing user access
- Maintaining a data object

About data objects

A data object is a BIRT object that contains all the information necessary to connect to an external data source, retrieve data from that data source, and structure the data in a way that supports business analysis. Data objects are similar to data marts, which are simplified repositories of data gathered from corporate data sources and designed to address specific business queries.

Data architects or report developers create data objects to provide data for the following items:

- Dashboards, which users create using Actuate BIRT 360 Studio, a dashboard application on Actuate BIRT iServer
- BIRT reports that are created using either Actuate BIRT Designer or Actuate BIRT Studio on iServer

Data objects use Actuate's in-memory analytics technology, which loads data in memory to speed up the processing of data.

Design considerations

A data object is a collection of the following BIRT objects:

- Data sources
- Data sets
- Data cubes
- Report parameters

A data object can include any number of data sources, data sets, data cubes, and report parameters. Although it is possible to create a single data object that contains all the data that dashboard users or report developers could possibly need, a data object that provides too much data can be confusing for users. In addition, the amount of memory that a data object uses increases with the number of data rows returned by data sets and the number of aggregations calculated by cubes.

If creating data objects for diverse groups of users or reports, evaluate how best to organize data into data objects and how much data to include in each data object.

The objects to include in a data object depend on which item—dashboard or BIRT report—is using the data object. Table 2-1 lists the objects that you typically include when creating a data object for a dashboard and for a report.

Table 2-1 Typical objects in a data object for a dashboard and a report

	Data source	Data set	Cube	Report parameter
Dashboard	✓	✓	✓	✓
BIRT report	✓	✓	✓	

The following sections describe in more detail the guidelines for designing data objects for dashboards and reports.

Designing data objects for dashboards

Actuate BIRT 360 is a web application designed for business users who want to measure the effectiveness of their business processes and have this decision-support information displayed graphically in a dashboard. The data objects you create for dashboards need to extract the right data and provide it in a structure suitable for dashboard gadgets.

Use the following guidelines when designing data objects for dashboard users:

- Identify the users and create a data object or series of data objects for each user group.

  A typical approach is to create data objects by groups of users, where each data object fulfills a different business need. For example, executives, sales managers, and customer support managers represent three distinct user groups with different data requirements. Executives might be interested in viewing revenue by month or quarter. Sales managers might need to evaluate the sales numbers of individual sales representatives by month or quarter. Customer support managers might need to monitor support call volume by days. Depending on how iServer user accounts are set up, you might be able to leverage the defined user roles and groups to identify the user groups by which to organize data objects. Contact the iServer administrator for this information.

- Provide users with sufficient data that they can use to analyze by different dimensions and at different levels of detail.

  Users often need to view data from different dimensions. For example, sales managers might need to view sales by product line, by region, or by sales representative, and by different time periods. If viewing sales data by region, sales managers might need to drill down to view sales by cities within each region.

- Design data sets and cubes to provide data that is suitable for the gadgets that will be used to display data.

  The dashboard provides a suite of gadgets for displaying data. Each gadget accesses data in the same way as the corresponding element does in a BIRT report.

  - All chart gadgets use data from either a data set or a cube.

- The cross tab gadget uses data from a cube.
- The table and summary table gadgets use data from a data set. In addition, summary tables require that each column in the data set be assigned the appropriate analysis type to provide the expected functionality. For more information, see "Configuring data set columns for summary tables," later in this chapter.

- Design data sets or report parameters to provide lists of values to display in data selector gadgets. Just as a report parameter supports run time filtering of data in a report, a data selector gadget enables a dashboard user to filter data in a chart, cross tab, table, or any other gadget that displays data.

Figure 2-1 shows a dashboard that uses five gadgets to display sales data. Descriptions of each gadget follow the illustration.

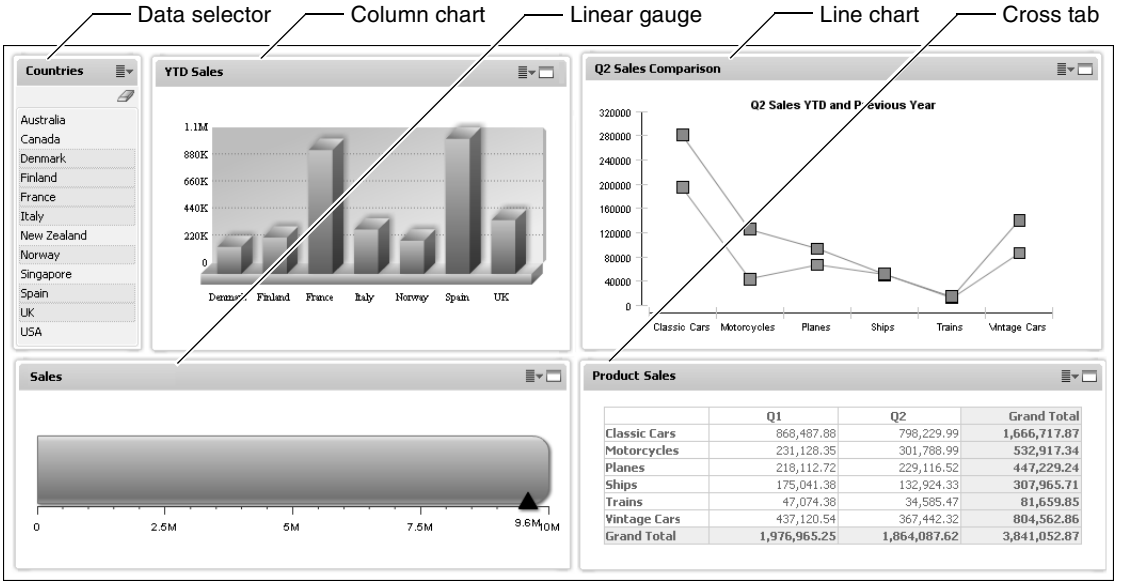


Figure 2-1 Sample dashboard displaying five gadgets

- The data selector gadget displays a list of countries. The data selector is linked to the column chart next to it. Dashboard users select values from the data selector to filter the data to display in the column chart. The data selector gets its values from a data set, a cube, or a report parameter.
- The column chart gadget is linked to the data selector, as described previously. The column chart derives its data from a data set or a cube.
- The line chart gadget derives its data from a data set or a cube.
- The linear gauge gadget derives its data from a data set or a cube.

- The cross tab gadget derives its data from a cube.

Designing data objects for reports created with Actuate BIRT Studio

Actuate BIRT Studio is a web-based report design tool on iServer designed for users who want to create reports quickly and easily without deep understanding of database architecture or report design techniques. BIRT Studio users create reports using predefined data sources and templates that provide the data and basic layout for their reports.

The data objects you create should provide the data in a structure that is appropriate for business users and for the report elements that users can add to a report. The design guidelines discussed in the previous section apply here as well, except for the following:

- In BIRT Studio, a chart uses data from a table in a report. The chart does not use data directly from a data set or a cube.
- Report parameters in a data object do not link to parameters created in BIRT Studio, so you typically do not include report parameters in a data object that you create for BIRT Studio users.

Designing data objects for reports created with Actuate BIRT Designer

A data object can be used as a data source for BIRT reports. As an alternative to defining data sources and data sets for each report, you can create a data object that multiple reports share. This principle is similar to reports sharing data sources and data sets stored in a library. One design option is to create data objects by types of reports. For example, if creating a set of sales reports and a set of customer reports, create one data object to provide data for the first set of reports and another for the second set of reports.

Report parameters in a data object cannot be reused in a report. A report parameter in a data object acts as a filter. You are prompted to specify a parameter value when you create a data source based on the data object, and the data source returns only the rows that meet the filter criteria.

Building a data object

Building a data object entails creating a data object file, then adding data sources, data sets, and cubes to the data object. To add these data items, you can:

- Create new data items within the data object.
- Export data items in reports or libraries to the data object.

How to create a data object

1. In the Report Design perspective, choose File➤New➤Data Object.
2. In New Data Object, do the following:
   1. Select the folder in which to store the data object.
   2. Edit the default file name to specify a new name. The extension must be .datadesign and the file name must not contain the following characters:

      ```
      [ ] * / \ : & ?
      ```

      Use a descriptive name that enables users to determine the contents of the data object. A descriptive name is particularly important if users have access to multiple data objects.
   3. Choose Finish. The report editor displays a blank data object design.
3. Start adding data sources, data sets, and cubes to the data object.

Creating new items for a data object

The procedures for creating data sources, data sets, cubes, and report parameters for a data object are the same as the procedures for creating these items for a report. Use Data Explorer to create, edit, and delete data items in a data object. Figure 2-2 shows a data object that contains one data source, two data sets, and two cubes.

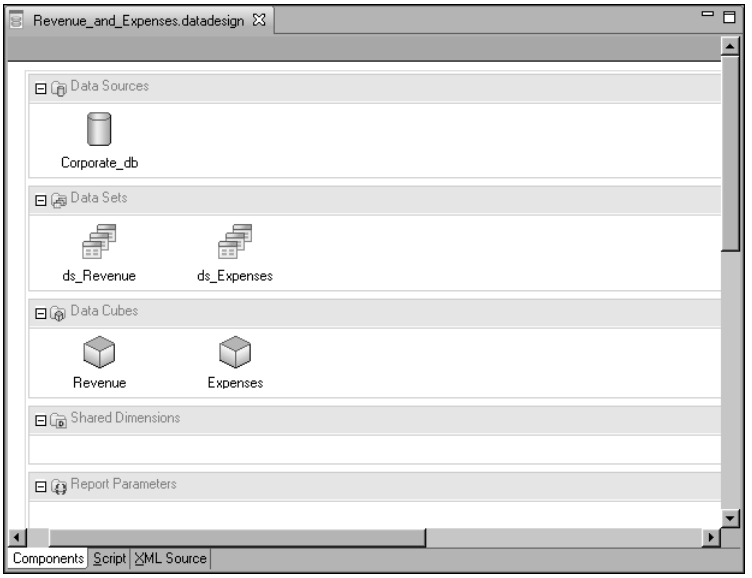


Figure 2-2 Contents of a data object

In this data object example, the Corporate_db data source connects to a corporate database. The ds_Revenue and ds_Expenses data sets retrieve data from the database. The Revenue and Expenses cubes use data from the ds_Revenue and ds_Expenses data sets, respectively. For information about creating data sources, data sets, and cubes, see *BIRT: A Field Guide*.

Exporting items to a data object

An organization that creates and uses BIRT reports has data sources, data sets, and cubes on hand. As a data object designer, you can reuse these items by exporting them from a report or a library to a data object.

The export utility copies the items to the data object. The exported items do not reference the original items in the report or the library. The export utility also detects and copies dependent items. For example, if you export a cube, the utility also exports the data source and data set that the cube uses.

Be careful when exporting multiple data sources, data sets, or cubes from different sources. If the data object contains an item with the same name, BIRT warns of the name conflict and asks whether or not to overwrite the item in the data object. Overwrite the item only if you want to replace it. The overwrite action cannot be undone. If you select a data set or a cube to export, and BIRT displays the name-conflict message, the duplicate names apply not only to the selected data set or cube, but to any dependent item. For a cube, for example, the data set or data source used by the cube might have the same name as a data set or data source in the target data object.

How to export data items to a data object

1. Open the report or library that contains the data items to export to a data object.
2. In Data Explorer, right-click a data item, then choose Export to Data Object, as shown in Figure 2-3.

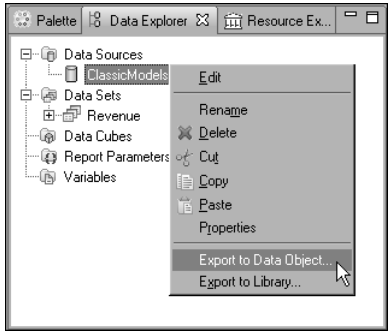


Figure 2-3 Exporting a data source to a data object

Export Elements to Data Object—Select Data Object displays the data objects (.datadesign files) in the resource folder, if any exists, as shown in Figure 2-4. By default, BIRT uses the current project folder as the resource folder.

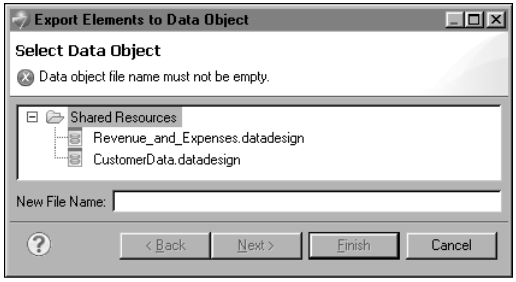


Figure 2-4 Export Elements displaying the data objects in the resource folder

3 Perform one of the following steps:

- Select an existing data object to which to export data items.
- Create a new data object by specifying a file name in New File Name. BIRT saves the data object in the resource folder.

4 Choose Next.

5 In Export Elements to Data Object—Select Elements, shown in Figure 2-5, select one or multiple data items to export. If you select a data set, BIRT also exports the data source that the data set uses. Similarly, if you select a cube, the associated data set and data source are exported.

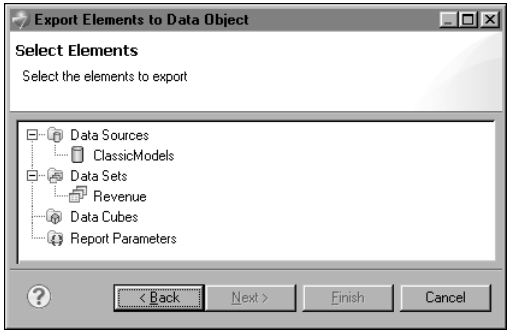


Figure 2-5 Export Elements displaying the data items that you can export

6 Choose Finish.

If BIRT does not detect any name conflicts between the items selected for export and the items in an existing data object, BIRT exports the items and asks if you want to open the data object.

7 Open the data object to review its contents. The exported data items appear in the data object.

Creating a shared dimension for cubes

If designing multiple cubes that contain the same dimension, create a shared dimension. For example, if one cube contains sales data by country, state, and city, and another cube contains budget data by country, state, and city, you can create a shared multi-level dimension that provides country, state, and city data. By using a shared dimension, you define and maintain dimension data in one place, and reuse the dimension in multiple cubes. Reusing a dimension also speeds up data processing.

There are two ways to create a shared dimension. You can:

- Create a new shared dimension.
- Convert an existing cube dimension into a shared dimension.

How to create a shared dimension

This procedure assumes that you have already created a data object, as well as, the data set that provides the data for the dimension.

1. Open the data object.
2. In Data Explorer, right-click Shared Dimensions, then choose New Shared Dimension.
3. In Dimension Builder, specify the following information:
   - In Dataset, select the data set that contains the columns to use in the dimension.
   - In Available Columns, drag a column and drop it in the following location:

     ```
     (Drop a field here to create a group)
     ```

   - If creating a multi-level dimension, drag and drop additional columns. Figure 2-6 shows an example of a multi-level dimension that contains country, state, and city data.

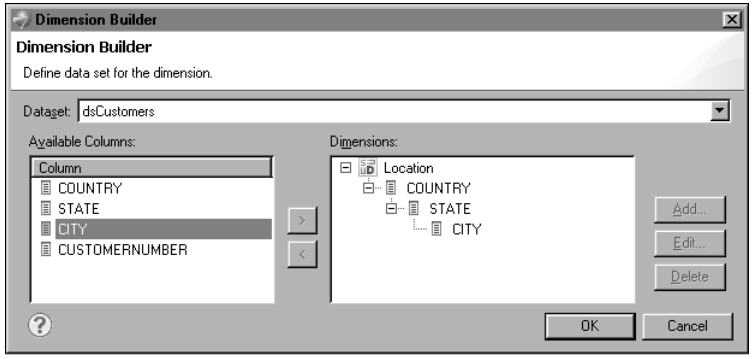


Figure 2-6 Dimension Builder displaying a defined shared dimension

Choose OK. The shared dimension appears under Shared Dimensions in Data Explorer and in the data object design.

How to convert a cube dimension into a shared dimension

This procedure assumes that you have already included cubes in a data object.

1. Open the data object.
2. In Data Explorer, expand the cube that contains the dimension to convert to a shared dimension, then right-click the dimension and choose Convert to Shared Dimension, as shown in Figure 2-7.

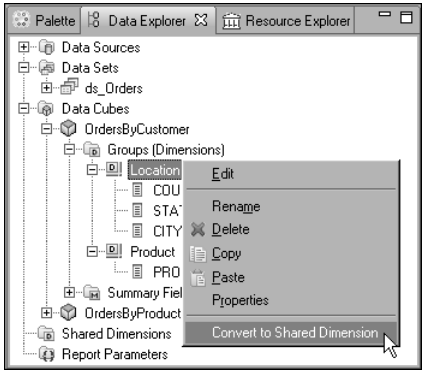


Figure 2-7 Converting a dimension to a shared dimension

The converted dimension appears under Shared Dimensions in Data Explorer and in the data object design, as shown in Figure 2-8. BIRT also replaces the original cube dimension with the shared dimension.

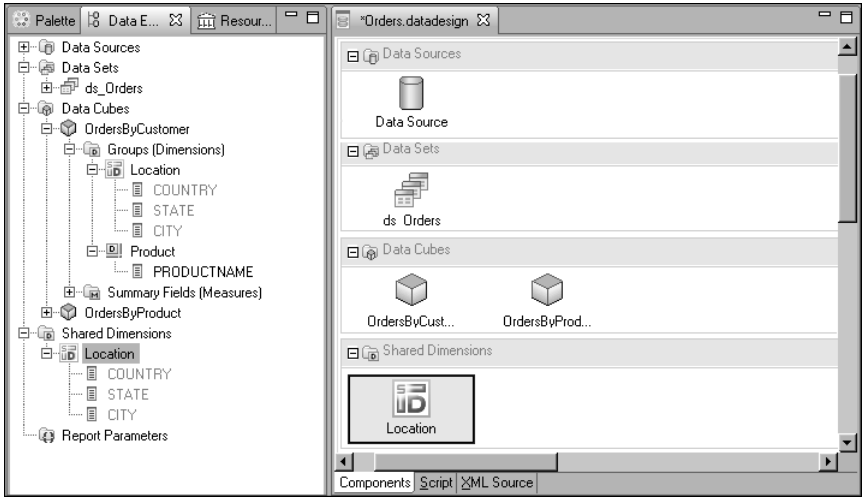


Figure 2-8 Data Explorer and data object showing the shared dimension

Configuring data set columns for summary tables

Summary tables are commonly used in dashboards and BIRT Studio reports to display key summary information. Figure 2-9 shows an example of a summary table created in BIRT Studio. Order data is grouped by date, in quarterly intervals, and by product line. The price of each order is summed to display subtotals for each quarter and product line, as well as, a grand total.

ORDERDATE	PRODUCTLINE	SUM (EXTENDED_PRICE)
Q1, 2003		
	Classic Cars	$25,833.14
	Trucks and Buses	$8,880.80
	Vintage Cars	$34,419.96
Sub Total (Q1, 2003)		*$69,133.90*
Q2, 2003		
	Classic Cars	$69,318.85
	Trains	$4,326.00
	Trucks and Buses	$22,602.03
	Vintage Cars	$33,303.52
Sub Total (Q2, 2003)		*$129,550.40*
Q3, 2003		
	Classic Cars	$168,294.05
	Trains	$8,914.09
	Trucks and Buses	$23,549.46
	Vintage Cars	$77,945.84
Sub Total (Q3, 2003)		*$278,703.44*
Q4, 2003		
	Classic Cars	$348,705.62
	Trains	$15,490.92
	Trucks and Buses	$77,538.49
	Vintage Cars	$188,673.66
Sub Total (Q4, 2003)		*$630,408.69*
Grand Total		*$1,107,796.43*

Figure 2-9 Summary table

To create a summary table quickly, users select a table's auto-summarize feature, then select the data set columns whose data to group and aggregate. When the auto-summarize feature is enabled, the software performs the grouping and aggregating. To support this feature, you must configure each data set column to provide the software with the appropriate information to perform these tasks. For example, it makes sense to group sales data by order date or product line, but not by revenue. Conversely, it makes sense to aggregate revenue values, but not order date or product line values.

To provide the appropriate information to generate a summary table, set the Analysis Type property of each data set column to one of the following values:

- Dimension

  Use this analysis type to support the grouping of data in the column. For example, to display revenue by product line, set the product line column as a dimension.

- Attribute

An attribute describes the items associated with a dimension. For a product dimension, for example, attributes might include color, size, and price. When you set a column as an attribute, you must also specify the dimension column of which it is an attribute. The summary table cannot group data in an attribute column.

- Measure

  Use this analysis type to support the aggregating of values in the column. For example, to calculate revenue totals, set the revenue column as a measure.

If you do not set a column's analysis type, BIRT Studio and the dashboard assign a value using the following criteria:

- If the column contains numeric values, the analysis type is measure.
- If the column contains date values, the analysis type is dimension.
- If the column contains string or Boolean values, the analysis type is attribute.
- If the column is a primary key, a foreign key, or an indexed column in a database, the analysis type is dimension even if the column contains numeric values.

In many cases, however, the default analysis type values do not provide usable data for a summary table. To create a well-designed data object, it is necessary to assign an analysis type for every column in the data set. The default values have the following limitations:

- Most columns containing data that users want to group, such as region and product line, are of string type. The default analysis type for these columns is attribute, and data in attribute columns cannot be grouped.
- Although BIRT assigns the attribute type to a column that contains string and Boolean data, it cannot specify the dimension column to which the attribute column is associated. As a result, the attribute column does not provide data that is usable in a summary table.
- Not all numeric data should be measures. Order or customer IDs, for example, are typically of numeric type, but, because it does not make sense to aggregate this type of data, a more appropriate analysis type is dimension or attribute.

How to set the analysis type of a data set column

This procedure assumes that you have already created a data object and added a data set to it.

1. Open the data object.
2. In Data Explorer, double-click the data set to edit it.
3. In Edit Data Set, choose Output Columns, then double-click the column whose analysis type to set. Edit Output Column displays the properties of the selected column, as shown in Figure 2-10.

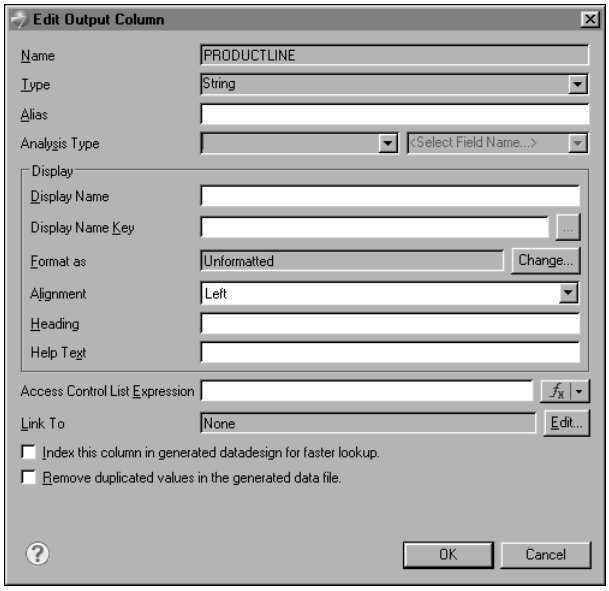


Figure 2-10 Properties of a data set column

4 In Analysis Type, select Dimension, Attribute, or Measure. If you select Attribute, in the list box that displays <Select Field Name...>, select a column of which this column is an attribute. The column you select must be a dimension column.

Creating hyperlinks to provide drill-down capability

Hyperlinks are commonly used in reports to enable users to find related information or drill down to more detailed data. For example, a summary report that displays sales totals by region can use hyperlinks to link each region to another report that displays detailed sales data. To provide this functionality in a dashboard or in a report, create hyperlinks in the following items in a data object:

- Data set columns
- Dimensions and measures in a cube

You can create the following types of hyperlinks:

- Drill-through, to link to a bookmarked location in a report
- URI, to link to a document or a web page

Figure 2-11 shows an example of a drill-through hyperlink definition that specifies a link to a bookmark, row["COUNTRY"], in a target report named SalesByCountryAndProduct.rptdesign.

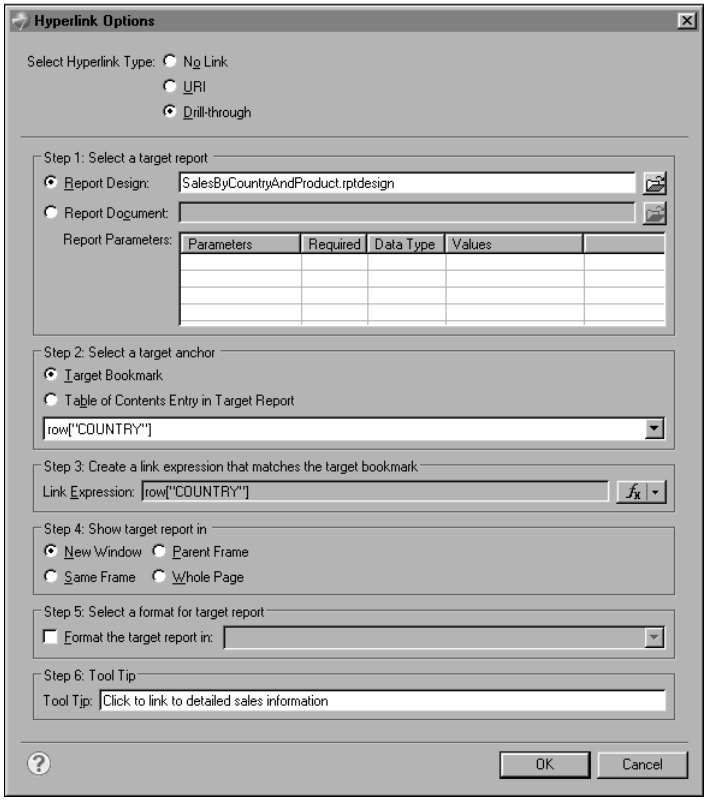


Figure 2-11 Definition of a drill-through hyperlink to link to a report

Figure 2-12 shows an example of a URI hyperlink definition that specifies a link to a document.

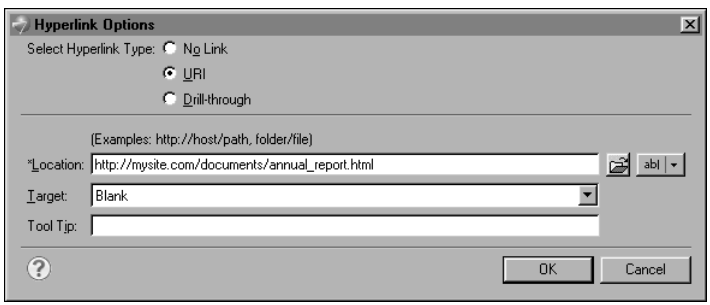


Figure 2-12 Definition of a hyperlink that uses a URI to link to a document

How to create a hyperlink from a data set column

This procedure assumes that you have already created a data object and added a data set to it.

1 Open the data object.

2 In Data Explorer, double-click the data set to edit it.

3 In Edit Data Set, choose Output Columns, then double-click the column to which to add a hyperlink. Edit Output Column displays the properties of the selected column, as shown in Figure 2-13.

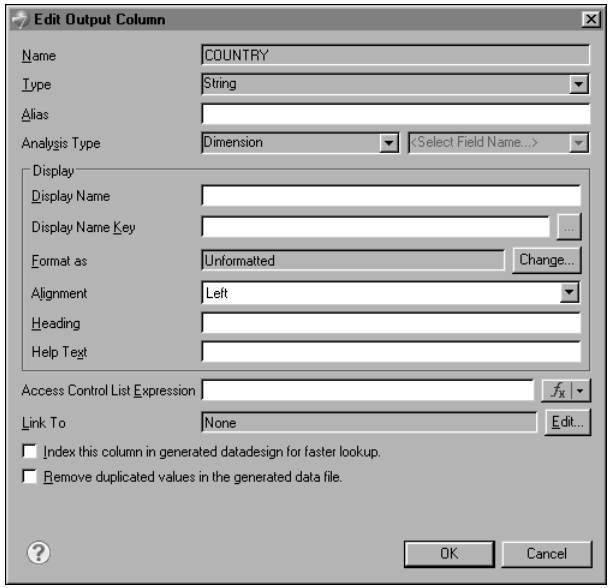


Figure 2-13 Edit Output Columns displaying the properties of a column

4 Choose Edit next to the Link To property.

5 In Hyperlink Options, select the type of hyperlink to create, then set the properties of the hyperlink. These steps are the same as the steps for defining a hyperlink in a report, and are described in *BIRT: A Field Guide*.

How to create a hyperlink from a dimension or measure in a cube

This procedure assumes that you have already created a data object and added a cube to it.

1 Open the data object.

2 In Data Explorer, double-click the cube to edit it.

3 In the cube builder, choose Groups and Summaries.

4 Under Groups and Summaries, double-click the dimension or measure to which to add a hyperlink.

The properties of the selected dimension or measure appear. Figure 2-14 shows the properties of a measure.

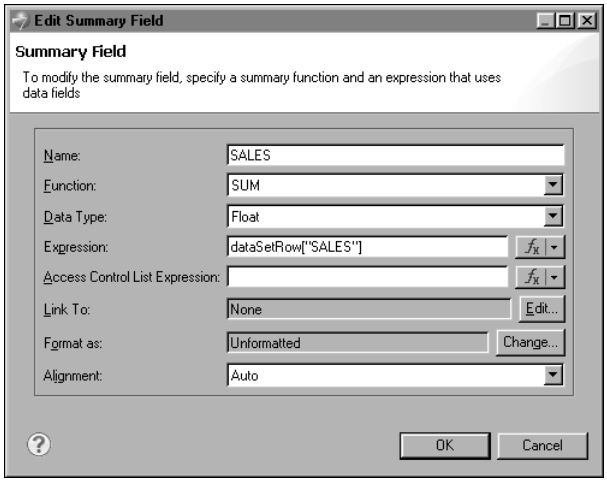


Figure 2-14 Edit Summary Field displaying the properties of a measure

5 Choose Edit next to the Link To property.

6 In Hyperlink Options, select the type of hyperlink to create, then set the properties of the hyperlink. These steps are the same as the steps for defining a hyperlink in a report, and are described in *BIRT: A Field Guide*.

Hiding data sets from users

When you publish a data object in iServer, dashboard and BIRT Studio users who are granted access to the data object can select any of the data sets and cubes in the data object as a source of data for their dashboard gadget or report. However, not all data sets return data that is suitable for a dashboard or a report.

Some data sets are created to provide data specifically for a cube or a report parameter, and the data usually is not useful for a dashboard or report. For example, a data set created for a report parameter that displays a list of countries would return only values from a country column. For some cubes, multiple data sets are linked to provide data for the cube, and the individual data sets do not provide sufficient data for a dashboard or report.

To present to users only data sets and cubes that are designed to provide data for dashboards and reports, you should hide data sets that only provide limited data for a report parameter or a cube.

How to hide data sets from users

1 In the data object design, right-click the data set to hide, and choose Edit.

2 In the data set editor, choose Settings.

3 In Visibility, deselect Include this data set in the generated data object store, as shown in Figure 2-15.

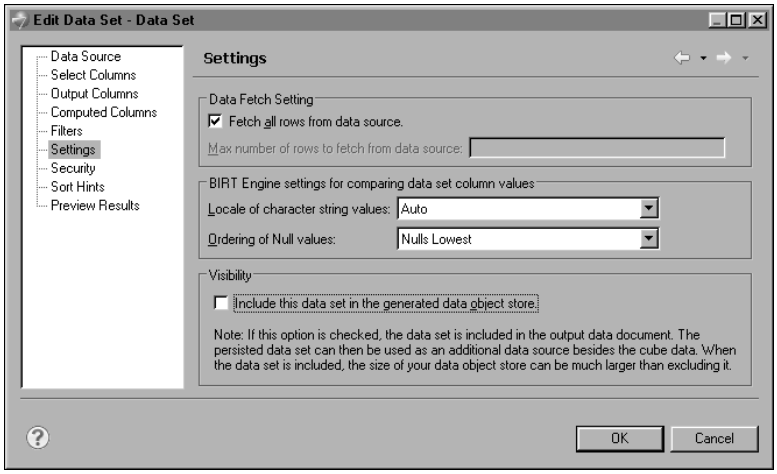


Figure 2-15 Data set editor displaying the Settings page

Providing cached data

When a dashboard or a report uses a data object, each time the dashboard is refreshed or the report is run, the data object connects to the underlying data source and retrieves data from it. This operation is typically resource-intensive. For a more efficient data access alternative, cache the data in a data object store, and provide dashboard and report users with access to the data object store.

How to create a data object store

1. Open the data object design (.datadesign) file.
2. Choose Run➛Generate Data Objects.
3. In Generate Data Objects, type a file name for the data object store. The file name must have a .data extension. Choose OK.

   BIRT creates a data object store (.data file) that contains the materialized data. The file is saved in the same folder in which the data object design file resides.

Publishing a data object

To make data objects available to report developers using Actuate BIRT Designer, place the data objects in the shared resource folder, just as you do with other shared resources, such as libraries and style sheets.

To make data objects available to dashboard and BIRT Studio users, you must publish the data objects in an iServer Encyclopedia volume. From the iServer

perspective, a data object is a resource, similar to Java classes, image files, or libraries, which are resources for reports published in iServer.

You can publish data object design (.datadesign) files or data object stores (.data). A common strategy is to publish the design files, then schedule those files to be run regularly to generate the data object stores. To manage system resources effectively, iServer volume administrators can make the data object stores generally available to users while limiting access to the data object design files.

How to publish a data object in an iServer volume

1. Create an iServer profile, which specifies the information to connect to an iServer volume.
2. Place the data object in the BIRT resource folder. The location of the resource folder is specified in the Preferences page, which you access by choosing Windows➤Preference from the main menu, then choosing Report Design—Resource. The default location is the current project folder.
3. Choose File➤Publish Resources to iServer, then select the data object to publish.

For more information about creating an iServer profile, assigning a resource folder in iServer, and publishing items to an iServer volume, see "Deploying BIRT reports to Actuate BIRT iServer," later in this book. For information about managing files in iServer, see *Managing an Encyclopedia Volume*.

Enabling incremental updates

Data object stores (.data) containing cached data are typically updated regularly to provide the latest data to dashboards and reports. In some cases, you can speed up the generation process by using the incremental updates option. Instead of retrieving the full set of data rows each time the data object store is generated, the incremental updates option retrieves only additional data rows that meet specified criteria.

Use this option to add new data rows on a regular basis to a large set of legacy data. For example, a data object retrieves order information. When the data object is generated initially, it contains all the order information to date. Each week, a new data object store is generated to capture new order data. For cases like this, adding only the new data each week improves performance significantly.

Observe the following guidelines when designing a data object that uses the incremental update option:

- For each update, you must specify which data rows to add to an existing data object store. To accomplish this task, create a parameter in the data object. In the example described previously, the data object would use a date-time parameter to specify which week of data to retrieve.

- Data rows can only be added to the result set returned by a data set, provided that the data set definition does not change between updates. If you change the definition of a data set, for example, by adding or deleting a column, you must generate a new data object store without using the incremental updates option.
- If the data object contains a cube and you want the cube to include the new data set rows, or if you change the cube definition, you must also generate a new data object store without the incremental updates option.

How to enable incremental updates

This procedure assumes that you have already created the data object design (.datadesign).

1. In the layout editor, right-click in an empty area of the data object design.
2. Choose Enable Incremental Update.

After you publish this data object to iServer, incremental updates occur each time the data object is generated. Depending on the job properties that you specify, iServer replaces the existing .data file with the updated version or maintains multiple versions of the .data file. Dashboards and BIRT Studio reports configured to use the latest .data file display the new data when users refresh the dashboard or report.

If you share the data object in a file system with other report developers using Actuate BIRT Designer, you replace the .data file manually.

How to apply incremental updates to a .data file stored in the file system

This procedure assumes that you have already created the data object design (.datadesign) and generated the initial data object store (.data).

1. In the layout editor, right-click in an empty area of the data object design, then choose Enable Incremental Update.
2. Choose Run➛Generate Data Objects.
3. In Incremental Update, do the following:
    1. Select Use incremental update.
    2. Select the .data file to update.
    3. In Target data file name, type a temporary .data file to which to write the new data. This step is required because a file saved in the file system cannot be read and written to at the same time.

       Figure 2-16 shows an example of values specified in Incremental Update. In this example, Employees.data is the file to update, and Employees_Temp.data is the temporary file to which to write the new data.

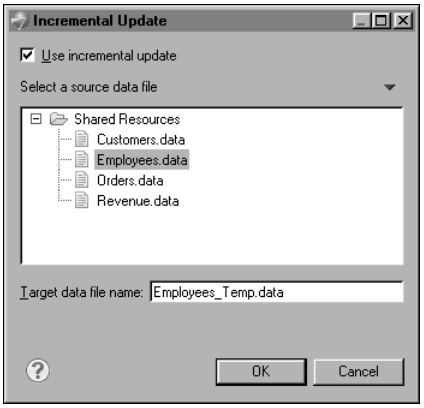


Figure 2-16 Incremental Update

4 Choose OK.

4 In Navigator, delete the original .data file. In the example shown in Figure 2-16, you would delete Employees.data.

5 Rename the temporary .data file to the name of the original .data file. In the example shown in Figure 2-16, you would rename Employees_Temp.data as Employee.data.

Managing user access

When you publish a data object to an iServer Encyclopedia volume, you grant specific users or user groups access to the data object. This type of security is implemented in iServer, and users either have access to all the items in the data object, or none at all. For information about implementing security in iServer, see *Managing an Encyclopedia Volume*.

Using Actuate BIRT Designer in conjunction with iServer, you can apply more granular security rules to control user access to individual items in a data object, down to which data set columns, cube dimensions and measures, and data rows are available to particular groups of users. For information about implementing this type of security, see Chapter 16, "Controlling user access to report pages and data," later in this book.

Maintaining a data object

Changes you make to any item in a data object propagate to the reports and dashboard gadgets that use that item. For example, if you add a dimension to a cube in a data object, all reports and gadgets that use that cube have access to the

new dimension. This automatic-update functionality is useful for applying necessary updates, such as connection properties in a data source, to all reports and gadgets.

However, the automatic-update functionality also means that you have to be careful with the type of change that you make. The following changes can cause errors in reports and gadgets:

- Renaming a data object, data source, data set, or cube
- Deleting a data object, data source, data set, or cube
- Deleting items within a data set or a cube, such as a column, dimension, or measure

To fix errors resulting from a renamed or deleted data object item, the user has to recreate the report element or dashboard gadget to use a different data object item.

Chapter

3

Accessing data in a data object

This chapter contains the following topics:

- Using data object data in a report
- Connecting to a data object
- Specifying the data to retrieve from a data object
- Using a cube in a data object

Using data object data in a report

A data object provides a report access to predesigned data sources, data sets, and cubes. Report developers create data objects to streamline the report creation process. Data objects provide the following benefits:

- Simplified data access and retrieval. The predesigned data sources, data sets, and cubes in a data object enable report developers to select the data to use in a report without knowledge of the underlying data source, how to connect to it, and how to extract data from it.
- Reusability across multiple reports. If a suite of reports require the same data, designing the data sources, data sets, and cubes once in a shared data object eliminates the need to design the same elements repeatedly for each report.
- Dynamic updates to data items. Changes to data items in a data object propagate to reports that use the data object, ensuring that reports have the latest updates, such as connection properties.

A report accesses data from a data object through either a data object design (.datadesign) file or a data object store (.data). The data design file retrieves data, on demand, each time the report is run. A data object store contains cached, or materialized, data, and provides much more efficient access to data. If getting real-time data is more important than report generation speed, use the data object design file. If data in the underlying data source does not change constantly, or if a data object store is generated regularly, use the data object store.

As with other types of data sources, for a report to use data from a data object, you must create the following BIRT objects:

- A data source that contains the information to connect to a data object
- A data set that specifies the data to use from the data object

Connecting to a data object

When creating a data source to connect to a data object, the only information required is the name of the data object.

How to connect to a data object

1. In Data Explorer, right-click Data Sources, then choose New Data Source.
2. In New Data Source, specify the following information:
    1. Select Actuate Data Object Data Source from the list of data source types, as shown in Figure 3-1.
    2. In Data Source Name, type a name for the data source.

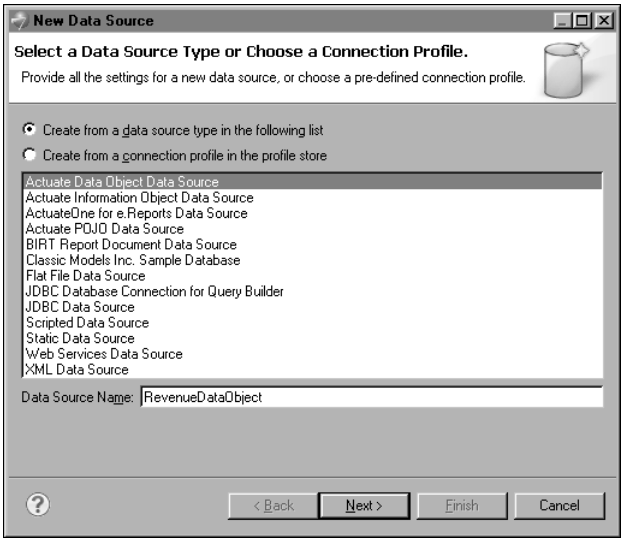


Figure 3-1 Selecting BIRT data object as a data source type

3 Choose Next.

3 In New Actuate Data Object Data Source, next to Data Object, choose Browse. Select Data Object File displays all the data objects (.datadesign and .data files) in the resource folder. Select the data object to use in the report, then choose OK.

Figure 3-2 shows an example of a data object data source that connects to a data object named Revenue.data.

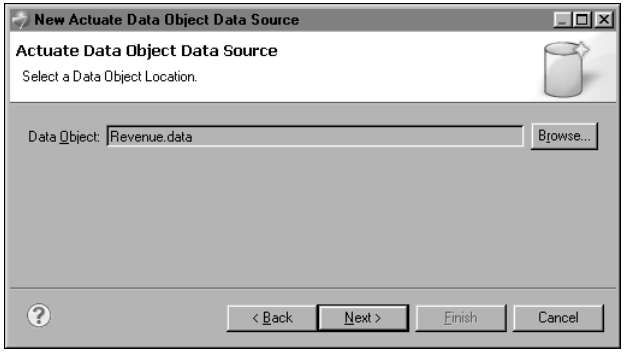


Figure 3-2 Data object selected

4 Choose Next. This button is enabled if the data object contains report parameters. Provide the parameter values.

5 Choose Finish.

The data source appears under Data Sources in Data Explorer. The report now has access to all the data sets and cubes defined in the data object.

Specifying the data to retrieve from a data object

Because a data object contains predesigned data sets, all you do is select a data set from the data object and the columns from the selected data set.

How to specify what data to retrieve from a data object

1. In Data Explorer, right-click Data Sets, then choose New Data Set.
2. In New Data Set, specify the following information:
   1. In Data Source Selection, select the data object data source to use. Data Set Type displays Actuate Data Object Data Set.
   2. In Data Set Name, type a name for the data set.
   3. Choose Next.
3. In New Actuate Data Object Data Set, in Available Data Sets, select one of the data sets defined in the data object.
4. Select the columns to retrieve, and move them to the right pane.
5. Choose Finish to save the data set. Edit Data Set displays the columns in the data set, as shown in Figure 3-3.

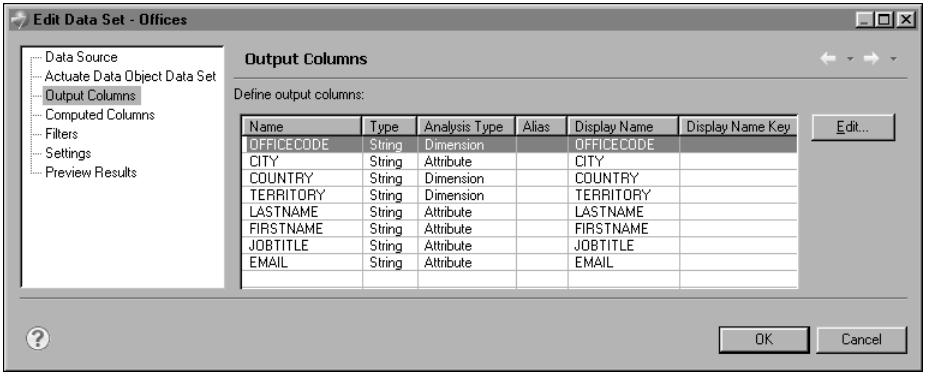


Figure 3-3 Columns in a data set

6. Choose Preview Results to view the data rows that the data set returns.
7. Choose OK to close the data set editor.

Using a cube in a data object

To add a cross tab to the report, a cube is required to provide data for the cross tab. You can create a cube, using data from a data set, or you can use a predesigned cube in the data object. The option you choose depends on how familiar you are with creating cubes, whether you want modifications to the original cube to propagate to the cross tab, or whether you need control over the cube data. You cannot edit a cube from a data object.

How to use a cube in a data object

1. In Data Explorer, right-click Data Cubes, then choose Use Data Object Cube.
2. In Use Data Object Cube, specify the following information. Then choose OK.
    1. In Name, type a name for the cube.
    2. In Select Data Source, select the data source that connects to the data object that contains the cube.
    3. In Available Data Cubes, select a cube.

    Figure 3-4 shows the selection of a cube named Revenue from a data object data source named Revenue_Expenses Data Object.

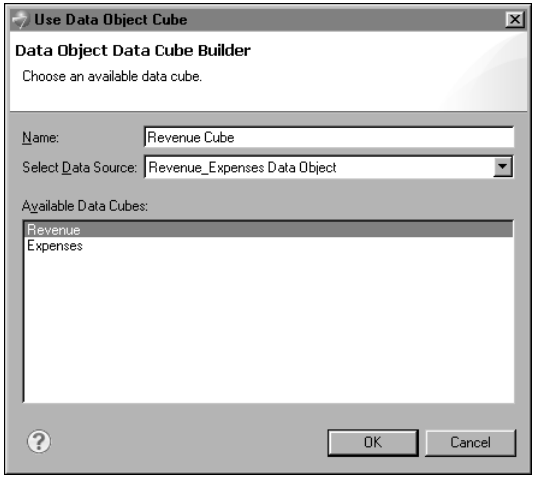


Figure 3-4 Cube selected from a data object

Chapter

4

Accessing data in an information object

This chapter contains the following topics:

- Using information object data in a report
- Connecting to an information object
- Specifying the data to retrieve from an information object

Using information object data in a report

An information object is an encapsulated SQL query designed for use with report design applications, such as Actuate BIRT Designer and Actuate BIRT Studio. Created using the IO Design perspective in Actuate BIRT Designer, information objects can extract and integrate data from a variety of sources, including relational databases, stored procedures, web services, and XML documents. For information about creating information objects, see *Designing BIRT Information Objects*.

Information objects provide the following benefits that data objects also provide:

- Simplified data access and retrieval. Report developers select the data to use without knowledge of the underlying data source, how to connect to it, and how to extract data from it.
- Reusability across multiple reports. Reports can use the same set of data, yet can use different columns, sorting and grouping rules, filters, and parameters.
- Dynamic updates to data properties. Changes to an information object propagate to reports that use the information object, ensuring that reports have the latest updates, such as modified connection properties and queries.

As with other types of data sources, for a report to use data from an information object, you must create the following BIRT objects:

- A data source that contains the information to connect to an information object
- A data set that specifies the data to use from the information object

Connecting to an information object

All information objects are stored in an Actuate BIRT iServer volume. To use an information object as a source of data for a report, you must have a user account on an iServer volume with the privileges that are required to access and run the information object. You need the following information to access an information object:

- The URL to the iServer and the name of the iServer volume that contains the information object
- Your login credentials

Contact the iServer administrator for this information.

How to connect to an information object

1. In Data Explorer, right-click Data Sources, then choose New Data Source.
2. In New Data Source, specify the following information:

1 Select Actuate Information Object Data Source from the list of data source types, as shown in Figure 4-1.

2 In Data Source Name, type a name for the data source.

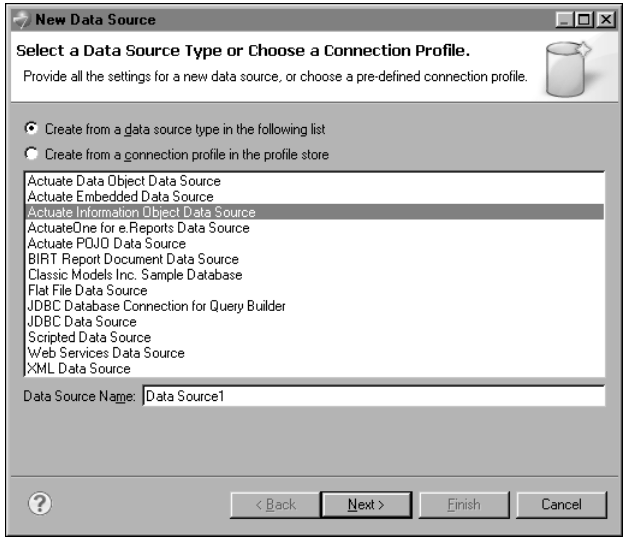


Figure 4-1 Selecting information object as a data source type

3 Choose Next.

3 In New Actuate Information Object Connection Profile, provide the following information to connect to the iServer volume on which the information object is stored:

1 In Server URI, type the URL to the iServer, using the following syntax:

```
http://<server name>:<port>
```

The following is an example of a URL:

```
http://athena:8700
```

2 In Volume, type the name of the iServer volume that contains the information object.

3 In User Name, type the user name required to log into the volume.

4 In Password, type the password required to log into the volume.

5 In Use logged in user credentials on iServer, select Yes or No.

- Select Yes to use the credentials of the user specified at login when the report is run on iServer. This option enables other users to run and view the report.

- Select No to use the iServer URI, volume, user name, and password specified in the report design when the report is run on iServer. This option restricts access to the report.

Figure 4-2 shows an example of connection information specified for an information object data source.

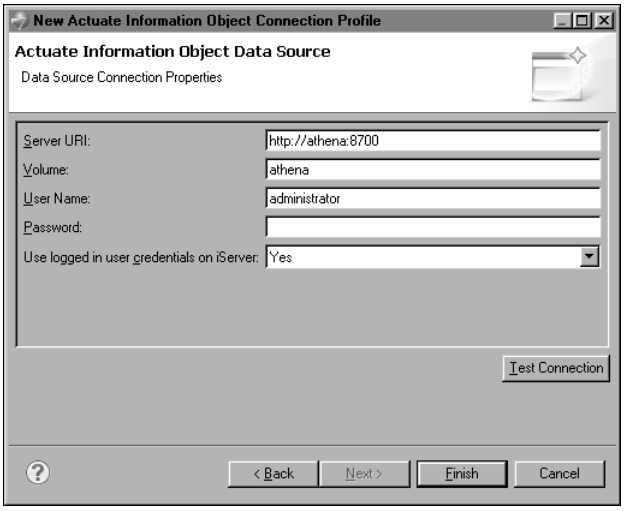


Figure 4-2 Information to connect to an information object

4 Choose Finish.

The data source appears under Data Sources in Data Explorer.

Specifying the data to retrieve from an information object

An information object returns data rows in the structure required by BIRT reports. To specify what data rows to retrieve from an information object, you create a query using the Information Object Query Builder, which is integrated with the data set editor. The query builder provides a graphical interface that you use to select an information object, select data columns, specify group, sort, and filter criteria, and view the resulting query. Alternatively, if you are familiar with the content of an information object and are well-versed in Actuate SQL (based on the ANSI SQL-92 standard), you can type the query.

How to specify what data to retrieve from an information object

1 In Data Explorer, right-click Data Sets, then choose New Data Set.

2 In New Data Set, specify the following information:

1 In Data Source Selection, select the information object data source to use. Data Set Type displays Actuate Information Object Data Set.

2 In Data Set Name, type a name for the data set.

3 Choose Next.

New Data Set—Actuate Information Object Query appears, as shown in Figure 4-3.

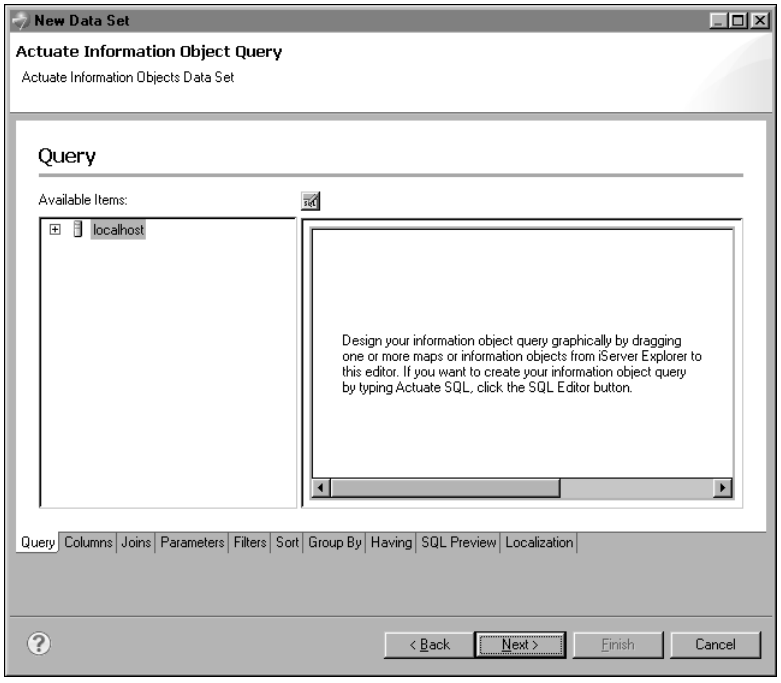


Figure 4-3 Information object query builder

3 Create a query to specify the data to retrieve from the information object. Use either the graphical tools to create the query or the SQL Editor to type the query. For information about creating an information object query and using Actuate SQL syntax, see *Designing BIRT Information Objects*. The query builder integrated with the data set editor has the same user interface and functionality as the query builder in the Information Object Designer.

Figure 4-4 shows an example of a graphical query in which three information objects are joined. Only a few columns from each information object are selected.

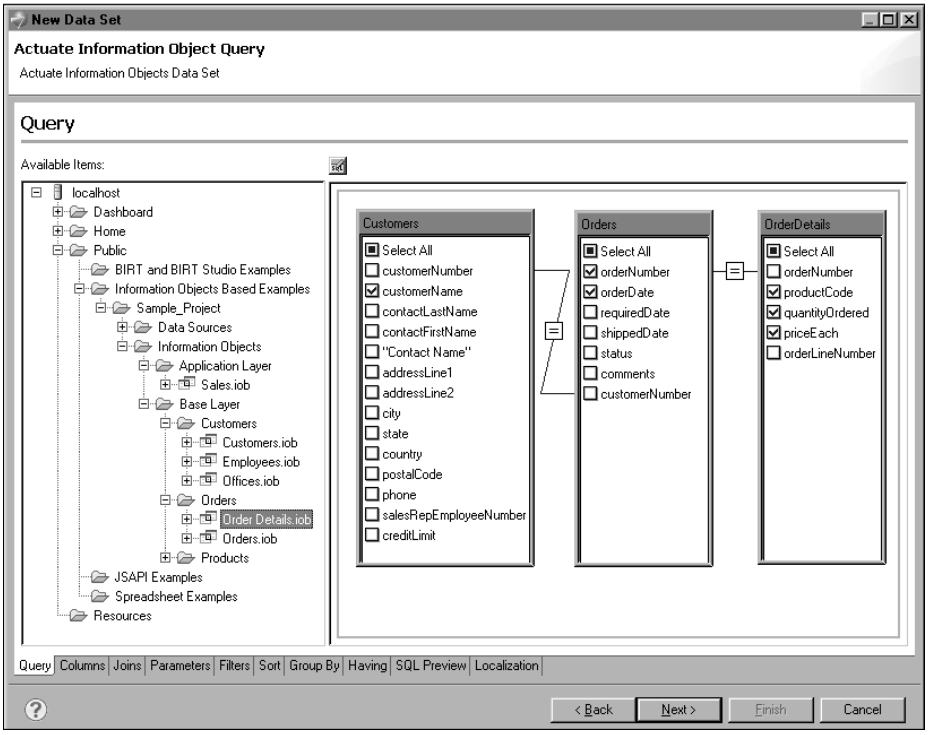


Figure 4-4 Information object query builder displaying a graphical query

4 Choose Next. New Data Set—Actuate Information Object Query displays a message indicating the status of the query.

5 If the query contains errors, choose Back to fix the query. If the query is valid, choose Finish. Edit Data Set displays the columns in the data set, as shown in Figure 4-5.

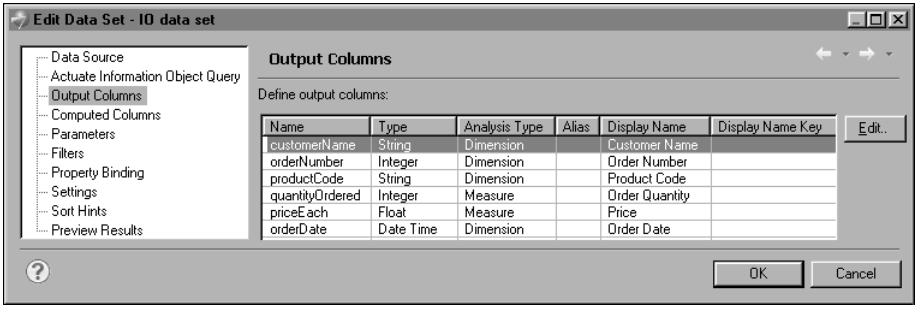


Figure 4-5 Columns in a data set

6 Choose OK to close the data set editor.

Chapter

5

Accessing data in a report document

This chapter contains the following topics:

- Using report document data
- Creating a report document
- Connecting to a report document
- Specifying the data to retrieve from a report document

Using report document data

A report document is a binary file that contains report design information and cached data. It has the file extension .rptdocument. You can generate a report document from a report design, then use the report document as a data source for other reports. Using data from a report document provides the following benefits:

- Faster report-generation time because the report does not have to connect to an external data source, such as a database or web service, to retrieve data.
- Reuse of calculated data. If a table or chart in a report document contains calculated data, such as aggregations, that data is available to your report design.
- Simplifies report creation. You do not need to gather all the information to connect to an external data source, nor do you need to create the query to retrieve data from the data source. Eliminating these steps is particular useful if data retrieval requires a complex SQL query to get data from a database, or a complex SOAP request to get data from a web service.

A report document provides efficient access to data, but at the cost of data possibly being out of date. It is most useful in cases where connecting to and querying a data source is resource-intensive, or when the data changes infrequently.

As with other types of data sources, for a report to use data from a report document, you must create the following BIRT objects:

- A data source that contains the information to connect to a report document
- A data set that specifies the data to use from the report document

Creating a report document

Generate a report document from a report design by choosing Run➛Generate Document. Specify the folder in which to save the report document. To share the report document with other report developers, put the file in a shared resource folder.

When used as a data source, the report document provides access to its result sets. At report generation, a result set is created for each table, chart, cross tab, and list. The data in a result set is defined by the data column bindings created for a table, chart, cross tab, or list. A report design has access to all result sets in a report document, except for a cross tab's result set.

For example, Figure 5-1 shows a report that displays sales data in two tables and a chart.

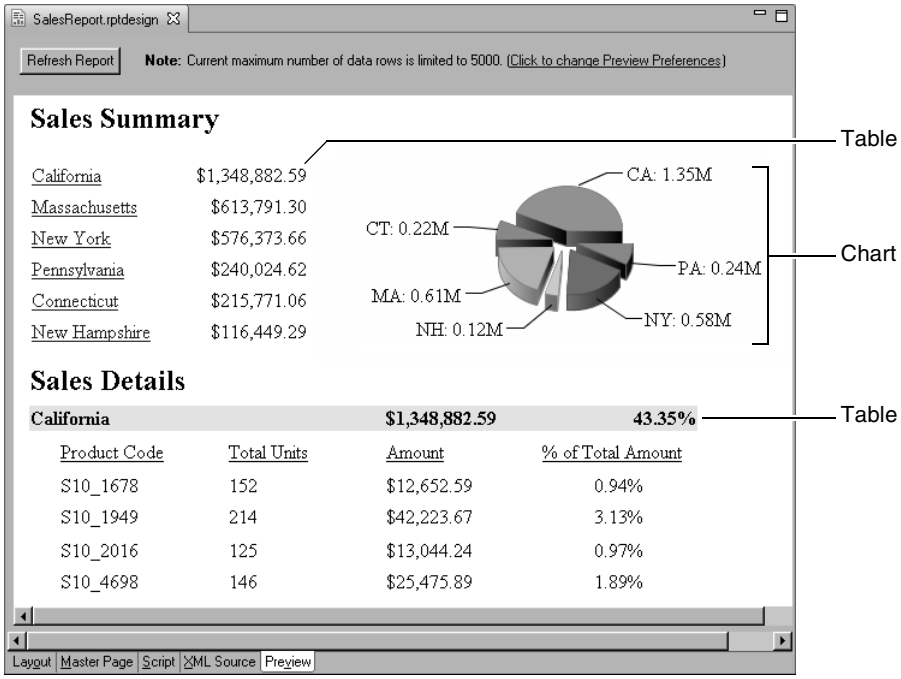


Figure 5-1 A report displaying data in two tables and a chart

A report design that uses a report document generated from this report has access to the result sets generated for the two tables and the chart. When you create a data set to specify which result set data to use, the data set editor displays all the available results sets, as shown in Figure 5-2.

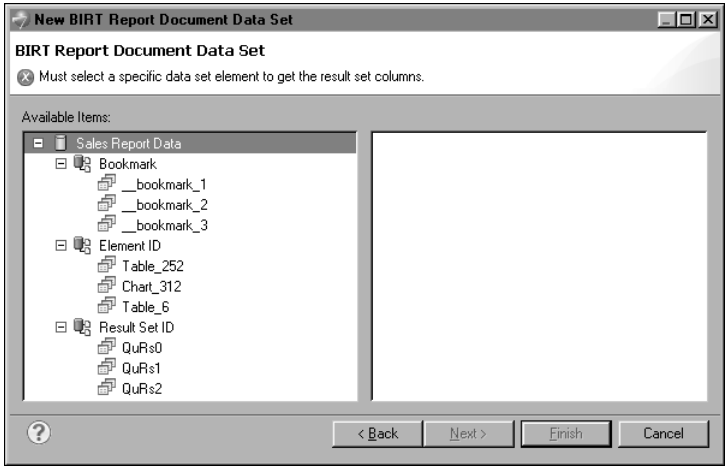


Figure 5-2 Data set editor displaying available result sets with default names

The result sets are organized under three categories: Bookmark, Element ID, and Result Set ID. The result sets named __bookmark_1, __bookmark_2, and __bookmark_3 correspond to the result sets for the first table, the chart, and the second table, respectively. Similarly, the result sets named Table_252, Chart_312, and Table_6 correspond to the result sets for the first table, the chart, and the second table. Finally, the result sets named QuRs0, QuRs1, and QuRs2 also correspond to the result sets for the first table, the chart, and the second table.

While it seems redundant to provide three methods to select any given result set, each method offers different benefits.

- For result sets that appear under Bookmark, the report developer designing the report from which a report document is generated can specify descriptive names. The name can be a literal string or an expression.
- For results sets that appear under Element ID, the report developer can also specify descriptive names, but can only use literal strings to do so. Result sets are also organized hierarchically if a report, such as a master-detail report, contains nested tables.
- For result sets that appear under Result Set ID, the report developer cannot specify alternate names. These generated names are useful for accessing a result set programmatically.

Figure 5-3 shows another example of the data set wizard displaying the result sets available to a report design.

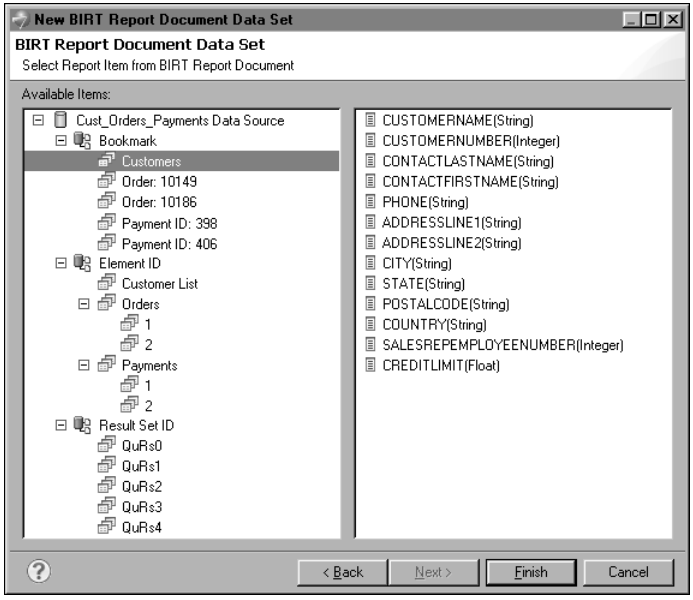


Figure 5-3 Data set editor displaying two result sets with custom names

This time, the result sets under Bookmark use custom names, rather than the default __Bookmark_#. Result sets under Element ID are organized hierarchically under Orders and Payments, which are custom names. This hierarchy reflects the structure of data in the report document.

Specifying bookmark names

As Figure 5-2 showed, BIRT generates a bookmark for each result set, and assigns bookmark names using the __bookmark_# format. If you are designing the report from which to generate a report document, you can specify more descriptive bookmark names to better identify the data in result sets. For example, instead of using the default name, __bookmark_1, to refer to the result set for the sales summary table, specify a name, such as Sales Totals by State.

How to specify a bookmark name

1. Open the report design to be used to generate a report document.
2. In the report layout, select the table, chart, or list for which to specify a bookmark.
3. In Properties Editor—Properties, choose Bookmark.
4. In Bookmark, specify one of the following:
   - A name, such as "Sales Totals by State". If typing a name, enclose it within double quotation marks, as shown in Figure 5-4.

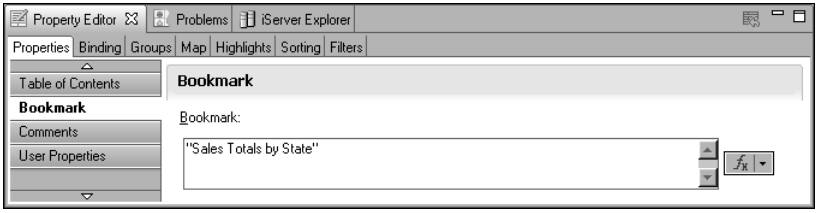


Figure 5-4 Specifying a bookmark name for an element

   - An expression, such as the following:

```
row["COUNTRY"]
"Order: " + row["ORDERNUMBER"]
```

     Specify an expression if, for example, a table is nested within another table or a list. In this case, the generated report document typically contains multiple instances of the inner table, and each table instance has a result set. If each table instance provides data about a particular sales order, for example, it is useful to identify each result set by order number, as shown in Figure 5-3.

5. Save the report.

Specifying element names

BIRT also assigns an element ID for each result set. As Figure 5-2 showed, each ID consists of the element type, an underscore, and a number, for example, Table_252 or Chart_312. If you are designing the report from which to generate a report document, you can specify more descriptive element IDs to describe the data in result sets.

How to specify an element name

1. Open the report design to be used to generate a report document.
2. In the report layout, select the table, chart, or list for which to specify a name.
3. In Properties Editor—Properties, choose General.

   Notice that the Name property is undefined. Notice, too, that the Element ID property has a read-only number, which BIRT automatically assigns to every report element. BIRT uses this number in the default result set name.
4. In Name, type a descriptive and unique name. A report cannot contain duplicate element names. Figure 5-5 shows an example of a name, Sales Chart, specified for a chart.

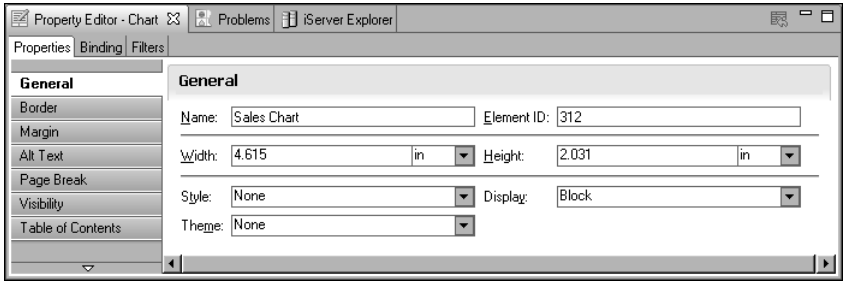


Figure 5-5 Specifying a name for a chart element

5. Save the report.

Connecting to a report document

When creating a data source to connect to a report document, the only information required is the location and name of the report document.

How to connect to a report document

1. In Data Explorer, right-click Data Sources, then choose New Data Source.
2. In New Data Source, specify the following information:
   1. Select BIRT Report Document Data Source from the list of data source types, as shown in Figure 5-6.

2 In Data Source Name, type a name for the data source.

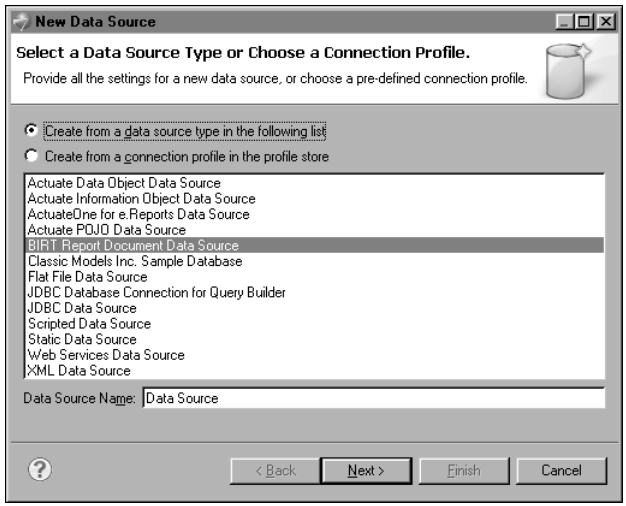


Figure 5-6 Selecting BIRT report document as a data source type

3 Choose Next.

3 In New BIRT Report Document Data Source Profile, use one of the following steps to provide a value for Report Document Path:

- To select a report document in the resource folder, choose Browse. Select the report document, then choose OK.
- To specify a report document in a location other than the resource folder, type the path to the file. The path must be relative to the current folder, for example:

```
../MyReportDocuments/CustomerReport.rptdocument
```

Figure 5-7 shows an example of a report document data source that connects to a file named SalesReport.rptdocument in the resource folder.

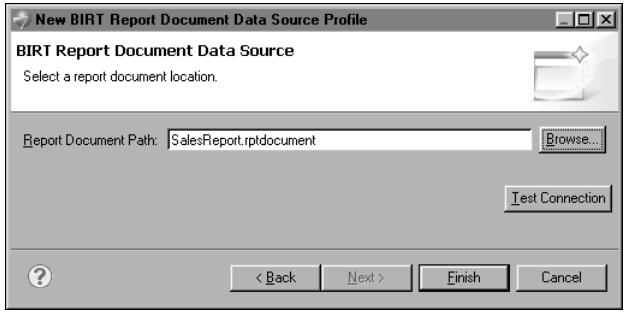


Figure 5-7 Report document selected

4 Choose Finish.

The data source appears under Data Sources in Data Explorer. The report now has access to the data saved in the report document.

Specifying the data to retrieve from a report document

Because a report document contains cached data in the form of result sets, all you do is select the result set or result sets whose data to use in a report design. You must create a data set for each result set.

How to retrieve data from a report document

1 In Data Explorer, right-click Data Sets, then choose New Data Set.

2 In New Data Set, specify the following information:

   1 In Data Source Selection, select the report document data source to use. Data Set Type displays BIRT Report Document Data Set.

   2 In Data Set Name, type a name for the data set.

   3 Choose Next.

   New BIRT Report Document Data Set displays all the result sets in the report document. These items are organized under Bookmark, Element ID, and Result Set ID.

3 Expand a category, then select a result set to see its data columns, as shown in Figure 5-8.

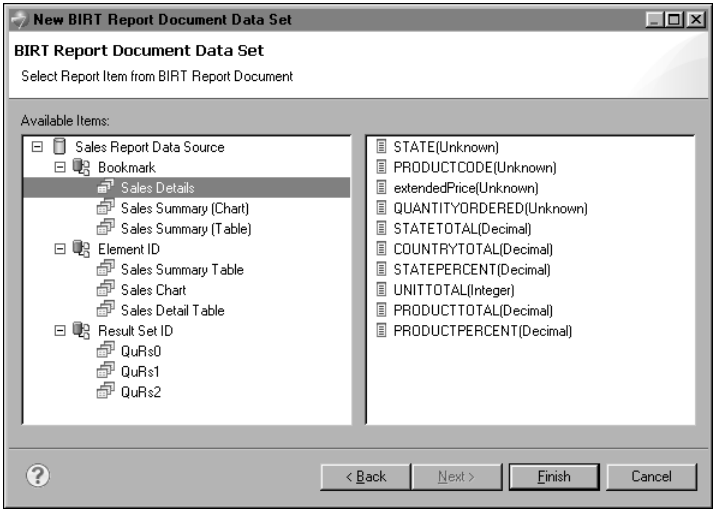


Figure 5-8 Selecting a result set

4 Select the result set that contains the data to use in the report, then choose Finish. Edit Data Set displays the columns in the data set, as shown in Figure 5-9.

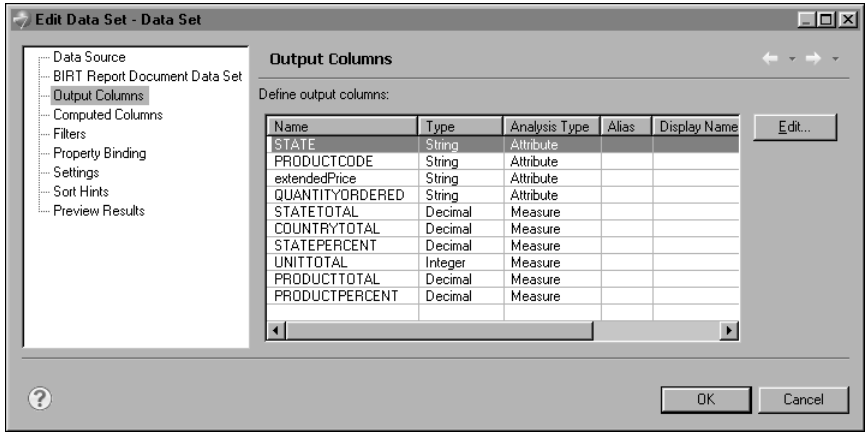


Figure 5-9 Columns in a data set

5 Choose Preview Results to view the data rows that the data set returns.

6 Choose OK to close the data set editor.

Chapter

6

Accessing data in an e.report

This chapter contains the following topics:

- Using ActuateOne for e.Reports Data Connector
- Connecting to an e.report
- Specifying the data to retrieve from an e.report

Using ActuateOne for e.Reports Data Connector

The ActuateOne for e.Reports Data Connector is an Open Data Access (ODA) driver that enables BIRT to retrieve data from an e.report, a report developed using Actuate e.Report Designer Professional. The e.report is a report object instance (.roi) file. Using the ActuateOne for e.Reports Data Connector driver to access the data and business logic from the existing e.report's data schema preserves the business logic in the report while BIRT consumes the data.

To apply BIRT features to data from an ROI file, a developer designs a BIRT report that uses the ActuateOne for e.Reports Data Connector driver to draw data from an existing e.report. After verifying the connection's functionality, the BIRT report design can be expanded using additional data sources and formatting as business needs require. For more information about e.reports, see *Working with Actuate e.Reports*.

About ActuateOne for e.Reports Data Connector functionality

A BIRT data object, information object, or report design can use the data connector to retrieve data from an existing ROI file in an Encyclopedia volume on Actuate BIRT iServer Release 11 or higher. The ROI file does not need to be on the same Encyclopedia volume as the BIRT report design.

The report object design (.rod) file used to generate the ROI file must have searchable fields defined. The ActuateOne for e.Reports Data Connector uses the display names for the searchable fields to name the columns in the BIRT data set.

An ActuateOne for e.Reports data source connects to an Encyclopedia volume. An ActuateOne for e.Reports data set maps controls in a specified e.report in that volume to data set columns. The user interface for creating the data set displays the control names as available columns. The names of the columns that the e.report data source uses are not available.

Accessing an e.report using Page Level Security

The ActuateOne for e.Reports Data Connector respects the Page Level Security (PLS) privileges implemented in the source e.report. The BIRT report developer must specify user credentials for an ROI file in order to retrieve data from that ROI file. PLS privileges of the specified user restrict the data that the BIRT report design retrieves from the ROI file.

Accessing an e.report having multiple sections

A BIRT report design can retrieve data from a complex, multi-section e.report. Each ActuateOne for e.Reports data set can access data from a single section only.

To determine which controls are available for use in the same data set, open the ROD file in e.Report Designer Professional. If the ROD file is not available, use the Output Columns page in Edit Data Set in BIRT Designer Professional to examine the partially scoped name of each column. The name's scope appears before the scope resolution operator (::).

Connecting to an e.report

When creating an ActuateOne for e.Reports data source in a BIRT report to connect to an e.report, you specify the Encyclopedia volume that contains the ROI files. You specify iServer connection properties and the name of the Encyclopedia volume. Specify the use of a trusted connection to improve performance. As you edit the data source and the data set, a trusted connection uses the same session to communicate with the iServer. A non-trusted connection uses the specified credentials to log in to the iServer for each communication.

How to create an ActuateOne for e.Reports data source

1. In Data Explorer, right-click Data Sources and choose New Data Source.
2. In New Data Source, select ActuateOne for e.Reports Data Source, as shown in Figure 6-1. Choose Next.

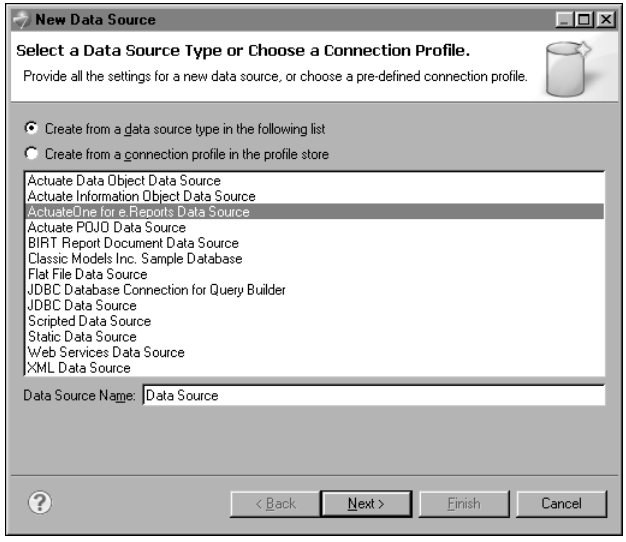


Figure 6-1 Selecting ActuateOne for e.Reports Data Source

3. In New ActuateOne for e.Reports Data Source Profile, type the URL of the server, the user credentials, the volume name, and whether the connection is trusted, as shown in Figure 6-2.

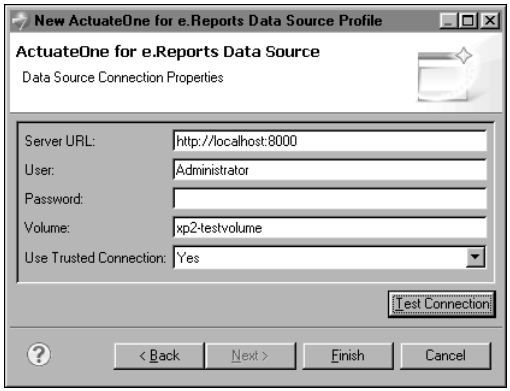


Figure 6-2 Creating a data source profile

4 Choose Test Connection. A message with the results of the test appears.

- If a Success message appears, as shown in Figure 6-3, choose OK.

Figure 6-3 Successful connection test

- If an Error message appears, as shown in Figure 6-4, choose OK. Then, check and correct the server connection properties. Choose Test Connection.

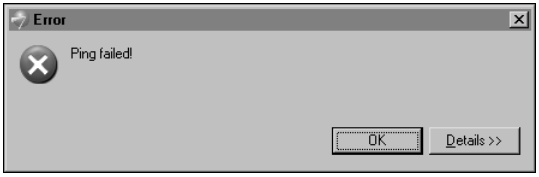


Figure 6-4 Failed connection test

5 Choose Finish to create the data connection.

Specifying the data to retrieve from an e.report

The controls in an e.report provide the columns for a data set. You must use columns from controls in only a single section in the e.report. The scoped name of the column shows the section that contains the control. This name is available in the user interface for editing the data set.

How to create an ActuateOne for e.Reports data set

1 In Data Explorer, right-click Data Sets and choose New Data Set.

2 In New Data Set, type a name for the new data set as shown in Figure 6-5. Then, choose Next.

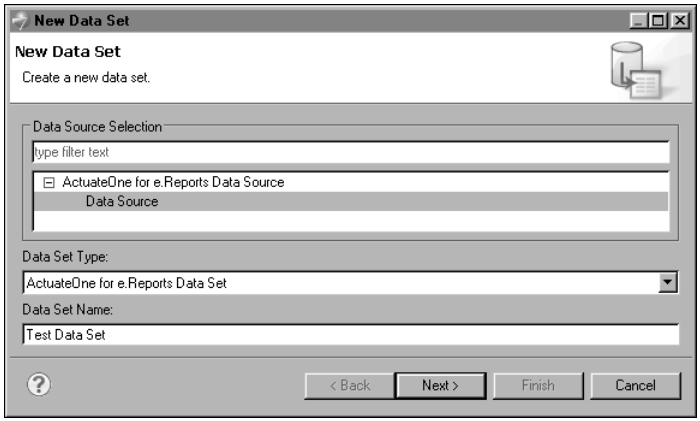


Figure 6-5 Naming the ActuateOne for e.Reports data set

3 In Select Columns, choose Browse.

4 Navigate to the location of the ROI file and select the ROI file, as shown in Figure 6-6. Choose OK.

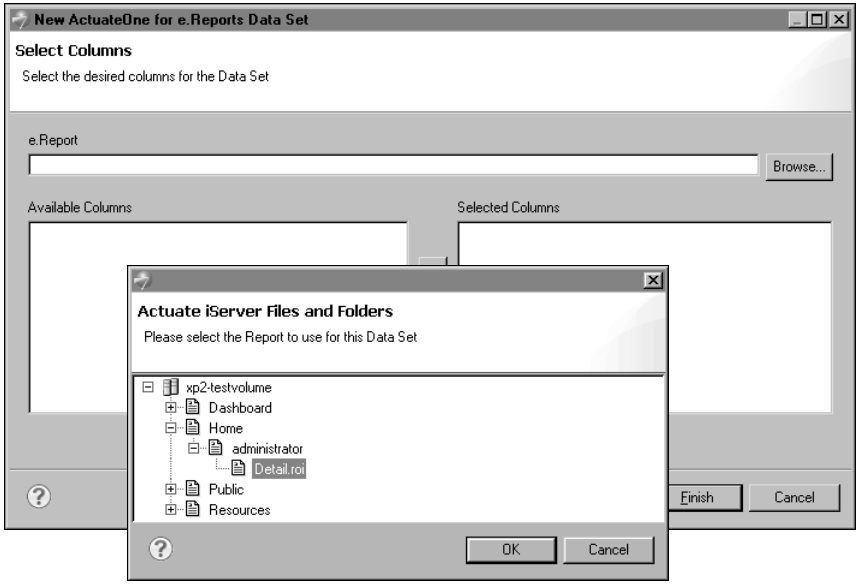


Figure 6-6 Selecting the ROI file

5 In Select Columns, choose Refresh Columns. The column names from the ROI file appear, as shown in Figure 6-7.

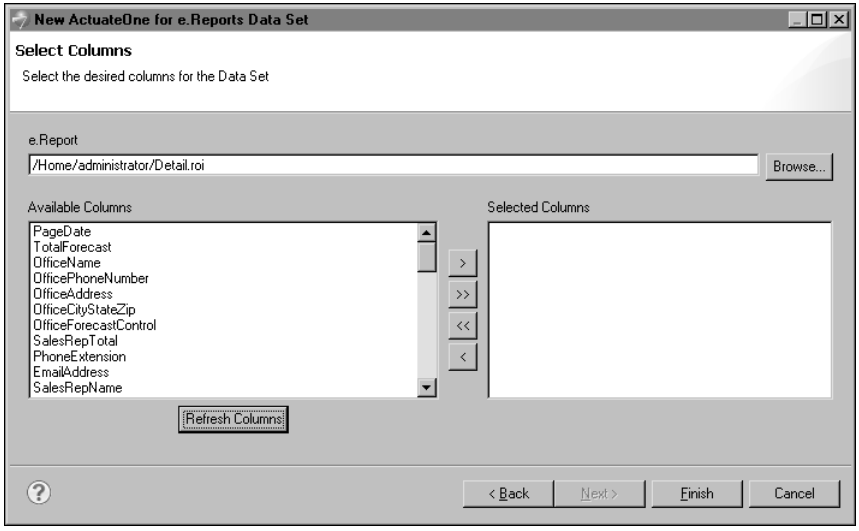


Figure 6-7 Available columns in Detail.roi

6 Select the columns that the BIRT report design requires. Choose the right arrow to move the columns to the Selected Columns pane, as shown in Figure 6-8. Choose Finish.

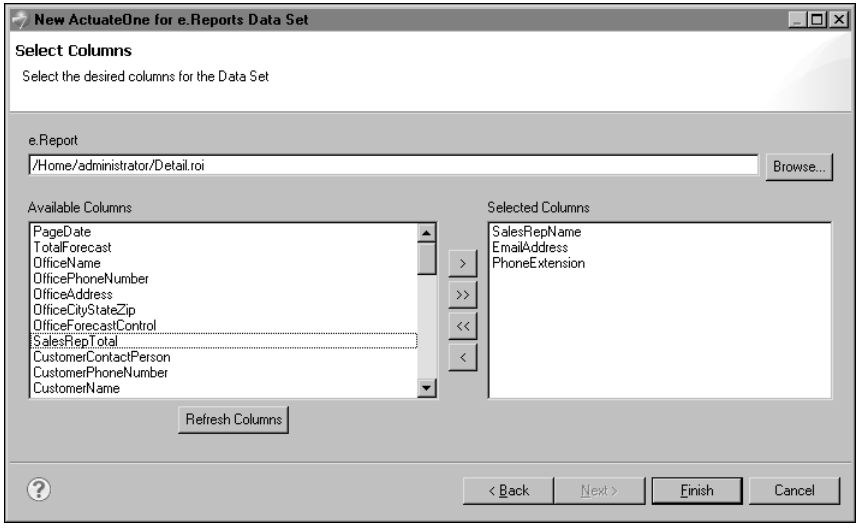


Figure 6-8 Selecting columns

7 In Edit Data Set, select Preview Results, as shown in Figure 6-9.

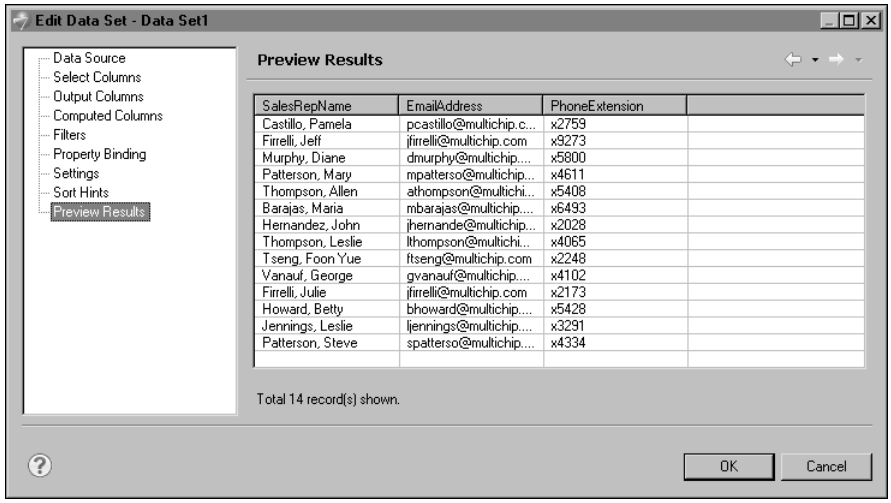


Figure 6-9 Preview of data set results

- If column values appear, select Output Columns.
- If either error message shown in Figure 6-10 appears, perform the following steps:

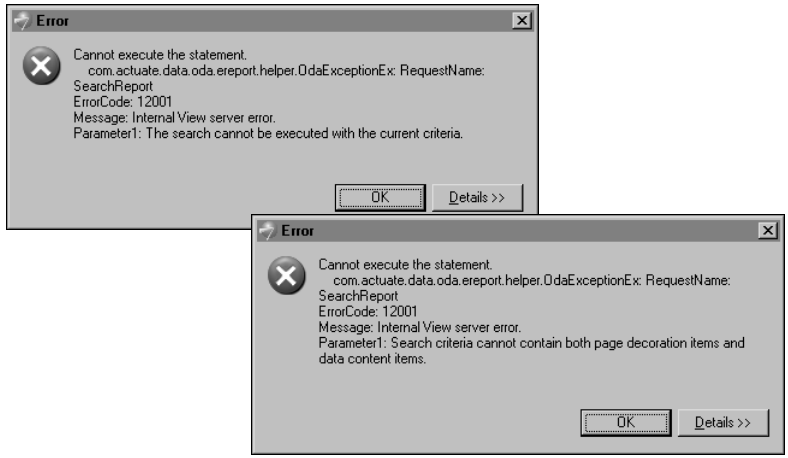


Figure 6-10 Data set preview error messages

1. In the error message, choose OK.
2. Select Output Columns.
3. Expand Name to make the full names of the output columns visible, as shown in Figure 6-11. Note each name's prefix, then select Select Columns.

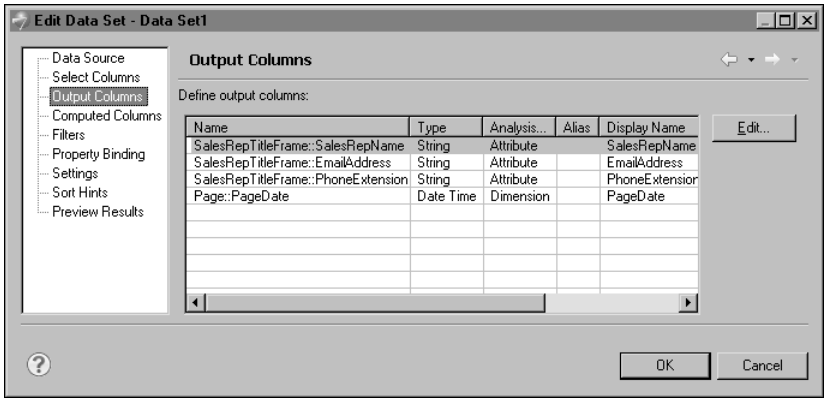


Figure 6-11 Checking scope prefixes

4 In Selected Columns, select columns that are in a different report section from other columns. See "Accessing an e.report having multiple sections," earlier in this chapter for information on using scope name prefixes to determine a column's report section. Choose the left arrow.

5 Repeat step 7.

8 In Edit Data Set, as shown in Figure 6-12, edit the data set column properties if desired. Choose OK. The data set appears in Data Explorer.

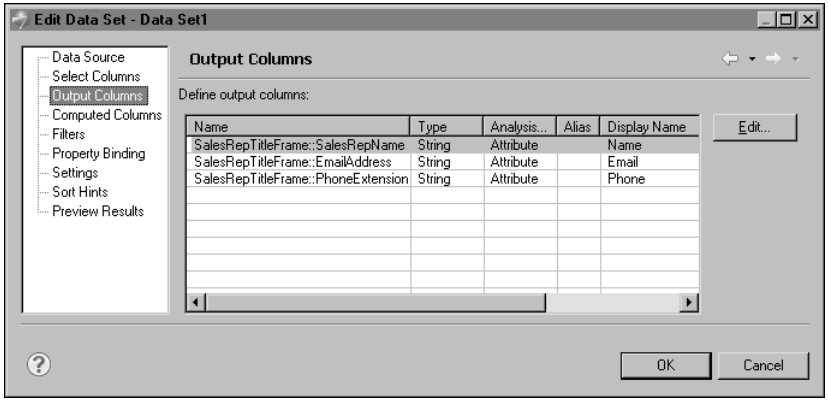


Figure 6-12 Editing the data set column properties

Chapter

7

Accessing data in a POJO

This chapter contains the following topics:

- Using POJO data in a report
- Connecting to a POJO
- Specifying the data to retrieve from a POJO

Using POJO data in a report

Plain Old Java Objects (POJOs) are simple Java objects that do not implement framework-specific interfaces such as those defined by the EJB framework. Java developers use POJOs to separate an application's business logic from infrastructure frameworks, which constantly evolve. POJOs are frequently used for data persistence—the storage and retrieval of data—in Java applications.

Actuate BIRT Designer supports the use of POJOs as a data source for reports. As with other types of data sources, such as databases, XML files, and web services, for a report to use data from a POJO, you must create the following BIRT objects:

- A POJO data source that contains the information to connect to a POJO
- A POJO data set that defines the data that is available to a report

No programming is required to create these BIRT objects. However, if using POJOs created by another developer, a basic understanding of what the classes do and the data they provide is necessary. A simple POJO example typically consists of the following classes:

- A class that describes the data object, for example, a books class that describes the properties of books, including book title, author, publisher, year published, and so on.
- A class that specifies how to retrieve data. For example, such a class can retrieve data about each book by using the Java interface, Iterator, and implementing the open( ), next( ), and close( ) methods to iterate through all the book objects.

Connecting to a POJO

When creating a POJO data source in a BIRT report to connect to a POJO, you specify the location of the JAR file that contains the POJO classes. You can specify either a relative or absolute path.

How to create a POJO data source

1. In Data Explorer, right-click Data Sources, then choose New Data Source.
2. In New Data Source, specify the following information:
    1. Select Actuate POJO Data Source from the list of data source types, as shown in Figure 7-1.
    2. In Data Source Name, type a name for the data source.

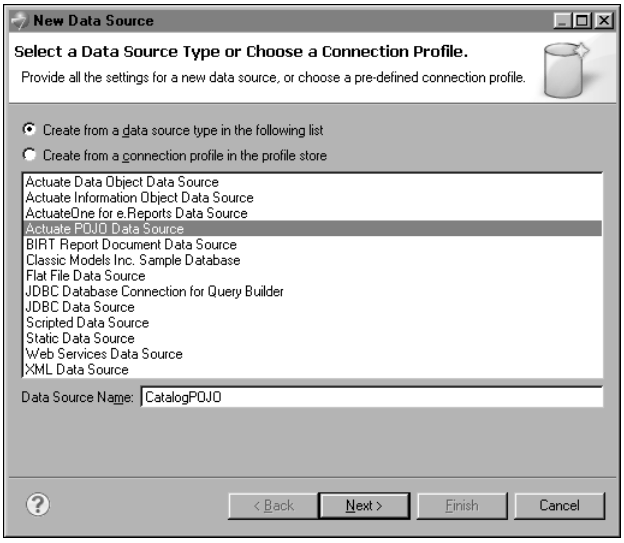


Figure 7-1 Selecting POJO as a data source type

3 Choose Next.

3 In New POJO Data Source Profile, shown in Figure 7-2, specify the properties to connect to the POJO.

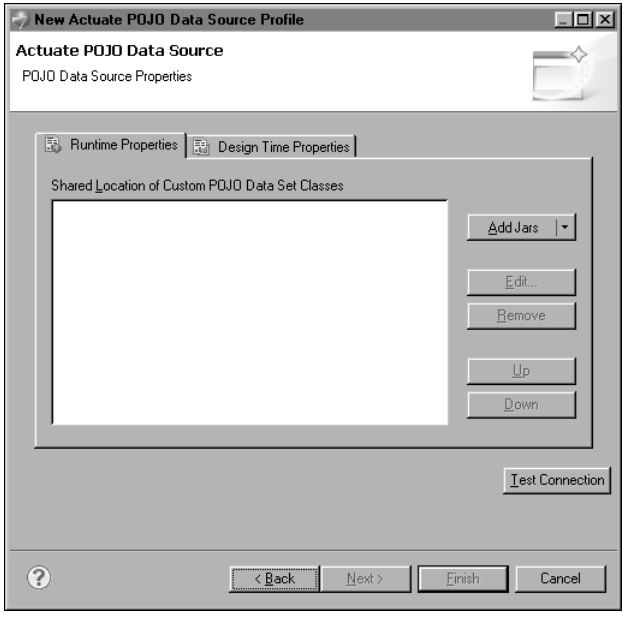


Figure 7-2 POJO data source properties

1 In Runtime Properties, specify the location of the POJO classes that define the run time properties of the data source. Click the arrow icon next to Add Jars, then select either Relative path or Absolute path.

- Select Relative path to specify a path that is relative to the folder designated as the resource folder. By default, BIRT uses the current project folder as the resource folder.

  In Select Jars/Zips, which displays the contents of the resource folder, as shown in Figure 7-3, select the JAR file, then choose OK.

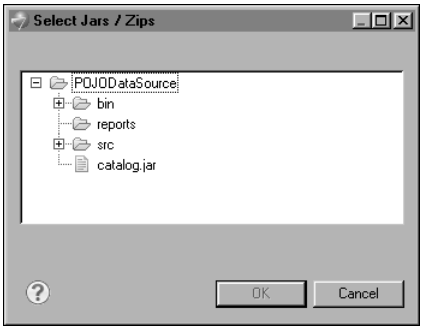


Figure 7-3 Select Jars/Zips displaying the contents of the resource folder

- Select Absolute path to specify the full path in the file system. Browse the file system and select the JAR file, then choose OK.

2 In Design Time Properties, specify the location of the POJO classes that define the design time properties of the data source. The data set editor uses this information to list the get methods, which you select to define the column mappings for the POJO data set. Use the instructions in the previous step to specify either a relative path or absolute path to the POJO classes.

4 Choose Test Connection to ensure that the connection information is correct. If Test Connection returns an error, repeat the preceding steps to correct the error.

5 Choose Finish. The new POJO data source appears under Data Sources in Data Explorer.

Specifying the data to retrieve from a POJO

BIRT reports must use data that is structured as a table consisting of rows and columns. For a POJO data set to return data in this format, you map methods or members of a POJO class to columns. Listing 7-1 shows an example of a class that represents music CDs. The class describes the members and uses pairs of get and

set methods to persist the data. To create a data set using this class, you would map the get methods to columns.

Listing 7-1 Class representing music CDs

```
package dataset;
public class CD {

   private String cdTitle;
   private String cdArtist;
   private String cdCountry;
   private String cdCompany;
   private String cdPrice;
   private String cdYear;

   public CD(String title) {
      this.cdTitle = title;
   }
   public String getCDTitle() {
      return cdTitle;
   }
   public void setCDTitle(String title) {
      this.cdTitle = title;
   }
   public String getCDArtist() {
      return cdArtist;
   }
   public void setCDArtist(String artist) {
      this.cdArtist = artist;
   }
   public String getCDCountry() {
      return cdCountry;
   }
   public void setCDCountry(String country) {
      this.cdCountry = country;
   }
   public String getCDCompany() {
      return cdCompany;
   }
   public void setCDCompany(String company) {
      this.cdCompany = company;
   }
   public String getCDPrice() {
      return cdPrice;
   }
```

(continues)

```
   public void setCDPrice(String price) {
      this.cdPrice = price;
   }
   public String getCDYear() {
      return cdYear;
   }
   public void setCDYear(String year) {
      this.cdYear = year;
   }
}
```

How to create a POJO data set

This procedure assumes you have already created the POJO data source that this data set uses. Examples in this section refer to the POJO example in Listing 7-1.

1. In Data Explorer, right-click Data Sets, then choose New Data Set.
2. In New Data Set, specify the following information:
   1. In Data Source Selection, select the POJO data source to use. Data Set Type displays Actuate POJO Data Set.
   2. In Data Set Name, type a name for the data set.
   3. Choose Next.
3. In New Actuate POJO Data Set, specify the following information:
   1. In POJO Data Set Class Name, specify the POJO class that retrieves the data at run time. Choose Browse to find and select the class.
   2. In Application Context Key, use the default key or delete it. This property is optional.

      Figure 7-4 shows an example of properties set for a POJO data set.

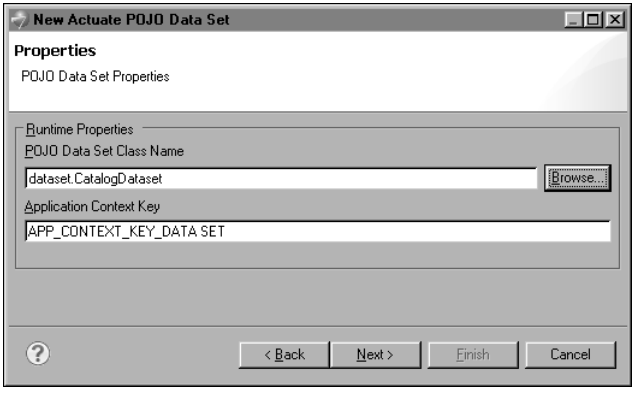


Figure 7-4 POJO data set properties

4 Choose Next.

5 Map methods or fields in a POJO class to data set columns, using the following steps:

1 In POJO Class Name, specify the POJO class that contains the get methods to map to columns. You can choose Browse to find and select the class.

The data set editor uses a get* filter to display all the get methods in the specified POJO class, as shown in Figure 7-5.

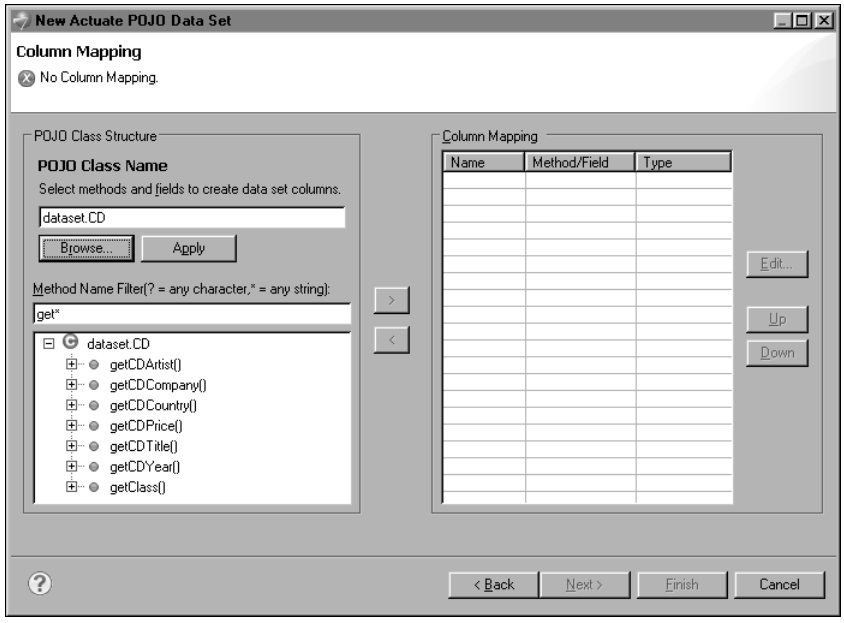


Figure 7-5 Data set editor displaying the get methods in a POJO class

2 Double-click the get method to map to a data set column.

Add Column Mapping displays the mapping information, as shown in Figure 7-6.

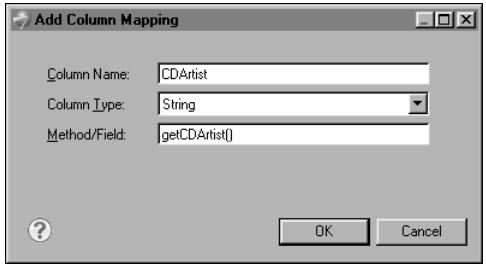


Figure 7-6 Column mapping information

Choose OK to accept the default values.

3 Repeat the previous step for every column to add to the data set. Figure 7-7 shows an example of column mappings defined in a POJO data set.

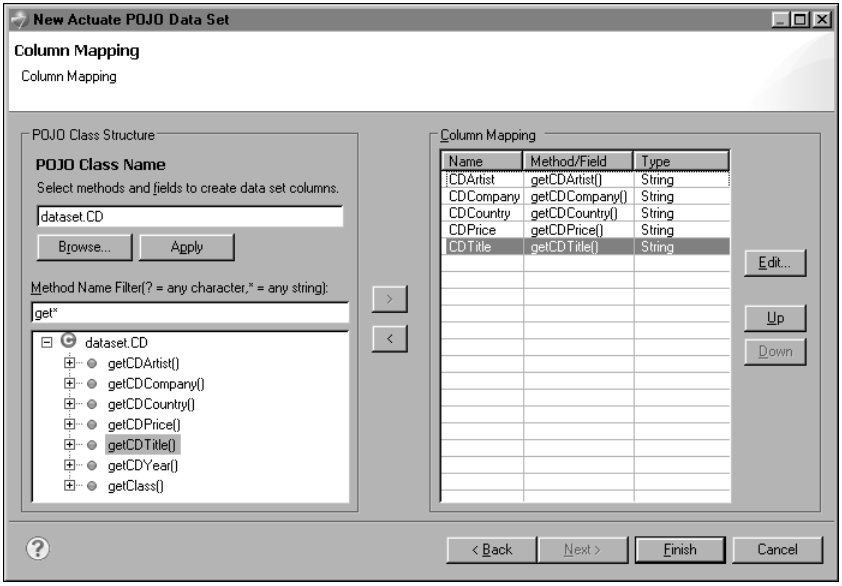


Figure 7-7 Data set editor displaying the column mappings

6 Choose Finish to save the data set. Edit Data Set displays the columns, and provides options for editing the data set.

7 Choose Preview Results to view the data rows returned by the data set. Figure 7-8 shows an example of data rows returned by a POJO data set.

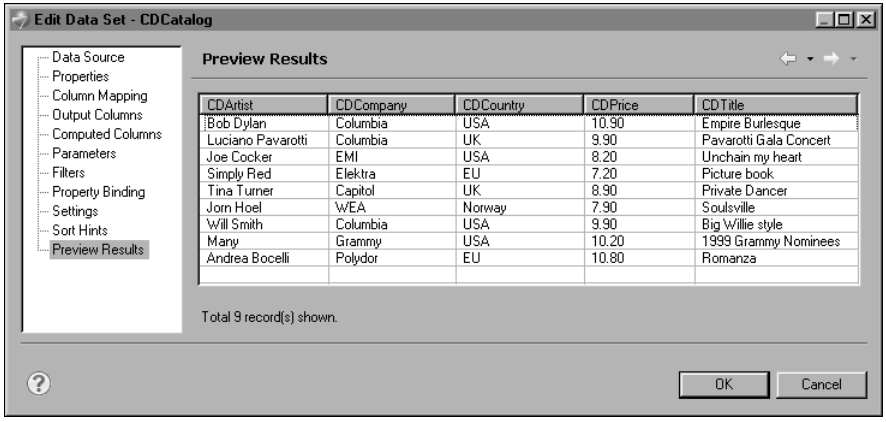


Figure 7-8 Data rows returned by a POJO data set

Chapter

8

Combining data from multiple data sources

This chapter contains the following topics:

- Ways to combine data
- Creating a union data set
- Creating a joined data set

Ways to combine data

Sometimes, the data that a report requires originates in several data sources. For example, an application system generates monthly transaction data and the data for each month is saved in a separate CSV file, or a system saves data in different formats, such as XML and CSV.

Actuate BIRT Designer provides two options for combining data from multiple sources. You can create a union data set or a joined data set. The option you choose depends on the data structures and the results you want. Both options entail combining data sets, so before creating a union data set or a joined data set, you first create the individual data sets. For example, to combine data from an XML file with data from a CSV file, you must first create the XML data set and the flat file data set.

Creating a union data set

A union data set combines the results returned by two or more data sets. Creating a union data set is similar to using a SQL UNION ALL statement, which combines the result sets of two or more SELECT statements into a single result set.

Create a union data set to consolidate data from multiple sources that have similar data structures. For example, a company maintains separate database tables to store contact information about employees and contractors. The structure of the tables are similar. Both contain Name and Phone fields. Suppose you want to create a master contact list for all employees and contractors. The solution is to create one data set to retrieve employee data, a second data set to retrieve contractor data, and a union data set that combines data from the previous data sets.

Figure 8-1 illustrates the data sets that return employee and contractor data.

Employees data set

Name	Phone	E-mail
Mark Smith	650-343-2232	msmith@acme.com
Patrick Mason	650-343-1234	mason@acme.com
Soo-Kim Yoon	650-343-5678	skyoon@acme.com
Maria Gomez	650-343-9876	gomez@acme.com

Contractors data set

Name	Phone
Sarah Brown	650-545-3645
Sean Calahan	415-242-8254
Paula Mitchell	650-662-9735
Michael Lim	408-234-2645

Figure 8-1 Data sets with common fields returning employee and contractor data

When creating a union data set, you select the fields to include. Figure 8-2 shows a union data set that includes all the fields from both Employees and Contractors

data sets. The Name field contains all employee and contractor names. The Phone field also contains all employee and contractor phone numbers. The E-mail field exists only in the employee data set, so only employee data rows have e-mail data.

Union data set

Name	Phone	E-mail
Mark Smith	650-343-2232	msmith@acme.com
Patrick Mason	650-343-1234	mason@acme.com
Soo-Kim Yoon	650-343-5678	skyoon@acme.com
Maria Gomez	650-343-9876	gomez@acme.com
Sarah Brown	650-545-3645	
Sean Calahan	415-242-8254	
Paula Mitchell	650-662-9735	
Michael Lim	408-234-2645	

Figure 8-2 Union data set that combines all data from Employees data set and Contractors data set

Figure 8-3 shows a union data set that includes only the common fields, Name and Phone, from the Employees and Contractors data sets.

Union data set

Name	Phone
Mark Smith	650-343-2232
Patrick Mason	650-343-1234
Soo-Kim Yoon	650-343-5678
Maria Gomez	650-343-9876
Sarah Brown	650-545-3645
Sean Calahan	415-242-8254
Paula Mitchell	650-662-9735
Michael Lim	408-234-2645

Figure 8-3 Union data set that combines data from common fields in Employees data set and Contractors data set

In the previous example, the two data sets used to create a union data set contained common fields with the same names. This condition is required for consolidating data into a single field. However, data sources often use different field names.

Suppose the Name field in the Employees and Contractors tables is EmployeeName and ContractorName, respectively. To create a union data set that consolidates employee and contractor names in a single field, rename the field names in the individual data sets to use the same name. When creating the

Employees data set, in Output Columns, use the Alias property to give the EmployeeName field another name. Figure 8-4 shows the EmployeeName field with the alias, Name.

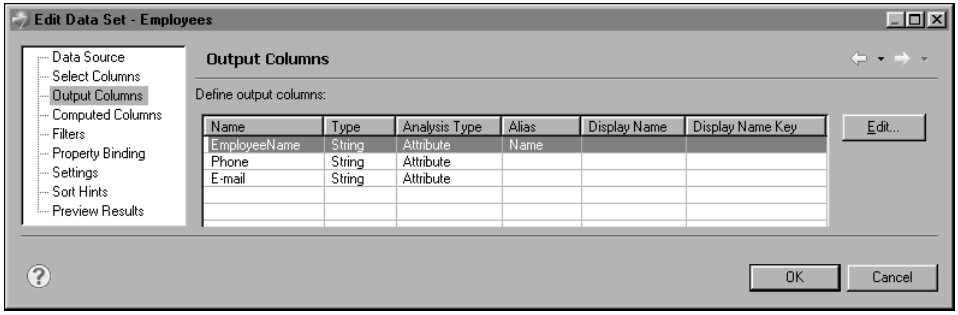


Figure 8-4 Alias specified for the EmployeeName field

Similarly, when creating the Contractors data set, edit the ContractorName field to use the same alias.

How to create a union data set

This procedure assumes that you have created the data sets to be included in the union data set.

1. In Data Explorer, right-click Data Sets, and choose Union Data Set.
2. In New Data Set, in Data Set Name, optionally type a name for the union data set.
3. Choose New.
4. In New Union Element, in Select Data Set, select the first data set that contains the data to include in the union data set.

   New Union Element displays the fields in the selected data set, as shown in Figure 8-5.

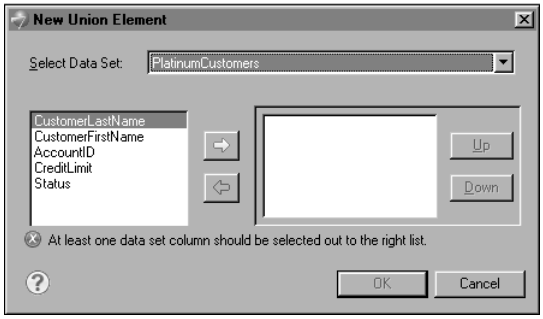


Figure 8-5 Fields in a data set selected for a union data set

5. Select the fields to include in the union data set, then choose OK.

6 Repeat steps 3 to 5 to add the next data set to the union data set.

Figure 8-6 shows a union data set named MasterCustomerList that consists of fields from two data sets, PlatinumCustomers and GoldCustomers.

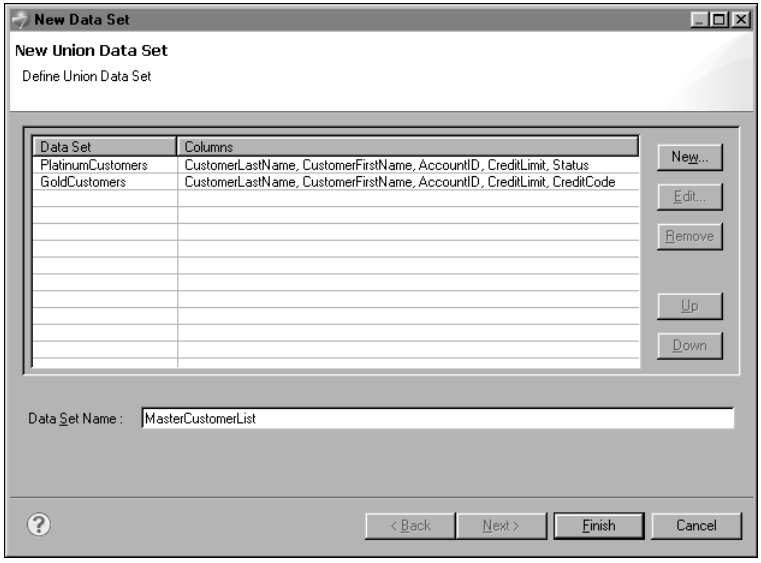


Figure 8-6 Definition of a union data set that combines two data sets

7 Choose Finish. Edit Data Set displays the selected fields, and provides options for editing the data set.

8 Choose Preview Results. Figure 8-7 shows the rows returned by the MasterCustomerList union data set.

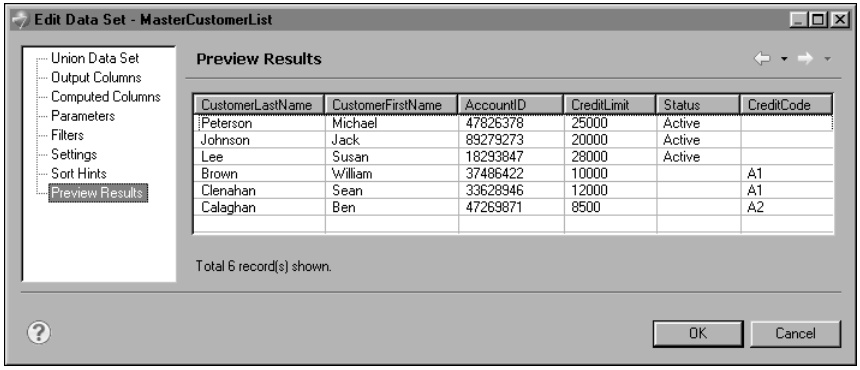


Figure 8-7 Data rows returned by the union data set

Creating a joined data set

A joined data set combines the results of two or more data sets that are related through a common key. Creating a joined data set is similar to joining tables in a database using a SQL JOIN clause. Use a joined data set to combine data from different data sources in which a relationship exists, as shown in the following example.

Figure 8-8 illustrates data sets that return data about customers and orders. The data sets are related through the CustomerID field. You can retrieve order information for each customer by joining the data sets.

Customers data set

CustomerName	CustomerID
Sarah Brown	1001
Sean Calahan	1002
Paula Mitchell	1003
Michael Lim	1004

Orders data set

OrderID	Amount	CustomerID
110	1500.55	1003
115	12520.00	1001
120	8450.50	1004
125	7550.00	1002

Figure 8-8 Data sets with a common field returning customer and order data

Figure 8-9 shows the results of joining the customers and orders data sets on the CustomerID key, and displaying only the CustomerName and Amount fields in the joined data set.

CustomerName	Amount
Sarah Brown	12520.00
Sean Calahan	7550.00
Paula Mitchell	1500.55
Michael Lim	8450.50

Figure 8-9 Data rows returned when the customers and orders data sets are joined

Actuate BIRT Designer supports the functionality of joined data sets available in the open-source version, and provides the following additional features in an updated user interface:

- The capability to join more than two data sets
- The capability to join on more than one key
- Support for new join operators: <>, <, >, <=, >=
- Support for a new type of join, the side-by-side join

Unlike the other types of supported joins (inner, left outer, right outer, and full outer), the side-by-side join links data sets without requiring a key. The resulting

joined data set displays the selected fields side by side. Figure 8-10 shows two data sets that do not share a common field. The first data set returns customer data, and the second data set returns order data.

Customers data set

CustomerName	CustomerID
Sarah Brown	1001
Sean Calahan	1002
Paula Mitchell	1003
Michael Lim	1004

Orders data set

OrderID	Amount
110	1500.55
115	12520.00
120	8450.50
125	7550.00

Figure 8-10 Data sets without a common field

Figure 8-11 shows the results of joining the customers and orders data sets using the side-by-side join. When using this type of join, do not misinterpret the results. As Figure 8-11 shows, the data from the two data sets appear side by side, implying that each customer has a relationship with an order when, in fact, no such relationship exists.

CustomerName	CustomerID	OrderID	Amount
Sarah Brown	1001	110	1500.55
Sean Calahan	1002	115	12520.00
Paula Mitchell	1003	120	8450.50
Michael Lim	1004	125	7550.00

Figure 8-11 Results of a side-by-side join

For information about the other types of supported joins, see *BIRT: A Field Guide*.

How to create a joined data set

This procedure assumes that you have created the data sets to be included in the joined data set.

1. In Data Explorer, right-click Data Sets, and choose Join Data Set.
2. In New Data Set, in Data Set Name, optionally type a name for the joined data set.
3. Specify the data sets to use in the joined data set. Under Available data sets, drag each data set to the editing area. Figure 8-12 shows three data sets in the editing area.

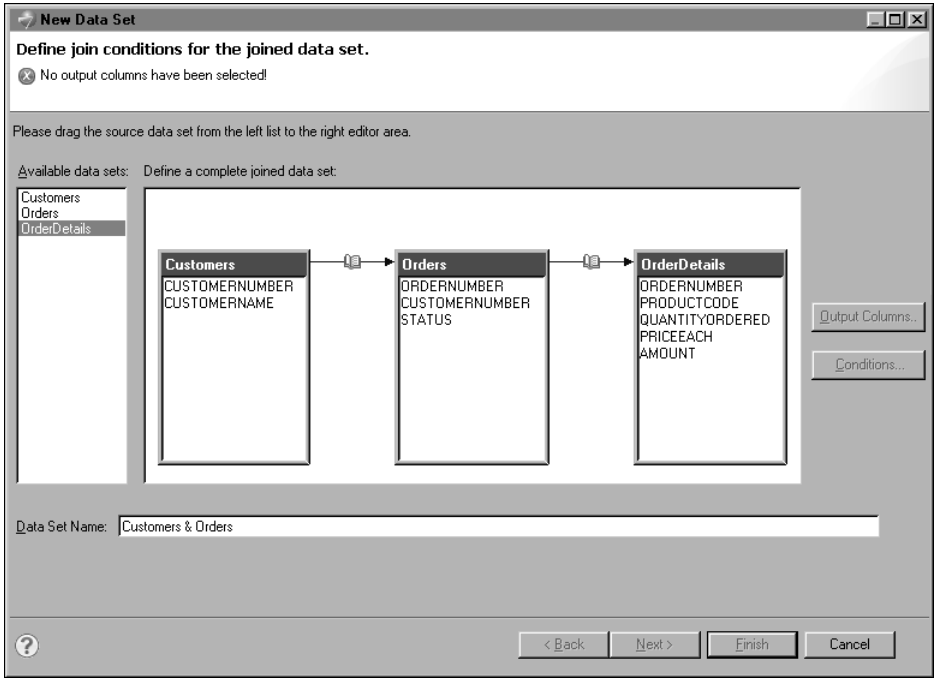


Figure 8-12 Three data sets selected for a joined data set

4 Specify the fields from each data set to include in the joined data set. Perform the following tasks for each data set:

   1 Select a data set by clicking anywhere in the image of the data set. Do not, however, click on a field name.

   2 Choose Output Columns.

   3 In Edit Data Set Properties, under Select output columns, select the desired data set fields, then choose OK.

5 Specify the conditions for joining the data sets. Perform the following tasks for each pair of data sets. In the example shown in Figure 8-12, specify a condition for joining the first and second data sets, and a condition for joining the second and third data sets.

   1 Select the arrow between two data sets.

   2 Choose Conditions.

   3 In Define join type and join conditions, specify the following information:

      1 In Join Type, select the type of join to use.

      2 If you select a join type other than sideBySide, define a join condition.

- Choose New.
- Select the fields on which to join, and select a operator that specifies how to compare the values in the fields being joined. Figure 8-13 shows a join condition that combines data when the CUSTOMERNUMBER value in the Customers data set is equal to the CUSTOMERNUMBER value in the Orders data set.

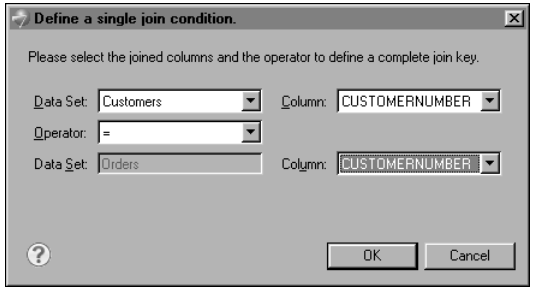


Figure 8-13 Joining data sets on a common field

- Choose OK.

The Define join type and join conditions dialog displays the specified condition, as shown in Figure 8-14.

Figure 8-14 Definition of an inner join

4 Choose OK.

6 Choose Finish to save the joined data set.

Joining on more than one key

You can specify more than one join condition when joining two data sets. For example, you can join a customers data set with a sales offices data set, shown in Figure 8-15, to find the names of customers and sales managers that are located in the same city and state.

Customers data set

Name	City	State
Mark Smith	San Francisco	California
Patrick Mason	Los Angeles	California
Paula West	New York	New York
Joanne Kim	San Diego	California

Sales Offices data set

City	State	SalesMgr
Los Angeles	California	Robert Diaz
New York	New York	Monica Blair
San Francisco	California	Susan Kline

Figure 8-15 Data sets with two common fields, City and State

You would create the following join conditions:

- The first condition, shown in Figure 8-16, compares the State values in the Customer and SalesOffices data sets and looks for a match.

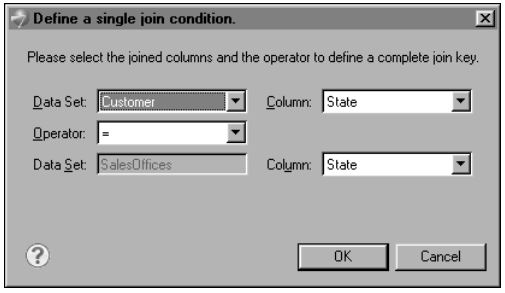


Figure 8-16 Joining on a State field

- The second condition, shown in Figure 8-17, compares the City values in both data sets and looks for a match.

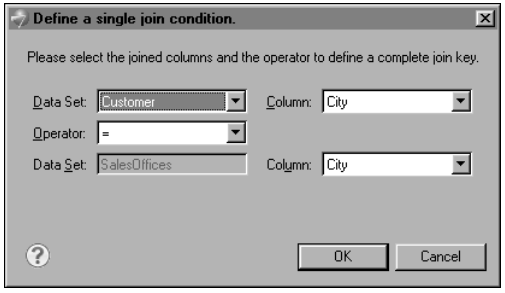


Figure 8-17 Joining on a City field

The joined data set returns the results shown in Figure 8-18.

Figure 8-18 Data rows returned by the joined data set

Specifying a join condition not based on equality

The condition for joining values in two fields is usually based on equality (=), as shown in all the examples so far. Less common are join conditions that use any of the other comparison operators: not equal (<>), greater than (>), less than (<), greater than or equal to (>=), and less than or equal to (<=).

The following example shows the use of joins that are not based on equality. In the example, a Sales data set is joined with a Commissions data set. The joined data set uses a >= join and a < join to look up the commissions to pay to sales managers, based on their sales totals and management levels.

Figure 8-19 shows the Sales and Commissions data sets. In the Commissions data set, each level has four commission rates. For level 1, a commission rate of 25% is paid if a sales total is between 75000 and 100000, 20% is paid if a sales total is between 50000 and 75000, and so on.

Sales data set

SalesMgr	Level	TotalSales
Susan Kline	1	55000
Robert Diaz	2	45000
Monica Blair	2	28000
Sean Calahan	1	23000

Commissions data set

Level	LowRange	HighRange	Commission
1	75000	100000	25
1	50000	75000	20
1	25000	50000	15
1	15000	25000	10
2	70000	100000	25
2	45000	70000	20
2	20000	45000	15
2	10000	20000	10

Figure 8-19 Data sets returning sales and commission rates data

The following join conditions specify the fields on which to join and how to compare the values in the fields being joined:

- The first condition, shown in Figure 8-20, compares the Level values in the Sales and Commissions data sets and looks for a match.

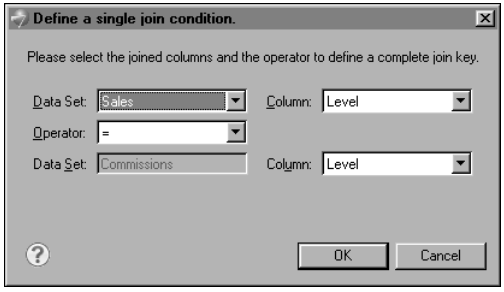


Figure 8-20 Joining on the Level field and looking for a match

- The second condition, shown in Figure 8-21, uses the >= operator to compare the TotalSales values in the Sales data set with the LowRange values in the Commissions data set.

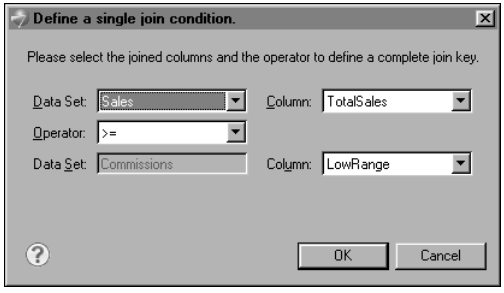


Figure 8-21 Joining on TotalSales and LowRange fields using the >= operator

- The third condition, shown in Figure 8-22, uses the < operator to compare the TotalSales values in the Sales data set with the HighRange values in the Commissions data set.

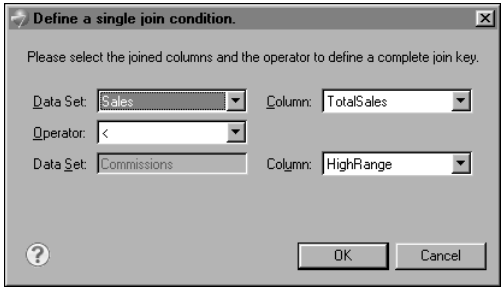


Figure 8-22 Joining on TotalSales and HighRange fields using the < operator

The second and third join conditions check if a sales total is greater than or equal to LowRange and less than HighRange.

The joined data set returns the results shown in Figure 8-23.

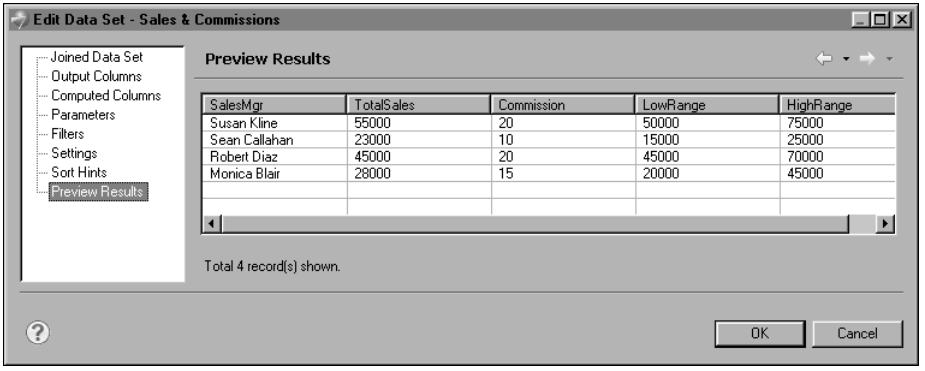


Figure 8-23 Data rows returned by the joined data set

Part Two

Designing reports

Chapter

9

Formatting a report

This chapter contains the following topics:

- Formatting features in Actuate BIRT Designer
- Removing the default themes
- Hiding columns in a table
- Designing for optimal viewer performance

Formatting features in Actuate BIRT Designer

Actuate BIRT Designer supports all the formatting features available in the open-source version, and provides a few additional features. The reports you create using Actuate BIRT Designer are typically published to the Actuate BIRT iServer, where they can be viewed in Interactive Viewer, opened in BIRT Studio, or added to a dashboard. Often, as you design a report, you consider how the report is viewed and used by these applications.

This chapter describes the additional report formatting options in Actuate BIRT Designer. For information about other formatting options, see *BIRT: A Field Guide*. This chapter also describes the design issues to consider when designing reports that users view in the web viewer.

Removing the default themes

By default, new reports that you create use a set of themes that apply formatting to charts, gadgets, tables, and cross tabs. Figure 9-1 shows a table with the default formats.

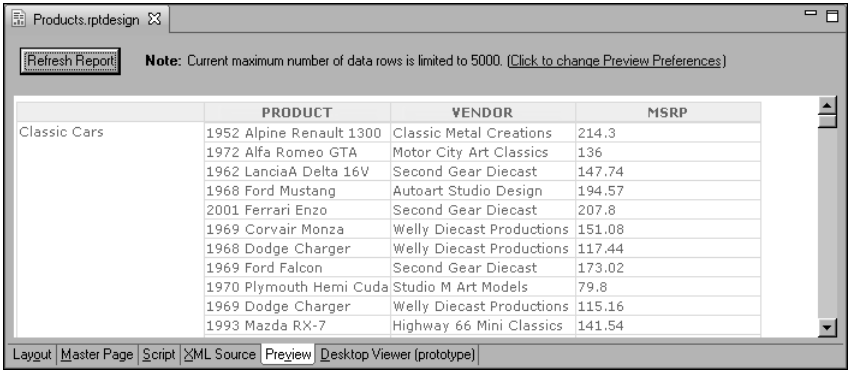


Figure 9-1 Table with the default formats

The themes are defined in a library, ThemesReportItems.rptlibrary, which is added to every new report.

To apply your own themes or styles in a report, disable the default themes by doing one of the following:

- When creating a new report, in the second dialog of the New Report wizard, deselect Include the default themes. Figure 9-2 shows this option selected, which is the default.

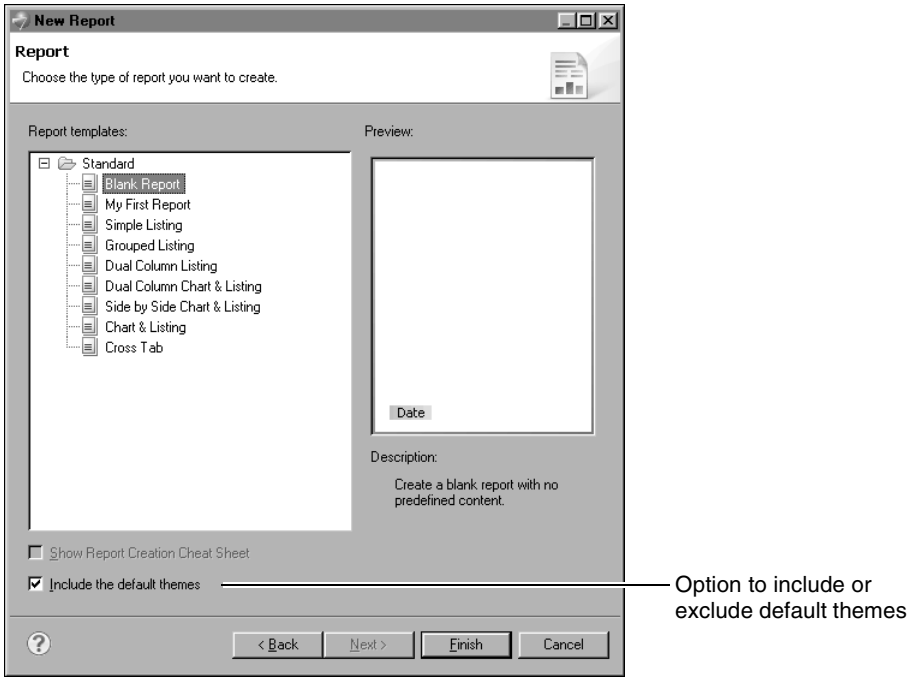


Figure 9-2 Include the default themes selected by default

- If a report already includes the default themes, in the Outline view, expand Libraries, then right-click ThemesReportItems and choose Remove Library, as shown in Figure 9-3.

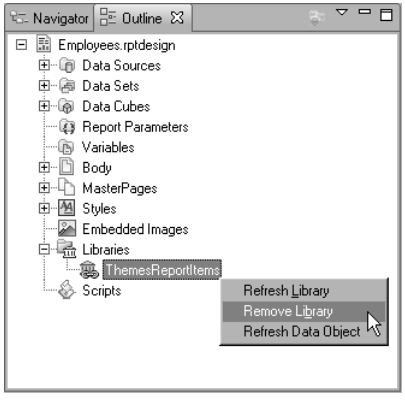


Figure 9-3 Removing ThemesReportItems.rptlibrary from a report

The previous procedures remove all the default themes from a report. You can, however, choose to remove themes from specific report elements while maintaining default themes for other report elements. Figure 9-4 shows an

example of removing a default theme, ThemesReportItems.default-table, from a table by setting the Theme property to None.

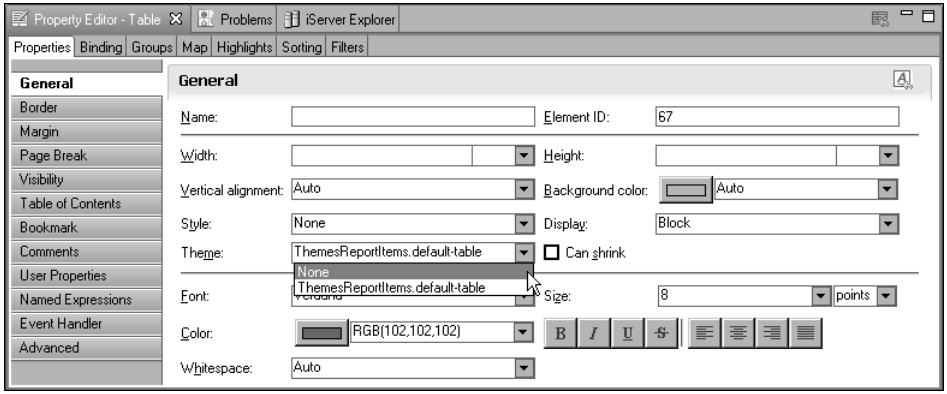


Figure 9-4 Setting a table's Theme property to None

Hiding columns in a table

There are two ways to hide a column in a table. You can:

- Set the column's Display property to No Display.
- Set the column's Visibility property to Hide.

Use the Display property if you are designing a report for Interactive Viewer and you want to hide one or more columns when the report is first displayed in Interactive Viewer. Users viewing the report can then choose to show the hidden columns. The Display property is available under Advanced properties in the Property Editor, as shown in Figure 9-5.

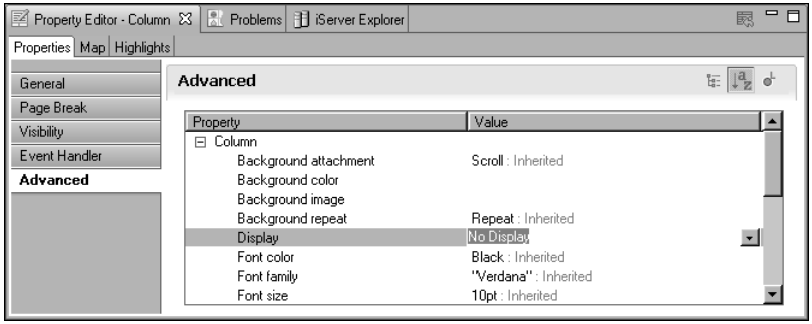


Figure 9-5 Display property of a table column set to No Display

Use the Visibility property to hide a column based on the output format or on a specific condition. For example, you can hide a column in all formats except PDF,

or hide a column if it contains no values. The Visibility property is available under Properties in the Property Editor, as shown in Figure 9-6.

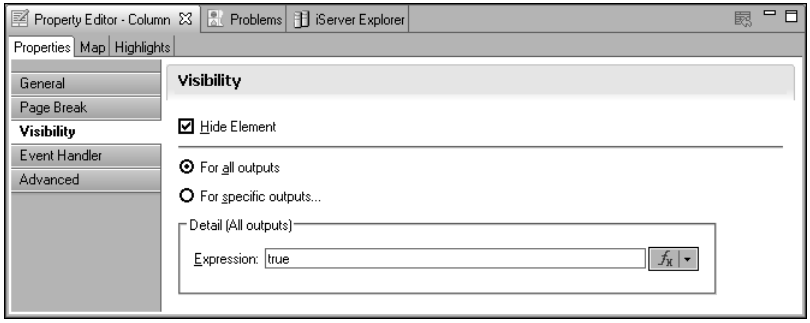


Figure 9-6 Visibility property of a table column set to Hide Element

In releases prior to 11SP1, columns hidden by the Visibility property were available for display in the Interactive Viewer. In releases 11SP1 and later, they are not. Reports created in a release prior to 11SP1 and which used the Visibility property to hide or display a column now exhibit different behavior in Interactive Viewer. To restore the original behavior, change the report to use the Display property instead of the Visibility property.

Designing for optimal viewer performance

Actuate BIRT viewers support a feature called progressive viewing, which displays the first few pages as soon as they are generated instead of waiting until the entire report is generated. For long reports, this feature can significantly reduce the amount of time a user waits before the first page appears.

The design and functionality of a report affect the time it takes for BIRT to generate the initial pages. A major factor that hinders performance is the retrieval of data from an underlying data source, and the storage and processing of all that data before BIRT can render the first report page. Optimal viewing performance occurs when BIRT renders a page as soon as the data for that page has been retrieved, before data for the entire report is processed.

To achieve optimal progressive viewing performance, observe the following guidelines:

- Ensure that data sets return only the data that you want to display in each report element (tables, lists, or charts).

  For example, if the data in a table must be filtered, grouped, sorted, or aggregated, perform these tasks at the data source level. To manipulate data at the table level, BIRT not only has to retrieve and store more data, it also has to spend more time processing the data.

- If, as recommended in the previous point, you create a data set to return data rows that are already grouped, disable the group sorting in BIRT, which occurs when you create a group using the group editor.

  To disable group sorting in BIRT, select the table in which grouping is defined. In Property Editor, choose Advanced, then set the Sort By Groups property to false.

- If creating nested tables (a table within another table) as is common in master-detail reports, create a data set for each table instead of creating a single data set that both the outer and inner tables use.

- Avoid the following items:

  - Top *n* or bottom *n* filters. These filters require that BIRT process an entire set of data to determine the subset of data to display.

  - Aggregations that require multiple passes through data, for example, subtotals as a percentage of a grand total.

  - Summary tables. Even though these tables do not display detail rows, BIRT must still process all the detail rows to calculate and display the summary data.

Chapter

10

Using Flash objects in a report

This chapter contains the following topics:

- About Flash
- Software requirements
- Ways to add Flash objects in a report
- Output formats that support Flash

About Flash

Flash, developed by Adobe Systems, is software commonly used for adding animation and interactivity to web pages, and to create rich Internet applications. Actuate BIRT Designer supports the use of Flash objects, such as Flash charts and gadgets, in reports. Figure 10-1 shows examples of the types of Flash objects that reports can display.

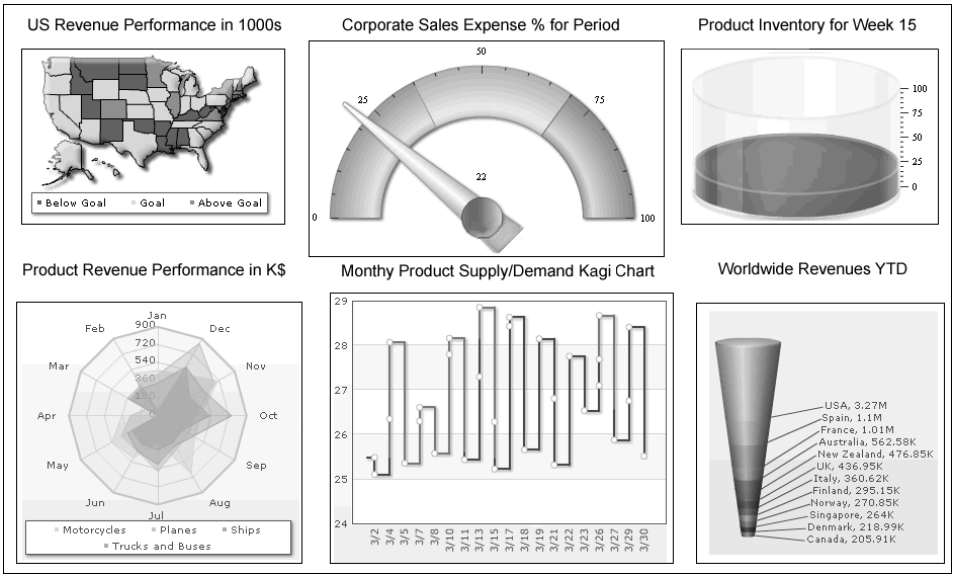


Figure 10-1 Flash charts and gadgets

Software requirements

You must install Adobe Flash Player to view and interact with Flash objects in a report design. Flash player installs as an ActiveX control or browser plug-in. It is available from Adobe at the following location:

`http://www.adobe.com/products/flashplayer`

Ways to add Flash objects in a report

Add Flash objects in a report in any of the following ways. The methods are listed in order of difficulty, from easiest to most difficult:

- Use a built-in Flash chart or Flash gadget. These elements are available in the palette and provide the basic types of charts and gadgets. Actuate BIRT Designer provides tools for creating these elements without any programming. For information about using these elements, see Chapter 11, "Using built-in Flash charts and gadgets."
- Use a Flash chart, gadget, or other object in the InfoSoft Flash Object Library, a third-party library that is packaged with Actuate BIRT Designer. Using a Flash object from this library requires programming in JavaScript or Java to convert data to the XML format required by the object and then to pass the converted data to the object. For information about using objects in the InfoSoft Flash Object Library, see Chapter 12, "Using the Flash object library."
- Use a Flash object from a third-party library other than InfoSoft. The procedure for using this type of Flash object is similar to the procedure for using objects in the InfoSoft Flash Object Library.
- Use a custom Flash object that you or another programmer develops using third-party software. This method provides full control and access to the underlying code of the Flash object, but requires knowledge of how the object is created, as well as, how to integrate and use the object in the report. The procedure for using a custom Flash object is similar to the procedure for using objects in the InfoSoft Flash Object Library.

To determine the method to use, consider the data to present and which type of Flash object is most suitable for the data, then look at the available types of built-in Flash charts and gadgets. For example, if you determine that a doughnut chart is best, you need look no further than the built-in Flash chart. However, if you decide that a Flash map is best, look at the maps included in the InfoSoft Flash Object Library. In most cases, the built-in Flash objects and the InfoSoft Flash Object Library provide all the objects suitable for presenting report data.

Output formats that support Flash

HTML reports display Flash content. Report users must have Flash Player installed. PDF reports can also display Flash content if the Adobe Reader supports Flash. Download a version of Adobe Reader that supports Flash from the following location:

`http://www.adobe.com/acrobat`

If creating a report that contains Flash content and that will be viewed in other formats, such as XLS or DOC, use the visibility property to hide Flash objects in formats that do not support Flash. If you do not hide the Flash objects, the report displays a message, such as "Flash report items are not supported in this report format." As a substitute for a Flash chart or gadget, use a standard chart and set it to appear in formats that do not support Flash content.

Chapter

11

Using built-in Flash charts and gadgets

This chapter contains the following topics:

About Flash charts and gadgets

Flash charts are charts that use visual effects and animation. Actuate BIRT Designer supports the creation of both Flash charts and standard charts. Standard charts are static images whereas flash charts add motion and more visual interest. For example, an animated Flash column chart can progressively draw its columns from the bottom to the top and its *x*-axis labels from left to right. Actuate BIRT Designer provides these Flash chart types: column, bar, line, pie, and doughnut.

Like Flash charts, Flash gadgets display data graphically and with animation. The difference between the two elements is that a gadget typically displays a single value whereas a chart plots multiple values for comparison. The supported Flash gadgets, shown in Figure 11-1, are meter, linear gauge, sparkline, cylinder, thermometer, and bullet.

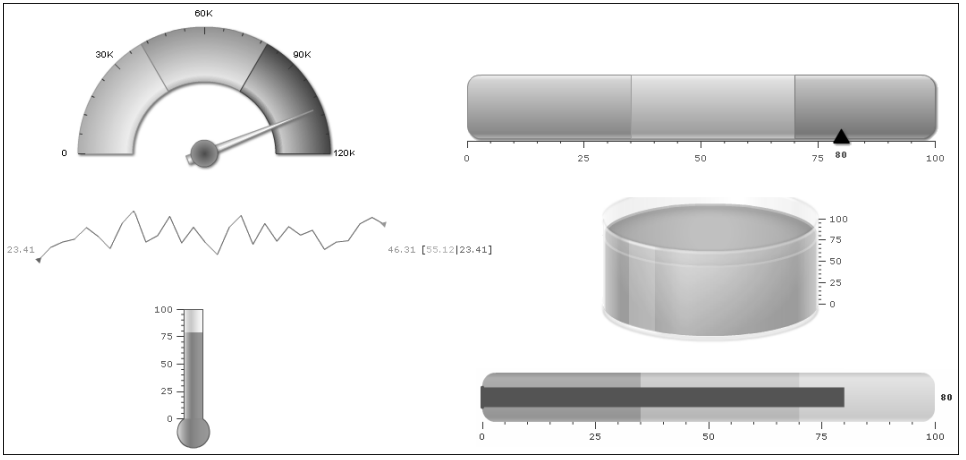


Figure 11-1 Flash gadgets

Creating a Flash chart and gadget

The procedure for creating a Flash chart and gadget is the same as the procedure for creating a standard chart. To create a Flash chart or gadget, perform the following tasks:

- Drag the Flash chart or Flash gadget element from the palette and drop it in the report.
- Choose a type of chart or gadget.
- Specify the data to present in the chart or gadget.
- Format the chart or gadget.

The formatting options available to the Flash elements are different from the formatting options available to standard charts. While many of the chart parts and formatting attributes are the same, Flash lets you add animation and special visual effects to parts of a chart or gadget.

This chapter describes the formatting options that are unique to Flash charts and gadgets. Flash gadgets are covered in more detail because gadgets have features that differ from those in a standard chart. For information about the different chart types, specifying data for a chart, and using the standard formatting options, see *BIRT: A Field Guide*.

Formatting a Flash chart

The Flash chart builder is similar to the standard chart builder. Both provide a separate page for formatting tasks. Figure 11-2 shows an example of the Format Chart page displaying Series properties for a Flash chart. This page is similar to the Format Chart page in the standard chart builder. The primary difference is the capability to add animation and special visual effects, such as bevels, glow, and blur, to a Flash chart. These tasks are described later in this chapter.

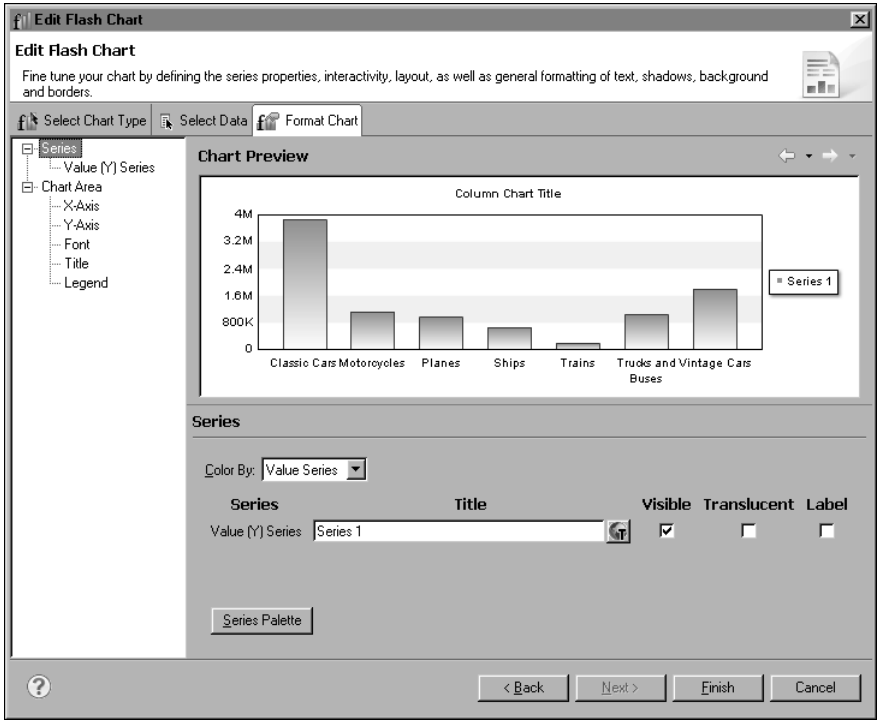


Figure 11-2 Format Chart page in the Flash chart builder

Formatting a Flash gadget

Like the Flash and standard chart builders, the Flash gadget builder provides a separate page for formatting tasks. Figure 11-3 shows an example of the Format Gadget page displaying the general properties for a linear gauge.

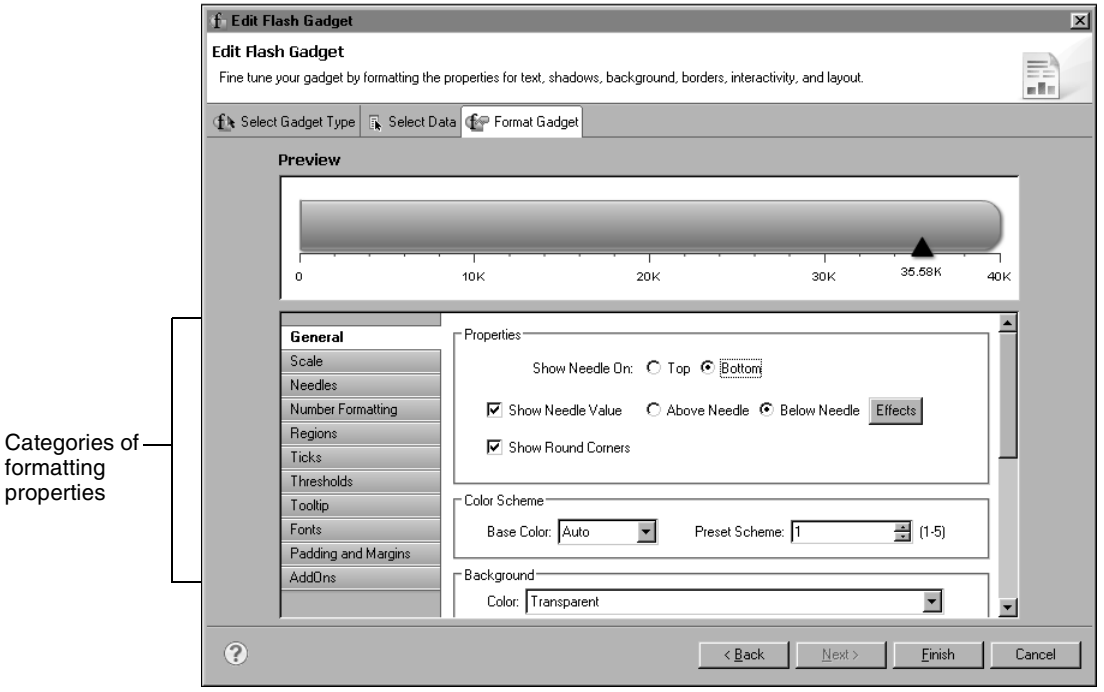


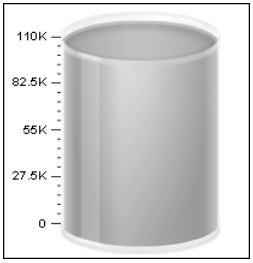
Figure 11-3 Format Gadget displaying a linear gauge and its general properties

Format Gadget lists formatting properties of each visual part of a gadget. As Figure 11-3 shows, for a linear gauge, you can format its scale, needle, numbers, regions, ticks, thresholds, and so on. Each gadget has a different set of formatting properties, which change specific aspects of the gadget's appearance.

General properties

The general properties of a gadget control overall appearance, such as color scheme, background and border style, and whether animation is enabled. General properties can also define the radius of a cylinder gauge, the needle position of a linear gauge, or the start and end angles of a meter gauge. For example, Figure 11-4 shows how changing the Radius, Height, and Viewing Angle properties affects the view of a cylinder gauge gadget. Radius and Height values are expressed as percentages of the gadget area.

Radius: 20%(default)
Height: 50% (default)
Viewing Angle: 30 (default)



Radius: 30%
Height: 60%
Viewing Angle: 0

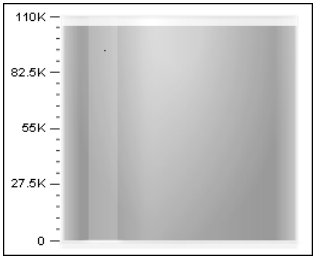
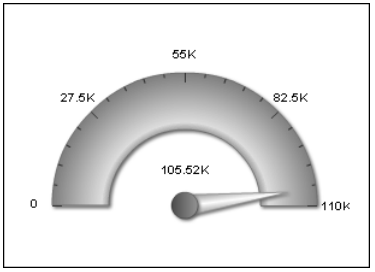


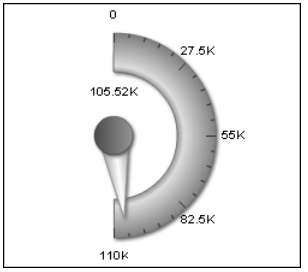
Figure 11-4 Examining results of setting properties for a cylinder gauge

Figure 11-5 shows examples of setting the Start Angle and End Angle properties to change the shape and orientation of a meter gauge. The examples also show how to use the Outer Radius and Inner Radius properties to set the thickness of the arc in the gauge.

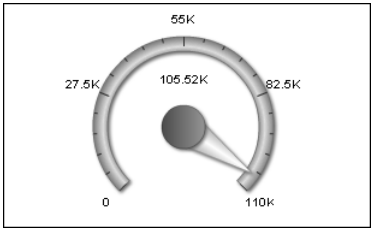
Start Angle: 180 (default)
End Angle: 0 (default)
Outer Radius: 70% of Height (default)
Inner Radius: 40% of Height (default)



Start Angle: 90
End Angle: -90
Outer Radius: 40% of Height
Inner Radius: 25% of Height



Start Angle: 225
End Angle: -45
Outer Radius: 30% of Height
Inner Radius: 25% of Height



Start Angle:45
End Angle: 135
Outer Radius: 50% of Height
Inner Radius: 50% of Height

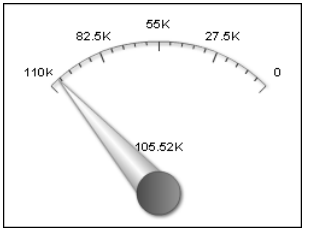


Figure 11-5 Examining results of setting properties for a meter gadget

Table 11-1 shows all the general properties and lists the gadgets to which they apply. Some properties appear for only one type of gadget. Other properties are common to multiple types of gadgets.

Table 11-1 General properties

Property	Gadget	Usage
Background Color	All	Sets the background color of the gadget.
Base Color	All	Sets the color scheme of the gauge. You can use either a base color or a preset color scheme. All other selections derive from this selection.
Center X Coordinate	Meter	Specifies the *x* coordinate of the gauge center.
Center Y Coordinate	Meter	Specifies the *y* coordinate of the gauge center.
Color	All	Specifies the color of the border around the gadget.
Connect Missing Data	Sparkline	Connects a line between missing points of data.
End Angle	Meter	Specifies the angle where the gauge ends drawing.
Fill color	Cylinder, thermometer	Specifies the color of the contained image within a filled type of gadget, such as a cylinder or thermometer.
Height	Cylinder, thermometer	Specifies the percentage of the gadget area that the gadget image height occupies.
Inner Radius	Meter	Specifies the radius of the inner portion of the gauge.
Outer Radius	Meter	Specifies the radius of the outer portion of the gauge.
Preset Scheme	All	Selects a preset color scheme for the gauge. You can use either a base color or a preset color scheme. All other selections derive from this selection.
Radius (or Bulb Radius)	Cylinder, thermometer	Species the percentage of the gadget area that the gadget image radius occupies.
Show Border	All	Enables or disables the border around the gadget.
Show Dial Values	Meter	Enables or disables the value display on the dial. The dial position can be selected to be above or below the dial.
Show Needle On	Linear gauge	Set to top to have needles appear on top of the gadget, set to bottom to have them appear on the bottom.

Table 11-1 General properties

Property	Gadget	Usage
Show Needle Value	Linear gauge	Enables or disables the display of the value at the needle. If enabled, set to Above Needle to display the value above the needle, or set to Below Needle to display the value below the needle.
Show Round Corners	Linear gauge, bullet	Enables or disables rounded corners on the gauge.
Show Value	Cylinder, thermometer	Enables or disables the display of the value the gadget is illustrating.
Start Angle	Meter	Specifies the angle where the gauge begins drawing.
Start X Coordinate	Cylinder	Chooses a starting *x* coordinate percentage that positions the image within the gadget. Selecting 0 starts the image at the left side of the gadget.
Start Y Coordinate	Cylinder	Chooses a starting *y* coordinate percentage that positions the image within the gadget. Selecting 0 places the starting y coordinate at the top of the gadget, selecting 100 places it at the bottom.
Style	All	Supports adding a style to the gadget.
Sub-Title	Sparkline, bullet	Adds a subtitle to the gadget.
Title	Sparkline, bullet	Adds a title to the gadget.
Turn Off All Animations	All	Enables or disables all animation effects.
Turn Off Default Animations	All	Enables or disables default animation.
Viewing angle	Cylinder	Specifies the angle at which the gadget is viewed. Valid values are 0 through 50. 0 appears flat, 50 is tilted towards the viewer.
Width	Linear gauge, meter	Specifies the thickness of the border around the gadget.

Scale properties

Scale properties define the range of values and the number of tick marks that a gadget displays. The scale properties affect the numbers displayed on the gadget, not its size. Minimum Value and Maximum Value specify the lowest and highest numbers, respectively. However, if the data set value (represented by the needle value) is lower than the minimum value or higher than the maximum value, the minimum or maximum value is ignored.

Figure 11-6 shows scale properties set for a linear gauge.

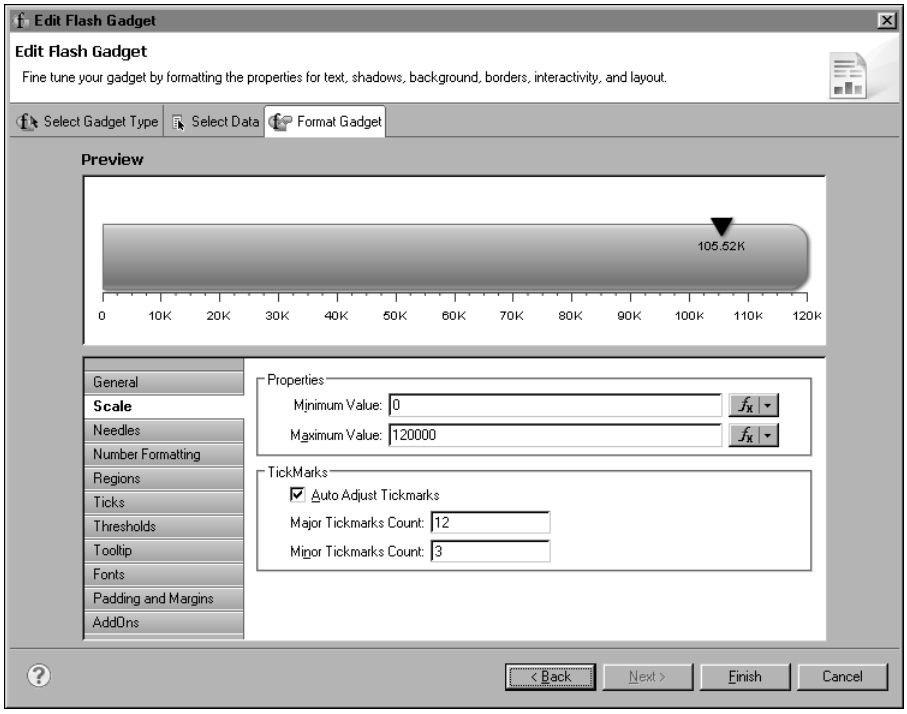


Figure 11-6 Format Gadget displaying a linear gauge and its scale properties

Table 11-2 shows all the scale properties and lists the gadgets to which they apply.

Table 11-2 Scale properties

Property	Gadget	Usage
Auto Adjust Tickmarks	All but sparkline	Enables or disables tick marks created evenly across the scale
Major Tickmarks Count	All but sparkline	Specifies the number of major tick marks to display on the scale
Maximum Value	All	Sets the highest value of the scale
Minimum Value	All	Sets the lowest value of the scale
Minor Tickmarks Count	All but sparkline	Specifies the number of minor tick marks to display between major tick marks

Needle properties

Needle properties define the shape, size, and color of a needle. A needle appears only in a linear gauge and in a meter gauge, and is used to point to a data value. Figure 11-7 shows the needle properties set for a meter gauge.

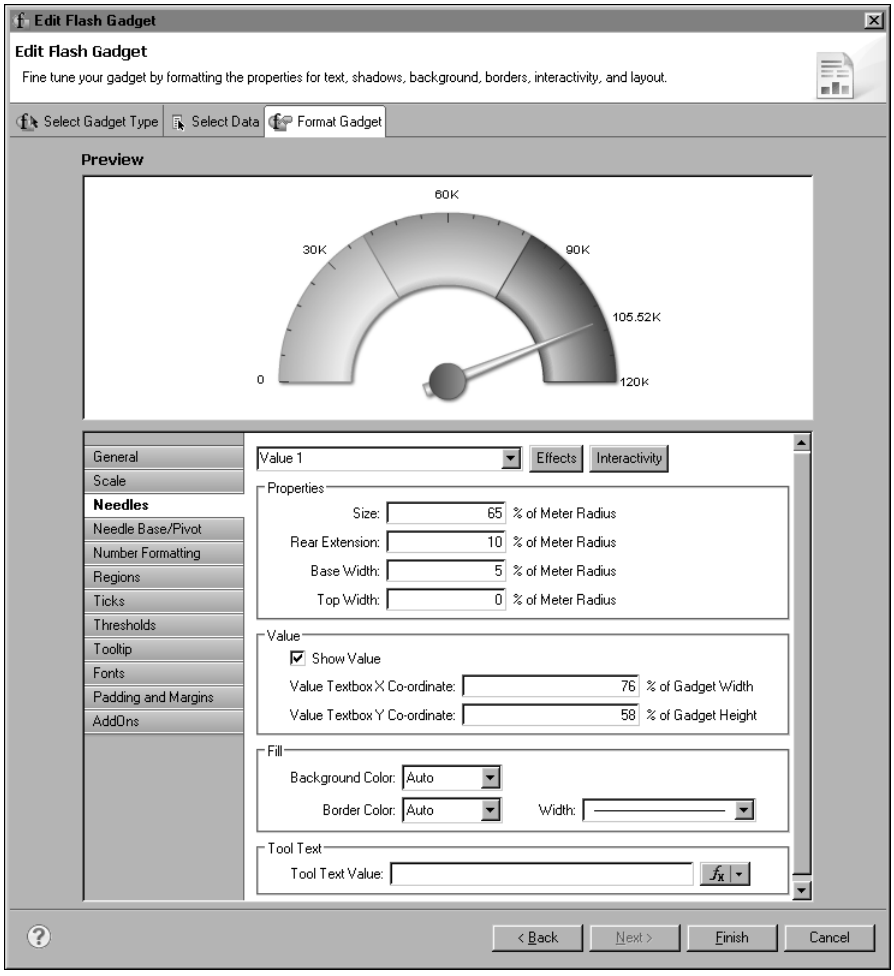


Figure 11-7 Selecting options for the needle of a meter gauge gadget

For a meter gauge, the needles properties apply only to the pointer part of the needle. To format the base, or pivot, of the needle (represented by the circle), choose Needle Base/Pivot.

Table 11-3 shows all the needle properties and lists the gadgets to which they apply.

Table 11-3 Needle properties

Property	Gadget	Usage
Base Width	Meter	Sets the size of the bottom part of the needle, as a percent of the size of the gadget.
Border Color	Linear gauge, meter	Sets the border color of the needle.
Border Width	Linear gauge, meter	Sets the thickness of the needle border.
Fill Background Color	Meter	Sets the background color of needle.
Fill Color	Linear gauge	Sets the interior color of the needle.
Rear Extension	Meter	Sets the size of the portion of the needle behind the pivot as a percent of the size of the gadget.
Shape	Linear gauge	Sets the shape of the needle.
Show Value	Meter	Enables or disables the display of the value to which the needle points.
Size	Linear gauge, meter	Sets the size in pixels, or in percent of gadget width, of the needle.
Tooltip	Linear Gauge, meter	Specifies text for the tooltip.
Top Width	Meter	Sets the size of the tip of the needle as a percent of the size of the gadget.
Value	Linear gauge, meter	Sets which needle to format. Several needles can co-exist, based on the data used to create the gadget.
Value Textbox X Co-ordinate	Meter	Sets the *x* coordinate of the value text, as a percent of gadget width.
Value Textbox Y Co-ordinate	Meter	Sets the *y* coordinate of the value text, as a percent of gadget height.

Needle base or pivot properties

Needle base or pivot properties define the appearance of a needle base, or pivot. Drawn as a circle, the base is the point around which the needle rotates. A needle base appears only for a meter gauge. Figure 11-8 shows the needle base properties set for a meter gauge. The size of the needle base is larger than the default size, and the fill color is set to a radial gradient.

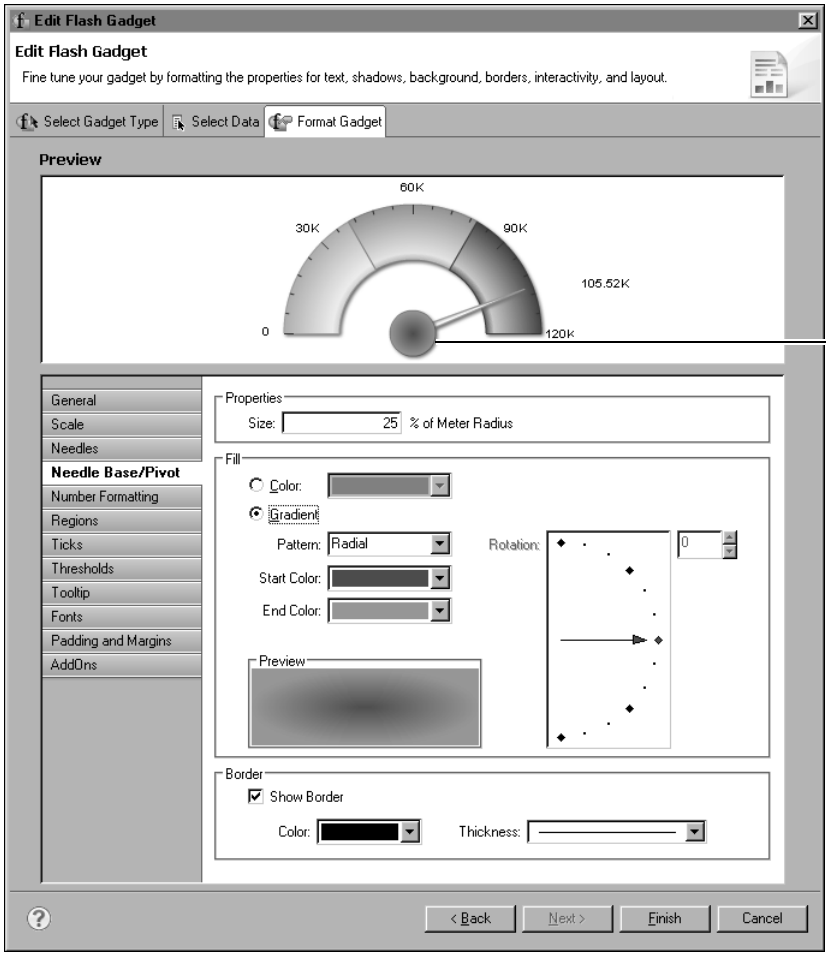


Figure 11-8 Selecting options for the needle base of a meter gauge

Table 11-4 shows all the needle base or pivot properties. These properties are used only in a meter gauge.

Table 11-4 Needle base/pivot properties

Property	Usage
Border Color	Sets the border color of the needle base.
Border Thickness	Sets the width of the needle base border.
End Color	Sets the ending color to use in a fill gradient.
Fill Color	Sets the interior color of the needle base to a solid color.

(continues)

Table 11-4 Needle base/pivot properties (continued)

Property	Usage
Fill Gradient	Sets the interior color of the needle base to a color gradient.
Pattern	Specifies the pattern of the fill gradient. Choose Radial or Linear.
Rotation	Sets the angle of a linear fill gradient.
Show Border	Displays or hides the border around the needle base.
Size	Sets the size of the needle base as a percent of the meter radius.
Start Color	Sets the starting color to use in a fill gradient.

Number formatting properties

Number formatting properties define how numbers are displayed in a gadget. Use these properties to abbreviate numbers, to add text before or after a number, or to specify the number of digits to display after a decimal point. Figure 11-9 shows the number formatting properties set for a thermometer gauge. Numbers display with the dollar symbol ($) before the number and they appear in abbreviated format, such as $30K instead of $30,000.

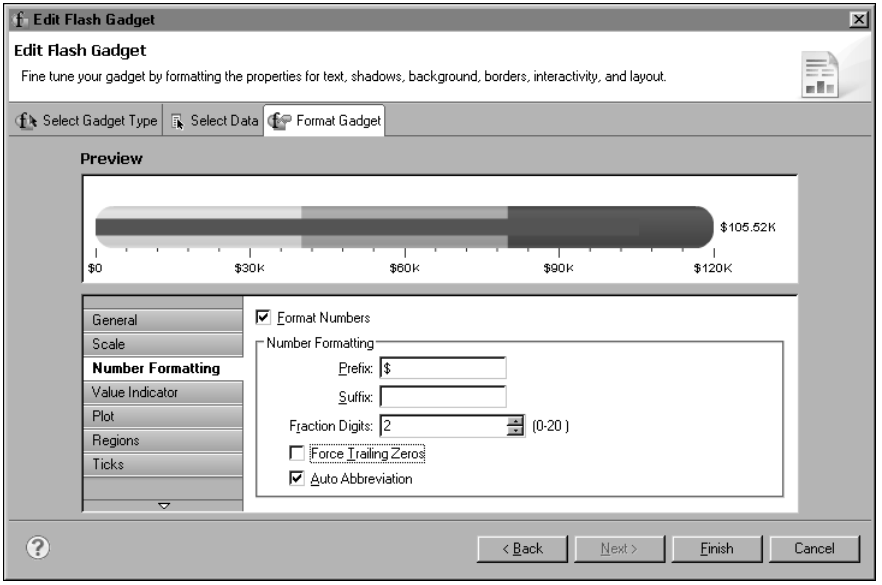


Figure 11-9 Examining a linear gauge and its number formatting properties

Table 11-5 shows all the number formatting properties. These properties are used in all the gadgets.

Table 11-5 Number formatting properties

Property	Usage
Auto Abbreviation	Abbreviates a number to an appropriate number factor. For example, 10,000 becomes 10K.
Force Trailing Zeros	Enables or disables the display of trailing zeros after the decimal point.
Format Numbers	Enables and disables number formatting.
Fraction Digits	Specifies the number of digits displayed after the decimal point.
Prefix	Specifies a text value to display before a number.
Suffix	Specifies a text value to display after a number.

Region properties

Region properties enable the division of the data plot into regions. Use regions to provide more information about values in a gadget. Compare the linear gauges in Figure 11-10 and Figure 11-11. The gauge in Figure 11-10 does not show regions. The gauge in Figure 11-11 displays three regions, labeled Fair, Good, and Excellent.

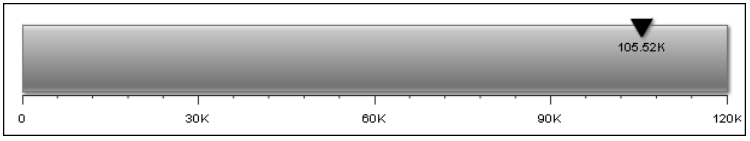


Figure 11-10 Linear gauge without regions

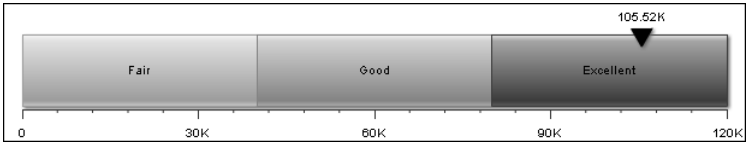


Figure 11-11 Linear gauge with three regions

Figure 11-12 shows the properties set for the region labeled Fair in Figure 11-11.

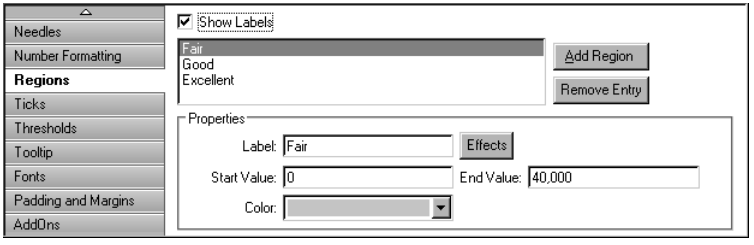


Figure 11-12 Properties specified for a region labeled Fair

Table 11-6 shows all region properties and lists the gadgets to which they apply.

Table 11-6 Region properties

Property	Gadget	Usage
Color	Linear gauge, meter, bullet	Specifies the color of the region.
End Value	Linear gauge, meter, bullet	Specifies where the region ends.
Label	Linear gauge, meter, bullet	Specifies the name of the region.
Region	Linear gauge, meter, bullet	Chooses the region for which the settings apply. You can also add or remove a region from the list.
Show Labels	Linear gauge	Display or hide the region labels.
Start Value	Linear gauge, meter, bullet	Specifies where the region starts.

Tick properties

Tick properties define the size, color, and position of tick marks on a gadget. Figure 11-13 shows the tick properties set for a linear gauge. Tick marks appear at the top and inside the gauge. The first and last tick values display Min and Max instead of numbers.

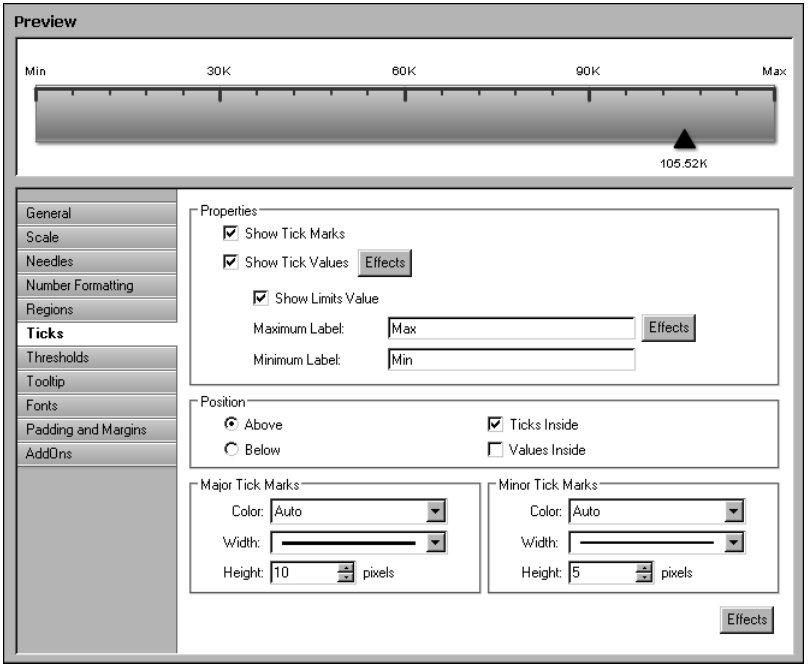


Figure 11-13 Format Gadget displaying a linear gauge and its tick properties

Table 11-7 shows all the tick properties and lists the gadgets to which they apply.

Table 11-7 Tick properties

Property	Gadget	Usage
Major Tick Marks Color	Linear gauge, meter, bullet, cylinder, thermometer	Sets the color of major tick marks.
Major Tick Marks Height	Linear gauge, meter, bullet, cylinder, thermometer	Sets the height of major tick marks.
Major Tick Marks Width	Linear gauge, meter, bullet, cylinder, thermometer	Sets the width of major tick marks.
Maximum Label	Linear gauge, meter, bullet, cylinder, thermometer	Sets the highest tick mark value. Text replaces the numeric value.
Minimum Label	Linear gauge, meter, bullet, cylinder, thermometer	Sets the lowest tick mark value. Text replaces the numeric value.
Minor Tick Marks Color	Linear gauge, meter, bullet, cylinder, thermometer	Sets the color of minor tick marks.
Minor Tick Marks Height	Linear gauge, meter, bullet, cylinder, thermometer	Sets the height of minor tick marks.
Minor Tick Marks Width	Linear gauge, meter, bullet, cylinder, thermometer	Sets the width of minor tick marks.
Position	Cylinder, thermometer	Positions tick marks on the right side of the gadget.
Position Above	Linear gauge, meter, bullet	Sets tick marks to appear above the gadget.
Position Below	Linear gauge, meter, bullet	Sets tick marks to appear below the gadget.
Position Left	Cylinder, thermometer	Positions tick marks on the left side of the gadget.
Show Limits Value	Linear gauge, meter, bullet, cylinder, thermometer	Enables or disables the display of the first and last values.
Show Tick Marks	Linear gauge, meter, bullet, cylinder, thermometer	Enables or disables the display of tick marks on the gadget.
Show Tick Values	Linear gauge, meter, bullet, cylinder, thermometer	Enables or disables the display of values on tick marks.
Ticks Inside	Linear gauge, meter, bullet	Sets tick marks to appear inside or outside of the gadget.
Values Inside	Linear gauge, meter, bullet	Sets tick mark values to appear inside or outside of the gadget.

Threshold properties

Threshold properties define thresholds, which you use to identify meaningful values. For example, in a linear gauge that displays a sales total, you can add a threshold that identifies the target sales amount, as shown in Figure 11-14. By displaying this threshold value, the gauge shows whether the actual sales total is over or under the sales target.

Figure 11-14 also shows the threshold properties set to create the threshold. You can specify a label, create a threshold line or a threshold zone, specify a threshold value or range of values, and format the line and marker. You can create multiple thresholds for a gadget.

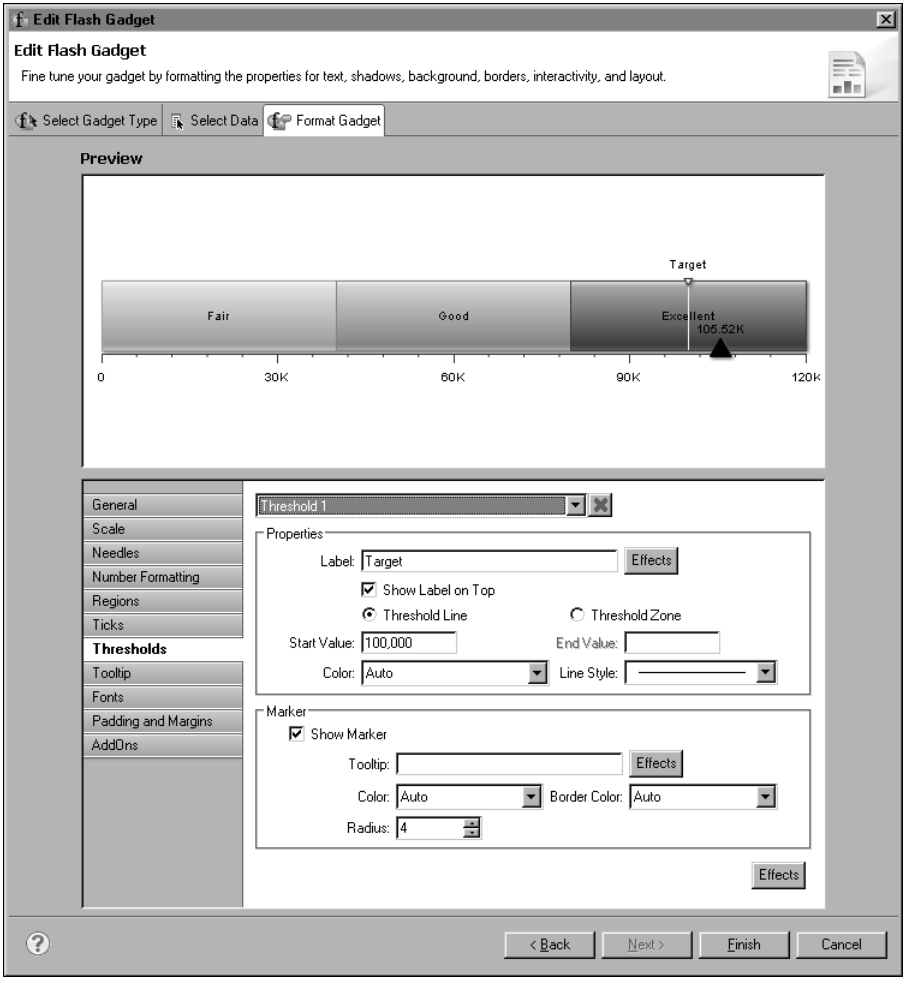


Figure 11-14 Examining a linear gauge and its threshold properties

Table 11-8 shows all the threshold properties and lists the gadgets to which they apply.

Table 11-8 Threshold properties

Property	Gadget	Usage
Arc Inner Radius	Meter	Specifies the inner radius of arc for the threshold area
Arc Outer Radius	Meter	Specifies the outer radius of arc for the threshold area
Border Color	Linear gauge, meter	Sets the border color of the threshold marker
Color	Linear gauge, meter, sparkline	Sets the color of the threshold area on the gadget
End Value	Linear gauge, meter, sparkline	Sets the end value of the threshold zone
Label	Linear gauge, meter	Specifies the text to apply to the threshold
Length	Bullet	Specifies the length of the threshold as a percent of gadget size
Line Style	Linear gauge, meter, sparkline	Sets the line style of the threshold
Marker Color	Linear gauge, meter	Sets the color of the threshold marker
Radius	Linear gauge	Sets the size of the threshold marker
Show as Zone	Sparkline	Enables or disables display of the threshold as a zone
Show Border	Meter	Enables or disables display of a border around the threshold
Show Marker	Linear gauge, meter	Enables or disables display of the marker on the threshold
Show Threshold	Sparkline, bullet	Enables or disables display of the threshold
Show Value	Meter	Enables or disables display of the threshold value
Show Value Inside	Meter	Displays value inside or outside of the arc on the gadget
Show Value on Top	Linear gauge	Enables or disables display of the threshold value
Size	Meter	Sets the size of the threshold marker
Start Value	Linear gauge, meter, sparkline	Sets start value of the threshold zone
Threshold	Linear gauge, meter	Sets which threshold the settings affect

(continues)

Table 11-8 Threshold properties (continued)

Property	Gadget	Usage
Threshold Line/ Threshold Zone	Linear gauge, meter	Sets whether the threshold is a single line or a zone
Tooltip	Linear gauge, meter	Sets tooltip text for the marker on the threshold
Width	Sparkline, bullet	Sets the width of the threshold

Anchor properties

Anchor properties control the shape, size, color, and visibility of markers, or anchors, in a sparkline gadget. Unlike other gadgets that display only one or two data values, a sparkline gadget plots multiple values and, by default, uses anchors to highlight the first, last, lowest, and highest values. Figure 11-15 shows the anchor properties set for a sparkline gadget.

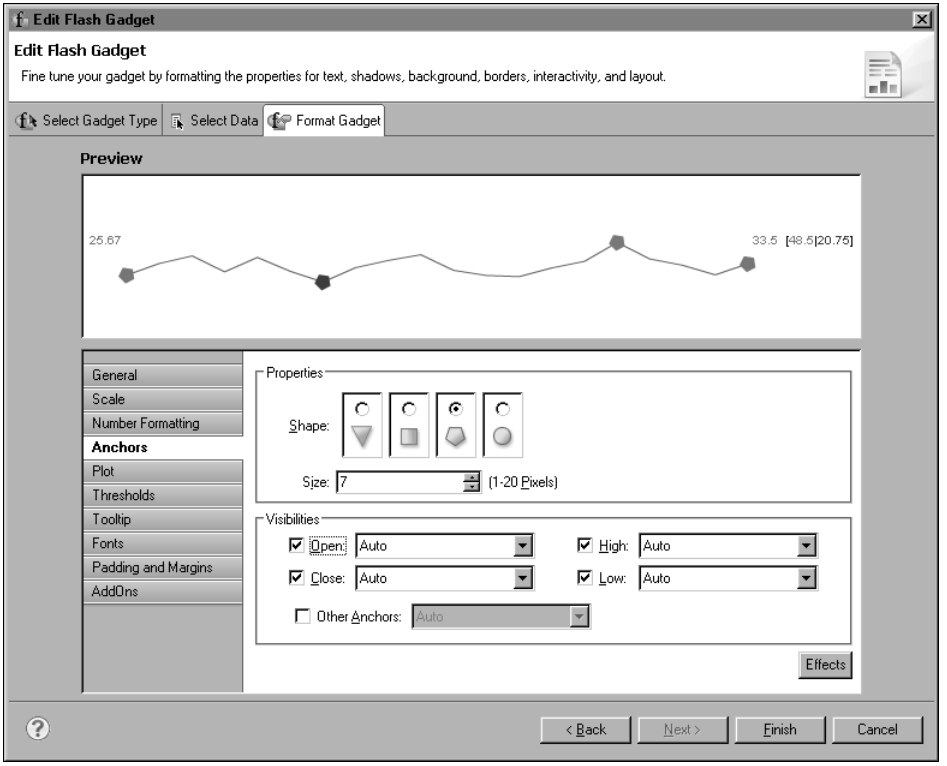


Figure 11-15 Examining a sparkline gadget and its anchor properties

Table 11-9 shows all the anchor properties. These properties are used only in a sparkline gadget.

Table 11-9 Anchor properties

Setting	Usage
Shape	Sets the shape of the anchors.
Size	Sets the size of the anchor in pixels.
Visibilities	Sets the visibility and color of the anchors. Open, Close, High, and Low anchors are visible by default. To display anchors for all the other values, select Other Anchors.

Plot properties

Plot properties control the appearance of elements in the data plot area of bullet and sparkline gadgets. For a bullet gadget, you can add a border around the gadget or a shadow below it. You can also specify whether to display the value label and whether to display the value indicator as a line or as a dot.

For a sparkline gadget, you can specify whether to display the first, last, lowest, or highest values, change the color and width of the data line, and add bars in the background to represent period blocks. For example, if a sparkline displays daily stock quotes over a month, you can show period blocks that have a length of 5 to divide the stock values into weeks.

For example, Figure 11-16 shows the preview of a sparkline gadget. The gadget displays period bars that span five values. Alternate bars appear in color.

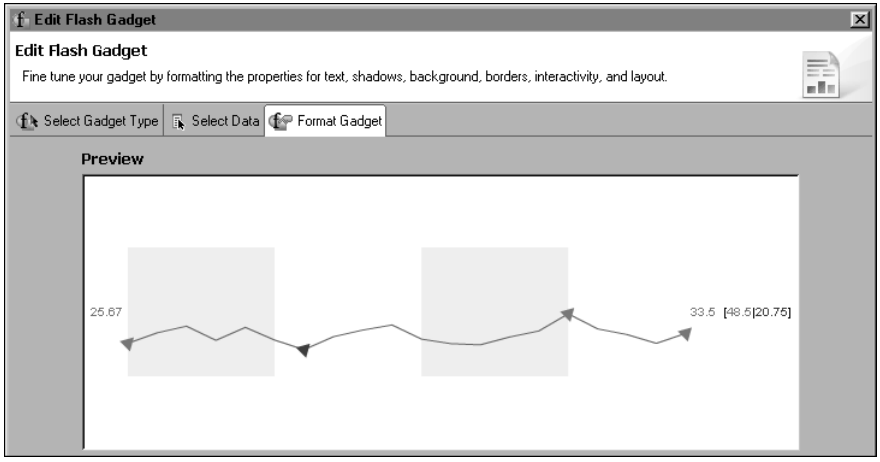


Figure 11-16 Format Gadget displaying a sparkline gadget

Figure 11-17 shows the plot properties specified for the plot that appears in the sparkline gadget example shown in Figure 11-16.

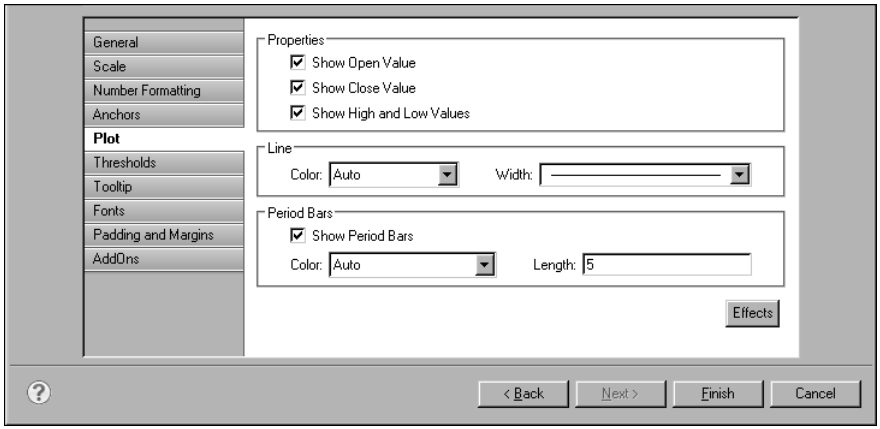


Figure 11-17 Examining the plot properties for a sparkline gadget

Table 11-10 shows all the plot properties.

Table 11-10 Plot properties

Property	Gadget	Usage
Border	Bullet	Enables or disables the border around the gadget.
Border Color	Bullet	Sets the color of the border around the gadget.
Border Width	Bullet	Sets the thickness of the border around the gadget.
Line Color	Sparkline	Sets the color of the plot line.
Line Width	Sparkline	Sets the thickness of the plot line.
Period Bars Color	Sparkline	Sets the color of the period bars. The color is applied to alternate bars.
Period Bars Length	Sparkline	Sets the number of values that each period bar highlights.
Show as Dot	Bullet	Enables or disables the display of the value indicator as a dot instead of a solid line.
Show Close Value	Sparkline	Enables and disables the display of the close value.
Show High and Low Values	Sparkline	Enables and disables the display of the high and low values.
Show Open Value	Sparkline	Enables and disables the display of the open value.

Table 11-10 Plot properties

Property	Gadget	Usage
Show Period Bars	Sparkline	Enables and disables the display of period bars.
Show Shadow	Bullet	Enables or disables the appearance of a shadow below the gadget.
Show Value Label	Bullet	Enables or disables the display of the value on the gadget.

Value indicator properties

Value indicator properties control the size, color, and border of the value indicator in a bullet gadget, as shown in Figure 11-18.

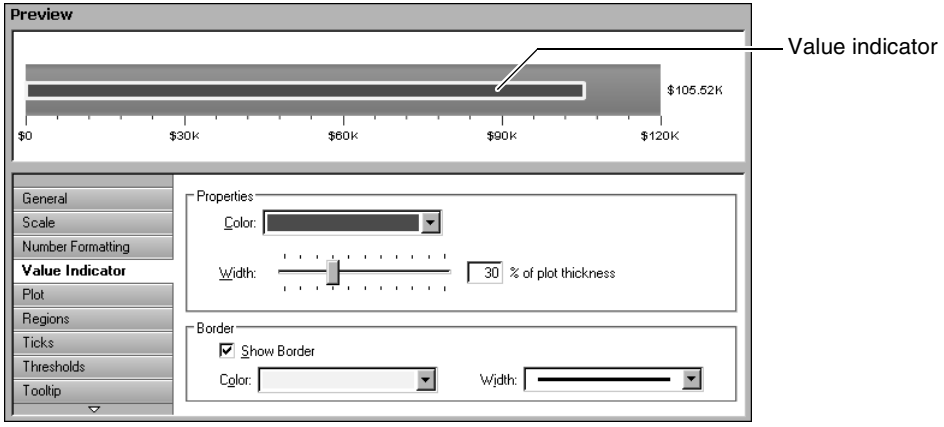


Figure 11-18 Examining a bullet gadget and its value indicator properties

Table 11-11 shows the value indicator properties. These properties are used only in a bullet gadget.

Table 11-11 Value indicator properties

Property	Gadget	Usage
Border Color	Bullet	Sets the color of the border
Border Width	Bullet	Sets the thickness of the border
Color	Bullet	Sets the color of the value indicator
Show Border	Bullet	Enables or disables a border around the value indicator
Width	Bullet	Sets the value indicator width as a percent of the plot thickness

Tooltip properties

Tooltip properties control the visibility and appearance of tooltips in a gadget. A tooltip displays a data value when the mouse pointer is placed over a value marker. Figure 11-19 shows a bullet gadget that displays a tooltip and the properties set for the tooltip.

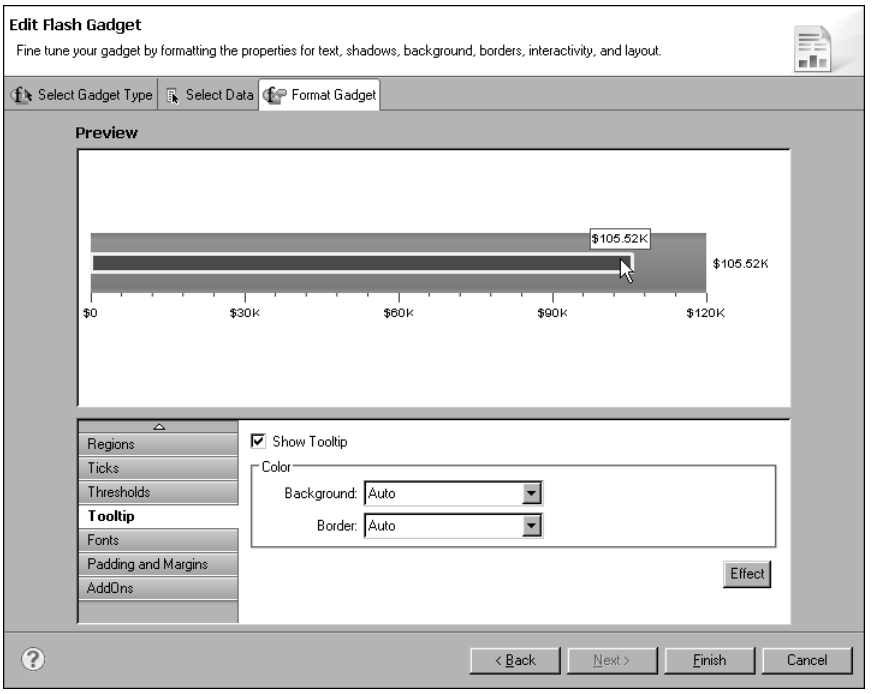


Figure 11-19 Format Gadget displaying a bullet gadget and its tooltip properties

Table 11-12 shows the tooltip properties. These properties are available to all the gadgets.

Table 11-12 Tooltip properties

Property	Usage
Show Tooltip	Enables and disables the display of a tooltip
Background	Sets the background color for the tooltip
Border	Sets the border color for the tooltip

Font properties

Font properties define the type, size, and color of the font used for any text in a gadget. Table 11-13 shows the font properties. These properties are available to all the gadgets.

Table 11-13 Font properties

Property	Usage
Font	Specifies the name of the font
Size	Specifies the font size in points
Color	Specifies the color of the text

Padding and margin properties

Padding and margin properties support the addition of space on all sides of a gadget, between a title and the plot, and between a data value and the plot. Compare the sparkline gadgets in Figure 11-20 and Figure 11-21. The gadget in Figure 11-20 uses default values for all the padding and margin properties.

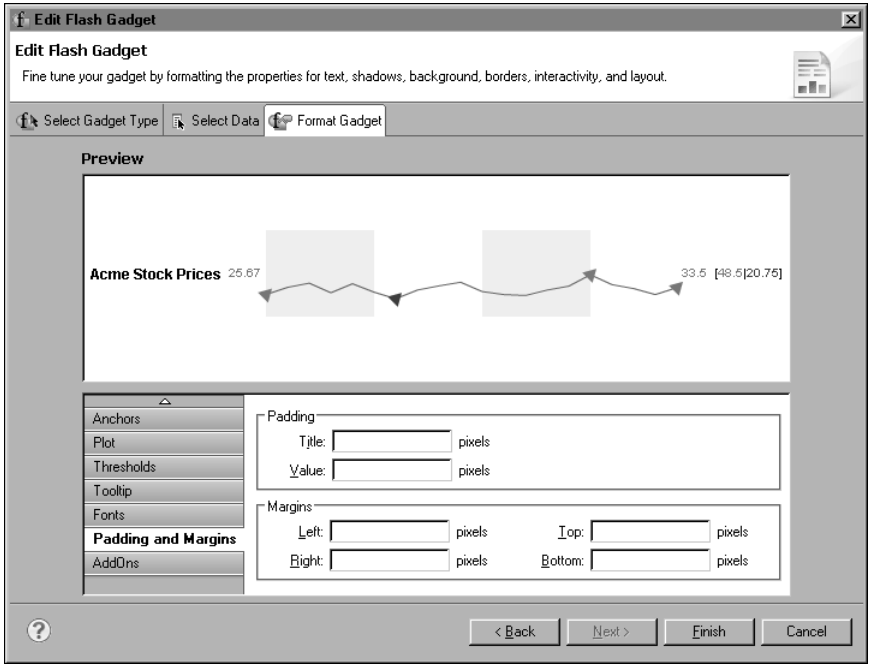


Figure 11-20 Format Gadget displaying a sparkline gadget and its default padding and margin property settings

The gadget in Figure 11-21 uses the margin and padding properties to add extra space between the elements in the gadget.

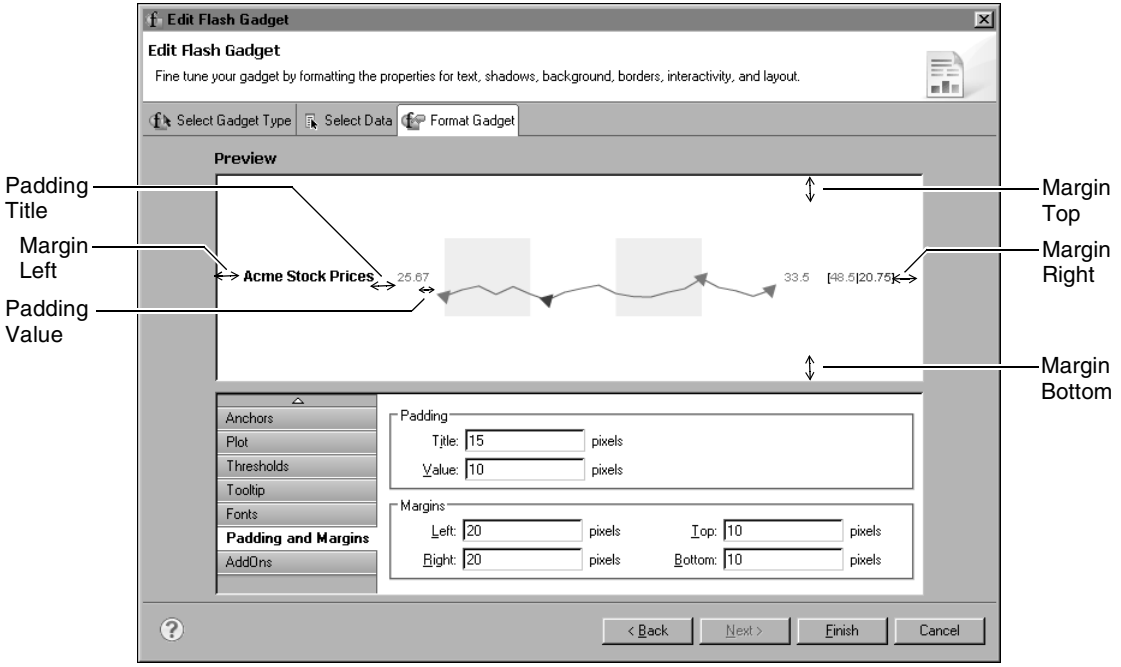


Figure 11-21 Format Gadget displaying a sparkline gadget that uses padding and margin properties to add extra space between elements

Table 11-14 shows the padding and margin properties.

Table 11-14 Padding and margin properties

Property	Gadget	Usage
Padding Title	Sparkline, bullet	Adds space, in pixels, between the title and the element next to it
Padding Value	All	Adds space, in pixels, between the data value and the element next to it
Margins Left, Right, Top, Bottom	All	Adds space, in pixels, around the entire gadget on the left, right, top and bottom sides

AddOn properties

AddOn properties support the creation of custom objects, called AddOns, to add to a gadget. You can add rectangles, polygons, circles, arcs, lines, text, and images

to any gadget to enhance its appearance. You can create any number of objects and arrange objects on top of or behind one another.

Figure 11-22 shows an example of adding two rectangles with rounded corners behind a meter gauge. To create this image, create one rectangle with a white border, then create another rectangle that is slightly larger. Use the same fill color for both rectangles. Place the larger rectangle behind the smaller rectangle.

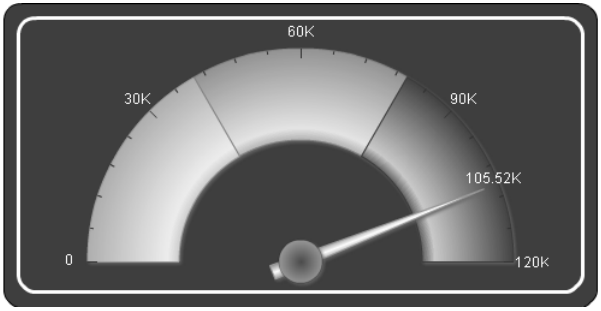


Figure 11-22 AddOn objects used to enhance a meter gadget

Figure 11-23 shows the AddOns page. AddOns lists the two rectangles added to the meter gauge. The objects are listed in z order, which is the order from front to back.

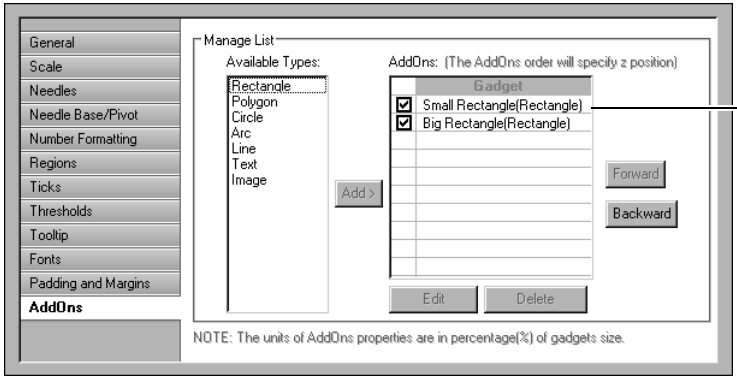


Figure 11-23 Format Gadget displaying a meter gauge and its AddOn properties

Figure 11-24 shows the properties set for the larger rectangle. Notice that the size of the rectangle is not fixed. Rather, the size is a percentage of the gadget's size. You define an AddOn's size by specifying values for these four properties: Start X coordinate, Start Y coordinate, End X coordinate, and End Y coordinate. By using a relative size, AddOns adjust to the size of the gadget area.

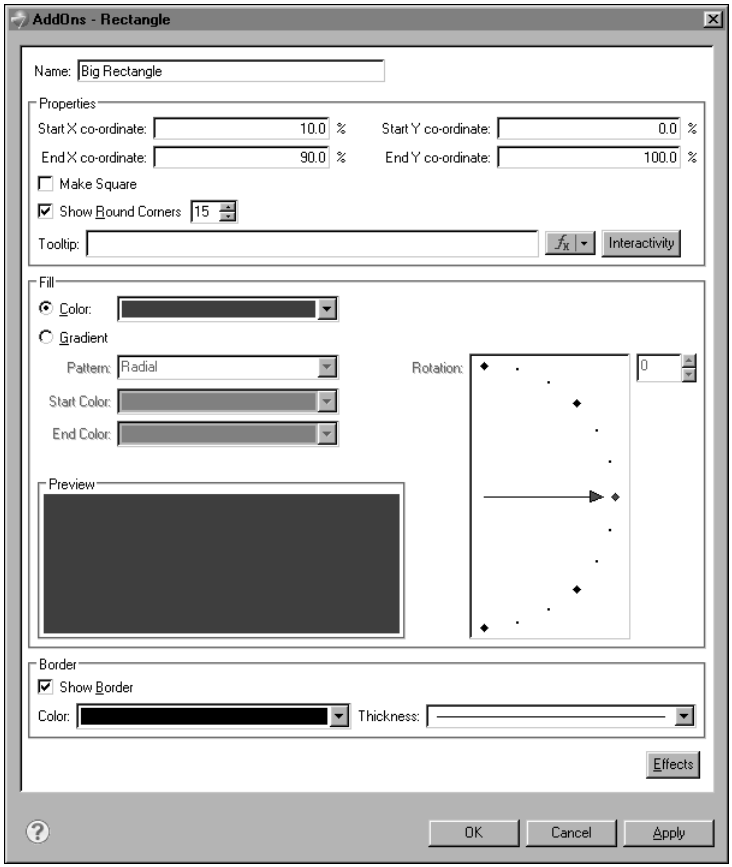


Figure 11-24 Properties of an AddOn object

Table 11-15 shows the properties for creating the different types of objects that you can add to a gadget.

Table 11-15 AddOn properties

Property	Object Type	Usage
Center X coordinate	Polygon, Circle, Arc	Specifies the location, as a percentage of the size of the gadget, of the *x* coordinate of the object.
Center Y Coordinate	Polygon, Circle, Arc	Specifies the location, as a percentage of the size of the gadget, of the *y* coordinate of the object.
Color	Line, Text	Specifies the color of the line.
Dash Gap	Line	Specifies the length of gaps between dashes, in pixels.
Dash Length	Line	Specifies the length of dashes, in pixels.

Table 11-15 AddOn properties (continued)

Property	Object Type	Usage
End Angle	Circle, Arc	Specifies the end angle of the object.
End Color	Rectangle, Polygon, Circle, Arc	Specifies the end color of the gradient fill.
End X coordinate	Rectangle	Specifies the location, as a percentage of the size of the gadget, of the end *x* value of the object.
End Y coordinate	Rectangle	Specifies the location, as a percentage of the size of the gadget, of the end *y* value of the object.
Font	Text	Specifies the font for the object. You can also select Bold, Italic, or Underline.
Font Size	Text	Specifies the font size in points.
Horizontal	Text	Supports the selection of horizontal text alignment within the gadget.
Inner Radius	Arc	Specifies the radius of the inner portion of the object, as a percent of the size of the gadget.
Gradient	Rectangle, Polygon, Circle, Arc	Select to have a gradient type of fill. Choose a Radial or Linear pattern.
Label	Text	Specifies the text that appears on the object.
Name	All	Specifies the name of the object. This name appears in the list on AddOns options.
Outer Radius	Arc	Specifies the radius of the outer portion of the object, as a percent of the size of the gadget.
Radius	Circle	Specifies the radius, as a percent of the gadget, of the object.
Rotation	Rectangle, Polygon, Circle, Arc	Specifies the rotation angle for the fill within the object.
Rotation Angle	Polygon	Specifies the rotation angle of the object.
Scale Image	Image	Enables or disables image scaling. Adjust the height and width of the image by percent.
Scale This Font	Text	Select to alter the size of the text. Adjust the scaling amount for width and height by percent.
Show as Dashed	Line	Enables or disables dashed lines.
Show Border	Rectangle, Polygon, Circle, Arc	Enables or disables the drawing of a border line around the object. Select the color and thickness of the border with the Color and Thickness drop-down menus.

(continues)

Table 11-15 AddOn properties (continued)

Property	Object Type	Usage
Show Round Corners	Rectangle	Enables or disables rounded corners. Type the percent of a circle to round the corner.
Sides	Polygon	Specifies the number of sides on the object.
Size	Polygon	Specifies the size, as a percent of the gadget, of the object.
Solid Color	Rectangle, Polygon, Circle, Arc	Select to use a solid fill color for the object. Select a color from the associated drop-down listbox.
Start Angle	Circle, Arc	Specifies the beginning angle of the object.
Start Color	Rectangle, Polygon, Circle, Arc	Specifies the start color of the gradient fill.
Start X coordinate	Rectangle	Specifies the location, as a percentage of the size of the gadget, of the beginning *x* value of the object.
Start Y coordinate	Rectangle	Specifies the location, as a percentage of the size of the gadget, of the beginning *y* value of the object.
TextBox Background Color	Text	Specifies the background color of the text box.
TextBox Border Color	Text	Sets the border color of the text box.
Text Wrap	Text	Disables or enables text wrap. Choose, by percent of the gadget, the maximum height and width for the wrap.
Thickness	Line	Specifies the thickness of the line.
Transparent	Image	Specifies the amount of transparency, in percent, of the image.
URL	Image	Specifies the location of the image for AddOn file types of .gif, .jpg, .png, or .swf.
Vertical	Text	Supports the selection of vertical text alignment within the gadget.

Using animation and other visual effects

By default, every Flash chart and gadget animates the data plot, the part of the chart or gadget that represents the data. For example, the columns in a column chart grow vertically and horizontally, and the pie and doughnut charts rotate.

You can change the default animation and define custom animations and visual effects. The types of visual effects you can apply are shadow, glow, bevel, blur, and font.

When defining custom animations and visual effects, do the following:

- Select the part of the chart or gadget to animate or to apply a visual effect.
- Select the type or types of effects to apply and set their properties.

If a standard formatting property and an effect property are set for the same chart or gadget part, the effect property takes precedence. For example, if the font property and a font effect are set for the *x*-axis labels, the font effect is used.

Creating effects

There are two approaches to designing effects for a Flash chart or gadget. You can create one or both of the following effects:

- A specialized effect that applies to a single chart or gadget part
- A general purpose effect that applies to multiple parts of a chart or gadget

The first approach is typical. For example, you might create one animation effect that draws a chart's *x*-axis labels horizontally, and a second animation effect that draws *y*-axis labels vertically.

Use the second approach to apply the same animation or visual effects to more than one chart or gadget part. For example, to apply the same font properties to the legend title, *x*-axis labels, and *y*-axis labels, create an effect with the desired font properties, then apply this effect to the three chart parts. Whenever you need to change the font for these chart parts, you modify a single effect. This approach enables you to reuse and maintain common effects easily.

How to create an effect

1. Select the part of the chart or gadget to which to apply an effect. If the selected part supports effects, the Effects button appears.
2. Choose Effects. Effects shows the part of the chart or gadget selected for an effect. Figure 11-25 shows an example of X-Axis Labels selected for a Flash chart.

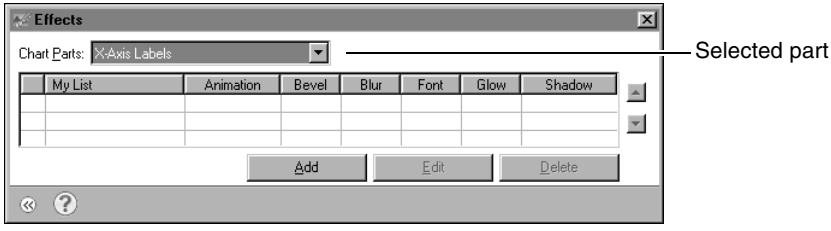


Figure 11-25 Effects displaying the chart part selected for an effect

3 To apply an effect to a different part of the chart or gadget, select a different part from the drop-down list.

4 Choose Add to create an effect.

5 In Add New Effect, type a name for the effect, then choose OK. Effect, shown in Figure 11-26, lists the types of effects that you can apply. Animation is selected by default.

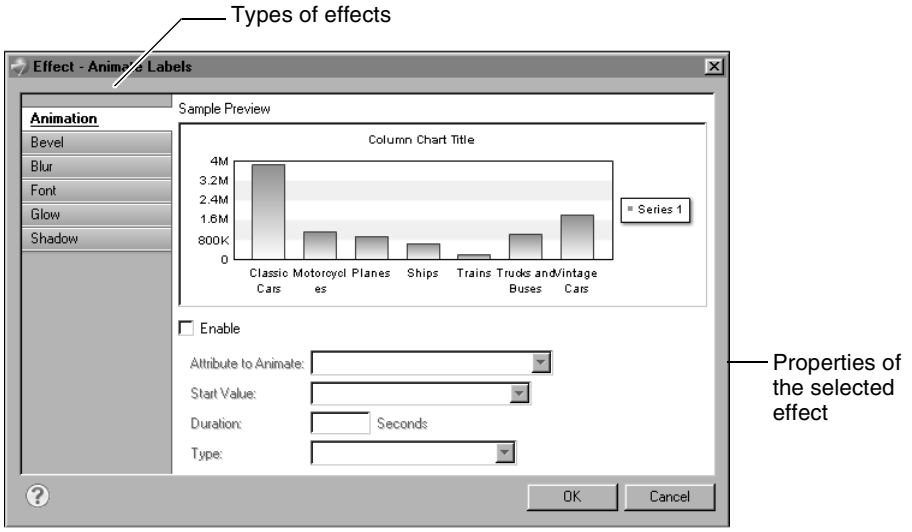


Figure 11-26 Types of effects that can be set for Flash object

6 Choose an effect type, then choose Enable.

7 Set the properties of the selected effect type. The properties of each effect type are described later in this chapter.

8 If desired, choose another effect type. For example, you can create an effect that uses the glow and shadow effect types.

9 Choose OK when you finish creating the effect. Effects displays information about the effect you created. Figure 11-27 shows an example.

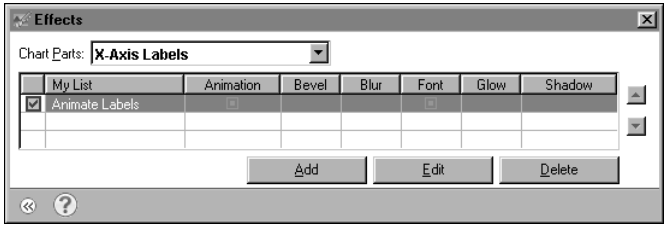


Figure 11-27 A defined effect named Animate Labels

My List shows an effect named Animate Labels. The check mark indicates that the effect applies to X-Axis Labels. The symbol under Animation and Font indicates that the Animate Labels effect uses animation and font effects.

For specific examples of creating effects, see the tutorials later in this chapter.

How to apply an effect to multiple parts in a chart or gadget

This procedure assumes that you have already created the effect.

1. Select the part of the chart or gadget to which to apply an existing effect, then choose Effects. Alternatively, choose Effects for any part that currently appears in the Format Chart or Format Gadget page. The point is to open the Effects dialog.

   Effects lists all the effects defined for the Flash chart or gadget.

2. In Effects, in Chart Parts (for a Flash chart) or Effect Target (for a Flash gadget), select the item to which to apply an effect, if necessary.

3. Under My List, select the effect to apply by clicking the checkbox next to the effect. Figure 11-28 shows an example of the Highlight effect selected for the needle in a Flash gauge.

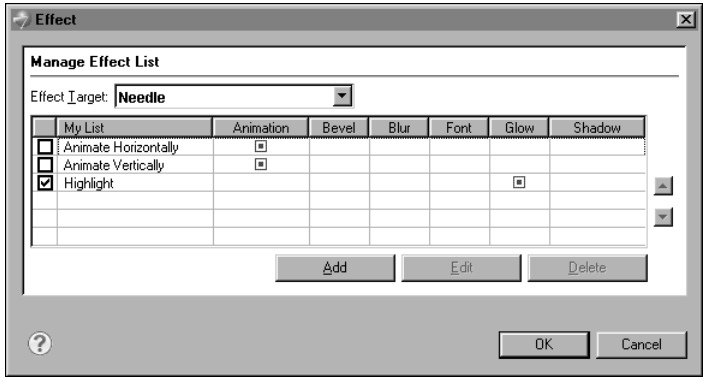


Figure 11-28 Applying an effect to a Flash object's needle

Managing effects

A single Flash chart or gadget can use any number of effects. The Effects dialog box, shown in Figure 11-28, shows all the effects created for a chart or gadget. In this dialog box, aside from creating a new effect as described previously, you can perform the following tasks:

- See which parts of the chart or gadget have effects applied. Open the drop-down list next to Chart Part or Effect Target. Items that have effects applied appear in bold.

- See which parts of the chart or gadget a particular effect applies to. Under My List, hover the mouse pointer over an effect. A tooltip displays the name of the chart or gadget part that uses the effect. For example, the following tooltip indicates that an effect applies to a chart's *y*-axis labels and title:

  ```
  Target: Y-Axis Labels, Title
  ```

- Edit an effect. Under My List, select the effect, then choose Edit.
- Delete an effect. Under My List, select the effect, then choose Delete.

Animation effect

Using the animation effect, you can animate different parts of a chart or gadget, including the background, title, data plot, data values, *x*-axis labels, *y*-axis labels, and more. After selecting the object to animate, you set properties to define how the object moves, including the direction, pattern, and duration of the animation.

For example, you can animate the data plot of a column chart so that the columns are drawn from the left side of the chart to the right side in five seconds, with a bouncing motion at the end. Figure 11-29 shows the properties set to create this type of animation. The Attribute to Animate property value of X coordinate specifies that the *x* (horizontal) position of the plot is animated. The Start Value property of Chart Start X specifies that the animation starts from the left side of the chart.

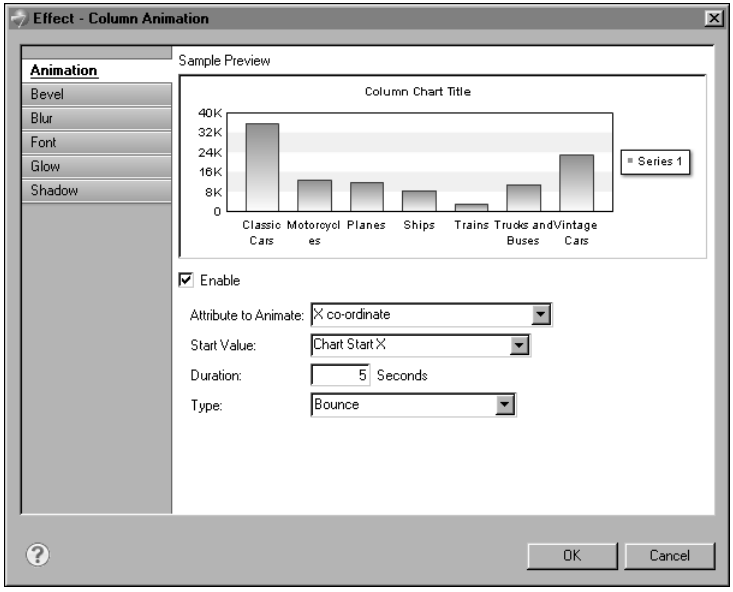


Figure 11-29 Definition of an animation effect

Table 11-16 describes the animation properties.

Table 11-16 Animation properties

Property	Description
Attribute To Animate	Specifies the property of the object to animate as described in Table 11-17. Each attribute produces different movements. Not all attributes apply to all chart objects. For example, the rotation attribute applies only to the data plot of a pie and doughnut chart.
Duration	Specifies the duration of animation in seconds.
Start Value	Specifies the start position of the animation. Either specify a fixed pixel location or select a macro. Macros are useful for setting a start *x* or *y* position at the start, center, or end position of chart. Without a macro, such as Chart Center X, you would have to experiment with many *x* values to find the center of the chart. Macros are described in Table 11-18.
Type	Specifies the type of animation as described in Table 11-19. The animation type determines acceleration and deceleration during animation. For example, a chart object might gradually increase its speed near the beginning of an animation, but slow down at the end of the animation.

Table 11-17 describes the attributes of a chart part, or object, that you can animate.

Table 11-17 Animation attributes

Attribute	Description
Horizontal Scale	Animates the *x* (horizontal) scale of the object. For example, for the data plot of a column chart, the columns are animated to grow widthwise.
Rotation	Animates pie and doughnut charts in a circular motion.
Transparency	Specifies alpha transition, or transparency fading.
Vertical Scale	Animates the *y* (vertical) scale of the object. For example, for the data plot of a column chart, the columns are animated to grow in height.
X coordinate	Animates the *x* position of the object.
Y coordinate	Animates the *y* position of the object.

Table 11-18 describes the macros that you can select for the Start Value property described in Table 11-16. Chart refers to the entire area of the Flash chart or gadget. Canvas refers to the plot area.

Table 11-18 Animation Start Value macros

Macro	Description
Chart Start X	The start *x* position of the chart, which is equal to 0
Chart Start Y	The start *y* position of the chart, which is equal to 0
Chart Width	The width of the chart
Chart Height	The height of the chart
Chart End X	The end *x* position of the chart, which is the same as the chart width
Chart End Y	The end *y* position of the chart, which is the same as the chart height
Chart Center X	The center *x* position of the chart
Chart Center Y	The center *y* position of the chart
Canvas Start X	The start *x* position of the canvas, which is the *x* coordinate of the left side of the canvas
Canvas Start Y	The start *y* position of the canvas, which is the *y* coordinate of the top of the canvas
Canvas Width	The width of the canvas
Canvas Height	The height of the canvas
Canvas End X	The canvas end *x* position, which is the *x* coordinate of the right side of the canvas
Canvas End Y	The canvas end *y* position, which is the *y* coordinate of the bottom of the canvas
Canvas Center X	The center *x* position of the canvas
Canvas Center Y	The center *y* position of the canvas

Table 11-19 describes the type of animation that you can select for the Type property described in Table 11-16.

Table 11-19 Animation types

Type	Description
Bounce	Adds a bouncing motion at the end of the animation. The number of bounces relates to the duration. Longer durations produce more bounces.
Elastic	Adds an elastic motion at the end of the animation. The range of motion is larger than that of the bounce. The amount of elasticity is unaffected by duration.
Linear	Adds a smooth movement from start to end of the animation without any changes in speed.

Table 11-19 Animation types

Type	Description
Regular	Adds slower movement at the end of the animation.
Strong	Adds an effect similar to regular, but the effect is more pronounced.

Bevel effect

Use the bevel effect to create bevels on chart and gadget objects. This effect is typically applied to a data plot, as shown in Figure 11-30. As the figure shows, the bevel makes the pie chart appear more three-dimensional. By setting properties, you can control the angle, depth, and color of the bevel.

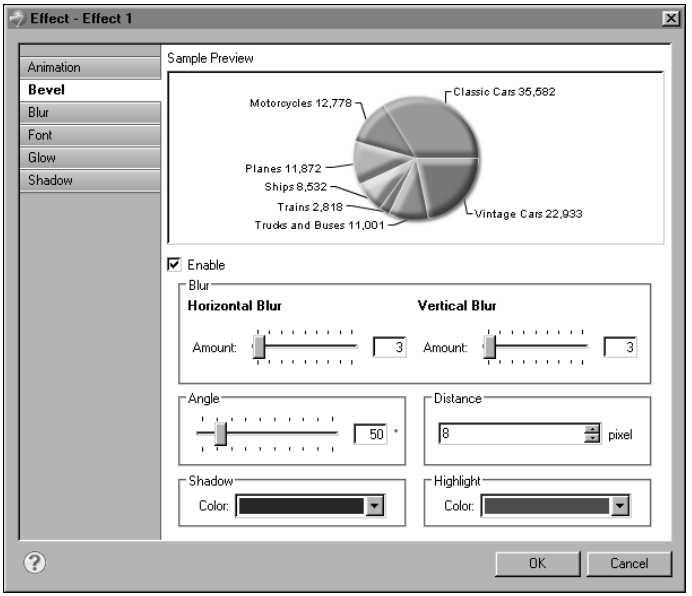


Figure 11-30 Definition of a bevel effect

Table 11-20 describes the bevel properties.

Table 11-20 Bevel properties

Property	Description
Angle	Specifies the angle of the bevel. Values are 0 to 360 degrees.
Distance	Specifies the offset distance of the bevel. Values are in pixels.
Highlight	Specifies the color of the highlight portion of the bevel.

(continues)

Table 11-20 Bevel properties (continued)

Property	Description
Horizontal Blur	Specifies the amount of horizontal blur in pixels.
Shadow	Specifies the color of the shadow portion of the bevel.
Vertical Blur	Specifies the amount of vertical blur in pixels.

Blur effect

Use the blur effect to blur a chart or gadget object. Figure 11-31 shows this effect applied to the gauge of a linear gauge gadget.

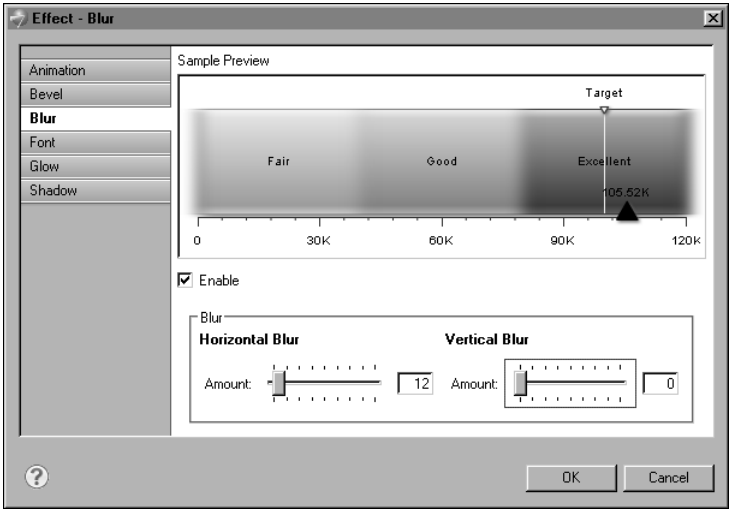


Figure 11-31 Definition of a blur effect

Table 11-21 describes the blur properties.

Table 11-21 Blur properties

Property	Description
Horizontal Blur	Specifies the amount of horizontal blur in pixels
Vertical Blur	Specifies the amount of vertical blur in pixels

Font effect

By default, all text in a Flash chart and gadget appear in the same font. Use the font effect to apply different fonts to different text objects. For example, you can use one font for the axes labels and another for the legend labels.

Table 11-22 describes the font effect properties.

Table 11-22 Font effect properties

Property	Description
Background color	Sets the background color for the text box
Bold	Sets text to bold
Border color	Creates a border around the text of specified color
Color	Sets the font color
Font	Sets the font family for the text
Italic	Sets text to italic
Size	Specifies the font size
Underline	Sets text to underline

Glow effect

Use the glow effect to add a glowing outline around a chart or gadget object. Figure 11-32 shows this effect applied to the data plot of a pie chart.

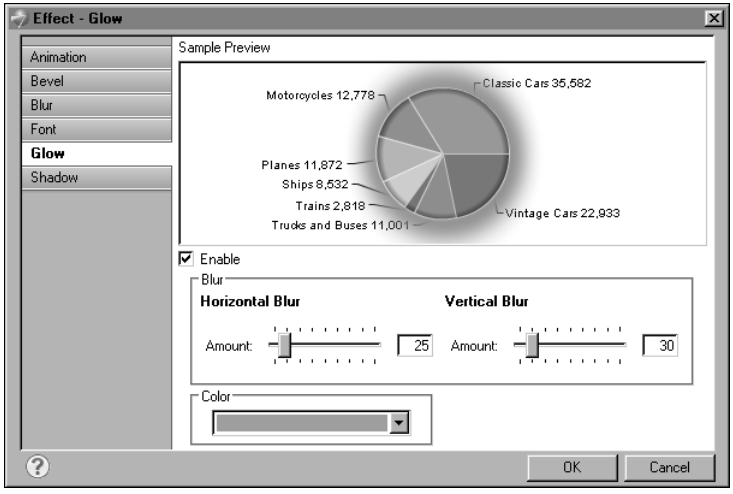


Figure 11-32 Definition of a glow effect

Table 11-23 describes the glow effect properties.

Table 11-23 Glow properties

Property	Description
Color	Specifies the color of the glow
Horizontal Blur	Specifies the amount of horizontal blur in pixels
Vertical Blur	Specifies the amount of vertical blur in pixels

Shadow effect

Use the shadow effect to add a shadow to a chart or gadget object. Figure 11-33 shows this effect applied to the data plot of a pie chart.

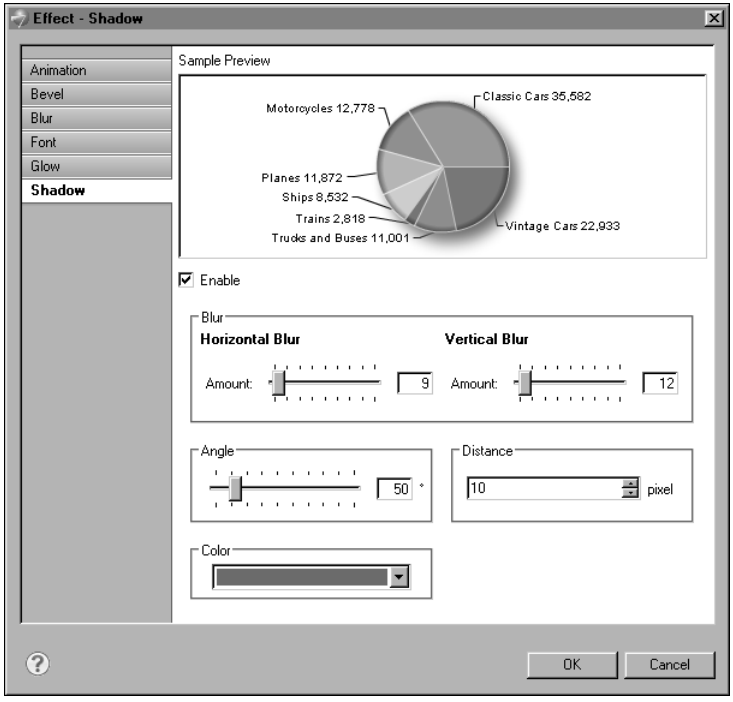


Figure 11-33 Definition of a shadow effect

Table 11-24 describes the shadow effect properties.

Table 11-24 Shadow properties

Property	Description
Angle	Specifies the angle of the shadow. Valid values are from 0 to 360 degrees.
Color	Specifies the color of the shadow.
Distance	Specifies the offset distance for the shadow in pixels.
Horizontal Blur	Specifies the amount of horizontal blur in pixels.
Vertical Blur	Specifies the amount of vertical blur in pixels.

Tutorial 1: Creating a Flash chart

This tutorial provides step-by-step instructions for building a report that uses an animated Flash column chart to display sales by product line. You perform the following tasks:

- Create a new report.
- Build a data source.
- Build a data set.
- Add a Flash chart to the report.
- Select data for the Flash chart.
- Animate the *x*-axis labels.
- Animate the *y*-axis labels.
- Change the animation effect of the columns.

Task 1: Create a new report

This task assumes you have already created a project for your reports.

1. Choose File➛New➛Report.
2. On New Report, type the following text as the file name:

   `ProductLineSales.rptdesign`
3. Choose Finish. A blank report appears in the layout editor.

Task 2: Build a data source

In this procedure, create a data source to connect to the Classic Models sample database.

1. Choose Data Explorer.
2. Right-click Data Sources, and choose New Data Source from the context menu.
3. Select Classic Models Inc. Sample Database from the list of data sources. Use the default data source name, then choose Next. Connection information about the new data source appears.
4. Choose Finish. The new data source appears under Data Sources in Data Explorer.

Task 3: Build a data set

In this procedure, build a data set to indicate what data to retrieve from the OrderDetails and Products table.

1 In Data Explorer, right-click Data Sets, and choose New Data Set.

2 In New Data Set, in Data Set Name, type the following text:

```
SalesTotals
```

Use the default values for the other fields.

- Data Source Selection shows the name of the data source that you created earlier.
- Data Set Type specifies that the data set uses a SQL SELECT query to retrieve the data.

3 Choose Next.

4 In New Data Set —Query, type the following SQL SELECT statement to indicate what data to retrieve:

```
select Products.ProductLine,
sum(OrderDetails.QuantityOrdered * OrderDetails.PriceEach) as
   TotalPrice
from OrderDetails, Products
where products.productcode = orderdetails.productcode
group by products.productline
order by products.productline
```

This statement calculates the total sales amount for each product line.

5 Choose Finish to save the data set. Edit Data Set displays the columns specified in the query, and provides options for editing the data set.

6 Choose Preview Results. Figure 11-34 shows the data rows that the data set returns.

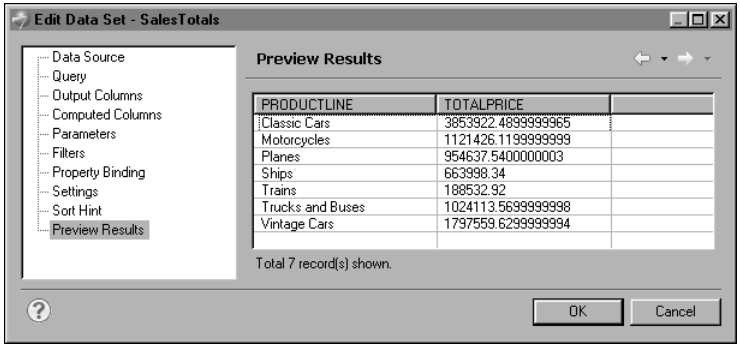


Figure 11-34 SalesTotals data set preview

7 Choose OK.

Task 4: Add a Flash chart to the report

Use the palette to insert a Flash chart, then select a chart type.

1 Choose Palette, then drag the Flash Chart element from the palette to the blank report design. The Flash chart builder opens and displays the Select Chart Type page, as shown in Figure 11-35.

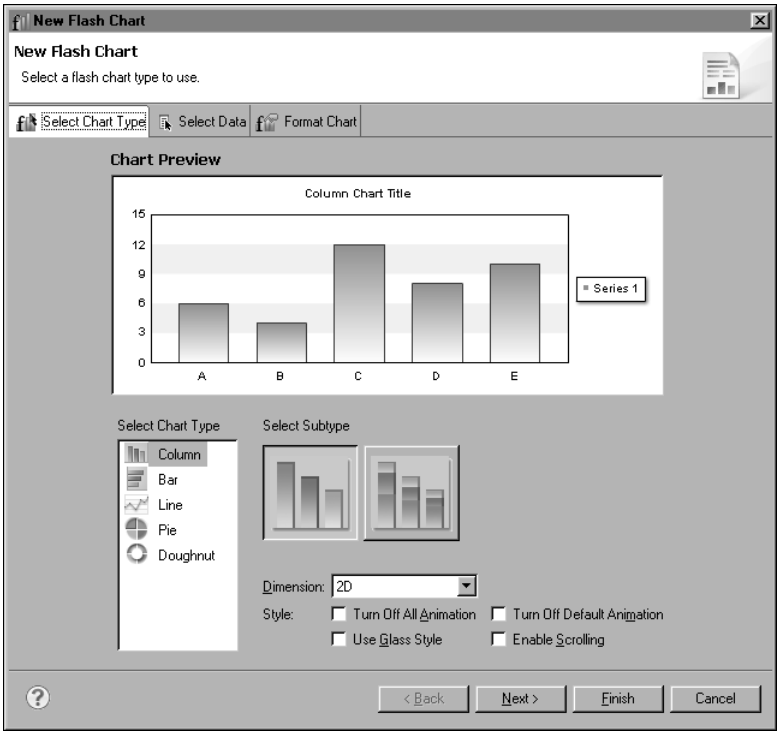


Figure 11-35 Flash chart builder displaying the Select Chart Type page

2 Create a column chart, using the default values for all the properties.

Task 5: Select data for the Flash chart

In this procedure, select the data to present in the chart.

1 In the Flash chart builder, choose Next to display the Select Data page.

On this page, under Select Data, Use Data From is selected by default and its value is set to SalesTotals, the data set you created earlier. Data Preview shows the columns and values returned by the data set.

2 In Data Preview, select the PRODUCTLINE column header and drag it to the empty field to the right of Category (X) Series, as shown in Figure 11-36.

3 Select the TOTALPRICE column header and drag it to the empty field in Value (Y) Series, as shown in Figure 11-36.

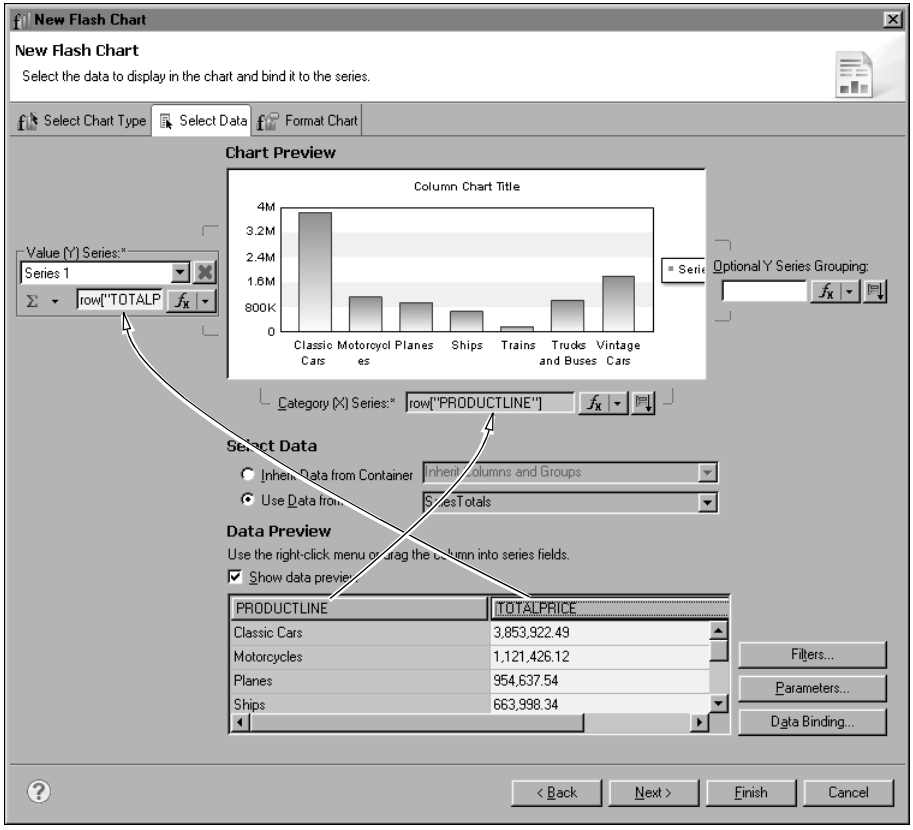


Figure 11-36 Specifying the data to use for the category and value series

The image in Chart Preview changes to use the specified data.

4 Before formatting the Flash chart, preview the chart.

1 Choose Finish to close the Flash chart builder.

2 In the layout editor, enlarge the Flash chart. Increase its width to seven inches, and increase its height to three inches.

3 Choose Preview.

The Flash chart is animated. Columns are drawn linearly from the bottom to the top. Figure 11-37 shows the generated chart.

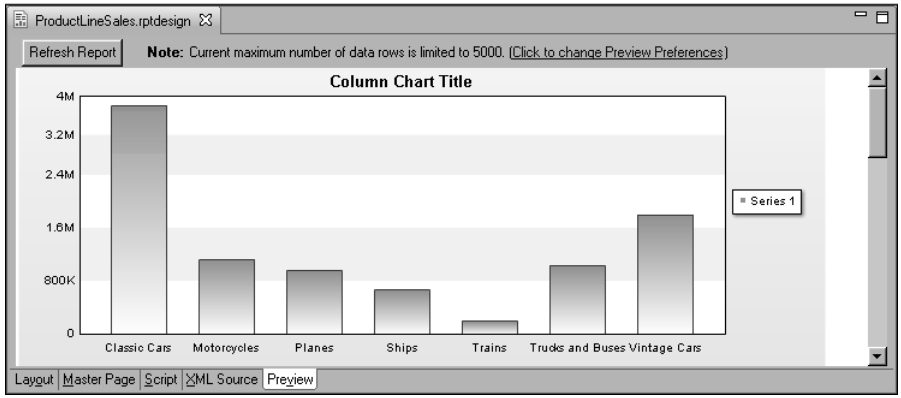


Figure 11-37 Preview of the Flash chart

Task 6: Animate the *x*-axis labels

In this procedure, animate *x*-axis labels to draw them linearly from left to right.

1. Choose Layout to resume editing the report.
2. Double-click the Flash chart to open the Flash chart builder.
3. Choose Format Chart.
4. Choose X-Axis in Chart Area, shown in Figure 11-38. Then, choose Effects.

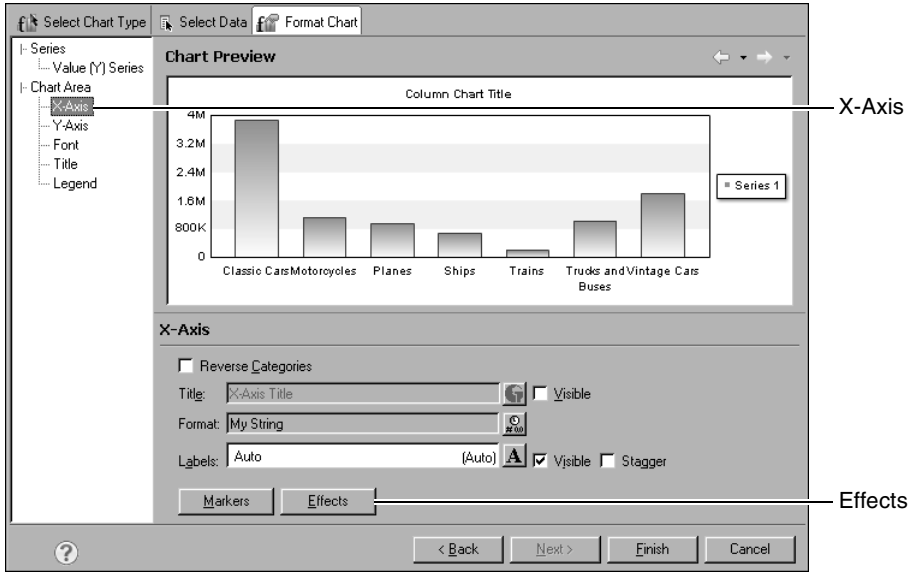


Figure 11-38 Select X-Axis and Effects to apply animation to *x*-axis labels

In Effects, Chart Parts displays X-Axis Labels, indicating that is the part of the chart selected for an effect.

5 Choose Add to create an effect.

6 In Add New Effect, type the following text as the name of the effect, then choose OK:

```
Animate Horizontally
```

In Effect—Animate Horizontally, Animation is selected by default.

7 Specify the following animation values, as shown in Figure 11-39:

- Select Enable.
- In Attribute to Animate, select X co-ordinate.
- In Start Value, select Canvas Start X.
- In Duration, type 3.
- In Type, select Linear.

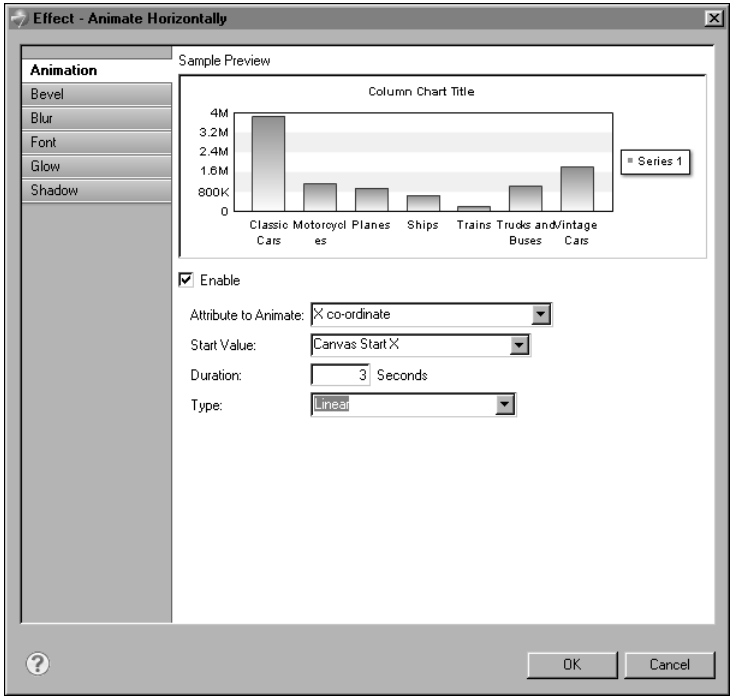


Figure 11-39 Animation values specified for the Animate Horizontally effect

8 Choose OK to save the effect.

In Effects, shown in Figure 11-40, My List shows Animate Horizontally, the effect you just created. The check mark indicates that the effect applies to X-Axis Labels. A symbol under Animation indicates that the effect uses animation.

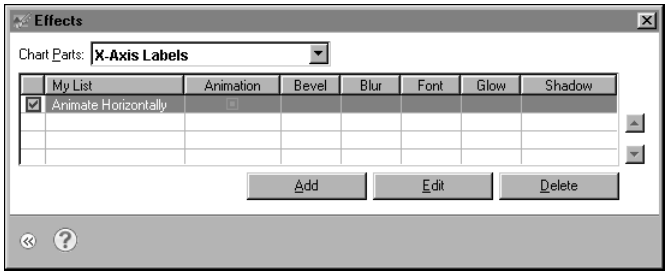


Figure 11-40 Effects lists the Animate Horizontally effect

Task 7: Animate the *y*-axis labels

In this procedure, animate the *y*-axis labels to display them from top to bottom with a bouncing movement.

1. In Effects, in Chart Parts, select Y-Axis Labels.
2. Choose Add to create a new effect.
3. In Add New Effect, type the following text as the name of the effect, then choose OK:

   ```
   Animate Vertically
   ```

4. Specify the following animation values:
   - Select Enable.
   - In Attribute to Animate, select Y co-ordinate.
   - In Start Value, select Chart Start Y.
   - In Duration, type 3.
   - In Type, select Bounce.
5. Choose OK to save the effect.

Task 8: Change the animation effect of the columns

As you saw earlier, the default animation for a column chart is the drawing of columns vertically from bottom to top. In this procedure, animate the columns to grow in both height and width.

1. In Effects, in Chart Parts, select Data Plot.

2 Choose Add to create a new effect.

3 In Add New Effect, type the following text as the name of the effect, then choose OK:

```
Animate Columns
```

4 Specify the following animation values:

- Select Enable.
- In Attribute to Animate, select Horizontal Scale
- In Start Value, select Canvas Start X.
- In Duration, type 3.
- In Type, select Linear.

5 Choose OK to save the effect.

In Effects, the Animate Columns effect is added to the list, as shown in Figure 11-41.

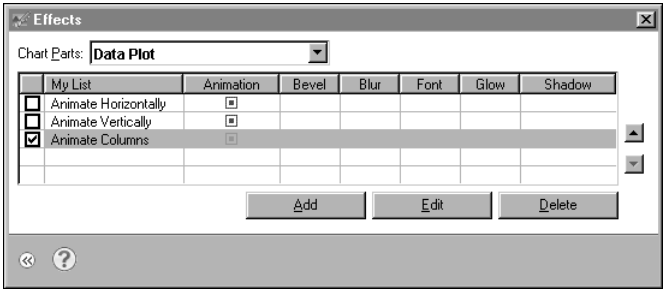


Figure 11-41 Effects listing the three effects created

6 Close Effects, then choose Finish to close the Flash chart builder.

7 View the report in the web viewer.

Tutorial 2: Creating a Flash gadget

This tutorial provides step-by-step instructions for creating an animated Flash gadget to display a sales grand total. This tutorial continues with the report built in the previous tutorial, which used a Flash chart to display sales totals by product line. The Flash gadget in this tutorial uses the data from the data set created in the previous tutorial. If you want to skip creating the Flash chart in the previous tutorial, set up the data required by the Flash gadget by running through the following tasks from the previous tutorial:

- Task 1: Create a new report

- Task 2: Build a data source
- Task 3: Build a data set

This tutorial covers the following tasks:

- Add a Flash gadget to the report.
- Select data for the linear gauge.
- Divide the data area into regions.
- Add thresholds.
- Animate the region labels.

Task 1: Add a Flash gadget to the report

1 Choose Palette, then drag the Flash Gadget element from the palette and drop it in the report, above the Flash chart. The Flash gadget builder opens and displays the Select Gadget Type page, as shown in Figure 11-42.

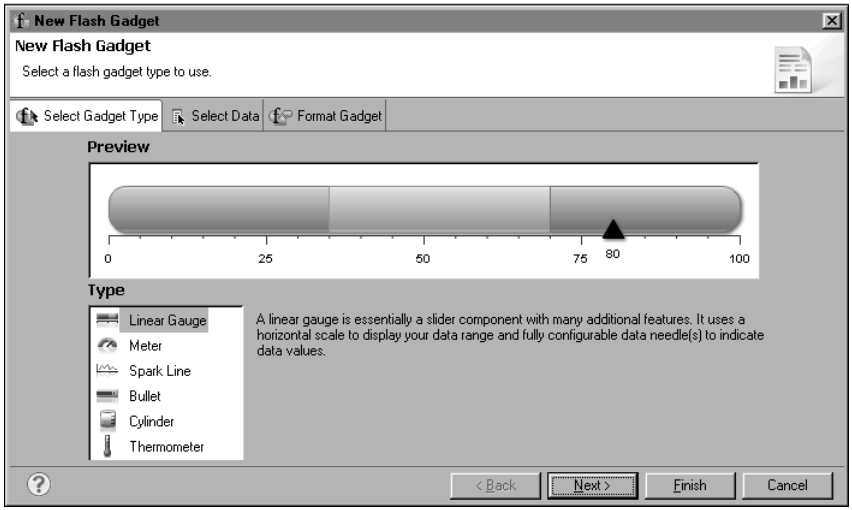


Figure 11-42 Flash gadget builder displaying the Select Gadget Type page

2 Create a linear gauge, the gadget selected by default.

Task 2: Select data for the linear gauge

In this procedure, specify the data to present in the gauge.

1 In the Flash gadget builder, choose Next to display the Select Data page.

On this page, under Select Data, Use Data From is selected by default and its value is set to SalesTotals, the data set you created earlier. Data Preview shows the columns and values returned by the data set.

2 In Data Preview, select the TOTALPRICE column header and drag it to the empty field in Value Definition, as shown in Figure 11-43.

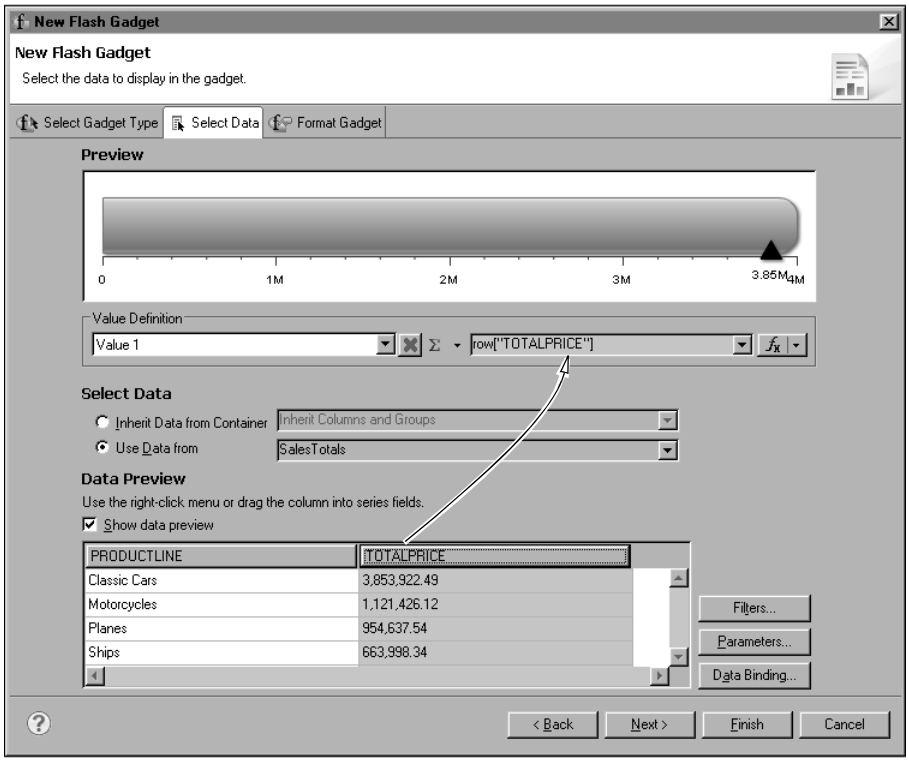


Figure 11-43 Specifying the data to use for the gauge value

The gauge in Preview changes to use the specified data. The needle shows a value of 3.85M, which is the total for Classic Cars, the value in the first row returned by the data set.

3 Specify that the gauge display the sum of sales across all product lines.

1. Click the down arrow button next to the sigma (Σ) symbol.
2. In Aggregate Expression, select Sum, then choose OK.

In Preview, the needle now shows a value of 9.6M.

4 Before formatting the gauge, preview the report.

1. Choose Finish to close the Flash gadget builder.

2 In the layout editor, resize the gauge. Increase its width to 7 inches, and decrease its height to 1.5 inches.

3 Choose Run➤View Report➤In Web Viewer. The gauge is animated. The gauge is drawn linearly from the left to the right, and the needle moves from the left edge of the gauge to its final position.

Task 3: Divide the data area into regions

In this procedure, divide the data area into three regions labeled Fair, Good, and Excellent.

1 Choose Layout to resume editing the report.

2 Double-click the gauge to open the Flash gadget builder.

3 Choose Format Gadget.

4 Choose Regions from the list of options. Figure 11-44 shows the default region properties. There are three predefined regions: A, B, and C. A is selected by default.

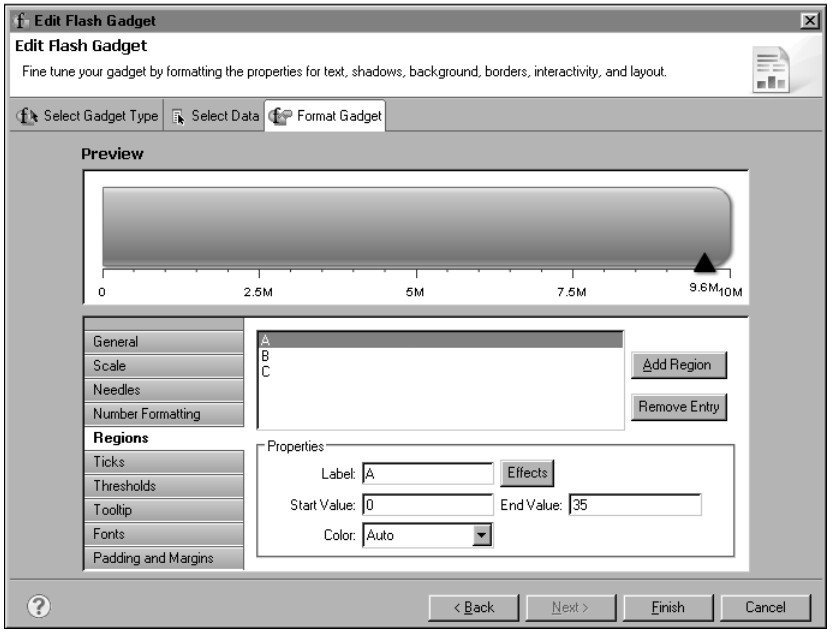


Figure 11-44 Format Gadget page displaying default region properties

5 In Properties, change the properties of region A as follows:

1 In Label, type:

```
Fair
```

2 In Start Value, use the default value 0.

3 In End Value, type 3,000,000.

4 In Color, select a red color.

6 Select B, then set its properties as follows:

1 In Label, type:

```
Good
```

2 In Start Value, type 3,000,000.

3 In End Value, type 7,500,000.

4 In Color, select a yellow color.

7 Select C, then set its properties as follows:

1 In Label, type:

```
Excellent
```

2 In Start Value, type 7,500,000.

3 In End Value, type 10,000,000.

4 In Color, select a green color.

Preview shows the data area of the gauge divided into three regions, as shown in Figure 11-45. The regions appear in the specified colors, but the region labels do not appear.

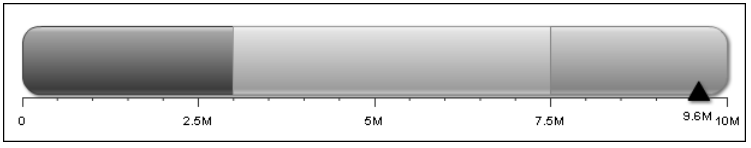


Figure 11-45 Gauge displaying three regions

8 Select Show Labels to display the region labels. The gauge displays Fair, Good, and Excellent in the corresponding regions, as shown in Figure 11-46.

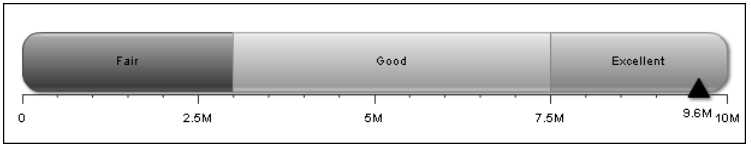


Figure 11-46 Gauge displaying region labels

Task 4: Add thresholds

In this procedure, add two threshold points to represent nominal and target sales values.

1. Choose Thresholds from the list of options.

   The page displays the properties for a predefined threshold, Threshold1.

2. Change the properties of Threshold1 as follows:

   1. In Properties—Label, type:

      ```
      Nominal
      ```

   2. In Start Value, type 2,500,000

   3. In Marker, select Show Marker.

   In Preview, the gauge displays a threshold line that displays the Nominal label above the marker.

3. Create a new threshold. In the drop-down list that displays the text Threshold1, click the down arrow button, then choose <New Threshold...>.

4. Set the properties of Threshold2 as follows:

   1. In Properties—Label, type:

      ```
      Target
      ```

   2. In Start Value, type 8,000,000.

   3. In Marker, select Show Marker.

   The gauge displays the Target threshold. Figure 11-47 shows the gauge with the two thresholds added.

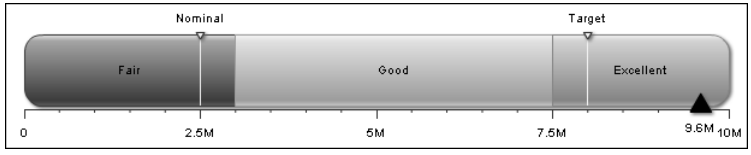


Figure 11-47 Gauge displaying thresholds

Task 5: Animate the region labels

In this procedure, animate the region labels to move from the left of the gauge to their final positions.

1. Choose Regions.

2. In Properties, Choose Effects, next to the Label property value. In Effects, Effect Target displays Gauge Labels, indicating that is the part of the gadget selected for an effect.

3. Choose Add to create an effect.

4 In Add New Effect, type the following text as the name of the effect, then choose OK:

```
Animate Horizontally
```

In Effect—Animate Horizontally, Animation is selected by default.

5 Specify the following animation values, as shown in Figure 11-48:

- Select Enable.
- In Attribute to Animate, select X co-ordinate.
- In Start Value, select Chart Start X.
- In Duration, type 3.
- In Type, select Linear.

When you finish specifying the values, Sample Preview shows the animation.

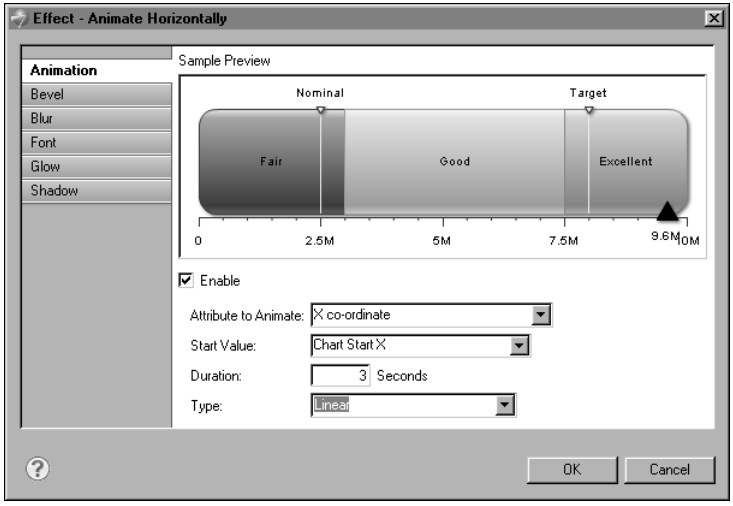


Figure 11-48 Animation values specified for the Animate Horizontally effect

6 Choose OK to save the effect.

In Effects, shown in Figure 11-49, My List shows Animate Horizontally, the effect you just created. The check mark indicates that the effect applies to Gauge Labels. A symbol under Animation indicates that the effect uses animation.

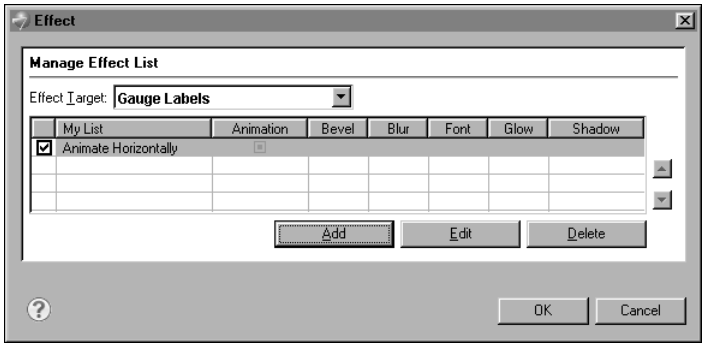


Figure 11-49 Effects lists the Animate Horizontally effect

Task 6: Animate the sales value

In this procedure, animate the sales value to move it from the top of the gauge to its final position above the needle.

1. In Effects, in Effect Target, select Value.
2. Choose Add to create a new effect.
3. In Add New Effect, type the following text as the name of the effect, then choose OK:

   ```
   Animate Vertically
   ```

4. Specify the following animation values:
   - Select Enable.
   - In Attribute to Animate, select Y co-ordinate.
   - In Start Value, select Chart Start Y.
   - In Duration, type 2.
   - In Type, select Linear.
5. Choose OK to save the effect, then choose OK to close Effect.
6. Display the sales value above the needle.
   1. In the Format Gadget page, choose General.
   2. In Properties, next to Show Needle Value, select Above Needle.

Task 7: Add a glow effect to the needle

In this procedure, highlight the needle by adding a glow effect.

1. Choose Needles from the list of options, then choose Effects.

2 In Effects, choose Add to create a new effect.

3 In Add New Effect, type the following text as the name of the effect, then choose OK:

```
Highlight
```

4 In Effect—Highlight, choose Glow.

5 Specify the following glow values:

- Select Enable.
- Use the default values for Horizontal Blur, Vertical Blur, and Color.

6 Choose OK to save the effect, then choose OK to close Effect.

7 Choose Finish to save the formatting changes to the gauge.

8 Preview the report. The gauge should look like the one in Figure 11-50.

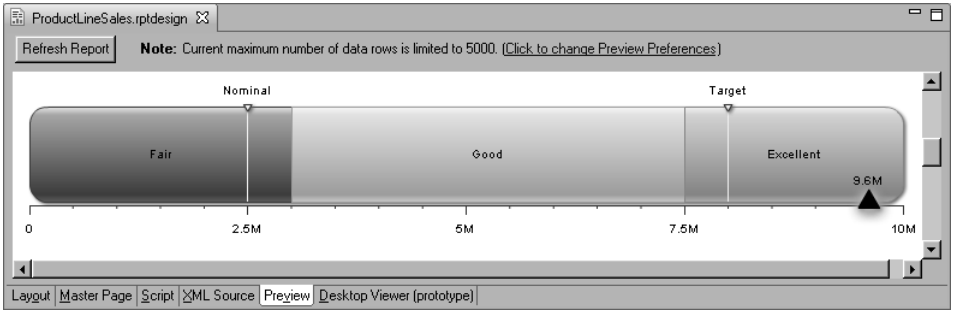


Figure 11-50 Preview of finished Flash gadget

Limitations

Due to certain aspects of using Flash objects with Actuate BIRT, there are situations where the Flash object does not work as expected. Data for the Flash object is embedded in the Flash object by default. If you create a Flash chart or gadget that contains data exceeding 64KB, you get an error, such as "A script in this movie is causing Adobe Flash Player to run slowly." This error can appear in either Actuate BIRT Designer or Actuate Interactive Viewer.

To fix this error, set the Embed Data property of the chart or gadget to false, and rebuild the report. This property setting prevents data from being embedded in the Flash object, and the object displays properly. To set the Embed Data property, in the report layout, select the Flash chart or gadget, and in Property Editor, select Advanced. Note, however, that setting a chart's Embed Data property to false displays the chart in an HTML report only. The chart does not appear in PDF.

Chapter

12

Using the Flash object library

This chapter contains the following topics:

About the Flash object library

Actuate BIRT Designer includes a library of Flash objects developed by InfoSoft. This third-party library provides hundreds of Flash objects organized in four categories—charts, power charts, widgets, and maps. If the Flash chart and Flash gadget elements, described in the previous chapter, do not provide the type of chart or gadget that you want to use in a report, look in the Flash object library.

Unlike the Flash chart and Flash gadget elements in the palette, using a Flash object from this library requires programming in JavaScript or Java. Knowledge of XML is also essential.

About Flash charts

The Flash object library provides all the basic chart types—bar, column, line, pie, and doughnut—supported by the built-in Flash chart element, as well as, an extensive gallery of advanced charts, including combination, multi-series, scroll, and XY plot charts.

Figure 12-1 shows examples of Flash charts.

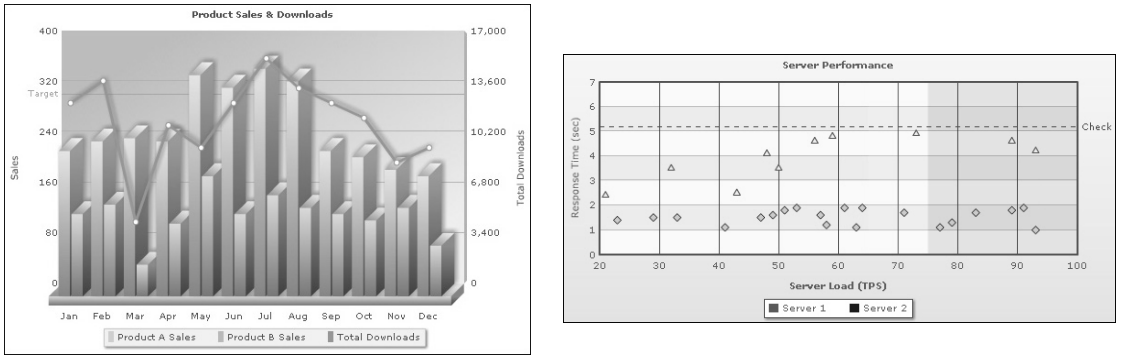
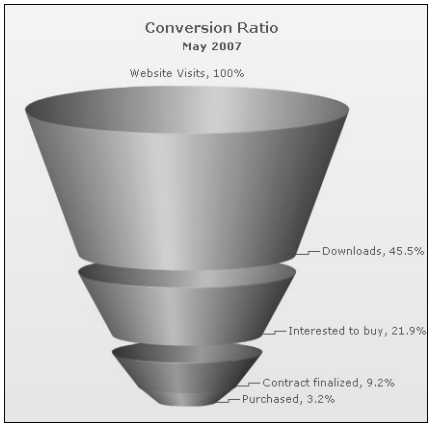


Figure 12-1 Column 3D line dual Y combination chart and scatter chart

About Flash gadgets

The Flash object library provides all the common gadgets—linear, meter, bullet, cylinder, and thermometer—supported by the built-in Flash gadget element, and many others, including funnel, pyramid, gantt, and LED gadgets. Gadgets, commonly used in dashboard applications, are suitable for displaying KPIs (Key Performance Indicators) and other critical data that are monitored in real time.

Figure 12-2 shows examples of Flash gadgets.



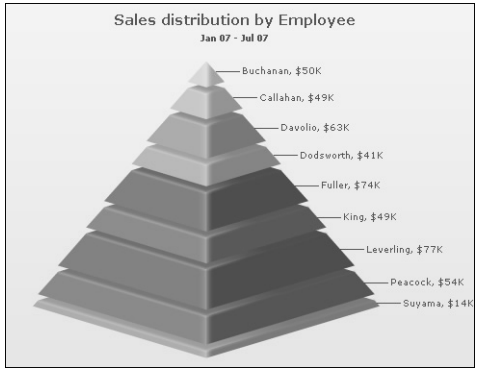
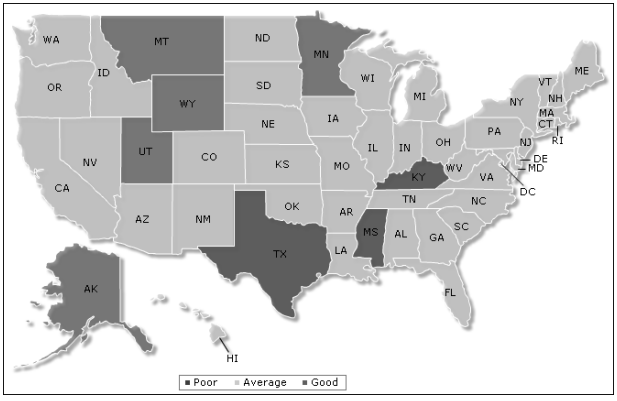


Figure 12-2 Funnel gadget and pyramid gadget

About Flash maps

Flash maps are vector maps suitable for displaying data by geographical divisions, such as population distribution, electoral results, office locations, survey results, weather patterns, and real-estate sales. The Flash object library provides hundreds of maps, including maps of the world, continents, countries, European regions, USA states, and so on.

Figure 12-3 shows examples of Flash maps.



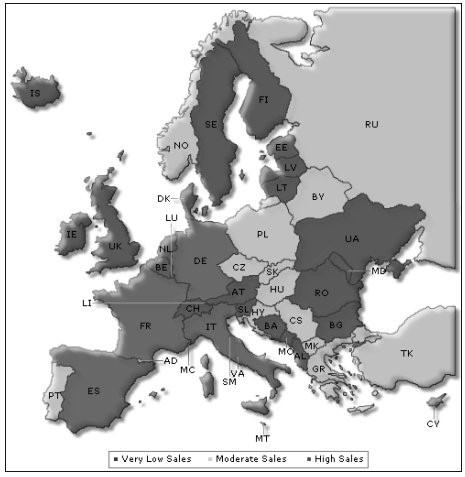


Figure 12-3 US map and Europe map

About Flash power charts

Flash power charts are specialized charting widgets that provide unique ways to present data. Charts, such as the drag-node, logarithmic, radar, kagi, and waterfall chart can be used for a wide variety of purposes, including simulations, scientific plotting, financial analysis, and hierarchical diagrams.

Figure 12-4 shows examples of power charts.

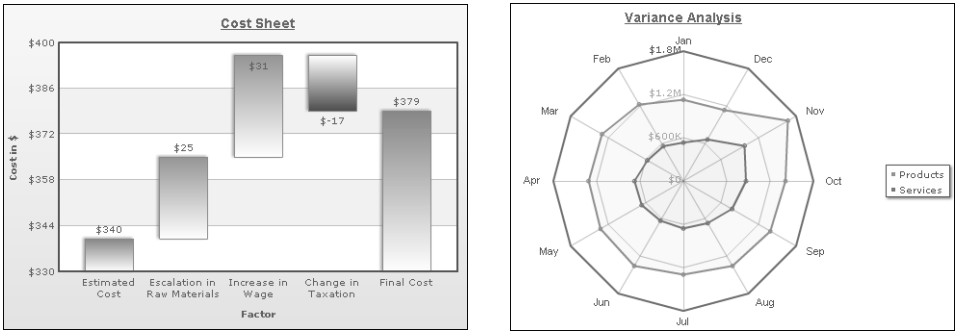


Figure 12-4 Waterfall chart and radar chart

Flash object components

Essentially, the Flash object library is a collection of Shockwave Flash (SWF) files that generate Flash charts, gadgets, or maps based on data and configuration settings provided in XML format. Each Flash object used in a report comprises the following components:

- An SWF file, which is a ready-to-use chart, gadget, or map
- XML data, which defines the data values and properties for rendering the Flash chart, gadget, or map

The Flash object library provides the SWF files. You provide the XML data in the format required by the Flash object.

Inserting a Flash object in a report

Just like the other report elements, you can insert a Flash object directly in the report page or in any of the container elements, which is the typical case. The location depends on various factors, including the position of the Flash object relative to other report elements, or whether the Flash object shares data in a data set that is used by other elements. For information about laying out a report, see *BIRT: A Field Guide*.

How to insert a Flash object

1 Drag the Flash Object element from the palette and drop it in the report layout.

2 In Flash Builder, specify the following information:

   1 In Select content from, select Flash Object Library, as shown in Figure 12-5.

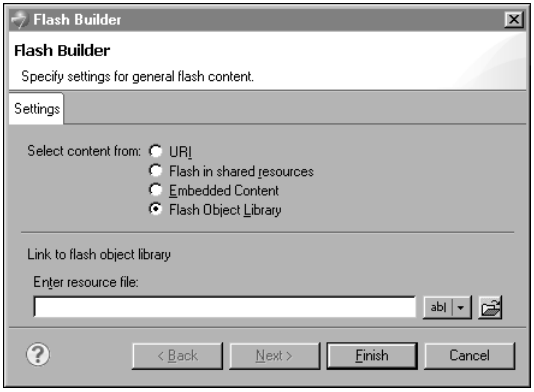


Figure 12-5 Selecting Flash Object Library

   2 In Enter resource file, choose the open folder button to select a Flash file from the library.

   Browse for Flash Files displays the top-level contents of the Flash Object Library, as shown in Figure 12-6.

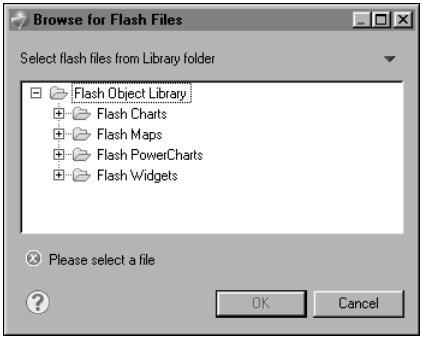


Figure 12-6 Top-level contents of the Flash Object Library

   3 Expand the folder that contains the type of Flash object you want, then select the SWF file for the object. Figure 12-7 shows some of the SWF files in the Flash Charts folder. The names of the SWF files reflect the chart types. For example, to insert a bubble chart in the report, select Bubble.swf.

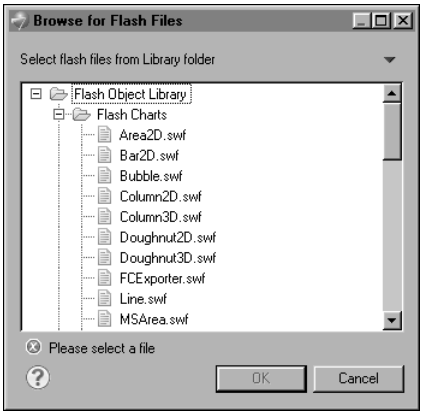


Figure 12-7 Available flash charts

4 Choose OK.

3 Choose Finish. The Flash object appears in the report layout.

Providing data to a Flash object

All Flash objects are controlled by XML properties. You must use XML to provide data to a Flash object and to define the visual and functional properties of a Flash object. Unlike the Flash charts and gadgets that you create using the Flash chart and Flash gadget elements, an object in the Flash library cannot access data directly from a data set. After creating a data set to retrieve data from a data source, you write code that accesses the data and converts it to the required XML format.

Before you can write this code, you need to know what XML is required for a given Flash object. The XML differs depending on the type of Flash object. The following example shows a basic single-series chart and the XML that defines its data and properties. Figure 12-8 shows a doughnut chart that displays a company's revenue by business division.

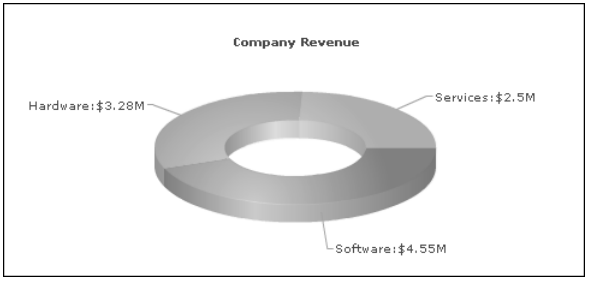


Figure 12-8 Doughnut chart displaying revenue by business division

Listing 12-1 shows the XML that defines the data and properties of the doughnut chart.

Listing 12-1 Sample XML for a doughnut chart

```
<chart caption='Company Revenue' numberPrefix='$'
   labelSepChar=':'>
   <set label='Services' value='2500000'/>
   <set label='Hardware' value='3280000'/>
   <set label='Software' value='4550000'/>
</chart>
```

In a BIRT report, the values highlighted in bold are data values that are derived from a data set. Other XML attributes and values define the appearance of the chart. For example:

- caption='Company Revenue' sets the title of the chart.
- numberPrefix='$' adds the dollar symbol as a prefix to all numbers on the chart.
- labelSepChar=':' specifies that the colon character be used to separate the data label and data values on the chart.

Even if you are not well-versed in XML, you quickly learn that chart data and formatting information are defined using the attribute='value' format. Notice that the sample XML is brief for a chart that looks presentable. Only three visual attributes are specified. The XML does not define attributes for fonts or colors. Every Flash object uses default values for visual attributes. You define attributes only to change default settings, or to add items that do not appear by default.

The next example shows a multi-series chart and the XML that defines its data and properties. Figure 12-9 shows a multi-series column chart that displays expenses and revenue from 2005 to 2009.

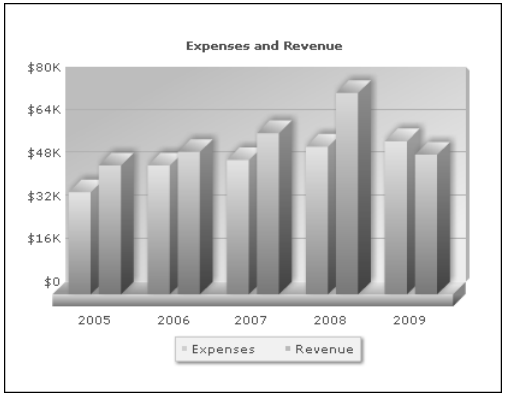


Figure 12-9 Multi-series column chart displaying expenses and revenue for five years

Listing 12-2 shows the XML that defines the data and properties of the multi-series chart.

Listing 12-2 Sample XML for a multi-series column chart

```
<chart caption='Expenses and Revenue' showValues='0' decimals='0'
   numberPrefix='$'>
  <categories>
    <category label='2005' />
    <category label='2006' />
    <category label='2007' />
    <category label='2008' />
    <category label='2009' />
  </categories>

  <dataset seriesName='Expenses'>
    <set value='38000' />
    <set value='48000' />
    <set value='50000' />
    <set value='55000' />
    <set value='57000' />
  </dataset>

  <dataset seriesName='Revenue'>
    <set value='48000' />
    <set value='53000' />
    <set value='60000' />
    <set value='75000' />
    <set value='52000' />
  </dataset>
</chart>
```

The values highlighted in bold are data values that are provided by a data set. Compared to the single-series doughnut chart, the XML for defining the data for a multi-series column chart is slightly more complex. The data for the multi-series chart is divided into three sections, whereas the data for the single-series doughnut chart is contained in one section.

Once you determine the XML needed to define the data and properties for a specific Flash object, perform the following tasks:

1. Write code to generate the XML.

   Use either JavaScript or Java, depending on the method you use to pass the XML to the Flash object, described next.

2. Pass the XML to the Flash object.

   There are two ways to accomplish this task. Use either the dataXML variable or the dataURL variable to pass the XML to the Flash object.

Generating the XML data

Defining the visual attributes is straightforward. Simply use the attribute='value' format for each attribute; for example, color='8BBA00'. Defining the data, however, requires programming to iterate through a data set and retrieve values from each row of data.

The following examples include code snippets, which show how to accomplish this task using JavaScript. For complete programming examples using JavaScript and Java, review the tutorials later in this chapter.

Doughnut chart example

In the doughnut chart example shown in Figure 12-8, data is defined using the <set> tag and the following format:

```
<set label='Services' value='2500000'/>
```

The code you write must get values from two columns, one that stores business division values, and another that stores revenue values. Assume that the data set providing the data values returns rows as shown in Figure 12-10.

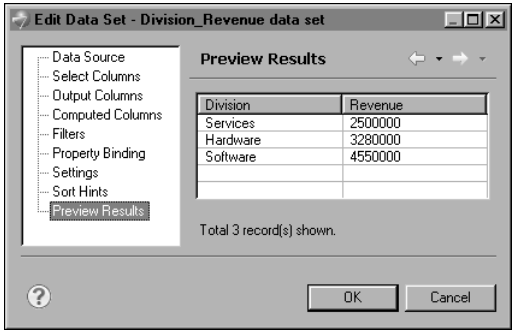


Figure 12-10 Data rows returned by data set in doughnut chart example

The JavaScript code for getting the division and revenue values from a data set row and creating a single <set> line would look like the following:

```
var setData = "<set ";
setData = setData + "label='" +
   this.getRowData().getColumnValue("Division") + "'";
setData = setData + "value='" +
   this.getRowData().getColumnValue("Revenue") + "'/>";
```

To generate all the <set> lines, add code, such as the following:

```
xmlData = xmlData + setData;
```

Multi-series chart example

In the multi-series chart example shown in Figure 12-9, category data is defined within the <categories> tag using the <category> tag in the following format:

```
<category label='2005' />
```

Value series data is defined within the <dataset> tag using the <set> tag in the following format:

```
<set value='38000' />
```

The code you write gets values from three columns in each data set row. Assume that the data set providing the data values returns rows as shown in Figure 12-11.

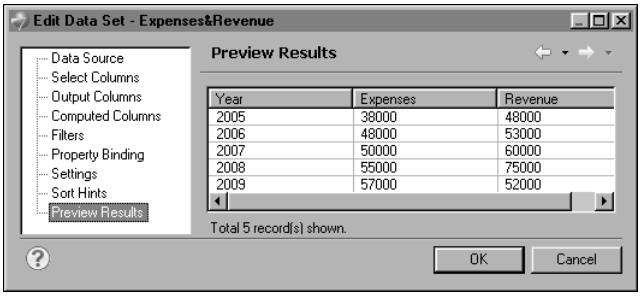


Figure 12-11 Data rows returned by data set in multi-series chart example

The JavaScript code for getting the column values from each data set row would look like the following:

```
var YearData = "<category label='" +
   this.getRowData().getColumnValue("Year")+ "'/>";

var ExpenseData = "<set value='" +
   this.getRowData().getColumnValue("Expenses") + "'/>";

var RevenueData = "<set value='" +
   this.getRowData().getColumnValue("Revenue") + "'/>";
```

To generate all the <category> and <set> lines, add code, such as the following:

```
dataPart1 = dataPart1 + YearData;
dataPart2 = dataPart2 + ExpenseData;
dataPart3 = dataPart3 + RevenueData;
```

Using the dataXML variable to pass XML data

Use the dataXML variable if the XML to pass to the Flash object is less than 64KB, a limit imposed by the Flash Player. Passing the data through the dataXML variable embeds the data in the Flash object. If the XML exceeds 64KB, the Flash Player displays an error. When using the dataXML variable, you write JavaScript code, as shown in the code examples in the previous section, to generate the XML.

Writing this code requires a basic understanding of BIRT events and their order of execution, as well as, BIRT elements and the functions for manipulating data.

How to use the dataXML variable to pass data to the Flash object

This procedure assumes you have already generated the data in XML format, as described in the previous section.

1. In the report layout, select the Flash object.
2. In Property Editor, choose the Flash Variables tab.
3. In the Flash Variables page, choose Add.
4. In Add Variables, do the following:
   1. In Name, type:

      ```
      dataXML
      ```

   2. In Expression, choose the JavaScript expression builder.
5. In the JavaScript expression builder, type an expression that passes the complete XML to the Flash object. The following expression passes the XML for creating the doughnut chart shown earlier in Figure 12-8:

   ```
   //Get the data generated and saved in the g_dataPart global var
   var g_dataPart =
      reportContext.getPersistentGlobalVariable("g_dataPart");

   //Build the complete XML
   "<chart caption='Company Revenue' showPercentageValues='1'>" +
      g_dataPart + "</chart>"
   ```

6. Choose OK to save the expression.
7. Choose OK to save the dataXML variable.

Using the dataURL variable to pass XML data

Use the dataURL variable if the XML to pass to the Flash object exceeds 64KB, which can occur for more complex objects, such as multi-series combination charts that require many rows of data or the definition of a large number of attributes.

When you use the dataURL variable, the XML data is stored in a separate file rather than embedded in the Flash object. To use this method, you write a Java class to generate the XML file, then pass the URL of the program to the Flash object.

Writing a Java class requires experience with the Eclipse Java development process. The Java class must be implemented as a plug-in, which is a modular component that provides a specific type of service within the Eclipse platform. In

fact, all the tools in Eclipse and Actuate BIRT Designer are plug-ins. For information about developing plug-ins, see the Eclipse documentation.

How to use the dataURL variable to pass data to the Flash object

This procedure assumes you have already implemented a plug-in that generates the XML file to pass to the Flash object.

1. In the report layout, select the Flash object.
2. In Property Editor, choose the Flash Variables tab.
3. In the Flash Variable page, choose Add.
4. In Add Variables, do the following:
   1. In Name, type:

      ```
      dataURL
      ```

   2. In Expression, choose the JavaScript expression builder.
5. In the JavaScript expression builder, type an expression that passes the URL of of the plug-in to the Flash object. Use the createDataURL( ) method of the flashContext object, as shown in the following example:

   ```
   flashContext.createDataURL("ComboChartXMLFormat", true, null);
   ```

   The first argument, ComboChartXMLFormat, is the format defined in the plug-in. The second argument, true, specifies that the URL is encoded. The third argument, null, specifies that there are no custom parameter names and values to pass to the URL.
6. Choose OK to save the expression.
7. Choose OK to save the dataURL variable.

Using the Flash object library documentation

This chapter describes the procedures for inserting a Flash object from the library in a report and passing data to the object. This chapter does not provide documentation about every Flash object, the structure of each object, or the XML elements and attributes that you use to create an object. This information, essential for generating the required XML, is available in the InfoSoft documentation, which is included in Actuate BIRT Designer's online help.

To access the InfoSoft documentation, in the main menu, choose Help➤Help Contents. In Help, expand Actuate BIRT Guide. The InfoSoft documentation is titled Flash Object Library Reference.

This reference is organized by Flash object type, as shown in Figure 12-12.

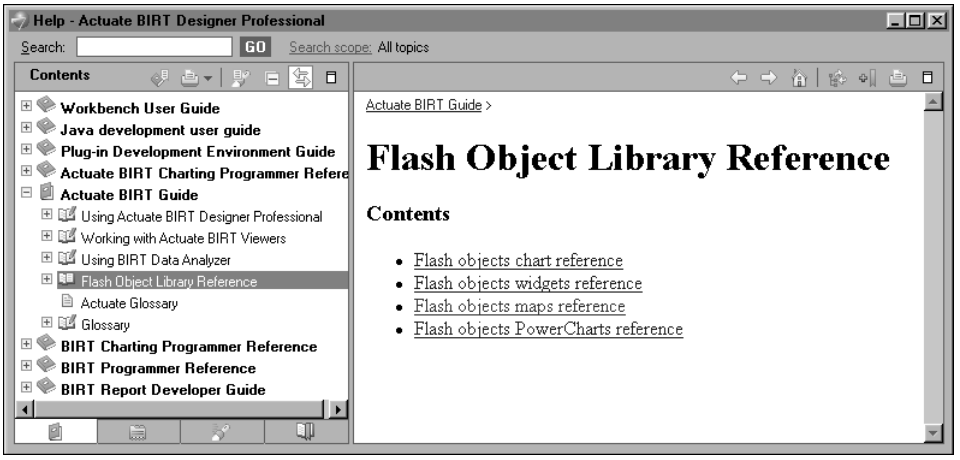


Figure 12-12 Contents of the Flash Object Library Reference

To find documentation about a particular Flash object, choose the corresponding reference, then drill down until you find the specification for the specific Flash object. The specification provides a complete reference to the object, including descriptions of all the parts of the object, and all the properties that can be set to manipulate and format the object. For example, let's say you want to see reference information about the 3D pie chart. First, choose Flash objects chart reference. In the InfoSoft documentation for charts, navigate through the following topic structure: Chart XML API—Single Series Charts—Pie 3D Chart.

If you prefer to learn by example, look at the sample Flash objects and view the XML for creating those objects. To see a sample of a 3D pie chart, for example, navigate through the following topic structure: Sample Charts—Single Series Charts—Pie 3D Chart.

Tutorial 3: Creating a Flash map that gets data through the dataXML variable

This tutorial provides step-by-step instructions for building a report that uses a Flash map from the Flash object library to display sales by territory. The map uses data from the Classic Models sample database, which you convert to XML and pass to the map through the dataXML variable.

You perform the following tasks in this tutorial:

- Create a new report.
- Build a data source.
- Build a data set.

- Find a suitable Flash map.
- Review the map specifications.
- Map the data set values to the Flash map entity values.
- Add the Flash map to the report.
- Generate an XML data string.
- Create the dataXML variable and pass the data.
- Format the Flash map.

Task 1: Create a new report

This task assumes you have already created a project for your reports.

1. Choose File➛New➛Report.
2. On New Report, type the following text as the file name:

   ```
   SalesByTerritory.rptdesign
   ```

3. Choose Finish. A blank report appears in the layout editor.

Task 2: Build a data source

In this procedure, create a data source to connect to the Classic Models sample database.

1. Choose Data Explorer.
2. Right-click Data Sources, and choose New Data Source from the context menu.
3. Select Classic Models Inc. Sample Database from the list of data sources. Use the default data source name, then choose Next. Connection information about the new data source appears.
4. Choose Finish. The new data source appears under Data Sources in Data Explorer.

Task 3: Build a data set

In this procedure, build a data set to specify what data to retrieve and combine from various tables in the database.

1. In Data Explorer, right-click Data Sets, and choose New Data Set.
2. In New Data Set, in Data Set Name, type the following text:

   ```
   Sales By Territory
   ```

   Use the default values for the other fields.

- Data Source Selection shows the name of the data source that you created earlier.
- Data Set Type specifies that the data set uses a SQL SELECT query to retrieve the data.

3 Choose Next.

4 In New Data Set —Query, type the following SQL SELECT statement to retrieve the sales total for each territory:

```
SELECT CLASSICMODELS.OFFICES.TERRITORY,
SUM(CLASSICMODELS.ORDERDETAILS.QUANTITYORDERED *
CLASSICMODELS.ORDERDETAILS.PRICEEACH) as SALES
FROM CLASSICMODELS.CUSTOMERS,
CLASSICMODELS.ORDERS,
CLASSICMODELS.ORDERDETAILS,
CLASSICMODELS.OFFICES,
CLASSICMODELS.EMPLOYEES
WHERE CLASSICMODELS.ORDERS.ORDERNUMBER =
   CLASSICMODELS.ORDERDETAILS.ORDERNUMBER
AND CLASSICMODELS.CUSTOMERS.SALESREPEMPLOYEENUMBER =
   CLASSICMODELS.EMPLOYEES.EMPLOYEENUMBER
AND CLASSICMODELS.EMPLOYEES.OFFICECODE =
   CLASSICMODELS.OFFICES.OFFICECODE
AND CLASSICMODELS.CUSTOMERS.CUSTOMERNUMBER =
   CLASSICMODELS.ORDERS.CUSTOMERNUMBER
GROUP BY CLASSICMODELS.OFFICES.TERRITORY
```

5 Choose Finish to save the data set. Edit Data Set displays the columns specified in the query, and provides options for editing the data set.

6 Choose Preview Results. Figure 12-13 shows the data rows that the data set returns.

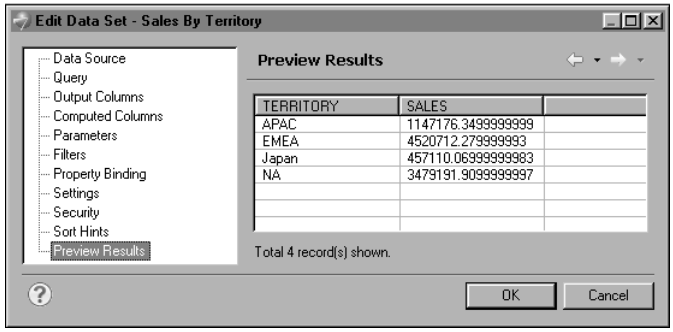


Figure 12-13 Sales By Territory data set preview

7 Choose OK.

Task 4: Find a suitable Flash map

The data set created in the previous task returns sales data for four worldwide territories: APAC (Asia Pacific), EMEA (Europe and middle east), Japan, and NA (North America). In this procedure, look in the Flash object library for a map suitable for representing data in these territories.

1. In Help, expand Actuate BIRT Guide, choose Flash Object Library Reference, then choose Flash objects maps reference.
2. Choose Map Gallery—World & Continents. This folder lists three world maps: World Map, World Map (Countries), and World Map (8 Regions). Review each map.
3. For this tutorial, either World Map or World Map (8 Regions) is suitable for the data. Use World Map. The help displays the image of World Map, as shown in Figure 12-14.

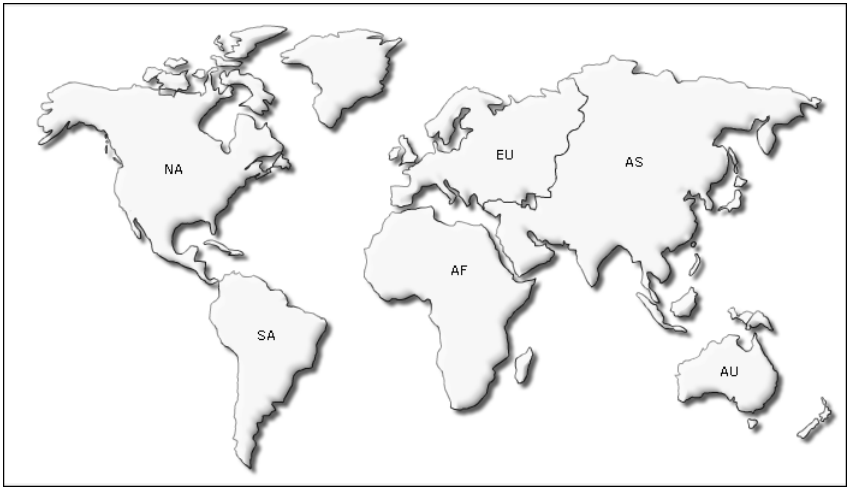


Figure 12-14 World Map available in Flash Object Library

World Map displays the names of the continents as two-letter abbreviations: NA, SA, AF, EU, AS, and AU. The Classic Models data uses these acronyms for the sales territories: APAC, EMEA, Japan, NA.

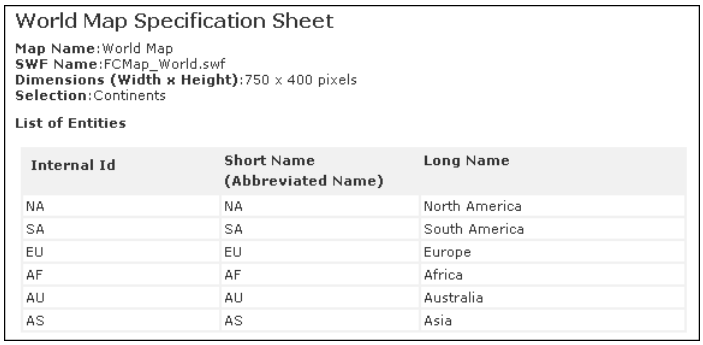
Task 5: Review the map specifications

Each map in the library displays different entities. An entity is the smallest item represented in a map. For example, in a world map that shows continents, each continent is an entity. In a continent map that shows countries, each country is an entity. Similarly, in a country map that shows states, each state is an entity. In this procedure, review the entities in World Map.

1 In the online Help for Flash maps, choose Map Specification Sheets—World & Continents—World Map.

2 The World Map Specification Sheet, shown in Figure 12-15, displays information about the map, including its list of entities. Each entity has the following properties:

- Internal Id—The ID through which an entity is referred to in the XML data document
- Short Name—The abbreviated entity name, which appears on the map
- Long Name—The full entity name, which appears as a tool tip



World Map Specification Sheet

Map Name:World Map
SWF Name:FCMap_World.swf
Dimensions (Width x Height):750 x 400 pixels
Selection:Continents

List of Entities

Internal Id	Short Name (Abbreviated Name)	Long Name
NA	NA	North America
SA	SA	South America
EU	EU	Europe
AF	AF	Africa
AU	AU	Australia
AS	AS	Asia

Figure 12-15 World Map Specification Sheet available in online help

Task 6: Map the data set values to the Flash map entity values

To display data from the data set in the Flash map, you need to map the territory values in the data set to the internal ID values used by World Map.

1 In Data Explorer, under Data Sets, right-click Sales By Territory, then choose Edit.

2 In Edit Data Set, choose Computed Columns, then choose New.

3 In New Computed Column, specify the following information:

   1 In Column Name, type Territory_ID.

   2 In Data Type, select String.

   3 In Expression, choose the JavaScript expression builder.

   4 In the expression builder, type the following statement, then choose OK.

   Each case statement replaces a territory value with the corresponding internal ID used by the map.

```
switch(row["TERRITORY"]) {
   case  "EMEA":
      name = "EU";
      break;
   case  "APAC":
      name = "AS";
      break;
   case  "Japan":
      name = "AS";
      break;
   case  "NA":
      name = "NA";
      break;
}
```

5 Choose OK.

4 Choose Preview Results. The data set returns the data shown in Figure 12-16. The Territory_ID values match Internal Id values in World Map.

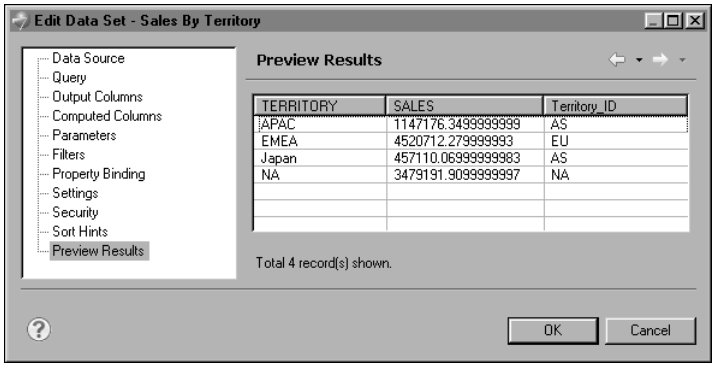


Figure 12-16 Sales By Territory data set preview includes Territory_ID values

5 Choose OK.

Task 7: Add the Flash map to the report

In this procedure, add World Map from the Flash object library to the report.

1 Insert a table that consists of one column and one detail row, and bind the table to the Sales By Territory data set.

2 Drag a Flash Object element from the palette and drop it in the table's footer row.

3 In Flash Builder, specify the following information:

   1 In Select content from, select Flash Object Library, as shown in Figure 12-17.

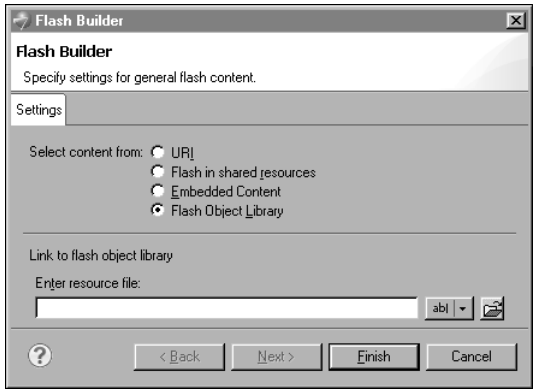


Figure 12-17 Selecting Flash Object Library

2 In Enter resource file, choose the open folder button to select a Flash file from the library.

3 In Browse for Flash Files, expand Flash Maps, and select FCMap_World.swf. Choose OK. In Flash Builder, the path to the Flash file appears in Enter resource file.

4 Choose Finish.

Task 8: Generate an XML data string

Data provided to any Flash object must be in the specific XML format that the object requires. In this procedure, look at the Flash map documentation for this information, then generate an XML data string that provides data in the required format.

1 In the online Help for Flash maps, choose How to use FusionMaps. This topic describes the procedure for displaying data in a map. It includes an example of displaying population data in the world map. The following is the sample XML:

```
<map borderColor='005879' fillColor='D7F4FF' numberSuffix='
   Mill.' includeValueInLabels='1' labelSepChar=': '
   baseFontSize='9'>
      <data>
           <entity id='NA' value='515' />
           <entity id='SA' value='373' />
           <entity id='AS' value='3875' />
           <entity id='EU' value='727' />
           <entity id='AF' value='885' />
           <entity id='AU' value='32' />
      </data>
</map>
```

Each XML document for maps starts with the <map> element. As the example shows, you can specify formatting attributes for the <map> element. Within the <map> element is the <data> element. The <data> element contains <entity> elements that define the data for each entity on the map. For example, <entity id='NA' value='515' /> assigns the population value 515 to the NA (North America) entity. The entity ID corresponds to the internal ID, which you saw earlier in the Map Specification Sheet for World Map.

2 Write code to generate an XML string that provides data defined as <entity> elements. The code needs to create the content within the <data> element. A logical place to put this code is in the OnCreate( ) method for the table's detail row because this method executes with each retrieval of a data row from the data set.

   1 In the report layout, select the detail row of the table, choose Script, then select OnCreate.

   2 Type the following code in the script editor:

```
var entityLine = "<entity id='" +
   this.getRowData().getColumnValue("Territory_ID") + "' "
   + "value='" + this.getRowData().getColumnValue("SALES")
   + "'/>";

dataPart = dataPart + entityLine;

reportContext.setPersistentGlobalVariable("g_dataPart",
   dataPart );
```

   The code iterates through the data rows in the data set, and builds an XML string using the Territory_ID and SALES values. The full XML string is stored as a persistent global variable so that it can be accessed anywhere in the report.

3 Initialize the dataPart variable by adding the following code to the table's OnCreate( ) method:

```
dataPart="";
```

   The table's OnCreate( ) method is typical for placing start-up or initialization code for report elements in a table.

Task 9: Create the dataXML variable and pass the data

As described earlier in this chapter, one of the ways to pass data to a Flash object is through the dataXML variable. In this procedure, create the dataXML variable and assign the XML content to the variable.

1 In the report layout, select the Flash object.

2 In Property Editor, choose Flash Variables, as shown in Figure 12-18.

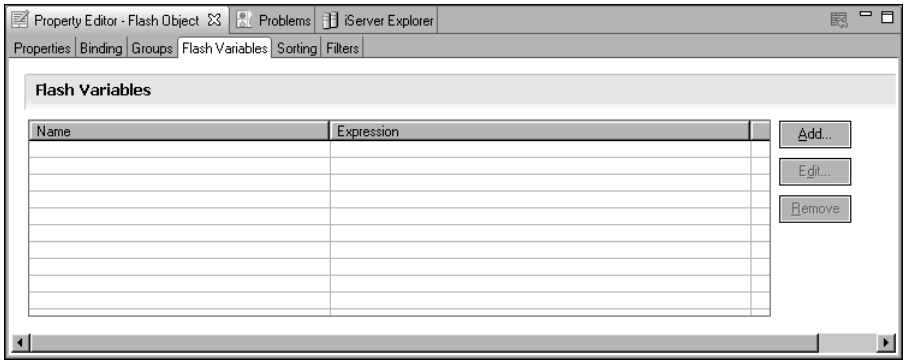


Figure 12-18 Flash Variables tab in Property Editor

3 Choose Add.

4 In Add Variables, do the following:

   1 In Name, type:

   ```
   dataXML
   ```

   2 In Expression, choose the JavaScript expression builder.

5 In the JavaScript expression builder, type the following expression:

```
var g_dataPart =
   reportContext.getPersistentGlobalVariable("g_dataPart");

"<map><data>" + g_dataPart + "</data></map>"
```

   The first statement retrieves the XML data string that you created earlier and stored in the persistent global variable g_dataPart. The second statement builds a bare-bones XML data document that contains only the essential elements. This line creates the required <map> and <data> elements, and appends the g_dataPart variable, which supplies the <entity> data. The XML does not include any formatting attributes.

6 Choose OK.

7 Choose Preview. The previewer displays the Flash map. Move the mouse pointer over each continent. A tooltip displays the continent's full name and the sales total for that continent (if sales data exists for the continent), as shown in Figure 12-19. This Flash map uses all the default data and formatting attributes.

Task 10: Format the Flash map

Now that you verified that the map displays the correct sales data for the territories, you can focus on adding functionality and visual interest to the map. Perform the following tasks in this section:

- Display sales values in a more readable format.
- Change the colors used in the map.
- Define data ranges and apply different colors to each range.
- Create city markers.

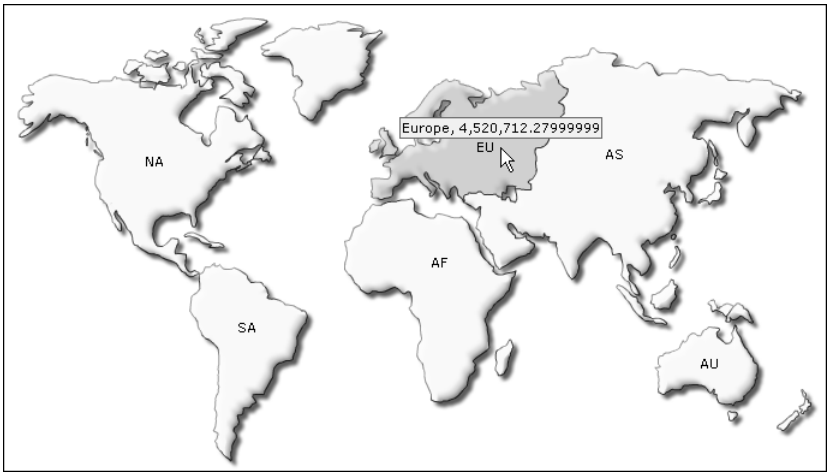


Figure 12-19 Preview of the Flash map with the mouse pointer over Europe

You specify formatting attributes by editing the XML string you typed in the previous task.

Display sales values in a more readable format

In this procedure, format the sales value so that it displays $4.52M instead of 4,520,712.27999999. Also, specify that the sales values appear on the map in the following format:

```
EU: $4.52M
```

1. Choose Layout to resume editing the map.
2. Select the Flash object. In Property Editor, choose Flash Variables, select the dataXML variable, and choose Edit.
3. In Edit Variables, choose the expression builder.
4. In the line that defines the XML (the line that begins with "<map>), add the text shown in bold. You must type the entire XML string in a single line.

```
"<map decimals='2' formatNumberScale='1' numberPrefix='$'
   includeValueInLabels='1' labelSepChar=': '><data>" +
   g_dataPart + "</data></map>"
```

For information about these attributes, see the “XML Attributes” topic in the Flash map help.

5 Choose Validate to verify the expression. If there are no syntax errors, choose OK.

6 In Edit Variables, choose OK.

7 Choose Preview. As Figure 12-20 shows, the sales values appear in $4.52M format on the map and in the tooltip.

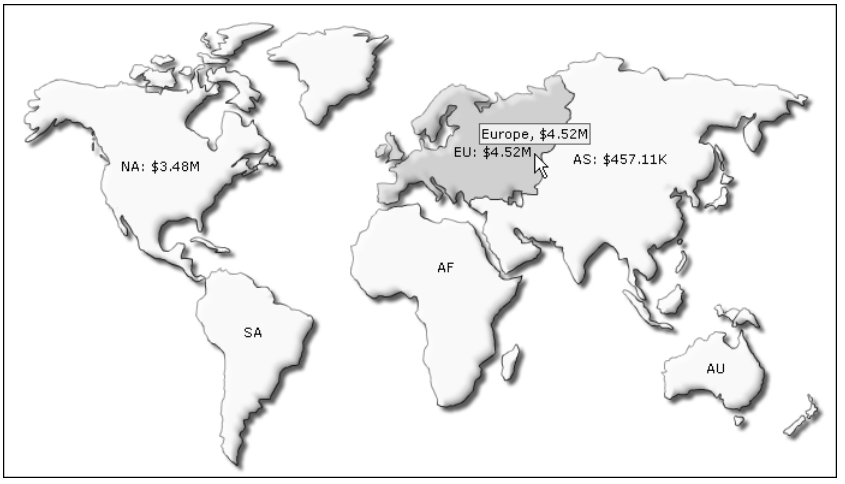


Figure 12-20 Flash map displaying sales values in new format

Building the XML string in readable pieces

As you add attributes, the XML string becomes increasingly difficult to type and read as a single line in the expression builder. This procedure shows how to build the XML string piece by piece.

1 Choose Layout to resume editing the map.

2 Edit the dataXML expression in the expression builder. Replace the line that defines the XML string (the line that begins with "<map>) with the following lines:

```
var str = "<map "
//Format sales numbers
str += "decimals='2' formatNumberScale='1' numberPrefix='$'"

//Display sales numbers in the map
str += "includeValueInLabels='1' labelSepChar=': '"
str += ">"
```

(continues)

```
//Define data
str += "<data>" + g_dataPart + "</data>"
str += "</map>"
```

The str variable stores the XML string. The += operator adds each successive piece of string to the current string. By building the XML string in pieces, you can read, edit, delete, and add attributes easily. As the example shows, you can also add comments about the purpose of the attributes.

3 Choose Validate to verify the syntax of the expression.

4 Preview the report to ensure that the map displays correctly. The expression builder's validation does not verify that the XML string contains valid content.

Change the colors used in the map

In this procedure, change the fill and background colors of the map. Use Hex codes for the color values.

1 Choose Layout to resume editing the map.

2 Edit the dataXML expression in the expression builder. Add the following lines before the str += ">" line:

```
//Colors in map
str += "fillColor='DDDDDD' bgColor='FFFFDC'"
```

3 Validate the expression, then preview the report.

Define data ranges and apply different colors to each range

In this procedure, categorize the sales data into the following ranges, and apply a different color to each range:

```
0 - 1000000, Below target
1000001 - 4000000, Within target
4000001 - 8000000, Above target
```

1 Choose Layout to resume editing the map.

2 Edit the dataXML expression in the expression builder. Add the following lines after the str += ">" line. Each str line must be a single line.

```
//Create data ranges
str += "<colorRange> "
str += "<color minValue='0' maxValue='1000000'
   displayValue='Below target' color='CCFF99' />"
str += "<color minValue='1000001' maxValue='4000000'
   displayValue='Within target' color='66CCFF' />"
str += "<color minValue='4000001' maxValue='8000000'
   displayValue='Above target' color='FFDDFF' />"
str += "</colorRange>"
```

3 Validate the expression, then preview the report.

Create city markers

In this procedure, display markers for these cities in which there are sales offices: New York, Paris, San Francisco, and Tokyo. To display markers, define the properties of each marker, including a user-specified ID, its XY position, the label to display, and the position of the label relative to the marker. You can also specify the shape, size, and color of each marker. After defining the markers, create the list of markers to display on the map.

1 Choose Layout to resume editing the map.

2 Edit the dataXML expression in the expression builder. Add the following lines after the data range definition, that is, after the str += "</colorRange>" line. Each str line must be a single line.

```
// Define city markers
str += "<markers>"
str += " <definition>"
str += "<marker id='NYC' x='210' y='140' label='New York'
   labelPos='bottom' />"
str += "<marker id='PAR' x='360' y='130' label='Paris'
   labelPos='bottom' />"
str += "<marker id='TOK' x='630' y='160' label='Tokyo'
   labelPos='right' />"
str += "<marker id='SFO' x='80' y='163' label='San Francisco'
   labelPos='left' />"
str += "</definition>"

//Specify the shape, size, and color of the markers
str += "<shapes>"
str += "<shape id='TOKdot' type='circle' radius='3'
   fillColor='ffd700' labelPadding='+1' /> "
str += "<shape id='PARdot' type='circle' radius='3'
   fillColor='ffd700' labelPadding='-2' /> "
str += "<shape id='NYCdot' type='circle' radius='3'
   fillColor='ffd700' labelPadding='+1' /> "
str += "<shape id='SFOdot' type='circle' radius='3'
   fillColor='ffd700' labelPadding='+1' /> "
str += "</shapes>"

//Specify which markers to display
str += "<application>"
str += "<marker id='TOK' shapeId='TOKdot' />"
str += "<marker id='NYC' shapeId='NYCdot' />"
str += "<marker id='PAR' shapeId='PARdot' />"
str += "<marker id='SFO' shapeId='SFOdot' />"
str += "</application>"
str += "</markers>"
```

The entire dataXML expression should look like the one shown in Figure 12-21.

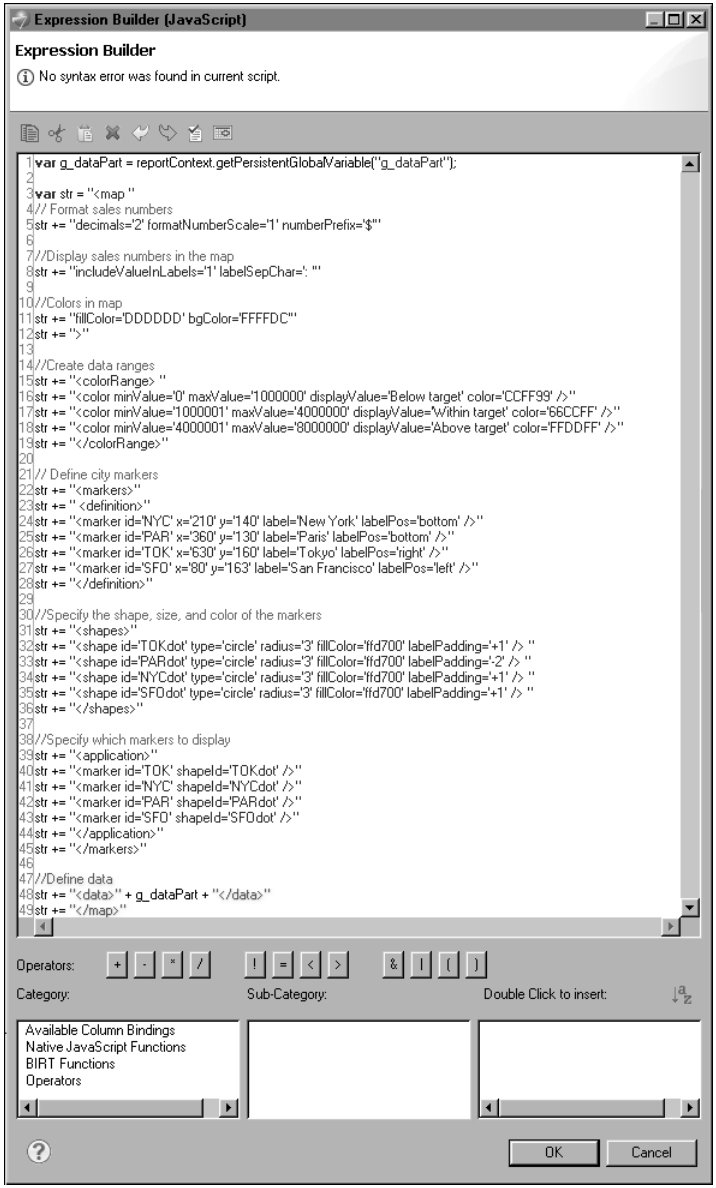


Figure 12-21 Expression builder displaying the entire dataXML expression

3 Validate the expression, then preview the report. The map should look like the one shown in Figure 12-22. The territories with sales data appear in different

colors. A legend displaying the data range colors and labels appears on the right. The map displays circular markers and labels for San Francisco, New York, Paris, and Tokyo.

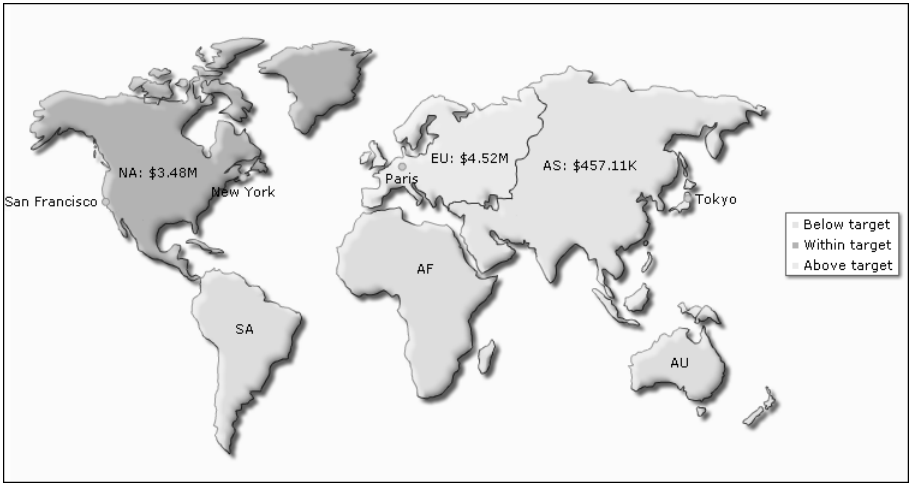


Figure 12-22 Flash map displaying all the formats you applied

This tutorial demonstrates only a few of the attributes that you can use to format and manipulate a Flash map. For a complete list and descriptions of the attributes, see the Flash map help.

Tutorial 4: Creating a Flash chart that gets data through the dataURL variable

This tutorial provides step-by-step instructions for building a report that uses a combination chart from the Flash object library to display revenue data by country. The chart uses data from the Classic Models sample database, which you convert to XML and pass to the chart through the dataURL variable.

As described earlier, using the dataURL variable to pass data requires Java programming and plug-in development experience. Although this tutorial provides detailed procedures accompanied by screen illustrations and conceptual explanations, Java programming experience is essential in the event basic troubleshooting is required or your Eclipse environment does not match exactly the environment shown in this tutorial.

You perform the following tasks in this tutorial:

- Create a new report.
- Build a data source.

- Build a data set.
- Add a Flash chart to the report.
- Create a plug-in.
- Define an extension.
- Create a Java class.
- Implement methods in the class.
- Deploy the plug-in.
- Create the dataURL variable.

Task 1: Create a new report

1 Choose File➤New➤Report.

2 On New Report, type the following text as the file name:

```
RevenueByCountry.rptdesign
```

3 Choose Finish. A blank report appears in the layout editor.

Task 2: Build a data source

In this procedure, create a data source to connect to the Classic Models sample database.

1 Choose Data Explorer.

2 Right-click Data Sources, and choose New Data Source from the context menu.

3 Select Classic Models Inc. Sample Database from the list of data sources. Use the default data source name, then choose Next. Connection information about the new data source appears.

4 Choose Finish. The new data source appears under Data Sources in Data Explorer.

Task 3: Build a data set

In this procedure, build a data set to indicate what data to retrieve from various tables in the database.

1 In Data Explorer, right-click Data Sets, and choose New Data Set.

2 In New Data Set, in Data Set Name, type the following text:

```
Revenue
```

Use the default values for the other fields.

- Data Source Selection shows the name of the data source that you created earlier.
- Data Set Type specifies that the data set uses a SQL SELECT query to retrieve the data.

3 Choose Next.

4 In New Data Set—Query, type the following SQL SELECT statement to retrieve the revenue and total of items sold for each country:

```
SELECT CLASSICMODELS.CUSTOMERS.COUNTRY,
SUM(CLASSICMODELS.ORDERDETAILS.QUANTITYORDERED *
   CLASSICMODELS.ORDERDETAILS.PRICEEACH) as SALES,
SUM(CLASSICMODELS.ORDERDETAILS.QUANTITYORDERED) as QUANTITY
FROM CLASSICMODELS.CUSTOMERS,
CLASSICMODELS.ORDERS,
CLASSICMODELS.ORDERDETAILS
WHERE CLASSICMODELS.CUSTOMERS.CUSTOMERNUMBER =
   CLASSICMODELS.ORDERS.CUSTOMERNUMBER
AND CLASSICMODELS.ORDERS.ORDERNUMBER =
   CLASSICMODELS.ORDERDETAILS.ORDERNUMBER
GROUP BY CLASSICMODELS.CUSTOMERS.COUNTRY
HAVING SUM(CLASSICMODELS.ORDERDETAILS.QUANTITYORDERED *
   CLASSICMODELS.ORDERDETAILS.PRICEEACH) > 200000
ORDER BY 2 DESC
```

5 Choose Finish to save the data set. Edit Data Set displays the columns specified in the query, and provides options for editing the data set.

6 Choose Preview Results. Figure 12-23 shows the data rows that the data set returns.

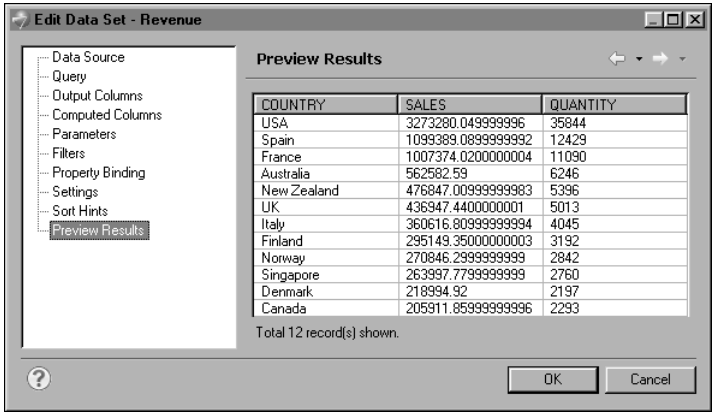


Figure 12-23 Revenue data set preview

7 Choose OK.

Task 4: Add a Flash chart to the report

In this procedure, add the 2D dual-Y combination chart from the Flash object library to the report.

1. Drag a Flash Object element from the palette and drop it in the report layout.
2. In Flash Builder, specify the following information:
    1. In Select content from, select Flash Object Library.
    2. In Enter resource file, choose the open folder button to select a Flash file from the library.
    3. In Browse for Flash Files, expand Flash Charts, and select MSCombiDY2D.swf, as shown in Figure 12-24.

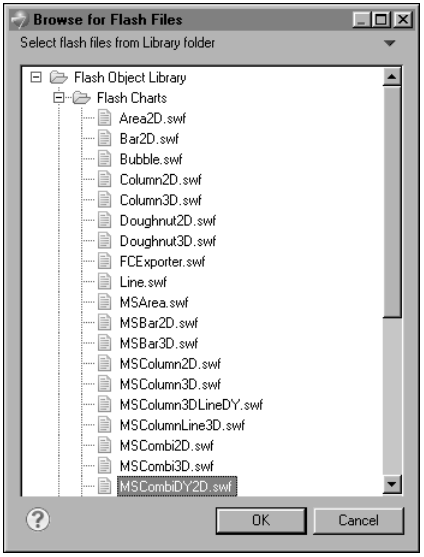


Figure 12-24 Selecting a Flash chart from the Flash Object Library

    4. Choose OK. In Flash Builder, the path to the Flash file appears in Enter resource file.
3. Choose Finish.
4. Bind the Flash object to the Revenue data set.
    1. While the Flash object is selected, in Property Editor, choose the Binding tab.
    2. In the Binding page, in Data Set, select Revenue.

       The Flash object has access to data in the selected data set.

Task 5: Create a plug-in

Data provided to any Flash object must be in the specific XML format that the object requires. To use the dataURL variable to pass data to the Flash object, you create a Java class to generate an XML document, and deploy the class as a plug-in. In this procedure, you create the plug-in using the Eclipse Plug-in Development Environment (PDE).

1 In the main menu, choose Window➝Open Perspective➝Other.

2 In Open Perspective, choose Plug-in Development, as shown in Figure 12-25.

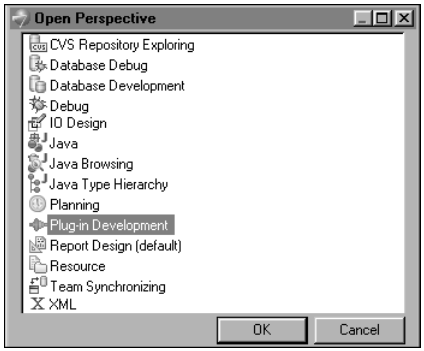


Figure 12-25 Selecting the Plug-in Development perspective

Choose OK. The Plug-in Development perspective displays the views and tools for creating and managing plug-ins.

3 Choose File➝New➝Project.

4 In New Project, expand Plug-in Development, and select Plug-in Project, as shown in Figure 12-26.

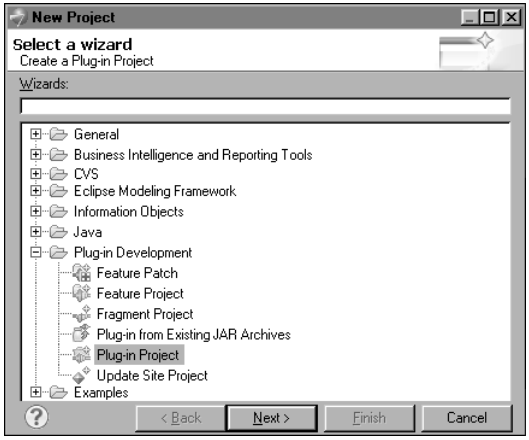


Figure 12-26 Selecting Plug-in Project

Choose Next.

5 In New Plug-in Project, specify the following project information:

1 In Project Name, type:

```
com.actuate.birt.flash.library.sample.CombinationChart
```

This name follows the naming convention used by Actuate BIRT plug-ins.

2 In Target Platform, select the following option:

```
OSGi framework: Equinox
```

OSGi is a framework specification for developing and deploying modular Java applications. Equinox is an Eclipse project that implements the OSGi framework and is the plug-in technology used by Eclipse and BIRT.

3 Use the default values for the other properties. Figure 12-27 shows the information specified for the plug-in project.

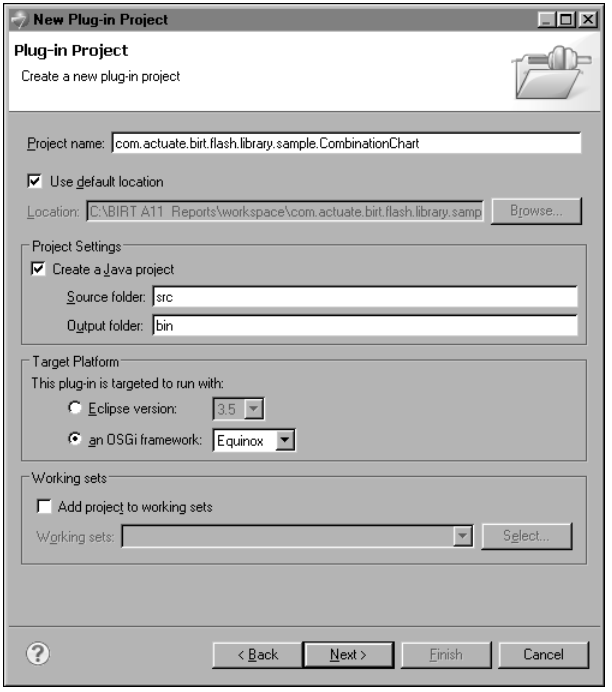


Figure 12-27 Properties of the plug-in project

Choose Next.

6 In New Plug-in Project—Content, specify the information for generating the plug-in.

1 In Properties, use all the default values.

- ID identifies the plug-in. By default, the value you specified as the project name in the previous step is used as the ID value.
- Version is the version number to assign to this plug-in.
- Name is the plug-in's display name, which appears in general descriptions about the plug-in.
- Provider is the name of the plug-in contributor.
- Execution Environment specifies the JRE (Java Runtime environment) to use.

2 In Options, uncheck the first option, Generate an activator, a Java class that controls the plug-in's life cycle.

Figure 12-28 shows the information for generating the plug-in.

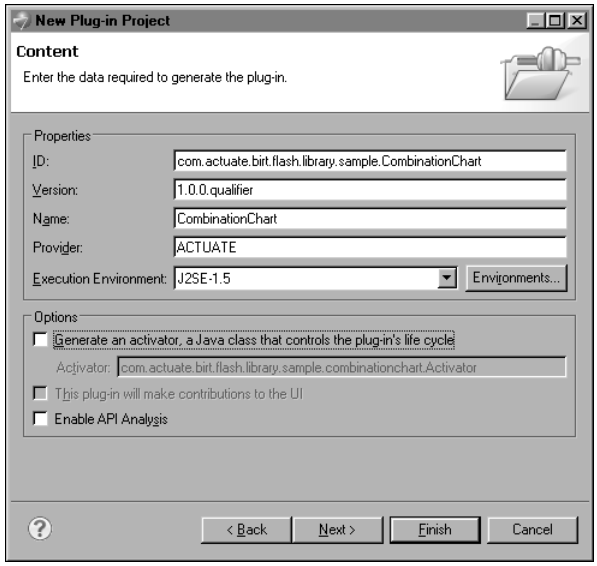


Figure 12-28 Information for generating the plug-in

Choose Finish.

Eclipse creates the plug-in. The Plug-in editor displays an Overview page, as shown in Figure 12-29. This page shows the properties of the plug-in and provides links to pages about developing, testing, and deploying a plug-in.

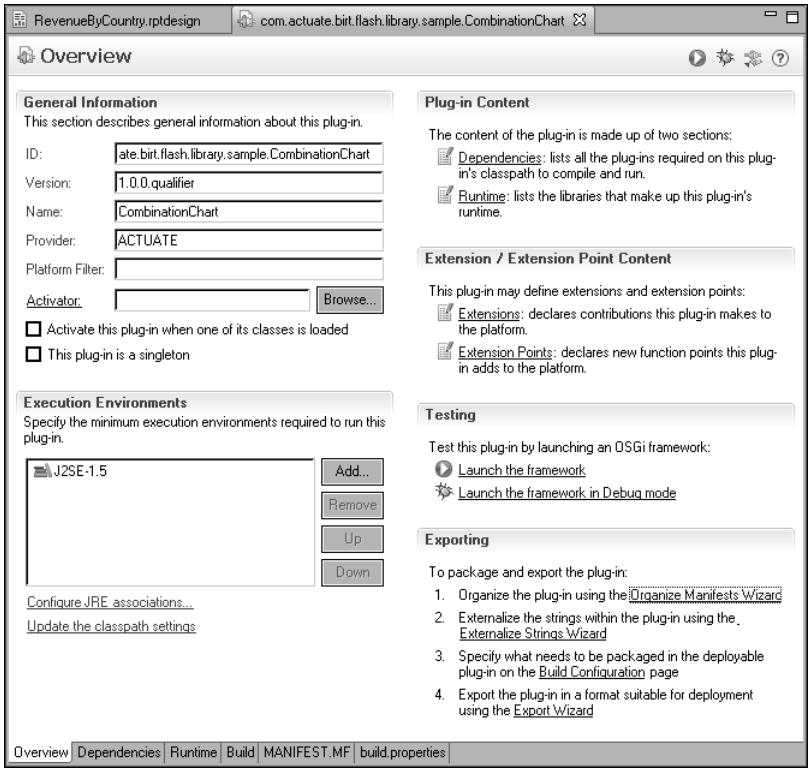


Figure 12-29 Plug-in editor displaying overview information

Task 6: Define an extension

In this procedure, define an extension that customizes the data extraction functionality provided by the org.eclipse.birt.report.engine.dataextraction plug-in. This plug-in defines an extension point, which your plug-in uses to define a custom extension to retrieve data from the data set and generate the XML data required by the Flash chart.

1. In the Plug-in editor, choose the Extensions tab. If an Extension tab is not available, do the following:
    1. In the Overview page, in the Extension/Extension Point Content section, choose the Extensions link.
    2. In the message that appears, choose Yes.
2. In the Extensions page, choose Add.
3. In New Extension, perform the following steps:
    1. Deselect Show only extension points from the required plug-ins.

2 Select org.eclipse.birt.report.engine.dataExtraction from the list of extension points, as shown in Figure 12-30.

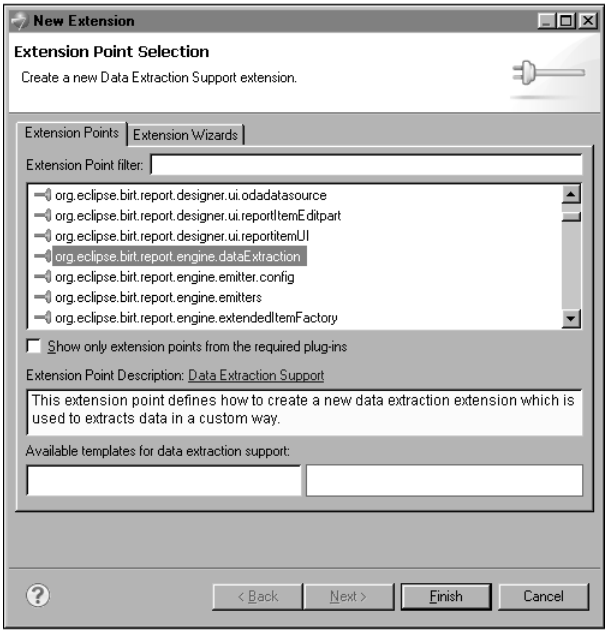


Figure 12-30 Selecting an extension point

Choose Finish. A message asks if you want to add a dependency to the org.eclipse.birt.report.engine plug-in. Choose Yes. The Extensions page displays the new extension to the plug-in.

4 In Extension Element Details, edit the extension properties using the values shown in Table 12-1.

Table 12-1 Extension properties

Property	Value	Description
id	com.actuate.birt.flash.library.sample.combchart.XMLGenerator	The extension identifier.
format	CombChartXMLFormat	The supported format of this data extraction extension. Later, when you create the Flash chart's dataURL variable, you pass the format value as an argument to the createDataURL( ) method.

(continues)

Table 12-1 Extension properties (continued)

Property	Value	Description
class	com.actuate.birt.flash.library.sample.combchart.XMLGenerator	The Java class that implements the IDataExtractionExtension interface. You create this class later.
mimeType	text/xml	Mime type of the file generated by the extension.
name	Combination Chart XML Format	The name of the extension. This name appears in the user interface.
isHidden	true	Specifies whether format is shown in the user interface.

Figure 12-31 shows the specified properties in the Extensions page.

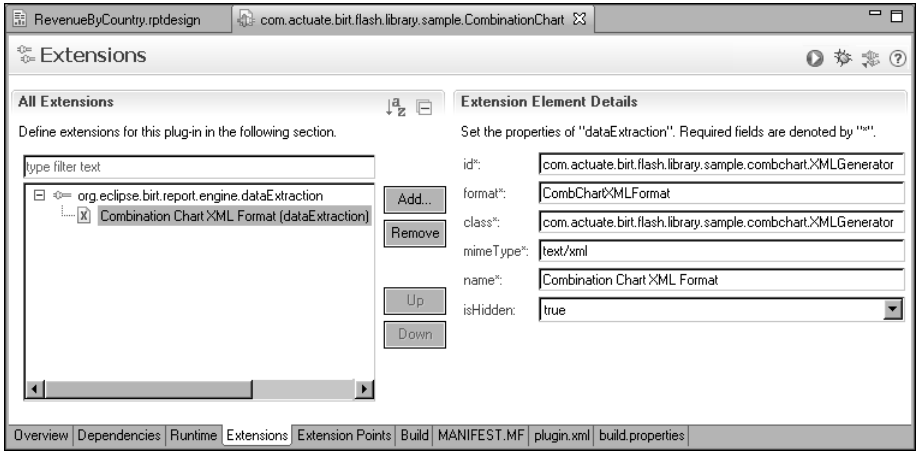


Figure 12-31 Properties of the extension

5 Save the plug-in. Package Explorer displays the folder structure of the plug-in, as shown in Figure 12-32.

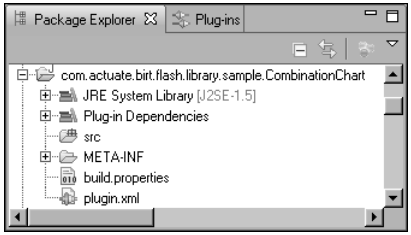


Figure 12-32 Package Explorer displaying the folder structure of the plug-in

Task 7: Create a Java class

In this procedure, create a Java class that contains the code to generate the XML data required by the Flash chart. This class implements the data extraction interface, IDataExtractionExtension.

1 In Package Explorer, right-click the src folder, then choose New➛Class.

2 In New Java Class, specify the following information:

   1 In Package, type:

   ```
   com.actuate.birt.flash.library.sample.combchart
   ```

   2 In Name, type:

   ```
   XMLGenerator
   ```

   3 In Interfaces, choose Add to add the data extraction interface.

   4 In Implemented Interfaces Selection, select IDataExtractionExtension. If the dialog box does not display any interfaces, do the following:

   - Under Choose interfaces, type:

     ```
     IDataE
     ```

     Matching items lists the interfaces that begin with IDataE.

   - Select IDataExtractionExtension.
   - Choose OK.

   5 Use the default values for the other options.

   6 Choose Finish. In Package Explorer, the class, XMLGenerator.java, appears in the plug-in's src folder. The content of XMLGenerator.java appears in the editor, as shown in Figure 12-33.

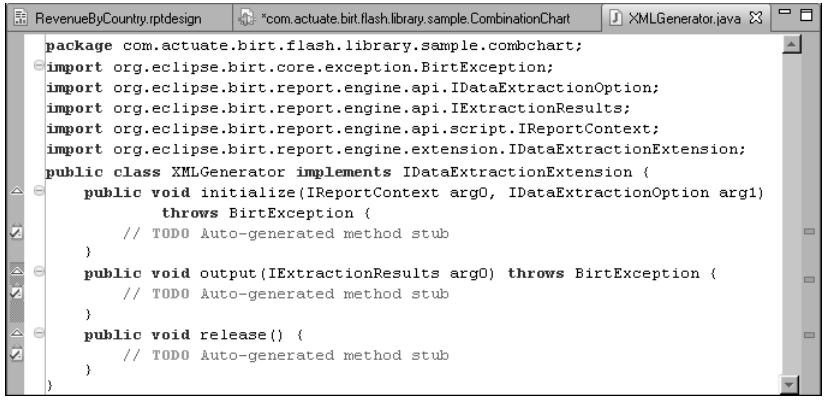


Figure 12-33 Code template for XMLGenerator.java

Task 8: Implement methods in the class

When you create a Java class using the wizard, Eclipse generates a code template, as shown in Figure 12-33. As the class declaration shows, the XMLGenerator class implements the IDataExtractionExtension interface. The interface defines three methods, which your class must implement. Perform the following tasks in this section:

- Import the required packages.
- Implement the initialize( ) method.
- Implement the output( ) method.
- Implement the release( ) method.

Import the required packages

The code template contains import statements to include the packages that contain the classes your code needs. Verify that the code contains the following import statements. If any are missing, add them.

```
import java.io.IOException;
import java.io.OutputStream;
import java.io.UnsupportedEncodingException;

import org.eclipse.birt.core.exception.BirtException;
import org.eclipse.birt.report.engine.api.IDataExtractionOption;
import org.eclipse.birt.report.engine.api.IDataIterator;
import org.eclipse.birt.report.engine.api.IExtractionResults;
import org.eclipse.birt.report.engine.api.script.IReportContext;
import org.eclipse.birt.report.engine.extension.
   IDataExtractionExtension;
```

Implement the initialize( ) method

The initialize( ) method is the first method that the BIRT report engine calls before rendering the Flash object. Use this method to initialize resources.

1. Add the following line after the class declaration line (the line that begins with public class XMLGenerator):

   ```
   private IDataExtractionOption option;
   ```

   This statement declares a private variable, option, of type IDataExtractionOption. The initialize( ) method takes an input argument of this type.

2. Add the following line after the initialize( ) method declaration:

   ```
   this.option = arg1;
   ```

   This statement assigns the option variable to the input argument arg1.

Listing 12-3 shows the edited code in the class definition and initialize( ) method.

Listing 12-3 Class definition and initialize( ) method implementation

```
public class XMLGenerator implements IDataExtractionExtension {

   private IDataExtractionOption option;

   public void initialize(IReportContext arg0,
   IDataExtractionOption arg1)
        throws BirtException {
        this.option = arg1;
   }
```

Implement the output( ) method

The output( ) method is where you write the code to build the XML data to pass to the Flash chart. Listing 12-4 shows the code.

Read the comments embedded in the code to understand what each section of code does. For information about the XML elements and attributes used to build the XML data, see the sample XML for the 2D dual-Y combination chart. In the online help for Flash charts, choose Chart XML API—Combination Charts—2D Dual Y Combination.

Listing 12-4 output( ) method implementation

```
public void output(IExtractionResults results) throws
   BirtException {
   //Get the handle of the OutputStream defined in the
   //IDataExtractionOption option interface
   OutputStream stream = option.getOutputStream();

   //If the stream is not null, define three string buffers.
   //The xml buffer is used to build the full XML.
   //The xmlSales and xmlQty buffers store the data for the
   //Revenue and Quantity series.
   if ( stream != null )
   {
      StringBuffer xml = new StringBuffer();
      StringBuffer xmlSales = new StringBuffer();
      StringBuffer xmlQty = new StringBuffer();

   //Start building the XML. This section defines chart attributes
      xml.append( "<chart caption='Revenue by Country'
         PYAxisName='Revenue' SYAxisName='Quantity'
         numVisiblePlot='8' showValues='0' numberPrefix='$'
         useRoundEdges='1' labelDisplay='ROTATE' >");
```

(continues)

```
//If results is not null, iterate through the data set rows.
//Add the values to the data series. Add the data and
//additional formatting attributes to the XML.
if ( results != null )
{
   IDataIterator itr = results.nextResultIterator();
   xml.append( "<categories >" );
   xmlSales.append("<dataset seriesName='Revenue' >");
   xmlQty.append("<dataset seriesName='Quantity'
      parentYAxis='S' >");
   while ( itr.next() )
   {
      String country =
         String.valueOf(itr.getValue("COUNTRY"));
      String sales = String.valueOf(itr.getValue("SALES"));
      String qty = String.valueOf(itr.getValue("QUANTITY"));
      xml.append( "<category label='"+ country + "' />" );
      xmlSales.append( "<set value='"+ sales + "' />" );
      xmlQty.append( "<set value='"+ qty + "' />" );
   }
   xmlSales.append("</dataset>");
   xmlQty.append("</dataset>");
   xml.append( "</categories>" );
   xml.append(xmlQty);
   xml.append(xmlSales);
   xml.append("<trendlines>");
   xml.append(" <line startValue='400000' color='91C728'
      displayValue='Target' showOnTop='1'/> ");
   xml.append("</trendlines>");
   xml.append("<styles> ");
   xml.append("<definition> <style name='CanvasAnim'
      type='animation' param='_xScale' start='0'
      duration='1' /> </definition> ");
   xml.append(" <application> <apply toObject='Canvas'
      styles='CanvasAnim' /> </application> ");
   xml.append("</styles>");
   xml.append("</chart>");

   //Write the buffer to the output stream.
   //Use the try/catch blocks to catch exceptions.
   try
   {
      stream.write( xml.toString().getBytes("UTF-8"));
      stream.flush();
   }
   catch ( UnsupportedEncodingException e )
   {
      e.printStackTrace();
```

```
            }
            catch ( IOException e )
            {
               e.printStackTrace();
            }
         }
      }
   }
```

Implement the release( ) method

Use the release( ) method to clean up allocated resources.

1. Add the following line after the release( ) method declaration:

   ```
   this.option = null;
   ```

   This statement releases the handle to the output stream.

2. Save your changes to XMLGenerator.java.

You have finished implementing the class and the plug-in.

Task 9: Deploy the plug-in

In this procedure, deploy the plug-in using Eclipse's Export utility. This utility creates a plug-in JAR file and copies it to a specified folder.

1. From the main menu, choose File➤Export.
2. In Export, expand Plug-in Development, and select Deployable plug-ins and fragments, as shown in Figure 12-34.

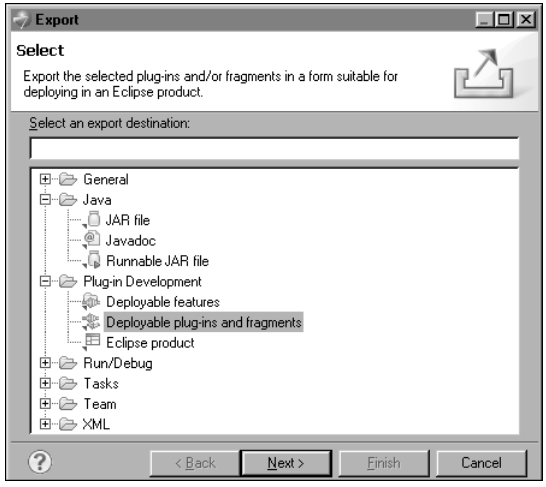


Figure 12-34 Selecting Deployable plug-ins and fragments

3 Choose Next. In Export, Available Plug-ins and Fragments displays the plug-in you created.

4 Select com.actuate.birt.flash.library.sample.CombinationChart.

5 In Destination—Directory, type the following path, if necessary:

```
C:\Program Files\Actuate11\MyClasses\eclipse
```

All custom plug-ins that Actuate BIRT Designer uses must be placed in this folder.

6 Choose Finish.

7 Restart Actuate BIRT Designer. This step is required for the new plug-in to take effect.

Task 10: Create the dataURL variable

In this procedure, create the dataURL variable to pass the XML data generated by the plug-in to the Flash chart.

1 Open the report design perspective by choosing Report Design in the toolbar.

2 Choose RevenueByCountry.rptdesign, the report you created earlier in this tutorial.

3 In the report layout, select the Flash object.

4 In Property Editor, choose the Flash Variables tab.

5 In Flash Variables, choose Add.

6 In Add Variables, do the following:

1 In Name, type:

```
dataURL
```

2 In Expression, choose the JavaScript expression builder.

7 In the JavaScript expression builder, type the following expression:

```
flashContext.createDataURL("CombChartXMLFormat", true, null);
```

The first argument, CombChartXMLFormat, is the format specified in the extension properties of the plug-in. The second argument, true, specifies that the URL is encoded. The third argument, null, specifies that there are no custom parameter names and values to pass to the URL.

8 Choose OK.

9 Preview the report. The Flash chart should look like the one shown in Figure 12-35. The chart has two *y* axes. The left axis displays revenue values and the right axis displays quantity values. The column chart presents revenue data and the line chart presents quantity data.

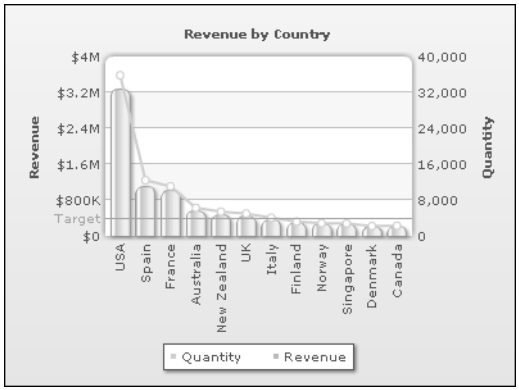


Figure 12-35 Preview of Flash chart

Debugging a Flash object

Because using a Flash object from the Flash object library requires programming and typing XML content, implementation errors are common. To troubleshoot errors, use the following debugging tools:

- The JavaScript debugger in Actuate BIRT Designer. Use this tool to debug the JavaScript code you write when using the dataXML variable to pass data to the Flash object.
- The Eclipse debugger. Use this tool to debug the entire report and the Java classes that the report uses. This tool is useful for debugging the Java class you write when using the dataURL variable to pass data to the Flash object.
- The debug mode provided by Flash objects. Use this method to see the processing that occurs in the Flash object.

Information about using the JavaScript debugger and the Eclipse debugger is provided in the Eclipse Series book, *Integrating and Extending BIRT*. This section provides instructions for using the debug mode in Flash objects.

Using the Flash object's debug mode

The debug mode provides a description and status of the Flash object. When you run a report in debug mode and there are errors generating the Flash object, the debugger lists the errors. If the Flash object runs without errors, the debugger shows the XML used to create the Flash object. Figure 12-36 shows an example of the type of information displayed by the debugger.

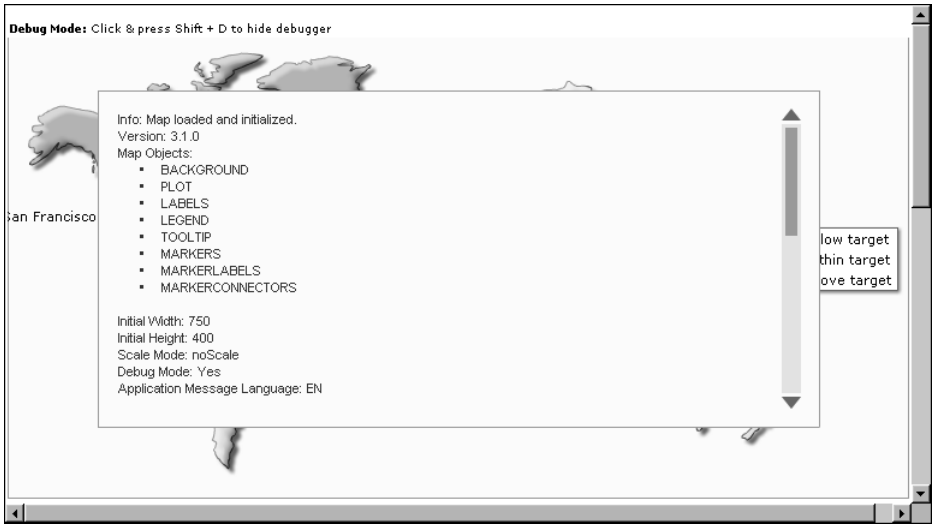


Figure 12-36 Information displayed by the Flash object debugger

How to enable debug mode

1. In the report layout, select the Flash object to debug.
2. In Property Editor, choose Flash Variables, then choose Add.
3. In Add Variables, specify the following information, then choose OK:
   1. In Name, type:

      `debugMode`

   2. In Expression, type:

      `1`
4. Preview the report. A debug window opens on top of the Flash object, as shown in Figure 12-36. To hide the debug window, click it while pressing Shift+D. Use the same keystrokes to redisplay the debug window.

How to disable debug mode

Edit the debugMode variable. Set Expression to 0.

Resolving errors

For information about the types of errors, their typical causes, and ways to resolve them, see the Debugging topics in the InfoSoft online documentation. Figure 12-37 shows a portion of the "Basic Troubleshooting" topic under "Debugging Your Maps."

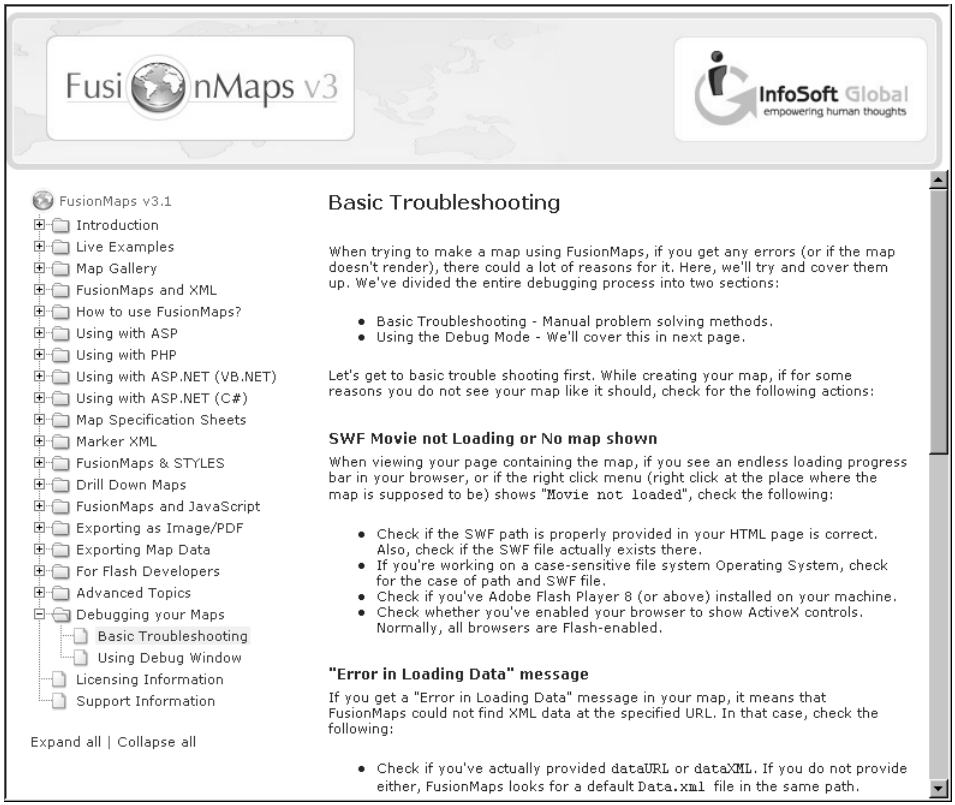


Figure 12-37 Troubleshooting topic in the InfoSoft online documentation

Chapter

13

Writing expressions using EasyScript

This chapter contains the following topics:

- About EasyScript
- Using the EasyScript expression builder
- Changing the default expression syntax
- Functions
- Operators

About EasyScript

EasyScript is an expression syntax similar to the syntax used in Excel formulas. Like Excel, EasyScript provides functions for performing calculations on report data. In Actuate BIRT Designer, EasyScript is supported in most places an expression is required. For example, when specifying an expression for a computed column, a column binding, a filter condition, or a map rule, you can use either JavaScript or EasyScript.

Choosing between EasyScript and JavaScript

You can use both JavaScript and EasyScript expressions in a report. For simple expressions or common calculations, the choice is often based on syntax familiarity or simplicity. Users who work with Excel functions will find EasyScript syntax familiar and easy to use.

The following example is an EasyScript expression that rounds values in a Price field to the nearest integer:

```
ROUND([Price])
```

The following example is the equivalent JavaScript expression:

```
Math.round(row["Price"])
```

Both expressions are straightforward, although one could argue that the EasyScript syntax is simpler. Now, compare the expressions used to round the Price values to 2 decimal places. In the following expressions, the first shows EasyScript syntax, and the second shows JavaScript syntax:

```
ROUND([Price], 2)
Math.round(row["Price"]*100)/100
```

In this case, the EasyScript syntax is clearly simpler and more intuitive. The EasyScript ROUND( ) function provides a second argument that lets you specify the number of decimal places to which to round the number. The JavaScript round( ) function does not, and, therefore, requires additional mathematical operations.

If a report needs complex calculations that require lines of code or calculations that cannot be done with EasyScript, use JavaScript. For information about writing JavaScript expressions, see *BIRT: A Field Guide*.

Syntax rules

When writing an EasyScript expression, observe the following rules:

- Enclose field names within square brackets ([ ]), for example, [CustomerID].

- Field names and function names are case-sensitive. All function names are uppercase.
- When creating an expression that contains a literal date, always type the date according to the conventions of the US English locale. For example, if working in the French locale, type 07/10/2010 to represent July 10, 2010. Do not type 10/07/2010, which is the convention for dates in the French locale. The following expression, which calculates the number of days from the current date to Christmas, includes a literal date:

  ```
  DIFF_DAY(TODAY(), "12/25/10")
  ```

- When creating an expression that contains a literal number, always type the number according to the conventions of the US English locale. Use a period (.), not a comma (,) as the decimal separator.

Using the EasyScript expression builder

When specifying an expression, the JavaScript syntax is the default. Figure 13-1 shows the icon that represents JavaScript syntax. Clicking on this icon opens the JavaScript expression builder.

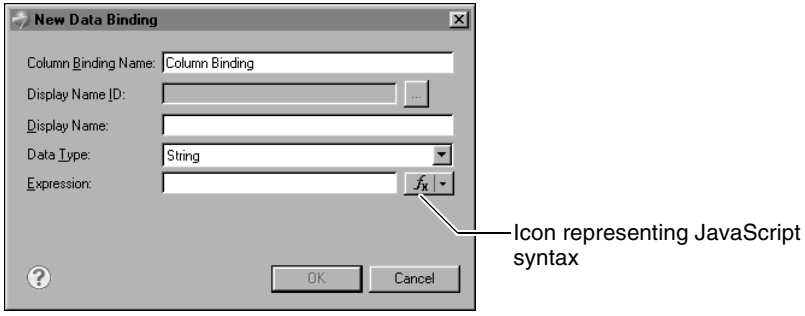


Figure 13-1 An expression property set to use a JavaScript expression

To switch to EasyScript syntax, click the arrow button next to the JavaScript syntax icon and choose EasyScript Syntax, as shown in Figure 13-2.

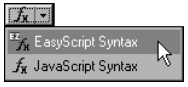


Figure 13-2 Switching to EasyScript

This action opens the EasyScript expression builder, shown in Figure 13-3.

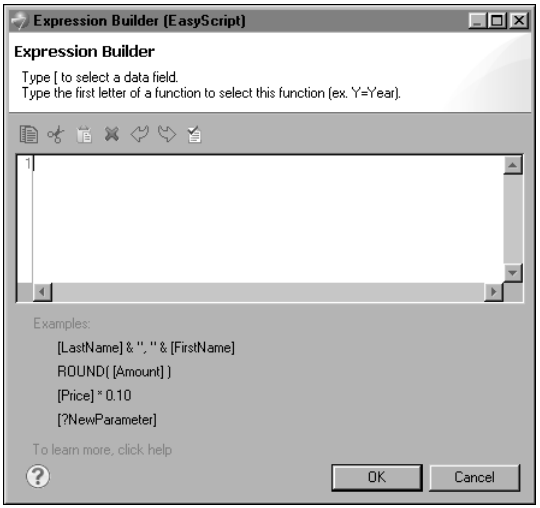


Figure 13-3 EasyScript expression builder

Like the JavaScript expression builder, the EasyScript expression builder provides help selecting functions and fields to use in an expression. To use a function in an expression, type the first letter of the function, then select a function from the list that appears. To use a field, type the left square bracket ([), then select a field from the list.

When you finish creating an expression, choose Validate to verify the expression.

Changing the default expression syntax

If you consistently use EasyScript or use it more than JavaScript, you can change the Default Syntax property in Preferences to EasyScript syntax. To access this property, select Window➤Preferences, and choose Report Design—Expression Syntax. After changing the default syntax, the EasyScript syntax icon appears by default every time you create a new expression.

The Default Syntax property does not convert existing JavaScript expressions to EasyScript expressions, and vice versa. To change the syntax of an expression, you must select the syntax type and edit the expression accordingly.

Functions

This section is a complete reference to all of the EasyScript functions in Actuate BIRT Designer. This reference organizes the functions alphabetically. Each function entry includes a general description of the function, its syntax, the

arguments to the function, the result the function returns, and an example that shows typical usage.

ABS( )

Returns the absolute value of a number without regard to its sign. For example, 6 is the absolute value of 6 and -6.

Syntax ABS(number)

Argument **number**
The number for which you want to find the absolute value.

Returns An integer that represents the absolute value of a specified number.

Example The following example returns the absolute value for each number in the TemperatureCelsius field:

```
ABS([TemperatureCelsius])
```

ADD_DAY( )

Adds a specified number of days to a date value.

Syntax ADD_DAY(date, n)

Arguments **date**
The date or date expression that represents the start date.

n
The number of days to add to the start date. If you specify a negative number, the result is as if the number is subtracted from the start date.

Returns The date value that results from adding the specified number of days to the start date.

Example The following example adds 15 days to each date value in the InvoiceDate field:

```
ADD_DAY([InvoiceDate], 15)
```

ADD_HOUR( )

Adds a specified number of hours to a date value.

Syntax ADD_HOUR(date, n)

Arguments **date**
The date or date expression that represents the start date. If a start date does not have a time value, the function assumes the time is midnight, 12:00 AM.

n
The number of hours to add to the start date. If you specify a negative number, the result is as if the number is subtracted from the start date.

Returns The date-and-time value that results from adding the specified number of hours to the start date.

Example The following example adds eight hours to each date value in the ShipDate field:

```
ADD_HOUR([ShipDate], 8)
```

ADD_MINUTE( )

Adds a specified number of minutes to a date value.

Syntax ADD_MINUTE(date, n)

Arguments **date**
The date or date expression that represents the start date. If a start date does not have a time value, the function assumes the time is midnight, 12:00 AM.

n
The number of minutes to add to the start date. If you specify a negative number, the result is as if the number is subtracted from the start date.

Returns The date-and-time value that results from adding the specified number of minutes to the start date.

Example The following example subtracts 30 minutes from each date in the StartTime field:

```
ADD_MINUTE([StartTime], -30)
```

ADD_MONTH( )

Adds a specified number of months to a date value.

Syntax ADD_MONTH(date, n)

Arguments **date**
The date or date expression that represents the start date.

n
The number of months to add to the start date. If you specify a negative number, the result is as if the number is subtracted from the start date.

Returns The date value that results from adding the specified number of months to the start date. This function always returns a valid date. If necessary, the day part of the resulting date is adjusted downward to the last day of the resulting month in the resulting year. For example, if you add one month to 1/31/08, ADD_MONTH( ) returns 2/29/08, not 2/31/08 or 2/28/08, because 2008 is a leap year.

Example The following example adds two months to each date value in the InitialRelease field:

```
ADD_MONTH([InitialRelease], 2)
```

ADD_QUARTER( )

Adds a specified number of quarters to a date value.

Syntax ADD_QUARTER(date, n)

Arguments

date
The date or date expression that represents the start date.

n
The number of quarters to add to the start date. If you specify a negative number, the result is the number subtracted from the start date.

Returns The date value that results from adding the specified number of quarters to the start date. A quarter is equal to three months. For example, if you add two quarters to 9/22/08, ADD_QUARTER( ) returns 3/22/09.

Example The following example adds two quarters to each date value in the ForecastClosing field:

```
ADD_QUARTER([ForecastClosing], 2)
```

ADD_SECOND( )

Adds a specified number of seconds to a date value.

Syntax ADD_SECOND(date, n)

Arguments

date
The date or date expression that represents the start date. If a start date does not have a time value, the function assumes the time is midnight, 12:00 AM.

n
The number of seconds to add to the start date. If you specify a negative number, the result is as if the number is subtracted from the start date.

Returns The date-and-time value that results from adding the specified number of seconds to the start date.

Example The following example adds 30 seconds to each date value in the StartTime field:

```
ADD_SECOND([StartTime], 30)
```

ADD_WEEK( )

Adds a specified number of weeks to a date value.

Syntax ADD_WEEK(date, n)

Arguments

date
The date or date expression that represents the start date.

n
The number of weeks to add to the start date. If you specify a negative number, the result is as if the number is subtracted from the start date.

Returns The date value that results from adding the number of weeks to the start date.

Example The following example adds two weeks to each date value in the OrderDate field:

```
ADD_WEEK([OrderDate], 2)
```

ADD_YEAR( )

Adds a specified number of years to a date value.

Syntax ADD_YEAR(date, n)

Arguments

date
The date or date expression that represents the start date.

n
The number of years to add to the start date. If you specify a negative number, the result is as if the number is subtracted from the start date.

Returns The date value that results from adding the number of years to the start date.

Example The following example adds five years to each date value in the HireDate field:

```
ADD_YEAR([HireDate], 5)
```

BETWEEN( )

Tests if a value is between two specified values.

Syntax BETWEEN(source, target1, target2)

Arguments

source
The value to test. The value can be a string, numeric, or date value.

target1
The first value in the range of values to compare to. String and date values must be enclosed in double quotation marks (" ").

target2
The second value in the range of values to compare to. String and date values must be enclosed in double quotation marks (" ").

Returns True if source is between target1 and target2, or equal to target1 or target2; returns false otherwise.

Examples The following example tests each value in the SalesTotal field to see if the value is between 10000 and 20000:

```
BETWEEN([SalesTotal], 10000, 20000)
```

The following example tests each value in the CustomerName field to see if the value is between A and M:

```
BETWEEN([CustomerName], "A", "M")
```

The following example tests each value in the ReceiptDate field to see if the value is between 10/01/07 and 12/31/07:

```
BETWEEN([ReceiptDate], "10/01/07", "12/31/07")
```

The following example uses BETWEEN( ) in conjunction with the IF( ) and ADD_DAY( ) functions to calculate a shipment date. If an orderDate value is in December 2007 (between 12/1/07 and 12/31/07), add five days to the orderDate value. If an orderDate value is in a month other than December, add three days to the orderDate value.

```
IF(BETWEEN([orderDate], "12/01/07", "12/31/07"),
   ADD_DAY([orderDate], 5), ADD_DAY([orderDate], 3))
```

CEILING( )

Rounds a number up to the nearest specified multiple.

Syntax CEILING(number, significance)

Arguments

number
The number to round up.

significance
The multiple to round number to.

Returns The number that results from the rounding. If the specified number value is an exact multiple of significance, no rounding occurs.

Examples CEILING( ) is commonly used to round up prices. For example, to avoid dealing with pennies, you can round prices in a Price field up to the nearest nickel with the following expression:

```
CEILING([Price], 0.05)
```

If the Price value is 20.52, CEILING( ) returns 20.55.

The following example rounds prices up to the nearest dime:

```
CEILING([Price], 0.1)
```

If the Price value is 20.52, CEILING( ) returns 20.60. If the Price value is 20.50, CEILING( ) returns 20.50. No rounding occurs because 20.50 is already a multiple of 0.1.

The following example rounds prices up to the nearest dollar:

```
CEILING([Price], 1)
```

If the Price value is 20.30, CEILING( ) returns 21.0.

DAY( )

Returns a number from 1 to 31 that represents the day of the month.

Syntax DAY(date)

Argument **date**
The date or date expression from which you want to extract the day.

Returns The number of the day of the month for the specified date value.

Example The following example gets the number of the day for each date value in the ShipDate field:

```
DAY([ShipDate])
```

DIFF_DAY( )

Calculates the number of days between two date values.

Syntax DIFF_DAY(date1, date2)

Arguments **date1**
The first date or date expression to use in the calculation.

date2
The second date or date expression to use in the calculation.

Returns The number of days between date1 and date2. If date1 is earlier than date2, the result is a positive number; otherwise the result is a negative number.

Example The following example calculates the time it takes to pay invoices by computing the number of days between each value in the invoiceDate field and each value in the paymentDate field:

```
DIFF_DAY([invoiceDate],[paymentDate])
```

The following example calculates the number of days from an order date to Christmas:

```
DIFF_DAY([orderDate], "12/25/10")
```

The following example calculates the number of days from the current date to Christmas. TODAY( ) is a function that returns the current date.

```
DIFF_DAY(TODAY(), "12/25/10")
```

DIFF_HOUR( )

Calculates the number of hours between two date values.

Syntax DIFF_HOUR(date1, date2)

Arguments

date1
The first date or date expression to use in the calculation. If the date does not have a time value, the function assumes the time is midnight, 12:00 AM.

date2
The second date or date expression to use in the calculation. If the date does not have a time value, the function assumes the time is midnight, 12:00 AM.

Returns The number of hours between date1 and date2.

Example The following example calculates the number of hours between each value in the startTime field and each value in the finishTime field:

```
DIFF_HOUR([startTime],[finishTime])
```

The following example calculates the number of hours from the current date to Christmas. NOW( ) is a function that returns the current date and time.

```
DIFF_HOUR(NOW(), "12/25/10")
```

DIFF_MINUTE( )

Calculates the number of minutes between two date values.

Syntax DIFF_MINUTE(date1, date2)

Arguments **date1**
The first date or date expression to use in the calculation. If the date does not have a time value, the function assumes the time is midnight, 12:00 AM.

date2
The second date or date expression to use in the calculation. If the date does not have a time value, the function assumes the time is midnight, 12:00 AM.

Returns The number of minutes between date1 and date2.

Example The following example calculates the number of minutes between each value in the startTime field and each value in the finishTime field:

```
DIFF_MINUTE([startTime],[finishTime])
```

The following example calculates the number of minutes from the current date to Christmas. NOW( ) is a function that returns the current date and time.

```
DIFF_MINUTE(NOW(), "12/25/10")
```

DIFF_MONTH( )

Calculates the number of months between two date values.

Syntax DIFF_MONTH(date1,date2)

Arguments **date1**
The first date or date expression to use in the calculation.

date2
The second date or date expression to use in the calculation.

Returns The number of months between date1 and date2. The function calculates the difference by subtracting the month number of date1 from the month number of date2. For example, if date1 is 8/1/08 and date2 is 8/31/08, DIFF_MONTH( ) returns 0. If date1 is 8/25/08 and date2 is 9/5/08, DIFF_MONTH( ) returns 1.

Example The following example calculates the number of months between each value in the askByDate field and each value in the ShipByDate field:

```
DIFF_MONTH([askByDate],[shipByDate])
```

The following example calculates the number of months from each value in the hireDate field to the end of the year:

```
DIFF_MONTH([hireDate], "12/31/10")
```

DIFF_QUARTER( )

Calculates the number of quarters between two date values.

Syntax DIFF_QUARTER(date1, date2)

Arguments

date1
The first date or date expression to use in the calculation.

date2
The second date or date expression to use in the calculation.

Returns The number of quarters between date1 and date2. DIFF_QUARTER( ) calculates the difference by subtracting the quarter number of date1 from the quarter number of date2. For example, if date1 is 1/1/10 and date2 is 3/31/10, DIFF_QUARTER( ) returns 0 because both dates are in quarter 1. If date1 is 3/31/10 and date2 is 4/15/10, DIFF_QUARTER( ) returns 1 because date1 is in quarter 1 and date2 is in quarter 2.

Example The following example calculates the number of quarters between each value in the PlanClosing field and each value in the ActualClosing field:

```
DIFF_QUARTER([PlanClosing],[ActualClosing])
```

The following example calculates the number of quarters from each value in the orderDate field to the end of the year:

```
DIFF_QUARTER([orderDate], "12/31/10")
```

DIFF_SECOND( )

Calculates the number of seconds between two date values.

Syntax DIFF_SECOND(date1, date2)

Arguments

date1
The first date or date expression to use in the calculation. If the date does not have a time value, the function assumes the time is midnight, 12:00 AM.

date2
The second date or date expression to use in the calculation. If the date does not have a time value, the function assumes the time is midnight, 12:00 AM.

Returns The number of seconds between date1 and date2.

Example The following example calculates the number of seconds between each value in the startTime field and each value in the finishTime field:

```
DIFF_SECOND([startTime],[finishTime])
```

The following example calculates the number of seconds from the current date to Christmas. NOW( ) is a function that returns the current date and time.

```
DIFF_SECOND(NOW(), "12/25/10")
```

DIFF_WEEK( )

Calculates the number of weeks between two date values.

Syntax DIFF_WEEK(date1, date2)

Arguments

date1
The first date or date expression to use in the calculation.

date2
The second date or date expression to use in the calculation.

Returns The number of weeks between date1 and date2. The function calculates the difference by subtracting the week number of date1 from the week number of date2. For example, if date1 is 1/1/10 (week 1 of the year), and date2 is 1/4/10 (week 2 of the year), DIFF_WEEK( ) returns 1.

Example The following example calculates the number of weeks between each value in the askByDate field and each value in the shipByDate field:

```
DIFF_WEEK([askByDate],[shipByDate])
```

The following example calculates the number of weeks from each value in the orderDate field to the end of the year:

```
DIFF_WEEK([orderDate], "12/31/10")
```

DIFF_YEAR( )

Calculates the number of years between two date values.

Syntax DIFF_YEAR(date1, date2)

Arguments

date1
The first date or date expression to use in the calculation.

date2
The second date or date expression to use in the calculation.

Returns The number of years between date1 and date2. The function calculates the difference by subtracting the year number of date1 from the year number of date2. For example, if date1 is 1/1/10 and date2 is 12/31/10, DIFF_YEAR( ) returns 0. If date1 is 11/25/09 and date2 is 1/5/10, DIFF_YEAR( ) returns 1.

Example The following example calculates the number of years between each value in the HireDate field and each value in the TerminationDate field:

```
DIFF_YEAR([HireDate],[TerminationDate])
```

The following example calculates the number of years from each value in the HireDate field to the current date. TODAY( ) is a function that returns the current date.

```
DIFF_YEAR([HireDate], TODAY())
```

FIND( )

Finds the location of a substring in a string.

Syntax FIND(target, source)

FIND(target, source, index)

Arguments

target
The substring to search for. The search is case-sensitive.

source
The string in which to search.

index
The position in str where the search starts.

Returns The numerical position of the substring in the string. The first character of a string starts at 1. If the substring is not found, FIND( ) returns 0.

Examples The following example searches for the substring, Ford, in each ProductName value:

```
FIND("Ford", [ProductName])
```

If the product name is 1969 Ford Falcon, FIND( ) returns 6.

The following example searches for the first hyphen (-) in each product code:

```
FIND("-", [ProductCode])
```

If the product code is ModelA-1234-567, FIND( ) returns 7.

The following example uses FIND( ) in conjunction with the LEFT( ) function to display the characters that precede the hyphen in a product code. The LEFT( ) function extracts a substring of a specified length, starting from the first character. In this example, the length of the substring to display is equal to the numerical position of the hyphen character.

```
LEFT([ProductCode], FIND("-", [ProductCode]))
```

If the product code is ModelA-1234, the expression returns the following string:

```
ModelA
```

IF( )

Returns one value if a specified condition evaluates to true, or another value if the condition evaluates to false.

Syntax IF(c, vt, vf)

Arguments

c
The condition to test.

vt
The value to return if the condition evaluates to true.

vf
The value to return if the condition evaluates to false.

Returns Returns the vt value if c is TRUE or the vf value if c is false.

Example The following example calculates and displays different discount amounts based on the value in the Total field. If the Total value is greater than 5000, the discount is 15%. Otherwise, the discount is 10%.

```
IF([Total]>5000, [Total]*15%, [Total]*10%)
```

The following example uses IF( ) in conjunction with the BETWEEN( ) and ADD_DAY( ) functions to calculate a shipment date. If an orderDate value is in December 2010 (between 12/1/10 and 12/31/10), add five days to the orderDate value. If a orderDate value is in a month other than December, add three days to the orderDate value.

```
IF(BETWEEN([orderDate], "12/1/10", "12/31/10"),
   ADD_DAY([orderDate], 5), ADD_DAY([orderDate], 3))
```

The following example checks each value in the Office field. If the value is Boston, San Francisco, or NYC, display U.S. If the value is something other than Boston, San Francisco, or NYC, display Europe and Asia Pacific.

```
IF([Office]="Boston" OR [Office]="San Francisco" OR
   [Office]="NYC", "U.S.", "Europe and Asia Pacific")
```

IN( )

Tests if a value is equal to a value in a list.

Syntax IN(source, target1,..., targetN)

Arguments

source
The value to test. The value can be a string, numeric, or date value.

target1, ..., targetN
The value or values to compare to.

Returns True if the source value is equal to one of the target values; returns false otherwise.

Example The following example tests if New Haven, Baltimore, or Cooperstown are values in the city field. If any one of the cities is in the field, IN( ) returns true.

```
IN([city], "New Haven", "Baltimore", "Cooperstown")
```

The following example tests if 9/15/08 or 9/30/08 are values in the payDate field:

```
IN([payDate], "9/15/08", "9/30/08")
```

The following example uses IN( ) in conjunction with the IF( ) function to test if Ships or Trains are values in the ProductLine field. If Ships or Trains is a value in the field, display Discontinued Item; otherwise, display the product line value as it appears in the field.

```
IF(IN([ProductLine], "Ships", "Trains"),"Discontinued Item",
   [ProductLine])
```

ISNULL( )

Tests if a value in a specified field is a null value. A null value means that no value exists.

Syntax ISNULL(source)

Argument **source**
The field in which to check for null values.

Returns True if a value in the specified field is a null value; returns false otherwise.

Example The following example uses ISNULL( ) in conjunction with the IF( ) function to test for null values in the BirthDate field. If there is a null value, display No date specified; otherwise display the BirthDate value.

```
IF(ISNULL([BirthDate]), "No date specified", [BirthDate])
```

LEFT( )

Extracts a substring from a string, starting from the left-most, or first, character.

Syntax LEFT(source)

LEFT(source, n)

Arguments **source**
The string from which to extract a substring.

n
The number of characters to extract, starting from the first character.

Returns A substring of a specific length.

- If you omit n, the number of characters to extract, the function returns the first character only.
- If n is zero, the function returns an empty string.
- If n is greater than the length of the string, the function returns the entire string.

Example The following example displays the first letter of each name in the CustomerName field:

```
LEFT([CustomerName])
```

The following example uses the LEFT( ) and FIND( ) functions to display the characters that precede the hyphen in a product code.

```
LEFT([ProductCode], FIND("-", [ProductCode]))
```

If the product code is ModelA-1234, the expression returns the following string:

```
ModelA
```

LEN( )

Counts the number of characters in a string.

Syntax LEN(source)

Argument **source**
The string expression to evaluate.

Returns The number of characters in the specified string.

Example The following example returns the length of each value in the ProductCode field:

```
LEN([ProductCode])
```

The following example uses LEN( ) in conjunction with the RIGHT( ) and FIND( ) functions to display the characters that appear after the hyphen in a product code. RIGHT( ) extracts a substring of a specified length, starting from the last character. In this example, the length of the entire string returned by LEN( ) minus the length up to the hyphen is the number of characters to display.

```
RIGHT([ProductCode], LEN([ProductCode]) - FIND("-" ,
   [ProductCode]))
```

If the product code is ModelA-Ford, the expression returns Ford.

LIKE( )

Tests if a string matches a pattern.

Syntax LIKE(source, pattern)

source
The string to evaluate.

pattern
The string pattern to match. You must enclose the pattern in double quotation marks (" "). The match is case-sensitive. You can use the following special characters in a pattern:

- A percent character (%) matches zero or more characters. For example, %ace% matches any string value that contains the substring ace, such as Facebook, and MySpace. It does not match Ace Corporation because this string contains a capital A, and not the lowercase a.
- An underscore character (_) matches exactly one character. For example, t_n matches tan, ten, tin, and ton. It does not match teen or tn.

To match a literal percent (%), underscore (_), precede those characters with two backslash (\\) characters. For example, to see if a string contains M_10, specify the following pattern:

```
"%M\\_10%"
```

Returns True if the string matches the pattern; returns false otherwise.

Example The following example returns true for values in the customerName field that start with D:

```
LIKE([customerName], "D%")
```

The following example returns true for productCode values that contain the substring Ford:

```
LIKE([productCode], "%Ford%")
```

The following example uses two LIKE( ) expressions to look for the substrings “Ford” or “Chevy” in each ProductName value. If a product name contains either substring, the expression displays U.S. Model; otherwise, it displays Imported Model.

```
IF(((LIKE([ProductName], "%Ford%") = TRUE) OR (LIKE([ProductName],
   "%Chevy%") = TRUE)), "U.S. model", "Imported Model")
```

LOWER( )

Converts all letters in a string to lowercase.

Syntax LOWER(source)

Argument

source
The string to convert to lowercase.

Returns The specified string in all lowercase letters.

Example The following example displays all the string values in the productLine field in lowercase:

```
LOWER([productLine])
```

MATCH( )

Tests if a string matches a pattern. The pattern must use JavaScript regular expression syntax.

Syntax MATCH(source, pattern)

Arguments

source
The string to evaluate.

pattern
The string pattern to match. You must enclose the pattern in quotation marks (" "). In JavaScript regular expression syntax, a pattern is enclosed within a pair of forward slash (/) characters. However, for this argument, the forward slash characters are optional. For example, the following values are equivalent:

```
"smith"
"/smith/"
```

You can use any special character supported by JavaScript regular expressions, such as the following:

- A question mark (?) matches zero or one occurrence of the character previous to it. For example, "te?n" matches tn, ten, and often. It does not match teen or intern.
- An asterisk (*) matches zero or any number of occurrences of the character precious to it. For example, "te*n" matches tn, ten, often, and teen. It does not match intern.
- A period (.) matches any character. For example, "te.*" matches ten, often, teen, and intern.

- A caret (^) specifies that the pattern to look for is at the beginning of a string. For example, "^ten" matches ten, tennis, and tense. It does not match often or pretend.
- An i character specifies a case-insensitive search. For example, "/smith/i" matches Smith, blacksmith, and Smithsonian. In this case, the pair of forward slashes is required.

To match a special character literally, precede the special character with two backslash (\\) characters. For example, to check if a string contains S*10, specify the following pattern:

```
"/S\\*10/"
```

Returns True if the string matches the pattern; returns false otherwise.

Examples The following example returns true for values in the ProductCode field that start with S18:

```
MATCH([ProductCode], "/^S18/")
```

The following example uses MATCH( ) to check if the values in the SKU field contain the letters EM followed by a number that ends with 99. If there is a match, display Discontinued; otherwise, display the SKU value.

```
IF(MATCH([SKU], "/EM.*99/"), "Discontinued", [SKU])
```

MOD( )

Returns the remainder after a number is divided by another.

Syntax MOD(number, divisor)

Arguments

number
The number to divide.

divisor
The number by which to divide the number value. You must specify a non-zero number.

Returns The remainder after the number value is divided by the divisor value. Different applications and programming languages define the modulo operation differently when either the dividend or the divisor are negative. For example, in EasyScript and Excel, MOD(-5, 3) returns 1. However, in JavaScript and most databases, the modulo operation returns -2.

Examples The following examples shows the results that the function returns for specific numbers:

```
MOD(10, 5) // returns 0
MOD(11, 5) // returns 1
MOD(12, 5) // returns 2
MOD(-10, 5) //returns 0
MOD(-11, 5) //returns 4
MOD(-12, 5) //returns 3
MOD(10, -5) //returns 0
MOD(11, -5) //returns -4
MOD(12, -5) //returns -3
```

The following example uses MOD( ) to check if numbers in the Grade field are odd or even. When the divisor is 2, MOD( ) returns 0 for even numbers, and 1 for odd numbers.

```
MOD([Grade], 2)
```

The following example uses MOD( ) and YEAR( ) to get the last digit of a year. YEAR( ) returns the year number of a date. Dividing a number by 10 returns the last digit of the number.

```
MOD(YEAR([BirthDate]), 10)
```

MONTH( )

Returns the month for a specified date value.

Syntax MONTH(date)

MONTH(date, option)

Arguments

date
The date or date expression whose month to get.

option
A number that represents the month format to return. Use one of the following values:

- 1 to get the month as a number from 1 to 12.
- 2 to get the full month name, for example, January. The result is locale-specific.
- 3 to get the abbreviated month name, for example, Jan. The result is locale-specific.

If you omit option, MONTH( ) returns the month as a number.

Returns The month for a specified date value.

Example The following example returns the month (1 - 12) for each value in the ShipDate field:

```
MONTH([ShipDate])
```

The following example returns the full month name for each ShipDate value:

```
MONTH([ShipDate], 2)
```

NOT( )

Negates a Boolean expression.

Syntax NOT(x)

Argument **x**
The Boolean value or expression to negate.

Returns True if the expression evaluates to false, and false if the expression evaluates to true.

Example The following example uses NOT( ) in conjunction with the IF( ) function. It tests if the value in the State field is not CA. If the value is not CA, it returns the value in the Markup field multiplied by 10%, and by 15% if it is.

```
IF(NOT([State]="CA"),[Markup]*10%,[Markup]*15%)
```

The previous IF( ) expression is semantically equivalent to the following expression:

```
IF([State]="CA",[Markup]*15%,[Markup]*10%)
```

NOTNULL( )

Tests if a value in a specified field is a non-null value.

Syntax NOTNULL(source)

Argument **source**
The field in which to check for non-null values.

Returns True if a value in the specified field is not a null value; returns false otherwise.

Example The following example uses NOTNULL( ) in conjunction with the IF( ) function to test for non-null values in the BirthDate field. If there is a non-null value, display the BirthDate value; otherwise display No date specified.

```
IF(NOTNULL([BirthDate]), [BirthDate], "No date specified")
```

NOW( )

Returns the current date and time.

Syntax NOW( )

Returns The current date and time. For example:

```
Feb 10, 2010 2:55 PM
```

Example The following example uses the DIFF_DAY( ) and NOW( ) functions to calculate the number of days from the current date and time to Christmas:

```
DIFF_DAY(NOW(), "12/25/10")
```

QUARTER( )

Returns the quarter number for a specified date value.

Syntax QUARTER(date)

Arguments

date
The date or date expression whose quarter number to get.

Returns A number from 1 to 4 that represents the quarter for a specified date value. Quarter 1 starts in January.

Examples The following example displays the quarter number for each value in the CloseDate field:

```
QUARTER([CloseDate])
```

The following example displays a string—Q1, Q2, Q3, or Q4—for each value in the CloseDate field:

```
"Q" & QUARTER([CloseDate])
```

RIGHT( )

Extracts a substring from a string, starting from the right-most, or last, character.

Syntax RIGHT(source)

RIGHT(source, n)

Arguments

source
The string from which to extract a substring.

n
The number of characters to extract, starting from the last character.

Returns A substring of a specific length.

- If you omit n, the number of characters to extract, the function returns the last character only.
- If n is zero, the function returns an empty string.
- If n is greater than the length of the string, the function returns the entire string.

Example The following example displays the last four characters of each value in the ProductCode field:

```
RIGHT([ProductCode], 4)
```

The following example uses RIGHT( ) in conjunction with the LEN( ) and FIND( ) functions to display the characters that appear after the hyphen in a product code. This example assumes that the number of characters after the hyphen varies. Therefore, the length of the entire string (returned by LEN( )) minus the length up to the hyphen (returned by FIND( )) is the number of characters to display.

```
RIGHT([ProductCode], LEN([ProductCode]) - FIND("-" ,
   [ProductCode]))
```

If the product code is ModelA-Ford, the expression returns Ford. If the product code is ModelCZ15-Toyota, the expression returns Toyota.

ROUND( )

Rounds a number to a specified number of digits.

Syntax ROUND(number)

ROUND(number, dec)

Arguments

number
The number to round.

dec
The number of digits to round number to. If you omit dec, ROUND( ) assumes 0.

Returns A number rounded to a specified number of digits.

Example The following example rounds the numbers in the PriceEstimate field to return an integer. For example, if the PriceEstimate value is 1545.50, ROUND( ) returns 1546. If the PriceEstimate value is 1545.25, ROUND( ) returns 1545.

```
ROUND([PriceEstimate])
```

The following example rounds the numbers in the PriceEstimate field to one decimal place. For example, if the PriceEstimate value is 1545.56, ROUND( ) returns 1545.6. If the PriceEstimate value is 1545.23, ROUND( ) returns 1545.2.

```
ROUND([PriceEstimate], 1)
```

The following example rounds the numbers in the PriceEstimate field to one digit to the left of the decimal point. For example, if the PriceEstimate value is 1545.56, ROUND( ) returns 1550. If the PriceEstimate value is 1338.50, ROUND( ) returns 1340.

```
ROUND([PriceEstimate], -1)
```

ROUNDDOWN( )

Rounds a number down to a specified number of digits.

Syntax ROUNDDOWN(number)

ROUNDDOWN(number, dec)

Arguments

number
The number to round down.

dec
The number of digits to round number down to. If you omit dec, ROUND( ) assumes 0.

Returns A number rounded down to a specified number of digits.

Example The following example rounds down the numbers in the PriceEstimate field to return an integer. For example, if the PriceEstimate value is 1545.25, ROUNDDOWN( ) returns 1545. If the PriceEstimate value is 1545.90, ROUNDDOWN( ) returns 1545.

```
ROUNDDOWN([PriceEstimate])
```

The following example rounds down the numbers in the PriceEstimate field to one decimal place. For example, if the PriceEstimate value is 1545.56, ROUNDDOWN( ) returns 1545.5. If the PriceEstimate value is 1545.23, ROUNDDOWN( ) returns 1545.2.

```
ROUNDDOWN([PriceEstimate], 1)
```

The following example rounds the numbers in the PriceEstimate field down to one digit to the left of the decimal point. For example, if the PriceEstimate value is 1545.56, ROUNDDOWN( ) returns 1540. If the PriceEstimate value is 1338.50, ROUNDDOWN( ) returns 1330.

```
ROUNDDOWN([PriceEstimate], -1)
```

ROUNDUP( )

Rounds a number up to a specified number of digits.

Syntax ROUNDUP(number)

ROUNDUP(number, dec)

Arguments

number
The number to round up.

dec
The number of digits to round number up to. If you omit dec, ROUND( ) assumes 0.

Returns A number rounded up to a specified number of digits.

Example The following example rounds up the numbers in the PriceEstimate field to return an integer. For example, if the PriceEstimate value is 1545.25, ROUNDUP( ) returns 1546. If the PriceEstimate value is 1545.90, ROUNDUP( ) returns 1546.

```
ROUNDUP([PriceEstimate])
```

The following example rounds up the numbers in the PriceEstimate field to one decimal place. For example, if the PriceEstimate value is 1545.56, ROUNDUP( ) returns 1545.6. If the PriceEstimate value is 1545.23, ROUNDUP( ) returns 1545.3.

```
ROUNDUP([PriceEstimate], 1)
```

The following example rounds up the numbers in the PriceEstimate field to one digit to the left of the decimal point. For example, if the PriceEstimate value is 1545.56, ROUNDUP( ) returns 1550. If the PriceEstimate value is 1338.50, ROUNDUP( ) returns 1340.

```
ROUNDUP([PriceEstimate], -1)
```

SEARCH( )

Finds the location of a substring in a string. The substring can contain wildcard characters.

Syntax SEARCH(pattern, source)

SEARCH(pattern, source, index)

Arguments

pattern
The string pattern to search for. You must enclose the pattern in double quotation marks (" "). You can use the following special characters in a pattern:

- An asterisk (*) matches zero or more characters, including spaces. For example, t*n matches tn, tin, and teen.
- A question mark (?) matches exactly one character. For example, t?n matches tan, ten, tin, and ton. It does not match teen or tn.

source
The string in which to search.

index
The position in source where the search starts.

Returns The numerical position of the string pattern in the string. The first character of a string starts at 1. If the substring is not found, SEARCH( ) returns 0.

Examples The following example searches for the string pattern, S*A, in each product code. If the product name is KBS5412A, SEARCH( ) returns 3.

```
SEARCH("S*A", [ProductCode])
```

The following example uses SEARCH( ) in conjunction with the LEFT( ) function to display the characters that precede the first space character in a product name. The LEFT( ) function extracts a substring of a specified length, starting from the first character. In this example, the length of the substring to display is equal to the numerical position of the space character.

```
LEFT([ProductName], SEARCH(" ", [ProductName]))
```

If the product name is 1969 Ford Falcon, the expression returns 1969.

SQRT( )

Calculates the square root of a number.

Syntax SQRT(number)

Argument **number**
The number for which you want to find the square root. The number must be a positive number.

Returns A number that is the square root of the specified number.

Examples The following example calculates the square root of each numeric value in the LotSize field:

```
SQRT([LotSize])
```

The following example uses SQRT( ) to calculate the actual distance travelled uphill, given the base distance and elevation values. This example applies the Pythagorean theorem, which states that $a^2 + b^2 = c^2$. Using this theorem, the actual distance traveled is c, which means we want to calculate

$c = \sqrt{a^2 + b^2}$

which translates to the following expression:

```
SQRT((([Distance] * [Distance]) + ([Elevation] * [Elevation])))
```

TODAY( )

Returns the current date that includes a time value of midnight, 12:00 AM.

Syntax TODAY( )

Returns The current date in the following format:

```
Feb 11, 2010 12:00 AM
```

Examples The following example calculates the number of days from the current date to Christmas:

```
DIFF_DAY(TODAY(), "12/25/10")
```

The following example calculates the number of years from each value in the HireDate field to the current date:

```
DIFF_YEAR([HireDate], TODAY())
```

TRIM( )

Removes the leading and trailing blanks from a specified string. TRIM( ) does not remove blank characters between words.

Syntax TRIM(source)

Argument **source**
The string from which to remove leading and trailing blank characters.

Returns A string with all leading and trailing blank characters removed.

Example The following example uses TRIM( ) to remove all leading and trailing blank characters from values in the FirstName and LastName fields. The expression uses the & operator to concatenate each trimmed FirstName value with a space, then with each trimmed LastName value.

```
TRIM([FirstName]) & " " & TRIM([LastName])
```

TRIMLEFT( )

Removes the leading blanks from a specified string.

Syntax TRIMLEFT(source)

Argument **source**
The string from which to remove the leading blank characters.

Returns A string with all leading blank characters removed.

Example The following example concatenates a literal string with each value in the customerName field. TRIMLEFT( ) removes all blank characters preceding the customerName value so that there are no extra blank characters between the literal string and the customerName value.

```
"Customer name: " & TRIMLEFT([customerName])
```

TRIMRIGHT( )

Removes the trailing blanks from a specified string.

Syntax TRIMRIGHT(source)

Argument **source**
The string from which to remove the trailing blank characters.

Returns A string with all trailing blank characters removed.

Example The following example concatenates each value in the Comment field with a semicolon, then with a value in the Action field. TRIMRIGHT( ) removes all blank characters after the Comment value so that there are no extra blank characters between the Comment string and the semicolon.

```
TRIMRIGHT([Comment]) & "; " & [Action]
```

UPPER( )

Converts all letters in a string to uppercase.

Syntax UPPER(source)

Argument **source**
The string to convert to uppercase.

Returns The specified string in all uppercase letters.

Example The following example displays all the string values in the customerName field in all uppercase:

```
UPPER([customerName])
```

WEEK( )

Returns a number from 1 to 52 that represents the week of the year.

Syntax WEEK(date)

Argument **date**
The date or date expression whose week of the year to get.

Returns A number that represents the week of the year for the specified date value.

Example The following example gets the week number of the year for each date value in the ShipDate field:

```
WEEK([ShipDate])
```

WEEKDAY( )

Returns the day of the week for a specified date value.

Syntax WEEKDAY(date, option)

Arguments **date**
The date or date expression from which you want to get the day of the week.

option
A number that represents the weekday format to return. Use one of the following values:

- 1 to get the day as a number from 1 (Sunday) to 7 (Saturday).
- 2 to get the day as a number from 1 (Monday) to 7 (Sunday).
- 3 to get the day as a number from 0 (Monday) to 6 (Sunday).
- 4 to get the full weekday name, for example, Wednesday. The result is locale-specific.
- 5 to get the abbreviated weekday name, for example Wed. The result is locale-specific.

If you omit option, WEEKDAY( ) assumes option 1.

Returns The day of the week for a specified date value.

Example The following example gets the full weekday name for each date value in the DateSold field:

```
WEEKDAY([DateSold], 4)
```

YEAR( )

Returns the four-digit year value for a specified date value.

Syntax YEAR(date)

date
The date or date expression from which you want to extract the year part.

Returns The number that represents the four-digit year for the specified date value.

Example The following example gets the four-digit year for each date value in the ShipDate field, and adds 15 to the four-digit year. For example, if the ShipDate value is Sep 16, 2008, YEAR( ) returns 2023.

```
(YEAR([ShipDate]) + 15)
```

Operators

Table 13-1 lists the operators in EasyScript.

Table 13-1 EasyScript operators

Operator	Use to	Example
+	Add two or more numeric values together	[OrderAmount] + [SalesTax]
-	Subtract one numeric value from another	[OrderAmount] - [Discount]
*	Multiply numeric values	[Price] * [Quantity]
/	Divide numeric values	[Profit]/12
^	Raise a numeric value to a power	[Length]^2
%	Specify a percent	[Price] * 80%
=	Test if two values are equal	IF([ProductName] = "1919 Ford Falcon", "Discontinued Item", [ProductName])
>	Test if one value is greater than another value	IF([Total] > 5000, [Total]*15%, [Total]*10%)
<	Test if one value is less than another value	IF([SalePrice] < [MSRP], "Below MSRP", "Above MSRP")
>=	Test if one value is greater than or equal to another value	IF([Total] >= 5000, [Total]*15%, [Total]*10%)
<=	Test if one value is less than or equal to another value	IF([SalePrice] <= [MSRP], "Below or equal to MSRP", "Above MSRP")
<>	Test if two values are not equal	IF([Country] <> "USA", "Imported product", "Domestic product")
AND	Test if two or more conditions are true	IF(([Gender] = "Male" AND [Salary] >= 150000 AND [Age] < 50), "Match found", "No match")

Table 13-1 EasyScript operators (continued)

Operator	Use to	Example
OR	Test if any one of multiple conditions is true	IF(([City] = "Boston") OR ([City] = "San Francisco"), "U.S.", "Europe and Asia")
&	Concatenate string values	[FirstName] & " " & [LastName]

YEAR( )

Chapter

14

Specifying filter conditions at report run time

This chapter contains the following topics:

- About report parameters and filters
- Enabling the user to specify a filter condition
- Getting information about queries

About report parameters and filters

Report parameters provide a mechanism for collecting values from a report user or a program. They are typically used in filters to collect information that determines the data to display in a report. Actuate BIRT Designer supports all the functionality of parameters and filters available in the open-source version, and provides additional features.

In open-source BIRT Report Designer, the following expression in the filter tool is an example of how a filter uses a report parameter to obtain the filter value at run time:

```
row["Total"] Greater than or Equal params[Sales Total].value
```

The field to evaluate (row["Total"]) and the operator that determines the type of filter test (Greater than or Equal) are specified at design time. At run time, the report user supplies the parameter value, which, in this example, is a sales total, such as 10000.

In Actuate BIRT Designer, report parameters and filters are enhanced to support dynamic filter conditions, which provide users more control over what data they see in the report. Instead of specifying only the value on which to filter, the report user can specify conditions, such as Total Less than 10000, or Total Between 10000 and 20000, or Total Greater than 20000. The user can also choose to view all totals; in other words, the user can choose to omit the filter condition.

Another enhancement is that these filters can modify the underlying query so that filtering occurs in the database. This functionality applies when accessing a database through an information object or a JDBC connection for query builder data source. When using these data source types, only data rows that meet the filter criteria are retrieved from the database. By retrieving a limited number of rows, Actuate BIRT Designer's performance improves.

This chapter describes how to create report parameters and filters to enable dynamic filtering. For information about other types of parameters and filters, see *BIRT: A Field Guide*.

Enabling the user to specify a filter condition

To enable users to specify a filter condition, complete the following tasks in the recommended order:

- Create a dynamic filter report parameter.
- Create a dynamic filter and bind it to the report parameter.

Creating a dynamic filter report parameter

A dynamic filter report parameter differs from a regular report parameter in one important aspect. Using a dynamic filter parameter, you can provide report users with a list of operators, which they can use to construct their own filter condition.

Figure 14-1 shows an example definition of a dynamic filter parameter where the display type is a text box. Aside from the Dynamic Filter Condition section, where you specify the column on which to filter and the operators to provide to users, the properties are similar to the properties for a regular report parameter.

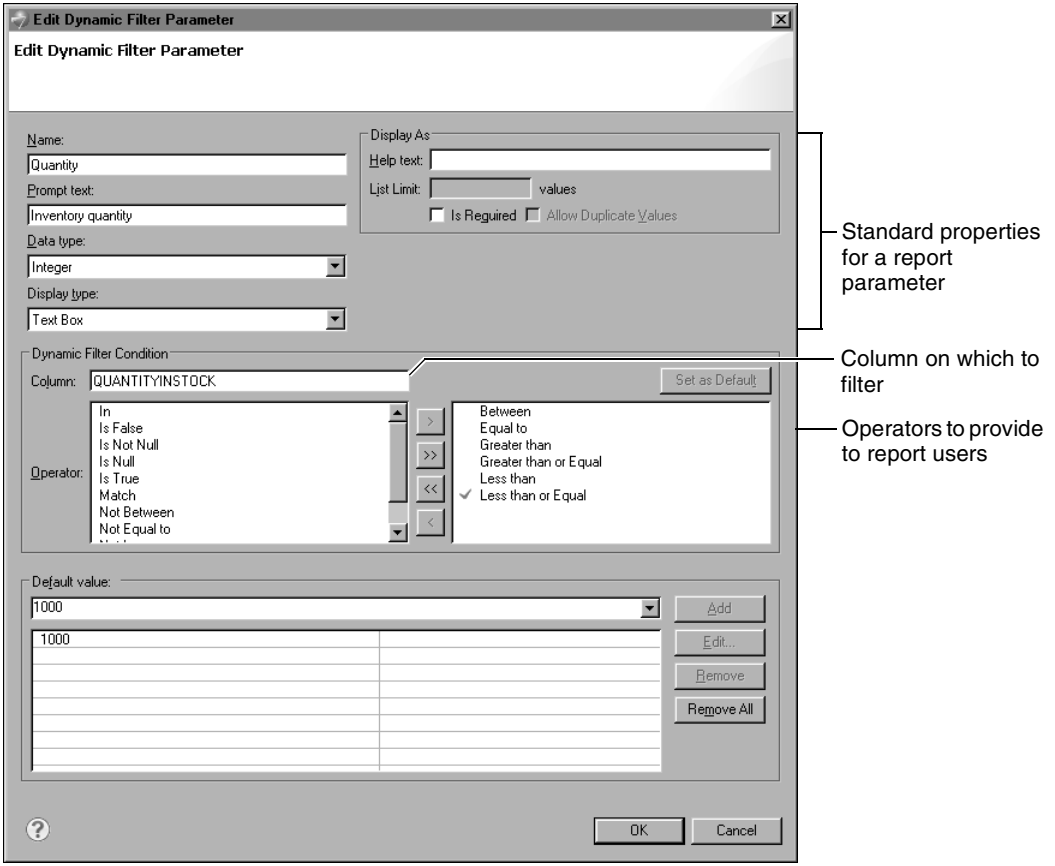


Figure 14-1 Properties of a dynamic filter parameter whose display type is a text box

Like the regular report parameter, a dynamic filter parameter can also provide the user with a list of values. However, values can be presented in a combo box or list box only. Figure 14-2 shows an example definition of a dynamic filter parameter that includes a list of values. The display type is set to Combo/List Box.

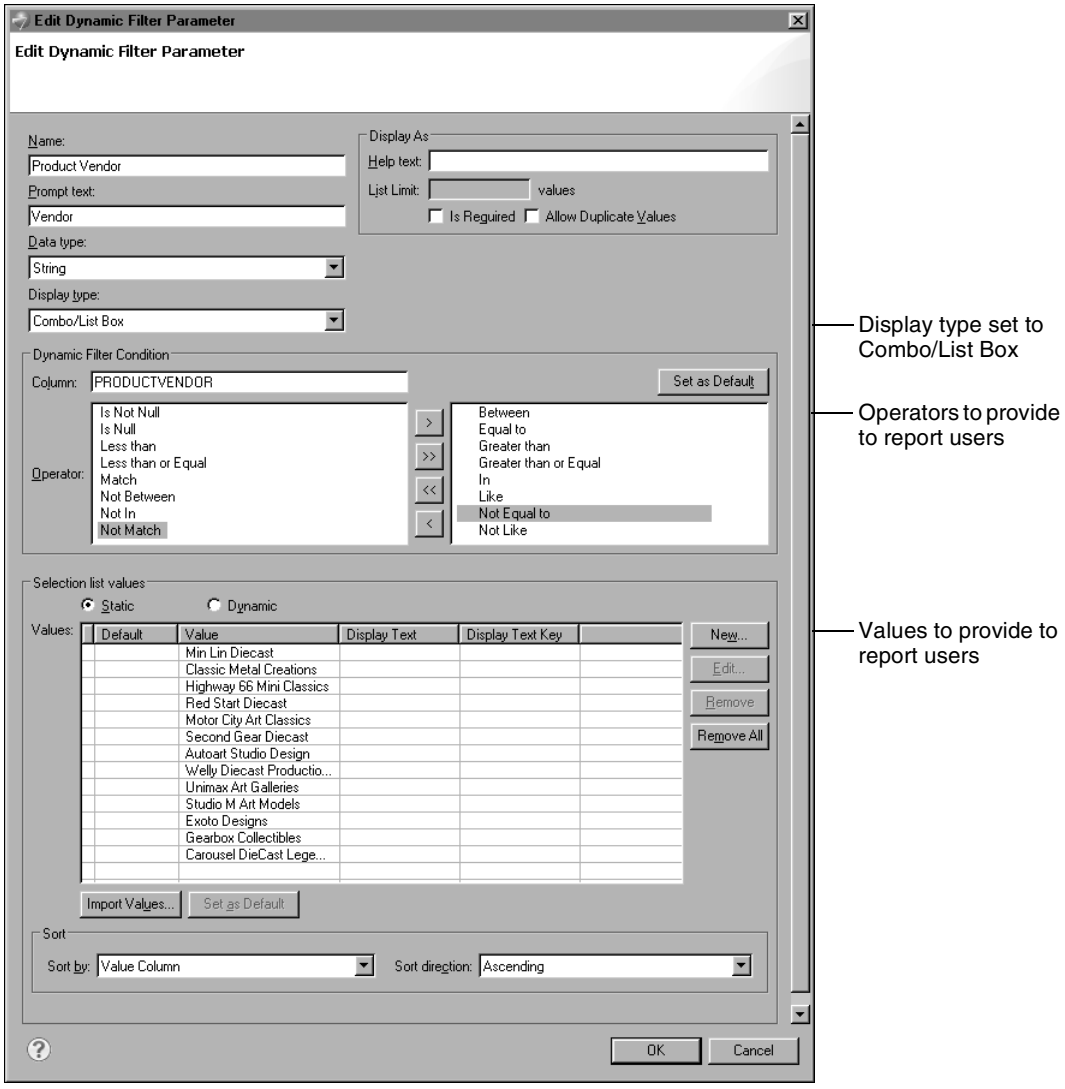


Figure 14-2 Properties of a dynamic filter parameter whose display type is a combo box or list box

How to create a dynamic filter report parameter

1. In Data Explorer, right-click Report Parameters, and choose New Dynamic Filter Parameter.
2. Specify the properties of the dynamic filter parameter. For information about the standard properties for a report parameter, see *BIRT: A Field Guide*. The

following Dynamic Filter Condition properties are specific to a dynamic filter parameter:

- In Column, type the name of the field on which to filter.
- In Operator, select the operators to provide to the user. By default, all the operators are selected. To remove an operator, select it and click <.
- Optionally, set one of the operators as the default. Select the operator, then choose Set as Default. A check mark appears next to the operator. If you specify a default operator, you must also specify a value in Default Value.

3 Choose OK.

Making a filter parameter optional

When you create a dynamic filter parameter, you can require the user to specify a value or you can make the filter optional. It is usually good practice to make the filter optional, so that the user can view a report with all the data. For example, if a report displays inventory data by vendor and you create an optional parameter to filter on vendors, the user can select No Condition to view inventory data for all vendors.

On the other hand, you can require that the user specify a value if displaying all the data results in a very long report. A report that runs into hundreds of pages is not only difficult to read, but the report takes longer to generate.

To make a filter parameter optional, deselect the Is Required property.

Accepting multiple values

Users often want to select any number of values for a filter condition. In an inventory report, for example, the user might need to view data for several vendors. To support the selection of multiple values, create a dynamic filter parameter as follows:

- Select Combo/List Box as the display type.
- Select the In operator as one of the operators to provide to the user.
- Create a list of values.

Creating a dynamic filter

Actuate BIRT Designer supports two types of filters: static and dynamic. Use a static filter to define a specific filter condition at design time. Use a dynamic filter to enable users to define a filter condition at report run time.

How to create a dynamic filter

This procedure assumes you have already created the dynamic filter report parameter to bind to the filter you are creating.

1 Select the element to which to apply a dynamic filter condition. For example, select a data set, a table, or a chart whose data you want to filter.

2 Choose Filters, then choose New or Add to define a filter.

3 In New Filter Condition, specify the following values:

   1 Choose Dynamic.

   2 In Column, select the field on which to filter.

   3 In Filter Parameter, select the dynamic filter parameter to update this filter with user-specified values at run time.

Figure 14-3 shows an example in which a QUANTITYINSTOCK field is bound to a dynamic filter parameter named Quantity.

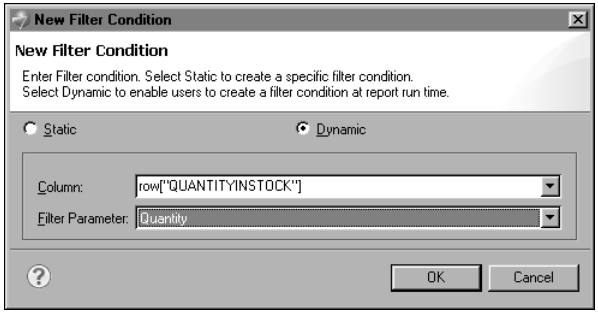


Figure 14-3 Definition of a dynamic filter condition

4 Choose OK.

The filter appears on the Filters page. Figure 14-4 shows the Quantity dynamic filter on the Filter page of the data set editor. Unlike a static filter, no values appear under Operator, Value 1, or Value 2, indicating that these values are specified at run time.

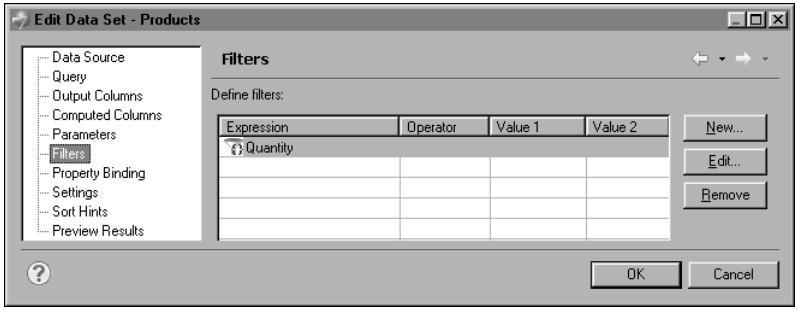


Figure 14-4 Dynamic filter in a data set

Getting information about queries

When a report accesses data from a database, it is useful to understand what queries the report sends to the database, and how charts and tables get their data. For example, if you create a dynamic filter on a table to display sales data for certain products only, does BIRT send a query to retrieve sales data for all products then filter at the table level to display data for specific products, or does BIRT send a query that retrieves only data for specific products? Answers to questions such as this can help you optimize the performance of a report.

To get information about the queries that are executed, right-click a report element, such as a table or a chart, then choose Show Query Execution Profile. Figure 14-5 shows an example of a query that is executed for a table. In this example, Query Execution Profile shows the following information:

- The data set (Products Data Set) that is bound to the table, the original query specified, and the query modified by BIRT and sent to the database
- A sort definition that sorts data rows by product name in ascending order
- A filter condition (row["QUANTITYINSTOCK"] > 5000
- A group definition that groups data by vendor
- Data bindings associated with the table

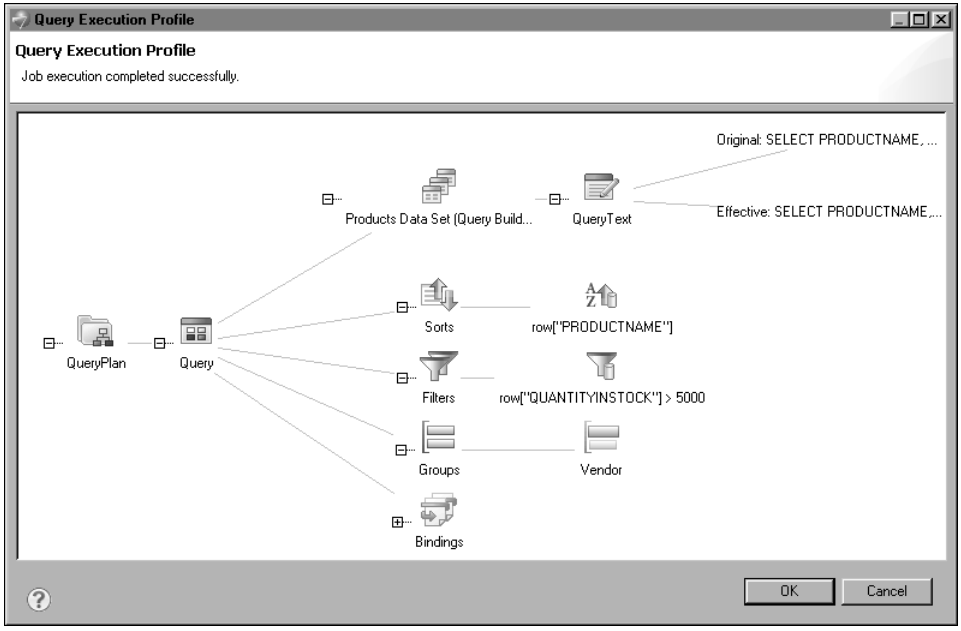


Figure 14-5 Query execution profile for a table

Select each item in the query execution profile to see more information about that item. For example, click the filter, as shown in Figure 14-6, to see whether the filter is executed in BIRT or at the database level. In the filter information, "Push Down: applied" means that the filter is pushed down to, or executed by, the database. Similarly, select the sort and group definitions to see where these tasks are executed.

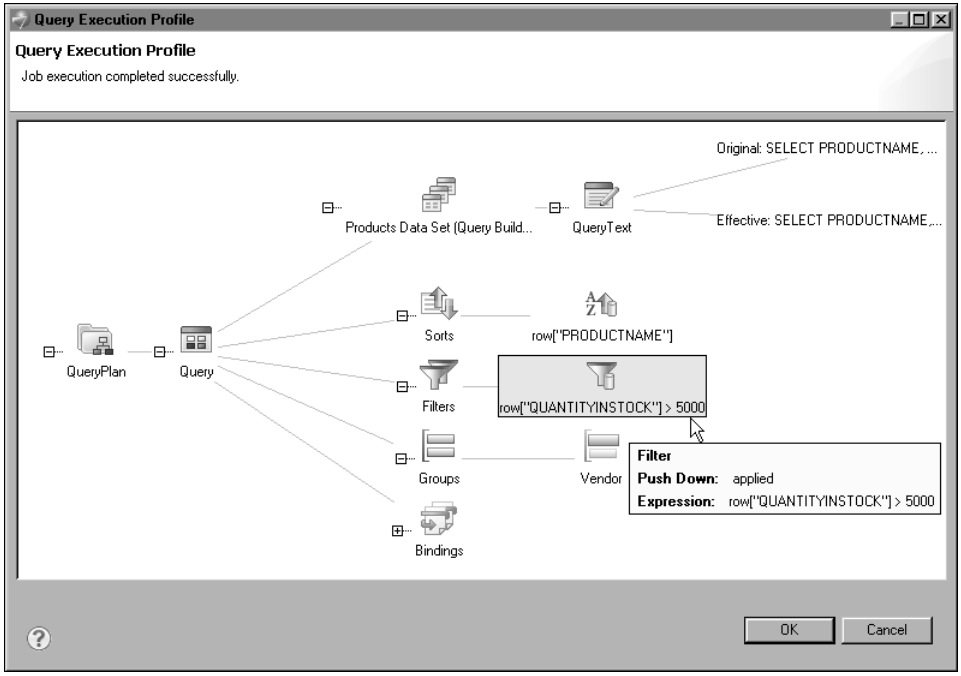


Figure 14-6 Filter information displayed in the query execution profile

Another piece of useful information that the query execution profile provides is whether, and how, BIRT modifies a query when you sort, group, or filter data using the graphical tools. As discussed at the beginning of this chapter, BIRT can modify a query to perform these tasks at the database level if the report accesses the database through an information object or a JDBC connection for query builder data source.

Select the Original: SELECT statement to see the query specified originally. Select the Effective: SELECT statement to see the query modified by BIRT. Figure 14-7 shows an example of the SELECT statement in the original query. Figure 14-8 shows an example of the SELECT statement in the modified query.

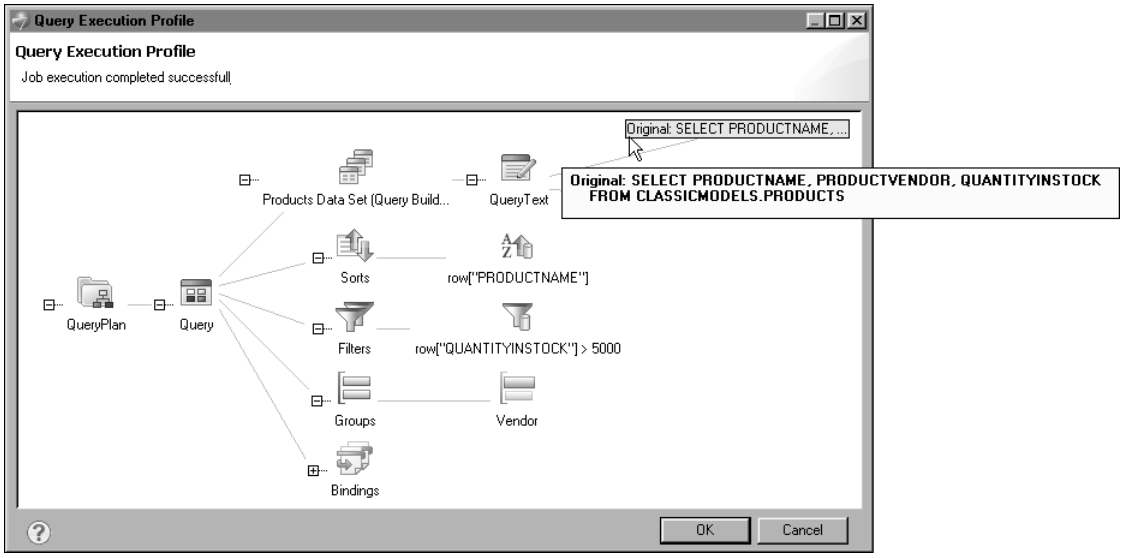


Figure 14-7 Original query displayed in the query execution profile

As Figure 14-8 shows, BIRT changes the original query to add a filter condition (WHERE clause) and a sort condition (ORDER BY clause).

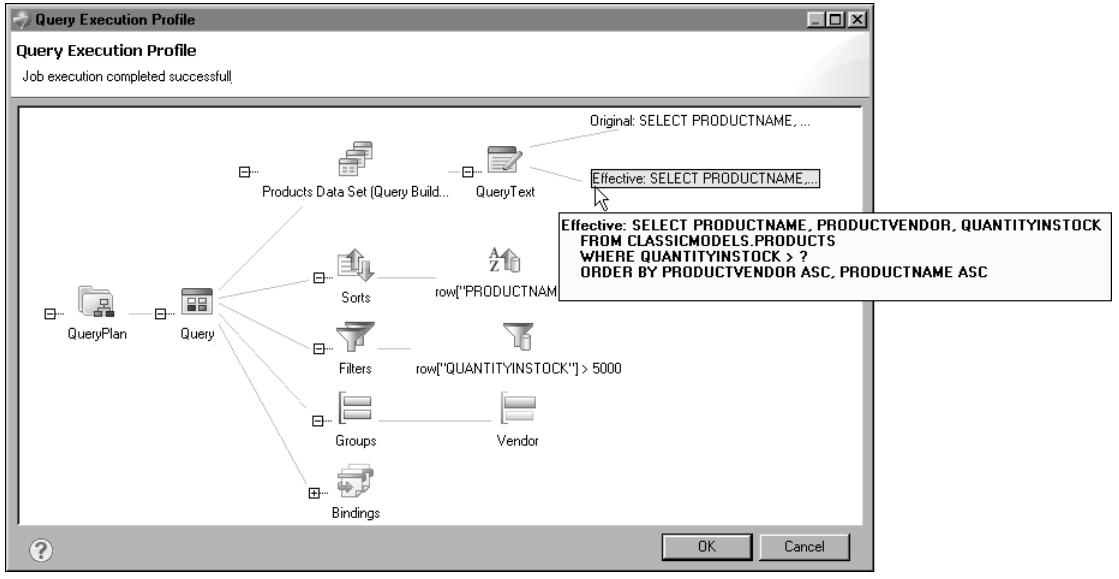


Figure 14-8 Modified (effective) query displayed in the query execution profile

The performance of a report improves when data is processed by the database rather than by BIRT. Data filtering in particular can affect performance significantly because filtering can mean the difference between retrieving hundreds or millions of rows of data.

When you create filters using the graphical filter tool, BIRT pushes a filter to the database if the filter condition can be mapped to a SQL expression (if using the JDBC connection for query builder data source) or an Actuate SQL expression (if using an information object data source). Using that criterion, the following are examples of when BIRT pushes a filter to the database:

- The filter uses an operator that is supported by the database, for example, <, >, =.

  BIRT-specific operators, such as Match, Top Percent, and Bottom Percent, do not have SQL equivalents, so a filter that uses any of these operators is not pushed to the database.

- The filter uses an expression that refers to a field in a database table. For example, the following filter condition is pushed to the database if SalesTotal is a column in the database table:

  ```
  row["SalesTotal"] Greater than 5000000
  ```

  On the other hand, the following filter condition is not pushed to the database if Profit is a computed column derived from other columns, for example, row["Sales"] - row["Cost"]:

  ```
  row["Profit"] Greater than 2000000
  ```

Chapter

15

Adding HTML buttons to a report

This chapter contains the following topics:

- About HTML buttons
- Creating an HTML button
- Writing code for an HTML button
- Changing the appearance of an HTML button

About HTML buttons

In a BIRT report, an HTML button is a report element that provides the same functionality as a button defined with the HTML <button> tag in a web page. The HTML button can execute client-side JavaScript code associated with button events, such as a button click or double-click.

You can use HTML buttons to provide users with custom interactive reporting functionality. For example, you can create HTML buttons that, when clicked, filter data, hide or show data, sort data, link to another report, or perform calculations.

Figure 15-1 shows Actuate Viewer displaying a product sales report that contains three buttons at the top. Each button provides a different data filtering option. The user can choose to view all product sales, the top ten products, or the bottom ten products. The report in Figure 15-1 shows the top ten products.

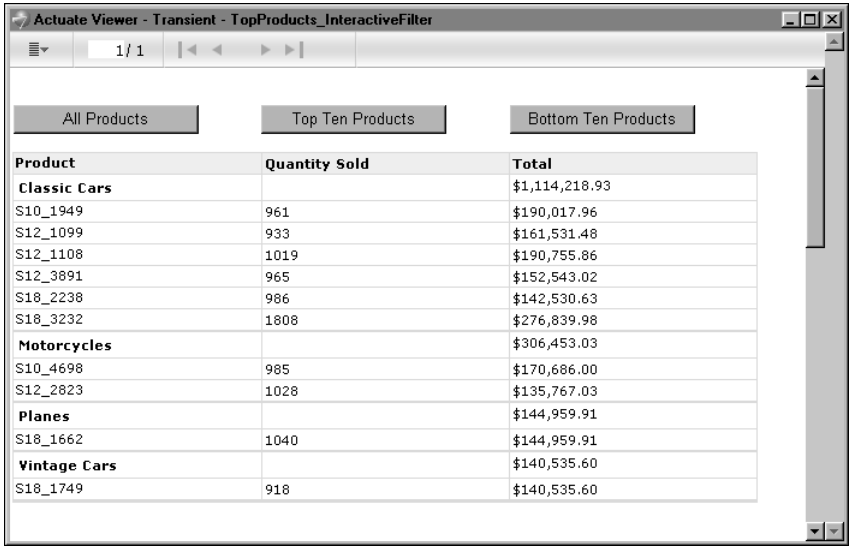


Figure 15-1 Report with HTML buttons that provide different data filtering options

You can also use HTML buttons to integrate a report with other enterprise applications. Figure 15-2 shows an example of a report that uses Check Inventory and Process Order buttons to link to business processes that run in a different application.

The HTML button is supported in HTML reports only. It does not work in other output formats, such as PDF or Excel, and appears as static text in those documents. If a report is to be viewed in formats other than HTML, use the Visibility property to hide HTML buttons in all output formats, except HTML.

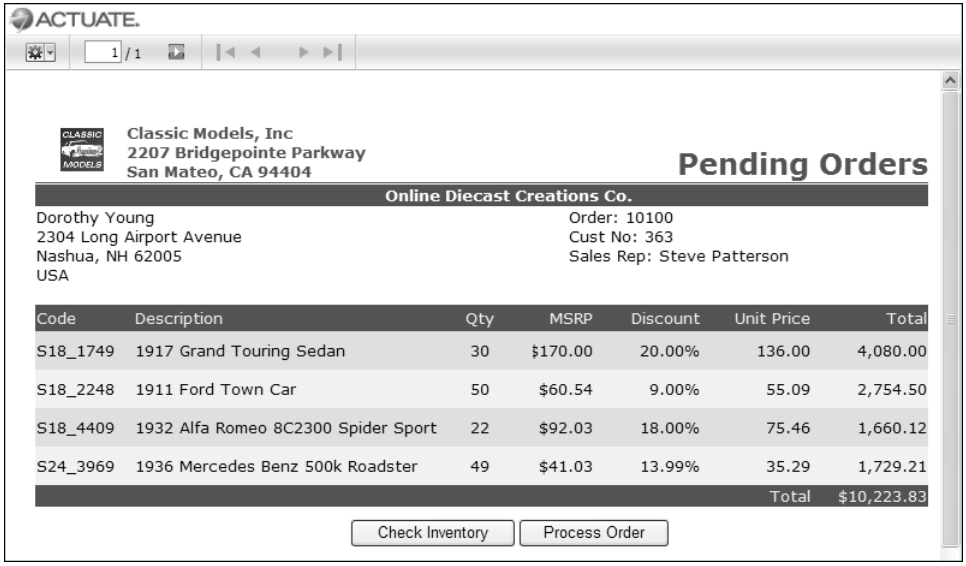


Figure 15-2 Report with HTML buttons that link to business processes

Creating an HTML button

Creating a functional HTML button entails inserting the HTML button element in the desired location in the report, specifying the text to display on the button, then programming the button's action. You can place an HTML button in the report page, a grid, table, list, and cross tab.

How to create an HTML button

1. Drag an HTML button element from the palette and drop it in the desired location in the report.
2. In HTML Button, specify the following values:
    1. In Name, type a different name for the element if you do not want to use the default name. Each HTML button must have a unique name.
    2. In Value, type the text to display on the button. Alternatively, select JavaScript Syntax or EasyScript Syntax to create an expression, which displays a dynamic or calculated value. Figure 15-3 shows an example of text specified for Value.

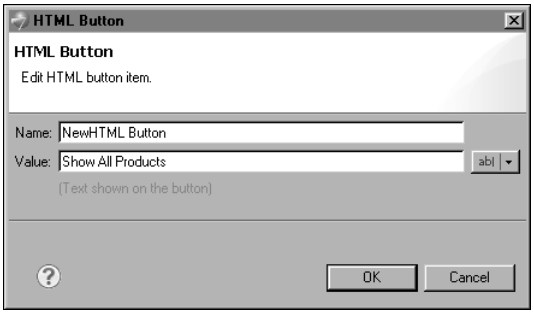


Figure 15-3 Definition of an HTML button

3 Choose OK. A message appears, providing information about writing code for the button. Choose OK.

The HTML button appears in the report.

3 While the button is selected, choose the Script tab.

4 In the script editor, click New Event Function and select a button event from the drop-down list, shown in Figure 15-4.

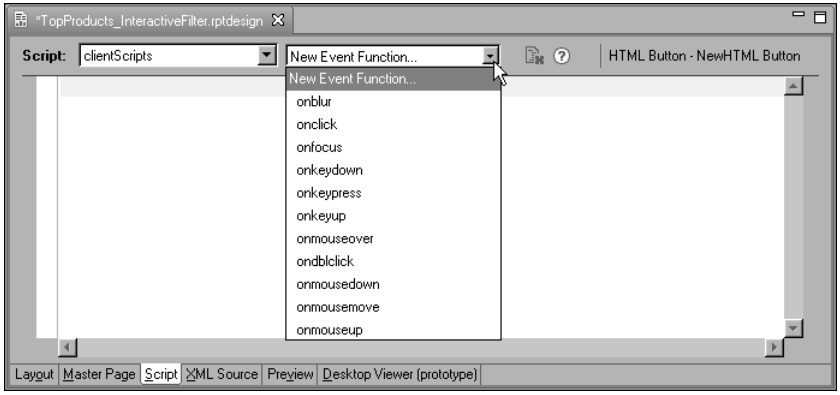


Figure 15-4 Click New Event Function to display the list of button events

5 Write JavaScript code to program the button's action for the selected event. The next section provides information about this task.

6 Run the report in the web viewer to test the button's functionality. If you do not receive expected results, or if you receive an error, check the event handler script for possible problems.

Writing code for an HTML button

After inserting an HTML button in a report, you use the script editor to write JavaScript code that specifies the task to perform when a particular button event occurs. This type of code is called an event handler. HTML button event handlers can consist of any valid JavaScript code, and typically access report data and the Actuate JavaScript API to provide interactive viewing functionality.

The HTML button supports multiple events, and you can write multiple event handlers for a single button to execute different routines based on the event that occurs. For example, you can write an event handler that displays help information when the user moves the mouse pointer over a button, and a second event handler to run a business process when the user clicks the button.

Table 15-1 lists and describes the events that the HTML button supports and for which you can write code.

Table 15-1 Supported events

Event	Description
onblur	Occurs when the button loses focus, or stops being active
onclick	Occurs when the button is clicked
ondblclick	Occurs when the button is double-clicked
onfocus	Occurs when the button gets focus, or becomes active
onkeydown	Occurs when a keyboard key is pressed
onkeypress	Occurs when a keyboard key is pressed and released
onkeyup	Occurs when a keyboard key is released
onmousedown	Occurs when a mouse button is pressed
onmousemove	Occurs when a mouse pointer moves when it is over the button
onmouseover	Occurs when a mouse pointer moves onto the button
onmouseup	Occurs when a mouse button is released

When you select an event for which to write code, the script editor provides a JavaScript code template, as shown in Figure 15-5.

The following line of code in the template is a signal to the software to execute the code within the braces that follow when a click, or button press, event occurs:

```
this.onclick = function(event)
```

Do not modify this line of code. Write your JavaScript code within the braces following that line.

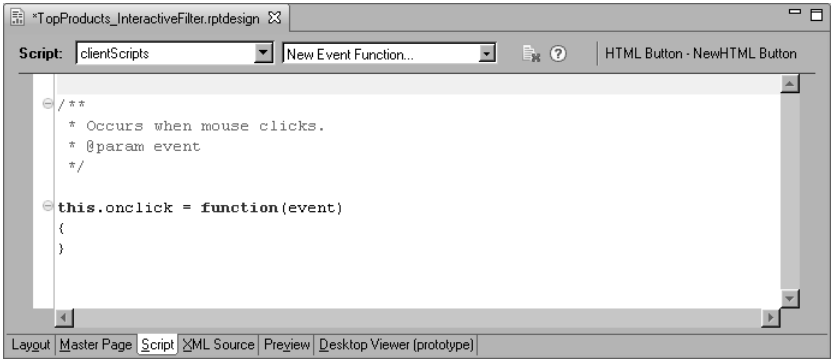


Figure 15-5 Script editor displaying a script template

If you write multiple event handlers for an HTML button, the script editor places all the event handlers serially, as shown in the following code example:

```
/**
 * Occurs when mouse clicks.
 * @param event */

this.onclick = function(event)
{
   /* onclick code here */
}

/**
 * Occurs when a mouse button is released.
 * @param event */

this.onmouseup = function(event)
{
 /* onmouseup code here */
}
```

Accessing report data

It is common to use HTML buttons to perform calculations on-demand or to present additional information. For example, rather than display customer notes that take up space in a report or that users view infrequently, you can create an HTML button that, when clicked, displays the information when users want to view the information.

These types of event handlers typically require access to data in the report, such as row data, aggregate data, or report parameter values. To provide event handlers with access to report data, you must do the following:

1 Insert the HTML button in a container, such as a table, that provides access to data.

2 For each type of data, create a variable for the HTML button using the Variables page on Property Editor. Figure 15-6 shows an HTML button variable named CustomerNotes whose value is set to the Notes column.

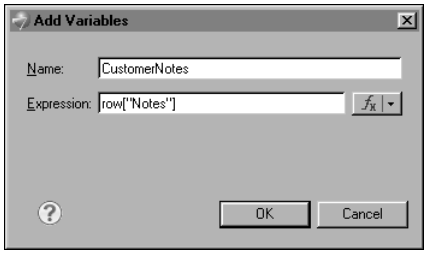


Figure 15-6 Variable associated with an HTML button

After you create a variable, use dataObject to access the variable in an event handler. For example, to access the variable CustomerNotes, use dataObject.CustomerNotes, as shown in the following event handler code:

```
/**
 * Occurs when mouse clicks.
 * @param event */
this.onclick = function(event)
{
   alert("Customer notes: " +
      "\n" + dataObject.CustomerNotes );
}
```

This example uses the JavaScript alert function to display customer notes in a message box, as shown in Figure 15-7.

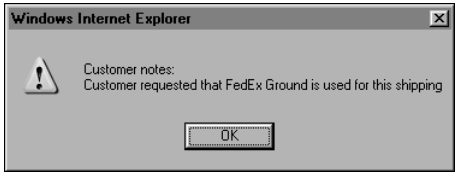


Figure 15-7 Message box displaying a note when the HTML button is clicked

The next example shows how to use an HTML button to calculate the price of a product after applying a discount specified by the user at report view time. Figure 15-8 shows the report in the web viewer. The report lists the products and their MSRP (Manufacturer's Suggested Retail Price). Each product row contains a Discounted Price button for calculating the discounted price for that product.

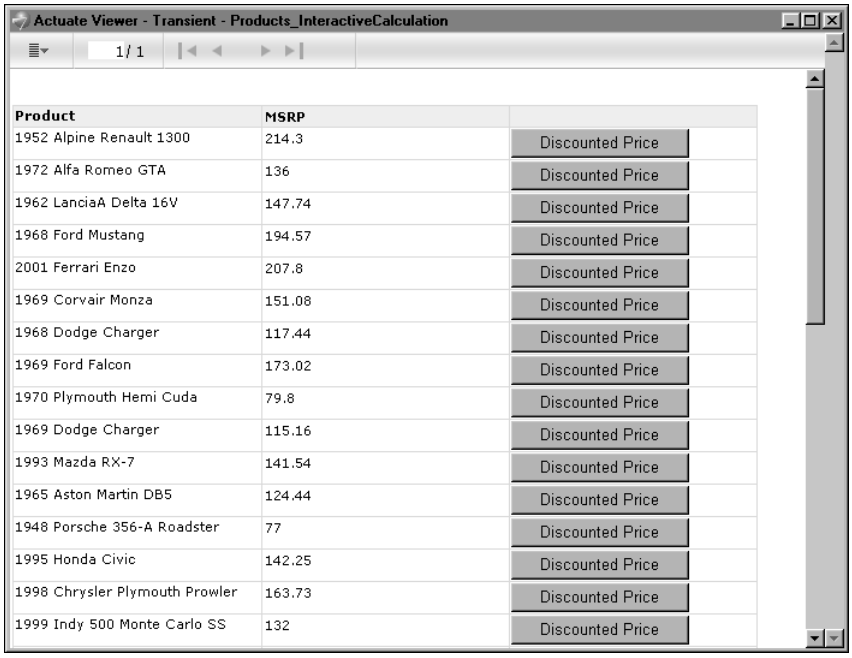


Figure 15-8 Product report using HTML buttons to calculate discounted prices

When the user clicks a button, a window prompts the user to enter a discount percent, as shown in Figure 15-9.

Figure 15-9 Window prompting for a discount percent

After the user enters a value, such as 10, and chooses OK, another window displays the product's discounted price, as shown in Figure 15-10.

Figure 15-10 Window displaying the discounted price

To create this HTML button, a button labeled Discounted Price is inserted in the detail row of a table. The HTML button's event handler code requires the MSRP

values to calculate the discounted price, so a variable is created. Figure 15-11 shows the definition of a variable named Price.

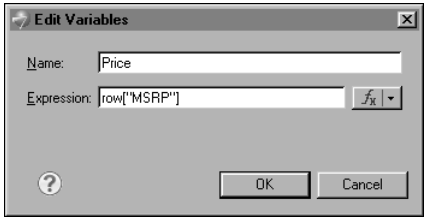


Figure 15-11 Price variable defined for the HTML button

The event handler code for the HTML button is as follows:

```
this.onclick = function(event)
{
   Discount = window.prompt('Enter the discount percent: ','');
   DiscountedPrice = dataObject.Price - (dataObject.Price *
      (Discount/100));
   alert("Discounted price: " + DiscountedPrice);
}
```

The first line after the opening brace prompts the user for a discount value and stores the value in the Discount variable. The second line calculates the discounted price using the values in the Price and Discount variables. The third line displays the results in a message box. Note that this event handler code covers only the main tasks. A more complete event handler would also perform data validation to ensure that the input value is a number, and handle the case if the user chooses Cancel at the prompt.

How to add a variable to an HTML button

1. In the layout editor, select the HTML button to which to add a variable.
2. Choose Property Editor, then choose the Variables tab.
3. In Variables, choose Add.
4. In Add Variables, specify the following values:
   1. In Name, type a unique name for the variable. JavaScript event handlers use the name to access the variable's information through dataObject. For example, the event handler accesses a variable named credit as dataObject.credit.
   2. In Expression, type the value that the variable contains. You can also use Expression Builder to create a value. Choose OK.

   Variables displays the variable you created, as shown in Figure 15-12.

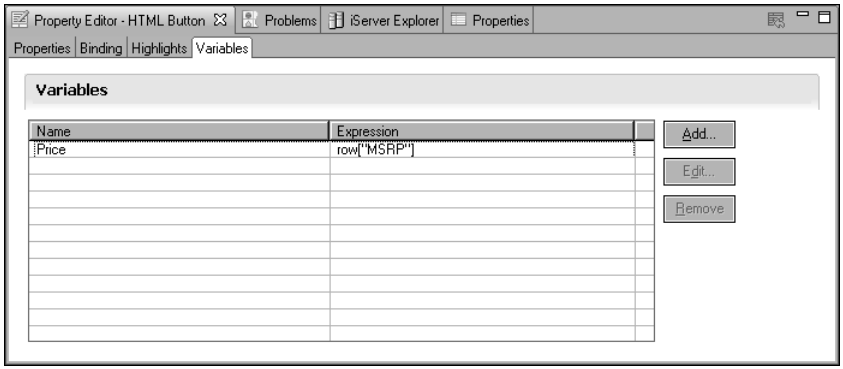


Figure 15-12 Variable defined for an HTML button

Using the Actuate JavaScript API

Actuate provides a JavaScript API (JSAPI) to support the integration of Actuate technology with web applications. Application developers can use the API to embed entire reports or individual report elements, such as charts or tables, into existing web pages.

HTML button event handlers can also use JSAPI to access report elements, manipulate data, and refresh a report in the Actuate viewer. For example, you can use the JSAPI to implement interactive functionality in the viewer, such as sorting and filtering data, linking to other report elements, and displaying or hiding report elements.

The three HTML buttons in the report shown in Figure 15-1, which provide three data filtering options, use methods in the JSAPI to get the current viewer, create the filters, and reload the report with new data each time the user clicks one of the buttons. The following is the event handler code for the Top Ten Products button:

```
this.onclick = function(event)
{
   //Get the current viewer object and the table with the
   //bookmark DetailTable on the current page.
   var viewer = this.getViewer();
   var pagecontents = viewer.getCurrentPageContent();
   var table = pagecontents.getTableByBookmark("DetailTable");

   //Create a top 10 filter on the table
   table.setFilters(new actuate.data.Filter("PRICE",
   actuate.data.Filter.TOP_N, "10"));

   //Reload the table
   table.submit();
}
```

The following is the event handler code for the All Products button:

```
this.onclick = function(event)
{
   var viewer = this.getViewer();
   var pagecontents = viewer.getCurrentPageContent();
   var table = pagecontents.getTableByBookmark("DetailTable");
   table.clearFilters("PRICE");
   table.submit();
}
```

The JSAPI provides many classes and methods that are useful for adding functionality to HTML buttons. For more information about using the JSAPI, see *Using Actuate JavaScript API*.

Testing an HTML button

As mentioned previously, HTML buttons are supported in HTML reports only. To test the functionality of an HTML button, run the report in the web viewer. The previewer in the report editor does not support the Actuate JavaScript API. You can view an HTML button that contains JSAPI code in the previewer, but it does not respond to any button events.

Changing the appearance of an HTML button

As with other report elements, you can modify an HTML button by changing property values, such as its name, its value, or aspects of its appearance. The tabs on the property sheet for the element support altering the appearance, visibility, and other features.

The general properties page provides the ability to change the size, color, and the appearance of the text of the button. By default, the button's size adjusts to the length of the text on the button. If a report contains multiple buttons and you want all the buttons to be the same size, specify values for the Width and Height properties.

The general properties page also supports adding an image to the face of the button. Use the Upload button next to the Image property to add an image. Before doing so, make sure the image file is the appropriate size for the button. A button expands to display the image in its full size unless you specify Width and Height values. So, if an image is large, and you use the default auto-sizing feature, the button is large. If you use explicit Width and Height values, and the image is larger than the specified button size, the image is truncated. To change the size of an image, you must edit the image using a graphic editing tool.

Figure 15-13 shows the general properties for an HTML button.

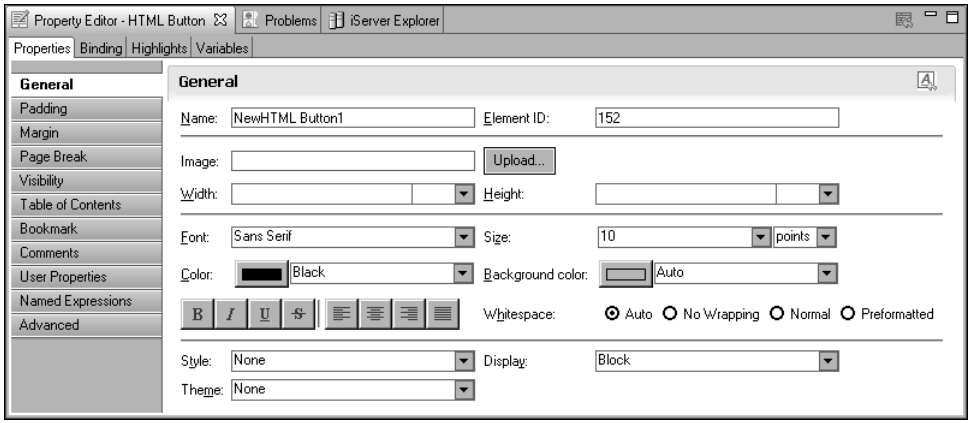


Figure 15-13 General properties for an HTML button

Use the padding settings, as shown in Figure 15-14, to add extra space around the text or image on the face of the HTML button.

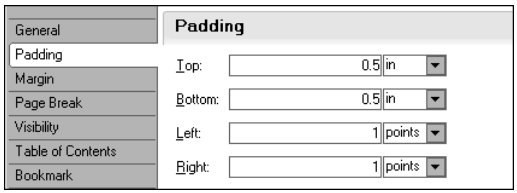


Figure 15-14 Padding properties for an HTML button

Padding supports the use of different units, such as inches or points. When you add padding, it affects the HTML button as shown in Figure 15-15. The button on the left uses the default padding values. The button on the right uses padding values of 0.5 inch at the top and bottom.

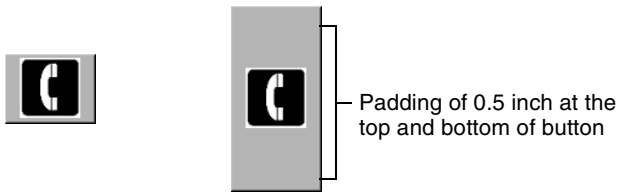


Figure 15-15 Padding added to an HTML button using an image

Use the margin settings to increase the space around the entire button. Specifying margin values is similar to specifying padding values, as shown in Figure 15-16.

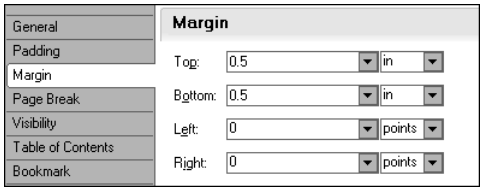


Figure 15-16 Margin properties for an HTML button

However, whereas padding modifies the size of the HTML button, margins modify the space around the button and do not change the button size. Figure 15-17 shows two buttons, each within a cell. The button on the left uses the default margin values. The button on the right uses margin values of 0.5 at the top and bottom.

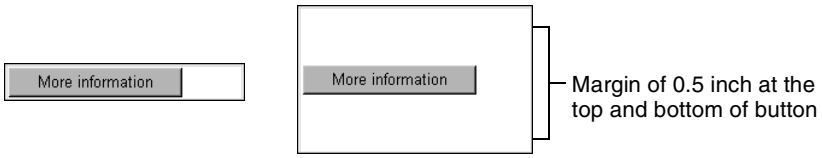


Figure 15-17 Margin space around an HTML button in a table

Visibility, Page Break, Table of Contents, and other properties operate in the same manner as they do for other report elements.

How to change the name or value of an HTML button

1. Double-click the HTML button.
2. In HTML Button, in Name, type the new button name. In Value, type the new value.
3. Choose OK. The HTML button displays the new value.

Chapter

16

Controlling user access to report pages and data

This chapter contains the following topics:

- About the security model
- Controlling user access to report pages
- Controlling user access to data

About the security model

All files stored in an Actuate BIRT iServer Encyclopedia volume are subject to a standard security procedure, which restricts file access to authorized users. The iServer security model is based on roles and privileges. The iServer administrator creates roles for various job functions in an organization, such as finance, marketing, and sales. The privileges, or permissions, to perform certain operations, such as read, write, and execute, are assigned to roles. Users are assigned roles, and, through these roles, acquire the privileges to perform particular operations on folders and files.

With this level of security, each user has access to files and folders on a need-to-know basis. For security at a more detailed level, iServer provides the following types of security:

- Page-level security, which controls user access to particular sections or pages in a report. This security feature requires the Page Level Security option on iServer. The published report must be assigned the Secure Read privilege.
- Data security, which controls user access to a particular set of data provided by a BIRT data object. This security feature is part of the core iServer functionality. The published data object must be assigned the Secure Read privilege.

The security procedure that applies to files and folders in an iServer volume is implemented entirely in iServer. Page-level security and data security, however, require implementation in Actuate BIRT Designer as well.

About access control lists (ACLs) and security IDs

Page-level and data security use the same security mechanism in Actuate BIRT Designer: access control lists.

An access control list (ACL) is a list of security IDs that tells iServer which users have access to a particular item in a report or data object. A security ID can be either a user name or a role defined in iServer. Typically, you use roles because they are more permanent than user names. A role can be assigned to different users at different times as employees leave or change positions.

To implement page-level and data security in Actuate BIRT Designer, perform the following tasks:

- In the report or data object, select the item to which to apply security.
- For the item's Access Control List Expression property, specify an expression that evaluates to a security ID or a list of security IDs.

ACL expression syntax

The ACL expression must evaluate to a string, and can be either a JavaScript or EasyScript expression. If specifying multiple security IDs, separate each with a comma.

The following expressions are examples of ACL expressions in JavaScript. The first expression specifies a literal role name. The second expression specifies two literal role names. The third expression evaluates to role names, such as Sales Manager France or Sales Manager Canada. The fourth expression specifies two literal role names and an expression that evaluates to role names.

```
"CFO"
"CFO, Sales VP"
"Sales Manager " + row["Country"]
"CFO, Sales VP" + "," + "Sales Manager " + row["COUNTRY"]
```

The following ACL expressions are the EasyScript equivalent:

```
"CFO"
"CFO, Sales VP"
"Sales Manager " & [Country]
"CFO, Sales VP" & "," & "Sales Manager " & row["COUNTRY"]
```

Controlling user access to report pages

In a report that uses page-level security, report users can view only pages to which they have access. You can design a single report that meets the needs of a range of users. The most common case is to create a hierarchy of ACLs where each successive level has access to more information. The ACL hierarchy can match the organizational hierarchy of a company.

For example, in a report that provides worldwide sales data by region and country, you can restrict user access to the content as follows:

- Each country sales manager can view only the pages that display sales data for her country.
- Each regional sales manager can view all the pages that display sales data for the countries in her region.
- The vice president of sales can view the entire report.

Figure 16-1 shows the page that the sales manager in Belgium can view.

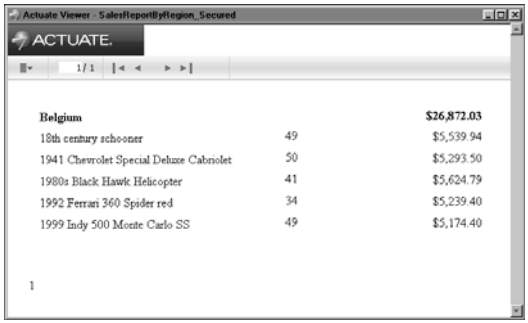


Figure 16-1 Page that the sales manager in Belgium can view

Figure 16-2 shows the pages that the regional sales manager for Europe can view.

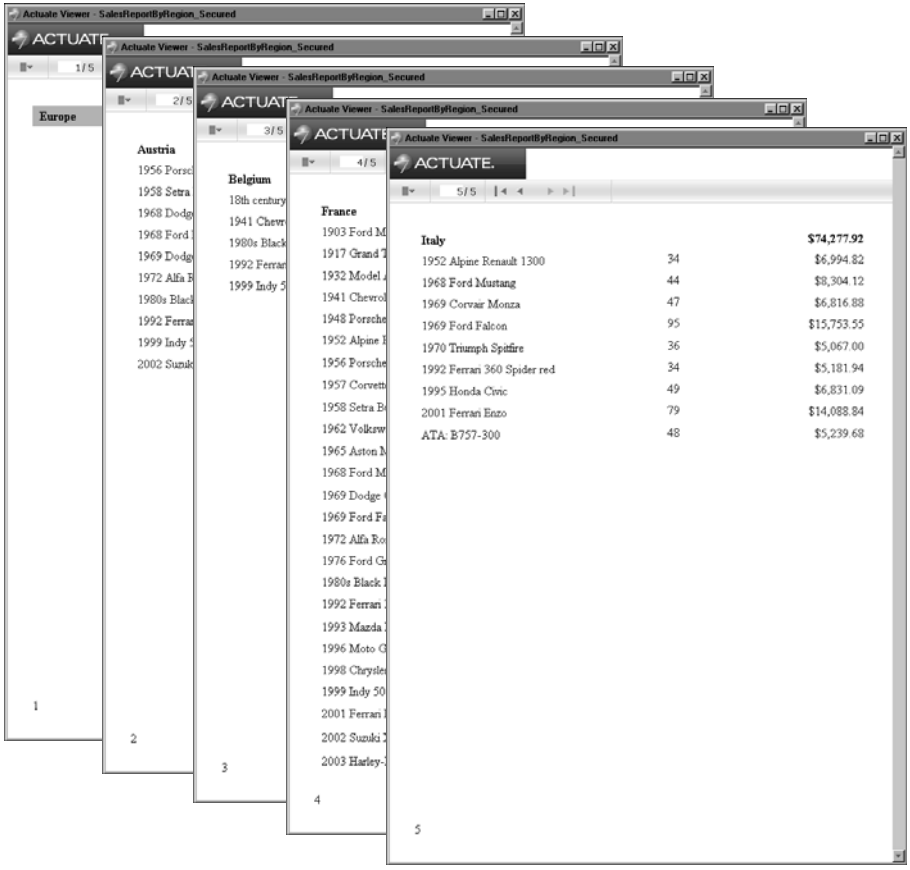


Figure 16-2 Pages that the regional sales manager for Europe can view

Figure 16-3 shows the full report, which only the vice president of sales can view.

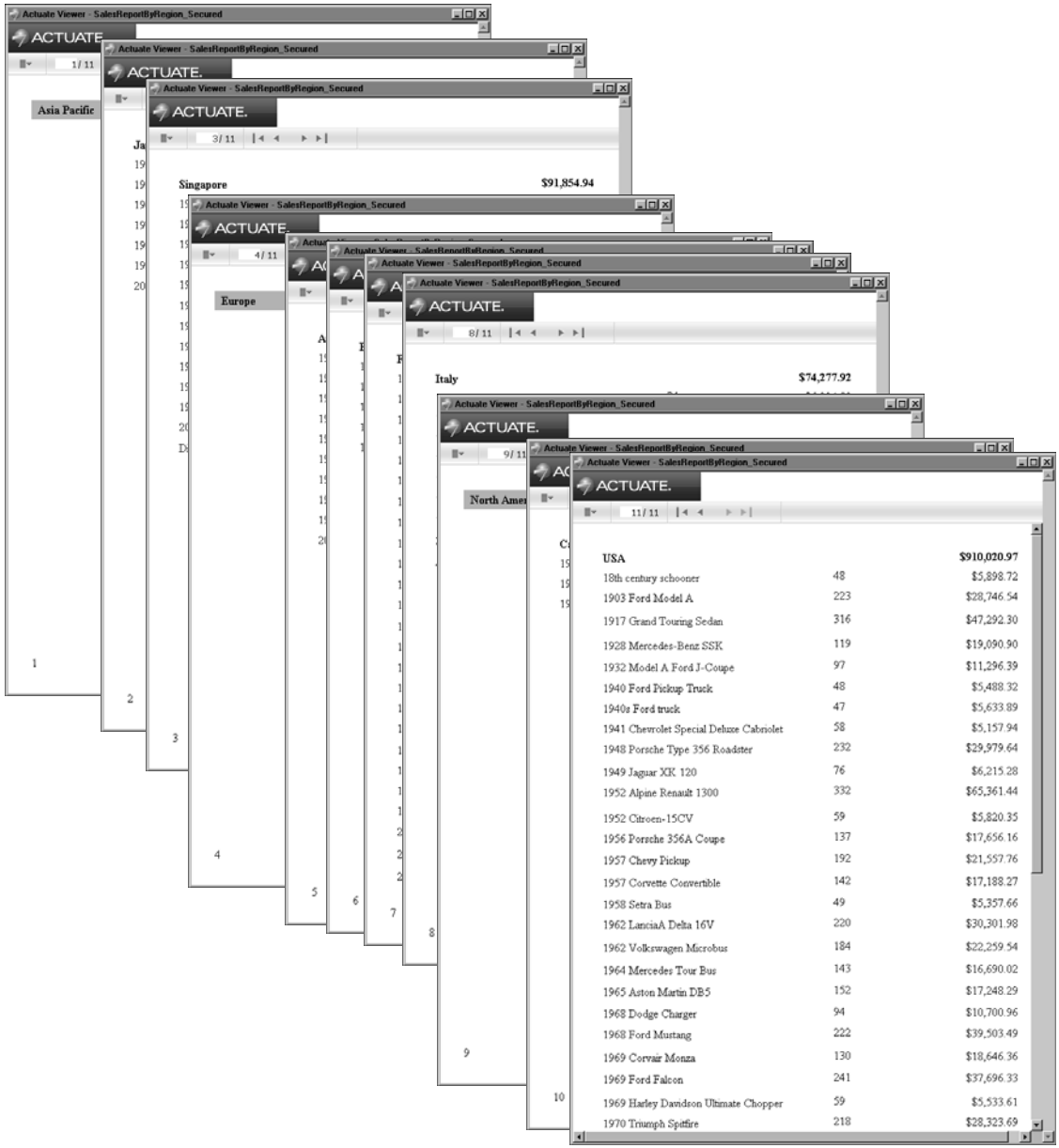


Figure 16-3 Pages that the vice president of sales can view

Without page-level security, you would need to create multiple reports—one report for each user—and the iServer administrator would then have to define

different security rules for each report, and manage multiple reports. In the sales report example, which presents data for three regions and eight countries, you would have to create twelve reports. For large companies, which typically have more hierarchical levels and more users, the number of reports increases.

Adding page-level security to a report

To implement page-level security in a report, perform the following tasks:

- Identify the sections that require security.
  The most common elements to which to apply security are tables, lists, grids, and groups defined in a table.
- Identify the users that can view each section.
  Obtain the security IDs, typically roles, from the iServer volume administrator.
- Set security.
  For each element that requires security, right click the element, then select Security from the context menu. Set the Access Control List Expression property to the security ID or IDs to which to grant access to the element's content.

Example 1

Figure 16-4 shows the design for the sales report shown in the previous section. The report design consists of a single table that groups data by region, country, and product.

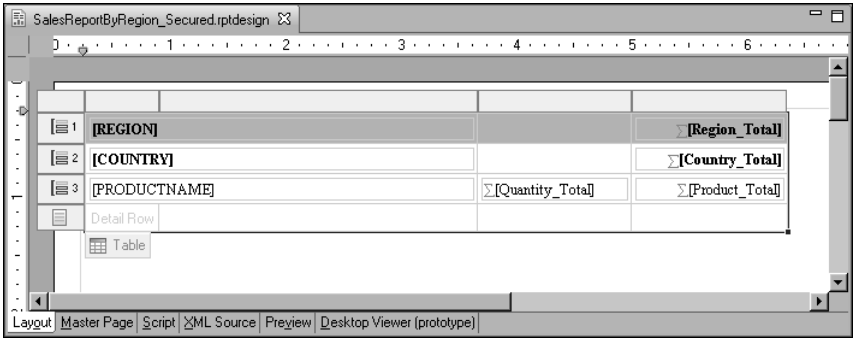


Figure 16-4 Design of report that groups sales data by region and country

Page-level security is applied to these elements: the table, the Region group, and the Country group. Figure 16-5 shows the Security dialog for the table element.

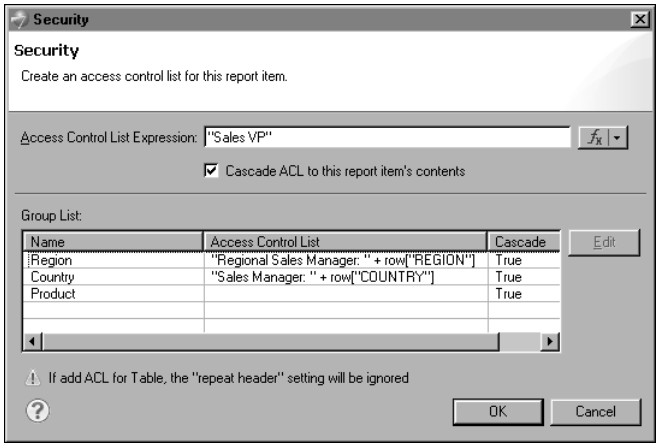


Figure 16-5 Page-level security applied to the table and two of its groups

- The Access Control List Expression property is set to the value "Sales VP".
- The Cascade ACL option is selected. This setting propagates the specified ACL to all the elements in the table.

  These settings specify that only the Sales VP has access to all of the table's contents.

- For the Region group:
  - The Access Control List expression is:

    ```
    "Regional Sales Manager: " + row["REGION"]
    ```

    This expression specifies that data for each region is restricted to a specific regional sales manager role. For example, only a user with the Regional Sales Manager: Europe role can view the sales data for Europe.
  - Cascade is set to True. This value propagates the ACL to the elements in the Region group, providing the regional sales manager access to all the data within the Region group.
- For the Country group:
  - The Access Control List expression is:

    ```
    "Sales Manager: " + row["COUNTRY"]
    ```

    This expression specifies that data for each country is restricted to a specific sales manager role. For example, only a user with the Sales Manager: France role can view the sales data for France.
  - Cascade is set to True. This value propagates the ACL to the elements in the Country group, providing the sales manager access to all the data within the Country group.

Example 2

This example shows how to implement page-level security in a report that contains multiple tables. Figure 16-6 shows a report design that contains four tables and identifies the roles that can view each table. The last table shows detailed sales data grouped by country and product. The CEO, Sales VP, and Sales Director can view all the content in this table. Each sales manager can view only the sales data for her country.

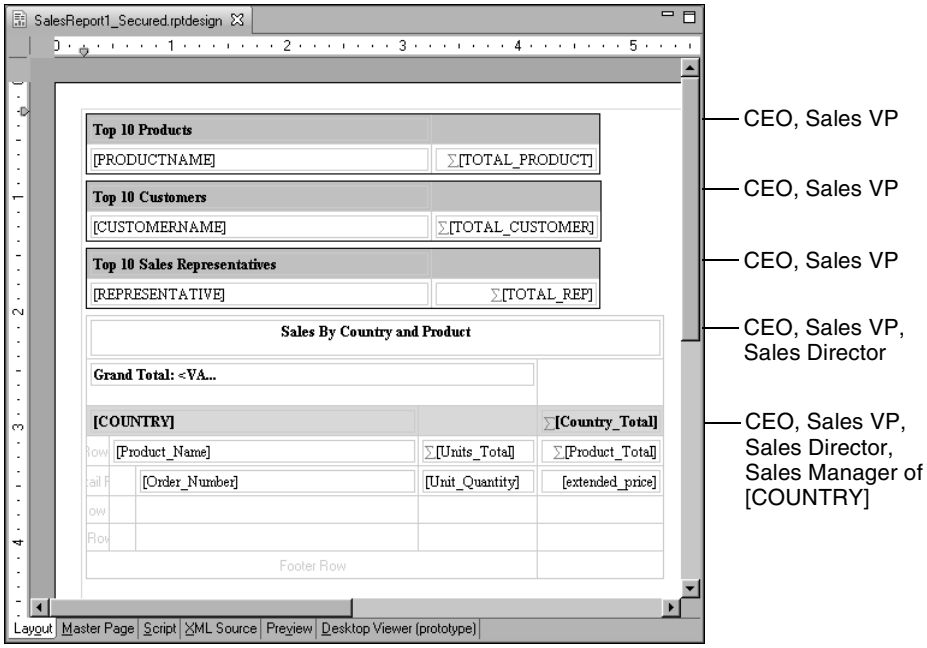


Figure 16-6 Design of report that contains four tables and the roles that can access each table

There are several ways to implement page-level security in this report. You can:

- Select each table and set each table's Access Control List Expression property to the roles identified in Figure 16-6.

  The ACL for the first, second, and third tables would be:

  ```
  "CEO, Sales VP"
  ```

  The ACL for the first fourth table would be:

  ```
  "CEO, Sales VP, Sales Director"
  ```

  The ACL for the Country group in the fourth table would be:

  ```
  "CEO, Sales VP, Sales Director" + "," + "Sales Manager of " +
    row["COUNTRY"]
  ```

- Use the Cascade ACL option to cascade security IDs from a container element to its contents. Because the CEO and Sales VP roles can view the entire report, it is more efficient to specify the ACL once at the topmost container, the report element, than it is to specify the same ACL multiple times.

  The ACL for the report element would be:

  ```
  "CEO, Sales VP"
  ```

  The ACL for the first fourth table would be:

  ```
  "Sales Director"
  ```

  The ACL for the Country group in the fourth table would be:

  ```
  "Sales Manager of " + row["COUNTRY"]
  ```

- Add a grid to the report, place all the tables in the grid, and cascade the "CEO, Sales VP" ACL expression from the grid instead of from the report element. This design is more versatile than the previous one because it is often practical to leave the ACL at the report level blank, which grants all users access to the report. For example, if you add new sections, such as a title page, for a broader range of users, it is easier to start with the rule that all users can view all the content in a report, then restrict access to particular sections.

The report examples in this section illustrate several key concepts about page-level security, which are summarized next. Understanding these concepts can help you design a report to use page-level security.

- When an element's ACL expression property is blank, there are no viewing restrictions for that element, except those restrictions (determined by the ACLs) that the element inherits from its container.
- An element inherits the ACLs from its container when the container's Cascade ACL option is selected. This option, selected by default, means that a user who is permitted to view a container can also view all elements within the container.
- The report element is the topmost container. If its ACL expression property is blank, BIRT assigns an internal ACL of "__all" to the report. This setting combined with the selected Cascade ACL option ensures that a report created initially is accessible to all users.
- BIRT generates one report document, inserting a page break between elements that have different ACLs. This concept explains why some pages display just a group header, as Figure 16-3 shows, when groups in a table have different ACLs.

Enabling and disabling page-level security

For ACLs to take effect when the report is run on iServer, you must enable page-level security in the report design. When enabled, BIRT generates a report that

consists of pages, which are restricted to users with specified security IDs. If you decide later to make the entire report available to users, all you do is disable the page-level security option. You do not have to remove the ACLs.

How to turn page-level security on or off

1 In the layout editor, right-click a blank area of the report, then select Security.

2 In Security, shown in Figure 16-7, either select or deselect Enable Page Level Security on generated report document.

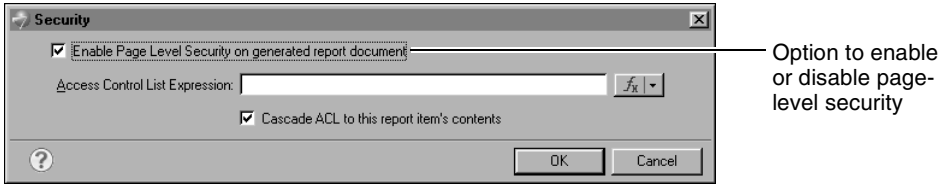


Figure 16-7 Enabling page-level security

Configuring page numbers

A report that uses page-level security provides two options for displaying page numbers. The report can display page numbers sequentially in the order that they appear to a user. For example, if a user can view pages 1, 5, 6, and 8 of a report, the page numbers that the user sees are 1, 2, 3, and 4. Alternatively, the report can display the actual page numbers 1, 5, 6, and 8.

Similarly, for page number formats that include the total page count, such as 1 of 4, the total page count can be the number of pages visible to the user or the number of pages in the report.

How to configure page numbers

This procedure assumes that the report already contains page number elements.

1 Choose Master Page to view the page number elements. Figure 16-8 shows an example of a master page where the footer contains three elements to display page numbers in the format 1 of 10.

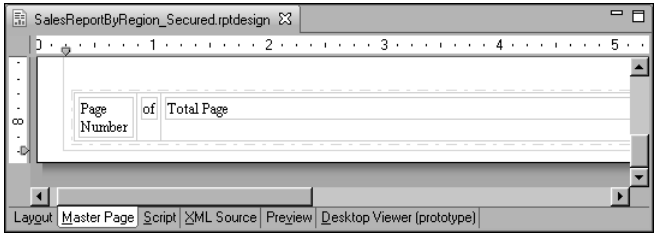


Figure 16-8 Master page containing page number elements

2 Right-click the Page Number element and choose Security.

3 In Security, shown in Figure 16-9, select a display option, then choose OK.

- Select Visible Page Number to display numbers sequentially in the order that the pages appear to the user.
- Select Actual Page Number to display the numbers as they appear in the entire report.

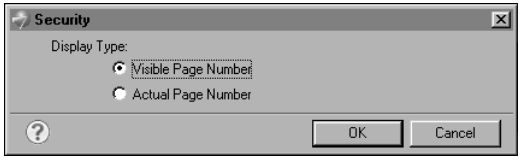


Figure 16-9 Selecting a page-numbering option

4 If the page number format includes a total page count, as shown in the sample master page in Figure 16-8, use the instructions in the previous step to select a display option for the Total Page element.

Testing page-level security

Actuate BIRT Designer supports the simulation of secure report viewing, so that you can test page-level security without having to publish the report to iServer, log in with different user credentials, run the report and verify its output.

How to test page-level security

1 Make sure page-level security is enabled. This procedure is described earlier in this chapter.

2 Choose Run➤View Report with Page Security, and select the output format in which to view the report.

3 In Run Report with Page Level Security, shown in Figure 16-10, type a security ID specified in an ACL. For example:

```
Sales Manager: France
```

Figure 16-10 Using a specified security ID

Choose OK. The report runs and displays only the page or pages that the specified security ID can view.

4 Repeat the previous step until you finish testing all the security IDs used in the report.

Controlling user access to data

In addition to page-level security, iServer also supports data security, which controls user access to a particular set of data provided by a BIRT data object. For example, you can design a data object that returns one set of data rows for one group of dashboard or report users, and a different set for another group of users.

You can limit access to the following items in a data object:

- A data set, its rows and columns
- A cube, its measures, dimensions, dimension levels, and level members

After designing the data object, generate a data object store (a .data file) and publish this file to an iServer volume. iServer supports data security on .data files, but not on .datadesign files.

A user can only see and use the data items to which she is granted access. The security rules apply to users designing a dashboard or a report in BIRT Studio, as well as, users running a dashboard or report.

Adding security to a data object

To implement security in a data object, perform the following tasks:

- Identify the data items that require security.
- Identify the users that can view each item. Obtain the security IDs, typically roles, from the iServer volume administrator.
- In the data object design (.datadesign), for each item that requires security, set the item's Access Control List Expression property to the security ID or IDs to which to grant access to the item.

Figure 16-11 shows security applied to the rows in a data set. The expression specified for the Row Access Control List Expression property is:

```
"HR Director" + "," + "Manager: Office " + row["OFFICECODE"]
```

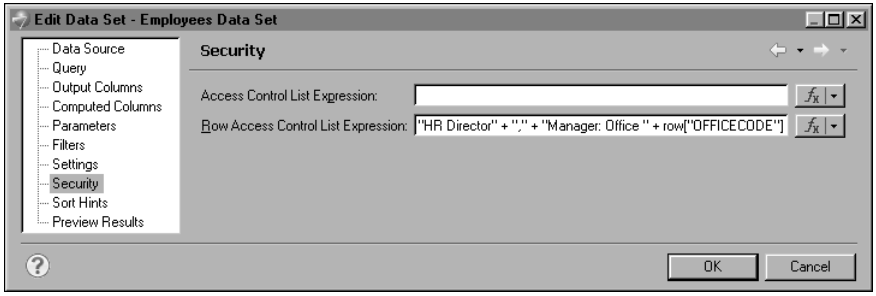


Figure 16-11 Data security applied to data set rows

Security applied to data set rows acts as a filter. In the example shown in Figure 16-11, the HR Director can view all rows in the data set. Managers can view only rows that pertain to their department as specified by the office code.

Figure 16-12 shows a report design that uses the secured data object. In the design, a table contains data elements that access the data set columns in the data object.

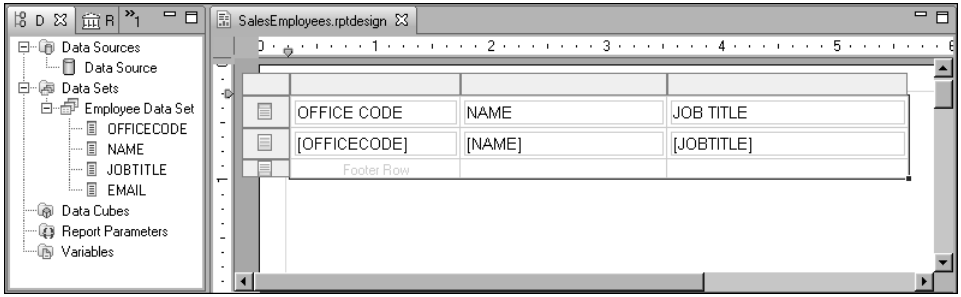
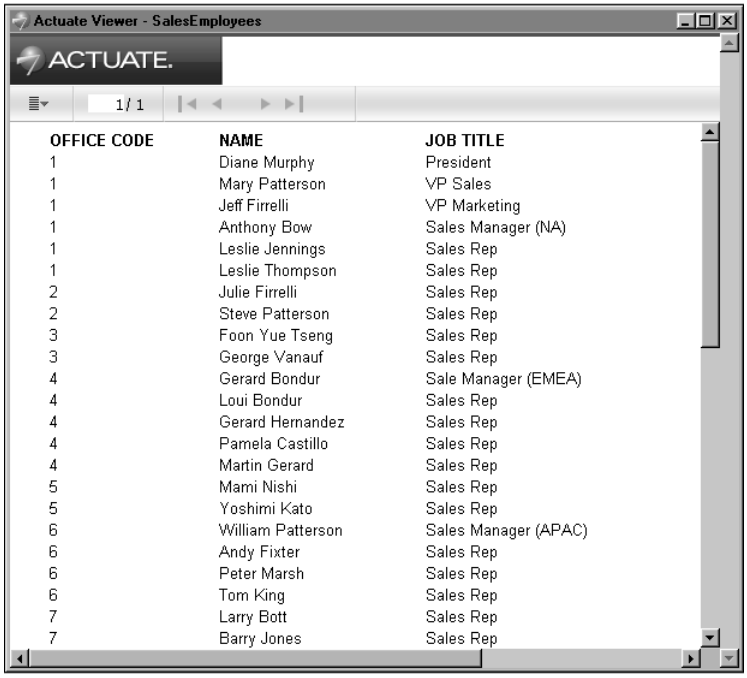


Figure 16-12 Report design that uses data from a data set in a secured data object

When run and viewed by the HR Director, the report displays all the rows in the data set, as shown in Figure 16-13.



OFFICE CODE	NAME	JOB TITLE
1	Diane Murphy	President
1	Mary Patterson	VP Sales
1	Jeff Firrelli	VP Marketing
1	Anthony Bow	Sales Manager (NA)
1	Leslie Jennings	Sales Rep
1	Leslie Thompson	Sales Rep
2	Julie Firrelli	Sales Rep
2	Steve Patterson	Sales Rep
3	Foon Yue Tseng	Sales Rep
3	George Vanauf	Sales Rep
4	Gerard Bondur	Sale Manager (EMEA)
4	Loui Bondur	Sales Rep
4	Gerard Hernandez	Sales Rep
4	Pamela Castillo	Sales Rep
4	Martin Gerard	Sales Rep
5	Mami Nishi	Sales Rep
5	Yoshimi Kato	Sales Rep
6	William Patterson	Sales Manager (APAC)
6	Andy Fixter	Sales Rep
6	Peter Marsh	Sales Rep
6	Tom King	Sales Rep
7	Larry Bott	Sales Rep
7	Barry Jones	Sales Rep

Figure 16-13 Preview of the report for the HR Director role

When run and viewed by the manager of a specific office code, the report displays only the rows for that office. Figure 16-14 shows the report that Manager: Office 4 sees.

Figure 16-14 Preview of the report for the Manager: Office 4 role

Enabling and disabling data security

For ACLs to take effect, you must enable data security in the data object. If you decide later to make all the data in the data object available to users, all you do is disable data security. You do not have to remove the ACLs.

How to turn data security on or off

1. In the layout editor, right-click in an empty area of the data object design, then select Security.
2. In Security, shown in Figure 16-15, either select or deselect Enable Data Security.

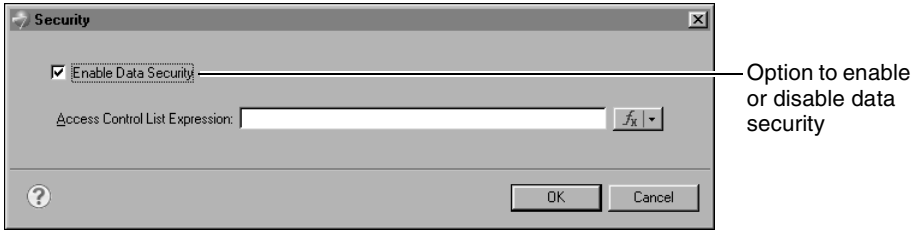


Figure 16-15 Enabling data security

Testing data security

Actuate BIRT Designer supports the simulation of viewing reports with data security. You can test data security in a report without having to publish the report to iServer, log in with different user credentials, run the report and verify its output.

To test data security from the perspective of a user designing a dashboard or a report in BIRT Studio, you need to run tests on the iServer. The testing procedure entails publishing the data object, assigning user privileges on the data object, logging in with user credentials, then launching the dashboard design tool or BIRT Studio, and using the data object as a source of data for the dashboard or report.

How to test data security in a report

1. Using Actuate BIRT Designer, build a report that uses a secure data object store (.data) as its data source. For information about this procedure, see Chapter 3, "Accessing data in a data object."
2. When you finish building the report, choose Run➤View Report with Data Security, and select the output format in which to view the report.
3. In Run Report with Data Security Enabled, shown in Figure 16-16, type a security ID specified in an ACL in the data object. For example:

   ```
   Manager: Office 4
   ```

   Figure 16-16 Running a report with data security using a specified security ID

   Choose OK.

   The report runs and displays only the content that the specified security ID can view.
4. Repeat the previous step until you finish testing all the security IDs used in the report.

Chapter

17

Accessing iServer environment information

This chapter contains the following topics:

- Writing event handlers to retrieve iServer environment information
- Debugging event handlers that use the iServer API
- iServer API reference

Writing event handlers to retrieve iServer environment information

Report developers distribute reports to users by publishing them to Actuate BIRT iServer. Sometimes a report requires information about the iServer environment to implement application or business logic based on, for example, the security credentials of the user running the report, the browser in which the report is viewed, the server volume on which the report is run, and so on. BIRT provides an API, referred to in this chapter as the iServer API, that enables access to this type of information.

To use the iServer API in a report, you write event handler scripts in either Java or JavaScript. BIRT event handlers are associated with all the elements that make up a report, such as data sources, data sets, tables, charts, and labels. When a report is run, BIRT fires events and executes event handlers in a specific sequence to generate and render the report.

Writing event handlers in a report requires knowledge of the BIRT event model. For information about the event model and details about writing event handlers in Java and JavaScript, see *Integrating and Extending BIRT.* This chapter describes the additional requirements for accessing and debugging the iServer API in an event handler.

Writing a JavaScript event handler

You write a JavaScript event handler that uses the iServer API the same way you write other event handlers. In Actuate BIRT Designer, you select an element, such as the report design or a table, then use the script editor to select an event, such as beforeFactory or onCreate, for which to write an event handler.

Figure 17-1 shows the script editor displaying event-handling code written for the report design's beforeFactory event.

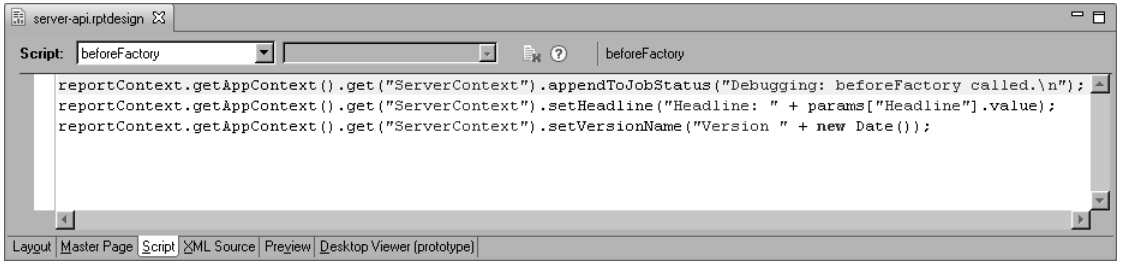


Figure 17-1 Event-handling code in the script editor

Writing a Java event handler

Writing a Java event handler that uses the iServer API is similar to writing other types of event handlers. You create a Java event handler class, make the class available to BIRT, and associate the class with a report element. The difference is the additional JAR files required to access the iServer API.

You must add the following JAR files in the build path and classpath when configuring the Java event handler project:

- $ACTUATE_HOME\iServer\Jar\BIRT\lib\scriptapi.jar

  This JAR file provides the event handler classes and access to the reportContext object. If you use the ULocale methods, com.ibm.icu_version.jar is also required. $ACTUATE_HOME is the location where iServer is installed.

- $ACTUATE_HOME\iServer\reportengines\lib\jrem.jar

  This JAR file contains the definitions of the classes and methods in the iServer API.

Figure 17-2 shows the build path of a Java project that uses the iServer API. In this example, Actuate BIRT Designer is installed in the same path where iServer is installed. If Actuate BIRT Designer is installed on a different machine, you must copy the JAR files from the iServer machine to your workspace.

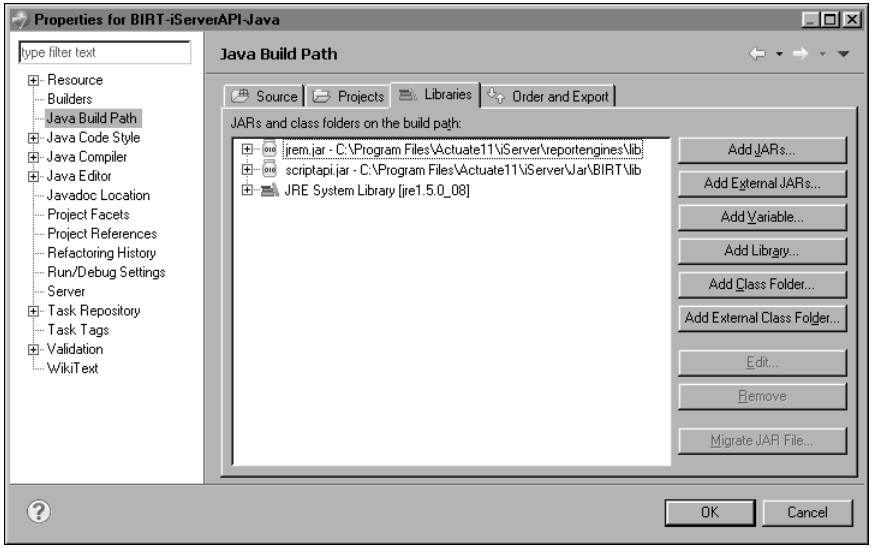


Figure 17-2 Build path of a Java project that uses the iServer API

About the serverContext object

The BIRT engine uses an object called serverContext to store information about the iServer environment. The serverContext methods and properties are defined in the IServerContext interface. The container for the serverContext object is the application context object appContext. The appContext object stores objects and values that are used in all phases of report generation and presentation.

The appContext object, in turn, is a property of the reportContext object. This object stores information associated with the instance of the report that is being generated or viewed. For example, the reportContext object stores information about report parameters, global variables, report output format, locale, the request that runs the report, and the application context. The report context class defines methods for setting and retrieving these properties. Every event handler in a BIRT report has access to the reportContext object. In Java, the report context object is an argument to all event-handler methods.

To call a method to retrieve iServer environment information, the code must reflect the relationships between the serverContext, appContext, and reportContext objects.

The following JavaScript code snippet shows how to call the getVolumeName( ) method to retrieve the name of the iServer volume in which a report runs:

```
reportContext.getAppContext().get("ServerContext").getVolumeName()
```

The following example shows the equivalent code snippet in Java:

```
IServerContext scontext;
scontext = (IServerContext)
   reportContext.getAppContext().get("ServerContext");
scontext.getVolumeName();
```

JavaScript event handler example

The code example in Listing 17-1 uses the getUserRoles( ) method to retrieve the user's roles and displays the contents of a report element if the user role is Manager. This code can be used, for example, in the onPrepare event of a table element to hide or display the table depending on the user role. The code example also uses the appendToJobStatus( ) method to write messages about the user's roles to the server job status.

Listing 17-1 JavaScript event handler

```
userRoles = reportContext.getAppContext().get("ServerContext")
   .getUserRoles();

reportContext.getAppContext().get("ServerContext")
   .appendToJobStatus("The user roles are:" + userRoles +"\n");
```

```
if (userRoles != null)
{
   for (i = 0; i < userRoles.size(); i++)
   {
      if ( userRoles.get(i) == "Manager")
      {
         reportContext.setGlobalVariable("HideDetails", "false");
         reportContext.getAppContext().get("ServerContext")
         .appendToJobStatus("The user has a Manager role\n");
         break;
      }
   }
}
```

Java event handler example

Like the JavaScript event handler in the previous section, the Java code example in Listing 17-2 uses the getUserRoles( ) method to retrieve the user's roles and displays the contents of a table if the user role is Manager. The TableEH class extends the TableEventAdapter class and implements the event-handler script in the onPrepare event method.

Listing 17-2 Java event handler class

```
package server.api.eh;

import java.util.List;

import org.eclipse.birt.report.engine.api.script.IReportContext;
import org.eclipse.birt.report.engine.api.script.element.ITable;
import org.eclipse.birt.report.engine.api.script.eventadapter
   .TableEventAdapter;

import com.actuate.reportapi.engine.IServerContext;

public class TableEH extends TableEventAdapter {

   public void onPrepare(ITable tbl, IReportContext reportContext)
   {

      IServerContext scontext;
      scontext = (IServerContext)
         reportContext.getAppContext().get("ServerContext");
      List<String> userRoles =  scontext.getUserRoles();
      scontext.appendToJobStatus("The user roles are:" + userRoles
         +"\n");
```

(continues)

```
      for (int i = 0; i < userRoles.size(); i++)
      {
         if ( userRoles.get(i).contentEquals("Manager"))
         {
            reportContext.setGlobalVariable("HideDetails", "false");
            scontext.appendToJobStatus("The user has a Manager role
               \n");
            break;
         }
      }
   }
}
```

Debugging event handlers that use the iServer API

A report that uses the iServer API returns the expected results only when it is run on iServer. When the report is run in Actuate BIRT Designer, the report cannot access the iServer to retrieve the server information, and the report typically returns null values. Therefore, you cannot debug the iServer API calls in the same way you debug other event handlers in Actuate BIRT Designer.

To debug iServer API calls, use the appendToJobStatus( ) method to write a debugging message for each event handler. For example, if you write a JavaScript event handler for the beforeFactory event, add the following line of debugging code:

```
reportContext.getAppContext().get("ServerContext")
   .appendToJobStatus("Debugging: beforeFactory called.\n");
```

In a Java event handler, write:

```
IServerContext scontext;
scontext = (IServerContext)
   reportContext.getAppContext().get("ServerContext");
scontext.appendToJobStatus("Debugging: beforeFactory called.\n");
```

The appendToJobStatus( ) method writes a specified string message in the status section of a job-completion notice. After running the report on iServer, you can view these messages in either iServer Management Console or iServer Information Console.

In Management Console, choose Jobs—Completed, then choose the job's details. The job's Status page displays the debug messages in the Status section, as shown in Figure 17-3.

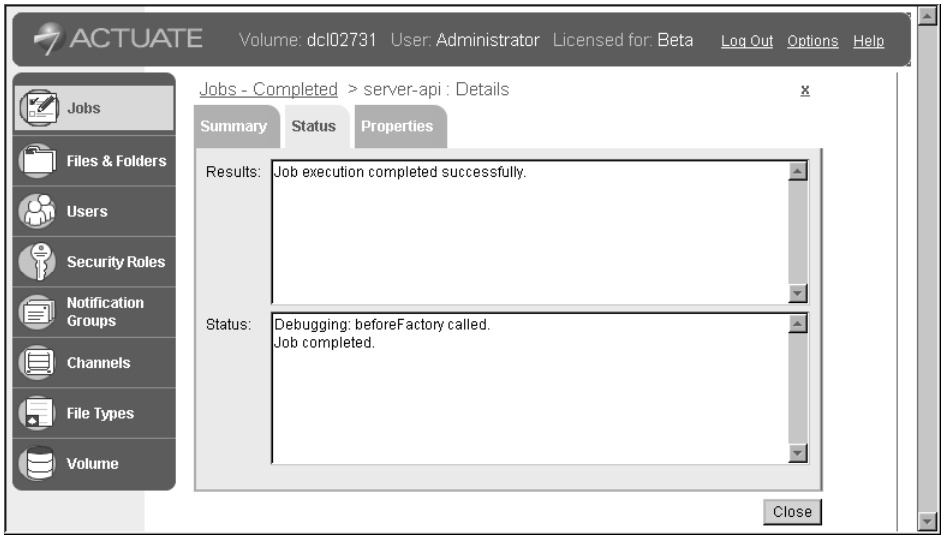


Figure 17-3 Debug message in the job status page in Management Console

In Information Console, choose My Jobs—Completed, then choose Details next to the job whose status you want to review. The debug messages appear in the Status section in the job details page, as shown in Figure 17-4.

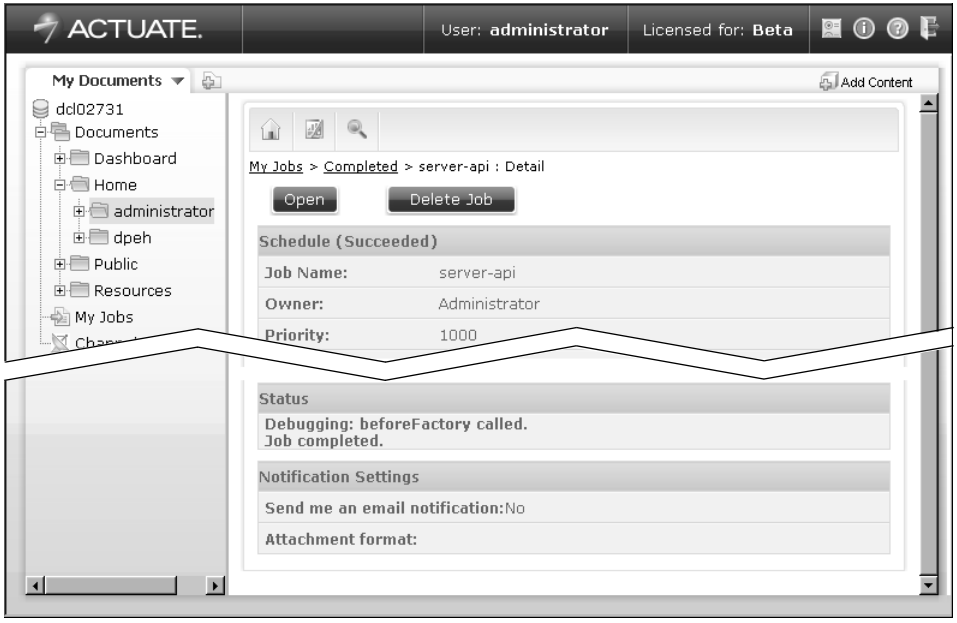


Figure 17-4 Debug message in the status section in the job detail page in Information Console

iServer API reference

This section lists all the methods in the iServer API in alphabetical order. Each method entry includes a general description of the method, the JavaScript and Java syntaxes, the result the method returns, and examples.

appendToJobStatus( )

Appends a specified string to the status of the current job. iServer writes status messages for each report-generation job.

JavaScript syntax

appendToJobStatus(statusString)

Java syntax

public void appendToJobStatus(String statusString)

Argument

statusString
The string to add to the job status.

Usage

Use to provide information for debugging purposes. For example, to verify that an event handler is executed, write a message indicating that the event method is called.

JavaScript example

```
reportContext.getAppContext().get("ServerContext")
   .appendToJobStatus("This message appears when beforeFactory is
   called.\n");
```

Java example

```
IServerContext scontext;
scontext = (IServerContext)
   reportContext.getAppContext().get("ServerContext");
scontext.appendToJobStatus("This message appears when
   beforeFactory is called.\n");
```

getAuthenticationId( )

Retrieves the current user's authentication ID.

JavaScript syntax

getAuthenticationId()

Java syntax

public String getAuthenticationId()

Usage

Use in cases when the report application needs to pass the ID to another application, such as IDAPI calls to iServer.

Returns

An authentication ID in String format.

JavaScript example

```
reportContext.getAppContext().get("ServerContext")
   .getAuthenticationId();
```

Java example

```
IServerContext scontext;
scontext = (IServerContext)
   reportContext.getAppContext().get("ServerContext");
scontext.getAuthenticationId();
```

getServerWorkingDirectory( )

Retrieves the path to the folder in the file system where temporary files are stored.

JavaScript syntax getServerWorkingDirectory()

Java syntax public String getServerWorkingDirectory()

Usage Use to read or write information from and to the file system.

Returns The full path to the iServer working directory.

JavaScript example

```
reportContext.getAppContext().get("ServerContext")
   .getServerWorkingDirectory();
```

Java example

```
IServerContext scontext;
scontext = (IServerContext)
   reportContext.getAppContext().get("ServerContext");
scontext.getServerWorkingDirectory();
```

getUserAgentString( )

Identifies the browser used to view a report.

JavaScript syntax getUserAgentString()

Java syntax public String getUserAgentString()

Usage Use in cases when an application requires different code for different browsers. The browser information is available only when the report is rendered, so use getUserAgentString( ) in a report element's onRender event.

Returns The browser type in String format. For Internet Explorer, for example, getUserAgentString( ) might return a string, such as:

```
Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; InfoPath.1;
   MS-RTC LM 8)
```

JavaScript example

```
reportContext.getAppContext().get("ServerContext")
   .getUserAgentString();
```

Java example

```
IServerContext scontext;
scontext = (IServerContext)
   reportContext.getAppContext().get("ServerContext");
scontext.getUserAgentString();
```

getUserRoles( )

Retrieves the roles assigned to the current user.

JavaScript syntax

getUserRoles()

Java syntax

public List<String> getUserRoles()

Usage

Use in cases when an application requires different code for different iServer security roles.

Returns

The current user's security roles.

JavaScript example

```
reportContext.getAppContext().get("ServerContext")
   .getUserRoles();
```

Java example

```
IServerContext scontext;
scontext = (IServerContext)
   reportContext.getAppContext().get("ServerContext");
List<String> userRoles = scontext.getUserRoles();
```

getVolumeName( )

Retrieves the name of the iServer volume on which the report runs.

JavaScript syntax

getVolumeName()

Java syntax

public String getVolumeName()

Usage

Use in cases when an application running in a multi-volume environment requires volume information, for example, to implement logging in a report.

Returns

The name of the iServer volume running a report.

JavaScript example

```
reportContext.getAppContext().get("ServerContext")
   .getVolumeName();
```

Java example

```
IServerContext scontext;
scontext = (IServerContext)
   reportContext.getAppContext().get("ServerContext");
scontext.getVolumeName();
```

setHeadline( )

Sets the headline of a generated report. A headline appears in a job completion notice that iServer writes to a channel.

JavaScript syntax

setHeadline(headline)

Java syntax

public void setHeadline(String headline)

Argument

headline
A string that represents the headline of a completed job.

Usage

Use to specify a headline based on the contents of a report, or on the value of a report parameter.

JavaScript example

```
reportContext.getAppContext().get("ServerContext")
   .setHeadline("Sales Report for " + params["Region"].value);
```

Java example

```
String region = (String)reportContext.getParameterValue("Region")
IServerContext scontext;
scontext = (IServerContext)
   reportContext.getAppContext().get("ServerContext");
scontext.setHeadline("Sales Report for " + region);
```

setVersionName( )

Sets the version name of a generated report.

JavaScript syntax

setVersionName(versionName)

Java syntax

public void setVersionName(String versionName)

Argument

versionName
A string that represents the report's version name.

Usage

Use to specify a version name that includes dynamic data, such as the contents of a report, the value of a report parameter, or the report-generation date.

JavaScript example

```
reportContext.getAppContext().get("ServerContext")
   .setVersionName("Version " + new Date());
```

Java example

```
IServerContext scontext;
scontext = (IServerContext)
   reportContext.getAppContext().get("ServerContext");
scontext.setVersionName("Version " + new Date());
```

Part Three

Deploying reports and resources

Chapter

18

Deploying BIRT reports to Actuate BIRT iServer

This chapter contains the following topics:

- About deploying BIRT reports
- Publishing a report resource to Actuate BIRT iServer
- Deploying Java classes used in BIRT reports
- Installing a custom JDBC driver
- Installing custom ODA drivers and custom plug-ins

About deploying BIRT reports

This chapter describes how to run and distribute BIRT reports in the Actuate business reporting system. To deploy BIRT reports, you need to understand the environment in which the reports run.

Actuate Information Console provides a central location from which business users can access, run, and view reports. You can use Actuate Information Console to run report executables, and to manage, generate, view, and print report documents.

Actuate Information Console is available in Actuate BIRT iServer, a report server that can store and manage the scheduling, versioning, and archiving of large numbers of reports. iServer uses the Actuate BIRT option to run and distribute BIRT reports.

Actuate BIRT Designer Professional, the tool that you use to develop BIRT reports, has built-in capabilities that facilitate the deployment process. The Actuate BIRT Designer integrates with iServer in several important ways to support performing the following tasks:

- Use an Open Data Access (ODA) information object data source that resides on an Encyclopedia volume.
- Publish a report design to an Encyclopedia volume.
- Publish a resource to an Encyclopedia volume.
- Install a custom JDBC driver for use by BIRT reports running in the iServer environment.

A user accesses BIRT Studio from Actuate Information Console. BIRT Studio is a licensed option of iServer. To deploy templates and reports to BIRT Studio you use the deployment features available in Actuate BIRT Designer Professional. The following sections describe these capabilities.

Publishing a report to Actuate BIRT iServer

The purpose of publishing a report to iServer is to make it accessible to a large number of users. A published report is available to manage, meaning you can schedule re-running the report to include updates from the data sources. You can also choose who can access part or all of the report.

Actuate BIRT Designer Professional provides tools for easy deployment of reports, templates and their resources to iServer. The designer connects directly to an iServer and deploys the reports to selected iServer folders. The designer provides an iServer Explorer view for managing iServer connections. Using iServer Explorer, you can create iServer connection profiles to store the

connection properties to a specific Encyclopedia volume. Figure 18-1 shows iServer Explorer displaying an iServer profile.

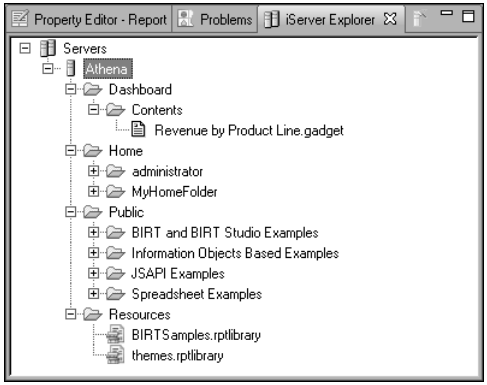


Figure 18-1 iServer Explorer view

How to create a new iServer profile

1. In Actuate BIRT Designer, open iServer Explorer. If you do not see the iServer Explorer view in the designer, select Windows➤Show view➤iServer Explorer.
2. In iServer Explorer, right-click Servers, and choose New iServer Profile.
3. In New iServer Profile, specify the connection information. Figure 18-2 displays an example of connection properties provided for an iServer named Athena.

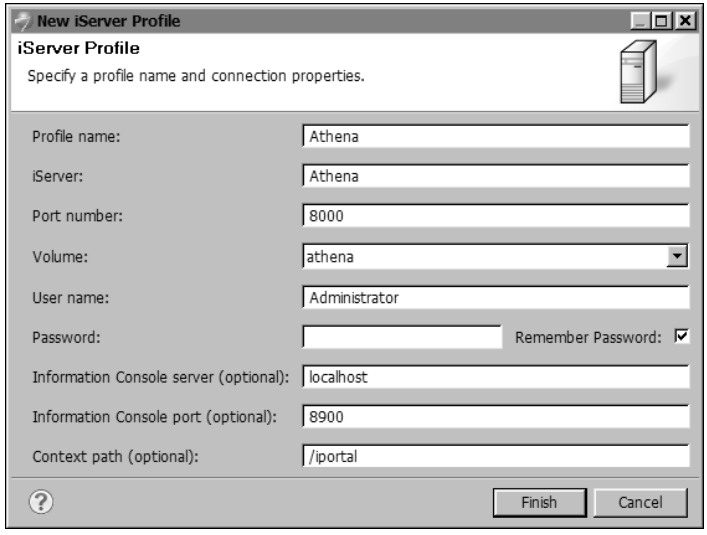


Figure 18-2 Setting properties in a new iServer profile

1. In Profile name, type a unique name that identifies the new profile.
2. In iServer, type the name or IP address of the computer on which iServer is installed.
3. In Port number, type the number of the port to access iServer.
4. In Volume, select the iServer Encyclopedia volume.
5. In User name, type the user name required to access the volume.
6. In Password, type the password required to access the volume.

4. Choose Finish to save the iServer profile. The iServer profile appears in the iServer Explorer as shown in Figure 18-1.

How to publish a report design to iServer

1. Choose File➤Publish Report to iServer.
2. On Publish Report Designs, select the report, as shown in Figure 18-3.

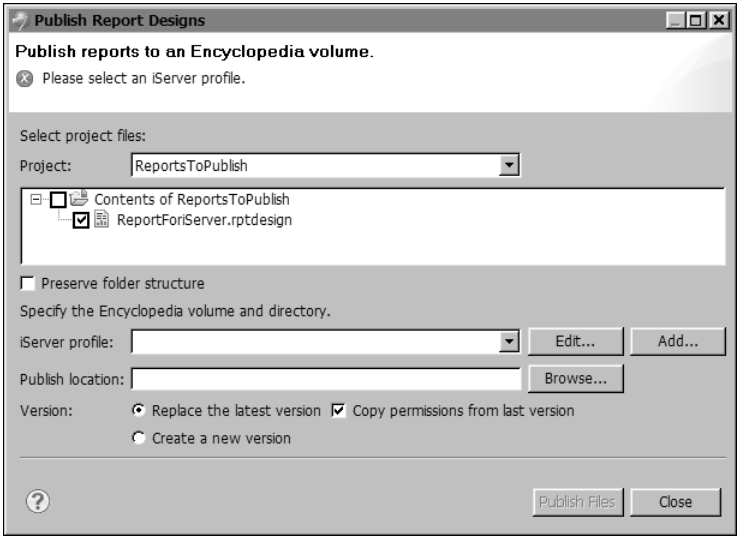


Figure 18-3 Selecting a report to publish

3. If there is no appropriate profile in the iServer profile list, create a new profile by choosing Add. In New iServer Profile, complete the information in the New iServer Profile, as shown in Figure 18-2. Then, choose Close.
4. In Publish Report Designs, in Publish location, type or browse for the location on the Encyclopedia volume in which to publish the report design, as shown in Figure 18-4.

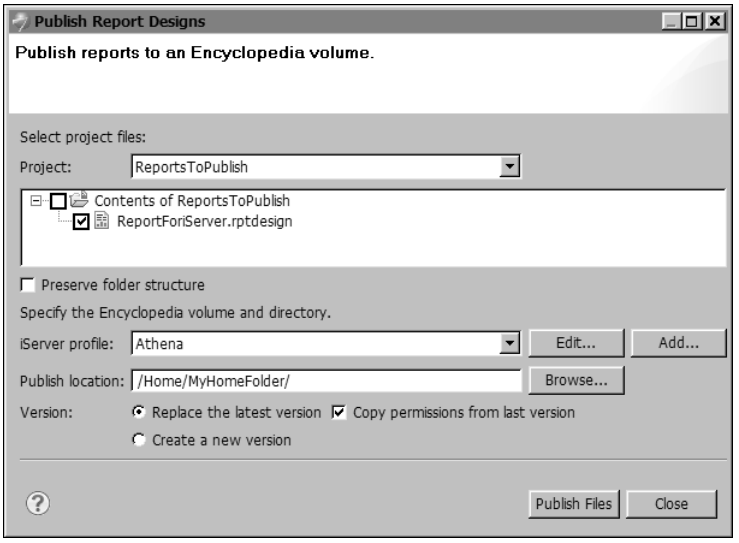


Figure 18-4 Selecting a server and location

5 Choose Publish Files. A window showing the file upload status appears.

In Publishing, wait until the upload finishes, then choose OK, as shown in Figure 18-5.

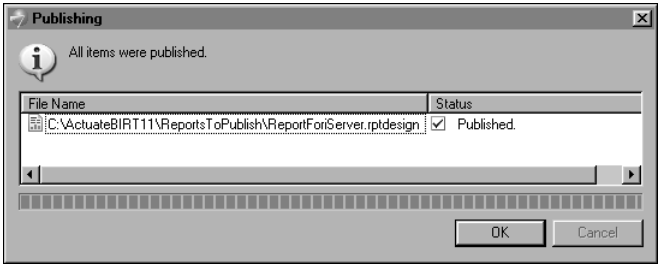


Figure 18-5 Confirming the report publishing

6 In Publish Report Designs, choose Close.

Publishing a report resource to Actuate BIRT iServer

BIRT reports frequently use files with additional information to present report data to the users. A BIRT resource is any of the following items:

- Static image that a BIRT report design uses
- Report library

- Properties file
- Report template
- Data object
- CSS file
- External JavaScript file
- SWF file of a flash object
- Java Event Handler class packaged as a Java Archive (JAR) file

You can publish BIRT resources from the BIRT Report Designer's local resource folder to iServer. By default, the Resource folder is the current report project folder. If you use shared resources with other developers and the resource files for your reports are stored in a different folder, you can change the default Resource folder. Use Windows➤Preferences➤Report Design➤Resource menu to set the resource folder.

In the Encyclopedia volume, the Resource folder is set to /Resources by default. In the sample Encyclopedia volume, the /Public folder contains sample reports. The libraries and templates used by these sample reports are stored in the /Resources folder.

If the resource folder in the Encyclopedia volume is different from the default, before publishing a resource, you need to set up the Resource folder in the Encyclopedia volume.

How to change the Resource folder on an Encyclopedia volume

1. Open Management Console and log in to the Encyclopedia volume.
2. Create a folder to designate as a resource folder.
3. Choose Volume.
4. On Volume, choose Properties.
5. On Properties—General, in Resource folder, type or browse to the folder to which you want to publish BIRT resources.

How to publish a resource from the Resource folder to iServer

1. In Actuate BIRT Designer, choose File➤Publish Resource to iServer.
2. On Publish Resources, expand the Actuate BIRT Designer's Resource Folder and select the resources to publish.
3. Select the iServer profile, as shown in Figure 18-6.

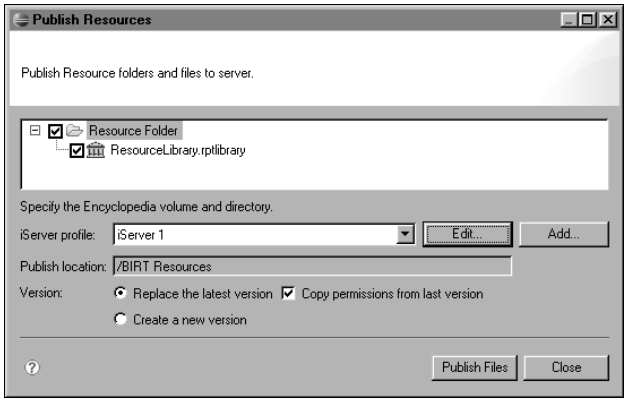


Figure 18-6 Publish Resources dialog

4 Choose Publish Files. A new window showing the file upload status appears.

5 Choose OK when the upload finishes.

6 In Publish Resources, choose Close.

Deploying Java classes used in BIRT reports

A BIRT report uses scripts to implement custom functionality. For example, you can use scripts to create a scripted data set or to provide custom processing for a report element. When you deploy a BIRT report to an Encyclopedia volume, you must provide iServer with access to the Java classes that the scripts reference. You package these classes as JAR files that can be recognized and processed from an iServer Java factory process. There are two ways to deploy Java classes:

- Deploy the JAR files to the Encyclopedia volume.

  This method supports creating specific implementations for each volume in iServer. This method of deployment requires packaging the Java classes as a JAR file and attaching the JAR file as a resource to the report design file. You treat a JAR file as a resource in the same way as a library or image. Using this method, you publish the JAR file to iServer every time you make a change in the Java classes.

- Deploy the JAR files to the following iServer subdirectory:

  `$ACTUATE_HOME\iServer\resources`

  This method uses the same implementation for all volumes in iServer. Actuate does not recommend deploying JAR files to an iServer /resources folder because iServer must be restarted after deploying the JAR file. Another disadvantage of this deployment technique is that the JAR file, deployed in the iServer /resources directory is shared across all volumes, which can cause

conflicts if you need to have different implementations for different volumes. When using this method, you do not have to add the JAR file to the report design's Resource property.

How to configure BIRT reports and deploy a JAR file to an Encyclopedia volume

1. Package the Java classes as a JAR file and copy the JAR file to the Actuate BIRT Designer resource folder.
2. Open the report design in Actuate BIRT Designer.
3. In Outline, select the root report design slot and select Resources property group in the Property Editor window.
4. In Resources, in JAR files, choose Add and navigate through the Resource folder to select the JAR file, as shown on Figure 18-7.

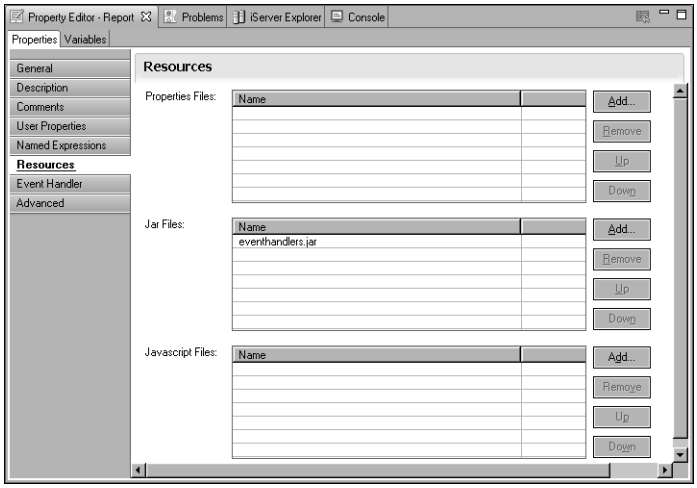


Figure 18-7 Add JAR file as a resource to a report

When the report executes, the engine searches for the required classes and methods only in this JAR file.

5. Choose File➛Publish Resource to iServer to publish the JAR file to the iServer. Select the server profile and the JAR file. The JAR file is stored in the Encyclopedia volume's Resource folder.
6. Choose File➛Publish Report to iServer to publish your BIRT report design to iServer.
7. Run the report from Information Console or Management Console.

How to deploy a JAR file to an iServer /resources folder

1. Copy the JAR file to the following iServer subdirectory:

```
$ACTUATE_HOME\iServer\resources
```

$ACTUATE_HOME is the folder where Actuate products install. By default, it is C:\Program Files\Actuate11 for version 11.

2. Publish the report to iServer as described in "Publishing a report resource to Actuate BIRT iServer," earlier in this chapter.
3. Restart iServer.
4. Run the report from Information Console or Management Console.

Installing a custom JDBC driver

In order to run a BIRT application in the iServer environment when the BIRT application uses a custom JDBC driver, you must install the JDBC driver in the following location:

```
$ACTUATE_HOME\Jar\BIRT\platform\plugins
   \org.eclipse.birt.report.data.oda.jdbc_<VERSION>\drivers
```

Installing custom ODA drivers and custom plug-ins

All custom ODA drivers and custom plug-ins must be installed in the following folder:

```
$ACTUATE_HOME\iServer\MyClasses\eclipse\plugins
```

By default, Actuate iServer and Information Console load custom plug-ins from this folder. If your application uses a different location to store custom plug-ins, you must set this location in each product's link file. Actuate products use link files to locate folders where the custom plug-ins are deployed. The name of the link files are customPlugins.link and customODA.link. As the file names suggest, the customODA.link file stores the path for custom ODA drivers, and customPlugins.link is for all plug-ins used by BIRT reports and the BIRT engine, such as custom emitters, or flash object library plug-ins. Typically, the link files are stored in a \links subfolder of the Eclipse instance of the product. For Actuate BIRT Designer, for example, the file is located in:

```
$ACTUATE_HOME\BRDPro\eclipse\links
```

You can change the path in customPlugins.link file and deploy the plug-ins to the new location.

When you install the InformationConsole.war file to your own J2EE application server, the shared folder MyClasses is not available. In this case, custom plug-ins should be copied to this folder:

```
WEB-INF/platform/dropins/eclipse/plugins
```

The locations of the link files for each product are listed in Table 18-1.

Table 18-1 .link files locations

Product	Paths of .link files
Actuate BIRT Designer Professional	$ACTUATE_HOME\BRDPro\eclipse\links
Actuate iServer	$ACTUATE_HOME\iServer\Jar\BIRT \platform\links
Information Console	WEB-INF/platform/dropins/eclipse/plugins

Chapter

19

Configuring data source connections in Actuate BIRT iServer

This chapter contains the following topics:

- About data source connection properties
- Using a connection configuration file
- Using a connection profile
- Accessing BIRT report design and BIRT resources paths in custom ODA plug-ins

About data source connection properties

Every BIRT data source object specifies the connection properties required to connect to an underlying data source. Typically, many report designs access the same data source. Rather than typing the same connection properties for each design, you can create a connection profile to reuse the same connection properties across multiple designs. The connection profile includes information about the database driver, a user ID, password, port, host, and other properties related to the type of data source.

Typically, you change database connection properties used in the development environment when you publish the reports to Actuate BIRT iServer. To change the connection properties dynamically when you design or deploy your reports, you can use one of two approaches, connection configuration file or connection profile. The following sections describe these two approaches.

Using a connection configuration file

A connection configuration file is an XML file that sets the data source connection properties to use when iServer runs a report. Externalizing data source connection information in this file enables an administrator to modify these settings without modifying the report.

iServer expects the configuration file to use UTF8 encoding. You also can use a file that only has ASCII characters. The settings that you use in a connection configuration file override the settings in a report.

Setting up the connection configuration file

In a BIRT report design, the configuration key that specifies a data source is the concatenation of the ODA plug-in's data source extension ID and the data source design name separated by an underscore (_) character.

The connection property names are the connection properties defined for your data source. To find the correct names for the connection properties, check the data source definition in the XML source of the BIRT report design file. You can view the report's XML source by selecting the XML Source tab in the report editor.

The code example in Listing 19-1 shows the XML definition of a MySQL Enterprise database in a report design. Using this data source, the report developer can test the report in development. This XML data source definition specifies the connection properties, odaDriverClass, odaURL, odaUser, odaPassword, for the data source, ClassicModels.

Listing 19-1 The XML definition of a MySQL Enterprise database

```
<data-sources>
   <oda-data-source extensionID=
   "org.eclipse.birt.report.data.oda.jdbc" name="Customers"
   id="4">
      <property name="odaDriverClass">
         com.mysql.jdbc.Driver
      </property>
      <property name="odaURL">
         jdbc:mysql://localhost/ClassicModels
      </property>
      <property name="odaUser">root</property>
      <property name="odaPassword">pwd</property>
   </oda-data-source>
</data-sources>
```

At run time, the report uploaded to iServer connects to a production database that resides on a different database server. The connection properties specify a machine IP address, 192.168.218.226, and a different username and password. To externalize the database connection information, create the configuration property file, DBConfig.xml, with the settings shown in Listing 19-2.

Listing 19-2 A configuration property file that connects to a production database

```
<Config>
<Runtime>
<ConnectOptions
   Type="org.eclipse.birt.report.data.oda.jdbc_Customers">
   <Property PropName="odaDriverClass">
      com.mysql.jdbc.Driver
   </Property>
   <Property PropName="odaURL">
      jdbc:mysql://192.168.218.226:3306/ClassicModels
   </Property>
   <Property PropName="odaUser">operator</Property>
   <Property PropName="odaPassword">pwd</Property>
</ConnectOptions>
</Runtime>
</Config>
```

Understanding how iServer uses the connection configuration file

When the report runs, iServer searches the path in the configuration file parameter for the configuration file that contains the valid ConnectOptions values. The Factory process reads the configuration file containing the ConnectOptions values when the process starts. Factory processes that are

running when you change the configuration file do not use the new information. Only Factory processes that start after the configuration file changes use the new information. To ensure that a report executable file uses the updated configuration file information, confirm that no reports are active and stop all currently running Factory processes before you change the configuration file. After you change the file, iServer starts a Factory process for the next report request using these settings.

Setting the location of a connection configuration file

There is no default location for the connection configuration file. iServer uses the value of the configuration file parameter to locate the connection configuration file.

If you do not specify a value for this parameter, iServer uses the database connection properties in the report executable file. When you set or change the value of the configuration file parameter, you must restart iServer for the change to take effect.

On a Windows operating system, the configuration file parameter can specify a path and file name or a URL. For example:

```
C:\BIRTRptConfig\DBConfig.xml
```

or:

```
http://myserver/configs/testconfig.xml
```

On a UNIX or Linux operating system, the parameter value can only be a path and file name. The parameter value cannot be a URL.

If you have an iServer cluster, each iServer in the cluster must have access to the file. You must use a local absolute path for each machine in the cluster. If you use a single copy of the file for a cluster, put the file in a shared location and set the path to that shared location for all iServers in the cluster.

How to set up a configuration file in iServer Configuration Console

To set up a connection configuration file, create the file and specify the name and location using the ConnConfigFile parameter in iServer Configuration Console.

1. Log in to iServer Configuration Console.
2. From the banner, select Advanced view.
3. From the side menu, select Server Configuration Templates.
4. In Server Configuration Templates, select the name of the template to modify.
5. In Properties Settings, select iServer to expand the property list.
6. In the iServer property list, choose Database Connection Configuration File.

7 In Servers—Properties—Runtime, in Configuration file for database connections and search path, type the configuration file's location, as shown in Figure 19-1.

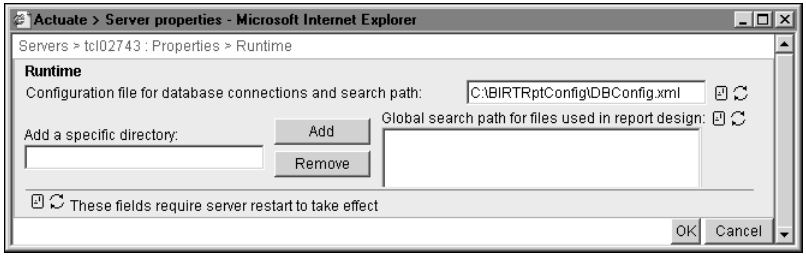


Figure 19-1 Specifying the location of a connection file

8 Restart iServer.

Encrypting the connection properties

Actuate BIRT supports the encryption of connection properties in the connection configuration file. The encryption conversion is created in Actuate BIRT Designer, using BIRT's encryption framework. The encryption user interface reads a user-specified configuration file, and writes the encrypted values for a specified property type to a new output file. The configuration file must have the file name extension .acconfig.

The runtime decryption processing runs in Actuate BIRT Designer, iServer, and Actuate Java Component. You must deploy the encrypted version of a configuration file to the iServer or Actuate Java Component environments, and set up the database configuration for iServer.

For more information about the BIRT encryption mechanism, see Chapter 21, "Working with BIRT encryption in Actuate BIRT iServer."

How to encrypt a configuration file in BIRT Designer

This procedure assumes you have already created a connection configuration file with an extension .acconfig. Listing 19-3 shows an example of connection properties specified for Microsoft SQL server. The properties are not encrypted.

Listing 19-3 Connection configuration file

```
<?xml version="1.0" encoding="UTF-8"?>
<Config>
   <Runtime>
      <ConnectOptions
         Type='org.eclipse.birt.report.data.oda.jdbc_Athena'>
```

(continues)

```
            <Property PropName='odaDriverClass'>
               com.actuate.jdbc.sqlserver.SQLServerDriver
            </Property>
            <Property PropName='odaURL'>
               jdbc:actuate:sqlserver://
                  Athena:1433;databasename=Financials
            </Property>
            <Property PropName='odaUser'>
               fmanager
            </Property>
            <Property PropName='odaPassword'>password</Property>
      </ConnectOptions>
   </Runtime>
</Config>
```

1. In Actuate BIRT Designer, choose File—Encrypt Property values from the main menu.
2. In Encrypt property values, in Connection configuration file name, choose Browse and select the connection file to encrypt.
3. Select the properties to encrypt.
4. In Encryption extension, select the encryption algorithm.
5. In Save as file name, type or click Browse to specify the file path and name for the encrypted connection file.

   Figure 19-2 shows an example of encryption options specified for a connection configuration file.

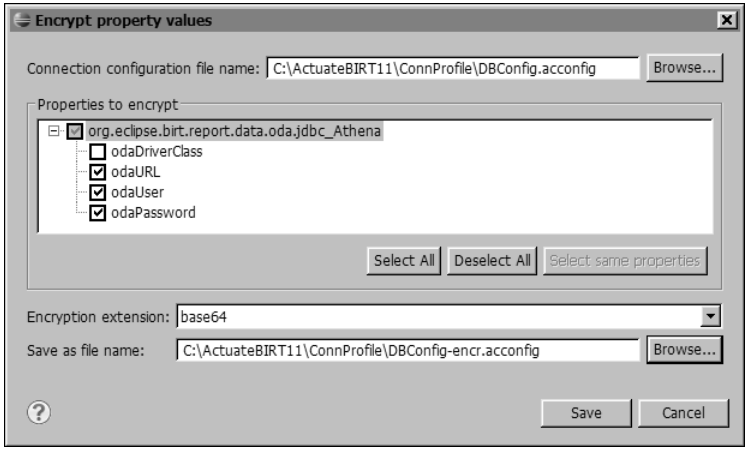


Figure 19-2 Encrypting property values

6. Choose Save to encrypt the properties. The encrypted configuration file looks like the one in Listing 19-4.

Listing 19-4 Encrypted connection configuration file

```
<?xml version="1.0" encoding="UTF-8"?>
<Config>
    <Runtime>
        <ConnectOptions
   Type='org.eclipse.birt.report.data.oda.jdbc_Athena'>
            <Property PropName='odaDriverClass'>
                com.actuate.jdbc.sqlserver.SQLServerDriver
            </Property>
            <Property PropName='odaURL'>
                amRiYzphY3R1YXRlOnNxbHNlcnZlcjov....
            </Property>
            <Property PropName='odaUser'>
                Zm1hbmFnZX...
            </Property>
            <Property PropName='odaPassword'>
                cGFzc3...
            </Property>
            <Property PropName='encryptedPropNames'>
                odaURL|odaUser|odaPassword
            </Property>
            <Property PropName='encryptionExtName'>
                base64
            </Property>
        </ConnectOptions>
    </Runtime>
</Config>
```

The encryption feature encrypts the selected properties and adds two new properties to the connection options. The encryptedPropNames property specifies the list of encrypted properties, each property separated by |. The encryptionExtName property specifies the encryption algorithm.

Using a connection profile

A connection profile contains all of the necessary information to allow a BIRT report to connect to a data source. BIRT supports using a connection profile when creating a data source in a report design. When the connection profile changes, the BIRT report picks up those changes. This behavior makes migration from a test to a production environment straightforward.

The connection profile is stored in the .metadata folder in your workspace. The default name of the connection profile is ServerProfiles.dat. You can use the Export feature in Data Source Explorer to create a connection profile with a

different name. All the connection profile properties can be bound to report parameters or expressions and changed when the report is generated.

When deploying reports that use connection profiles, you must deploy the connection profile to the correct folder in the file system on the iServer machine. For example, if a report uses a connection profile stored in a folder C:\ActuateBIRT11\ConnProfile\MySQL.dat, as shown in Figure 19-3, you must manually create the same folder structure on the iServer machine and copy the MySQL.dat file there.

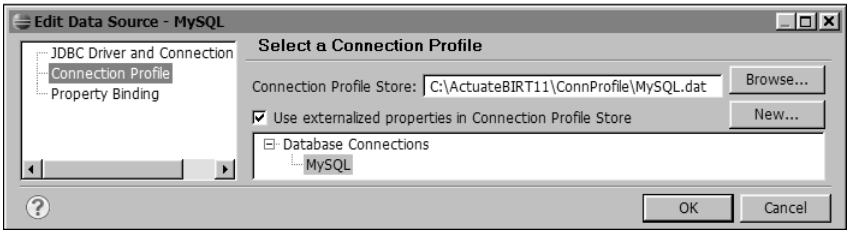


Figure 19-3 Connection profile properties in BIRT Designer

The next section shows how to bind a parameter to change the connection information. You can also use the Property Binding feature to specify a JavaScript expression for the value of OdaConnProfileStorePath property. This feature provides the flexibility to define a different root path for different file properties. For example, the JavaScript expression can include a variable to control the root path:

```
config[ "birt.viewer.working.path" ].substring(0,2) +
   "../../data/ProfileStore.dat"
```

Alternatively, you can use a reportContext object to pass session information and build the path expression.

For more information about creating and managing connection profiles, see *BIRT: A Field Guide*.

Binding connection profile properties

All the connection profile properties can be bound to report parameters or expressions and updated when the report is generated.

Binding Connection Profile Store URL property

The report developer can use the Property Binding feature in the BIRT Data Source editor to assign a dynamic file path or URL to the Connection Profile Store URL property at report run time without changing the report design itself.

The connection profile store URL's property name is OdaConnProfileStorePath. The next example shows you how to bind the connection profile store URL property to a report parameter.

How to bind the connection profile store URL property to a report parameter

This example shows how to create a report design that uses a CSV file as a data source, using Actuate BIRT Designer Professional. At design time, the report design uses the CSV file in the folder, C:\ConnProfile.

Typically, the design time CSV file contains only a few records. In the production environment, the CSV file, which contains more records, is in the folder, S:\ConnProfile. This example report design takes the full path to the connection profile as a parameter. In this way, the report runs as expected in development and production environments.

1. In Actuate BIRT Designer, to ensure that Data Source Explorer is open, choose Window➤Show View➤Other.
2. On Show View, expand Data Management. If Data Source Explorer is active, select this item. Then, choose OK. If Data Source Explorer is not active, close Show View.
3. In Data Source Explorer, right-click Flat File Data Source. Choose New.
4. Name the Flat File Data Source connection profile CSVFlatFile. Choose Next.
5. In folder, type C:\ConnProfile. This folder in your development environment contains the csvTestODA.csv file. Choose Next.
6. Choose Finish.

   The CSVFlatFile connection profile properties are shown on Figure 19-4.

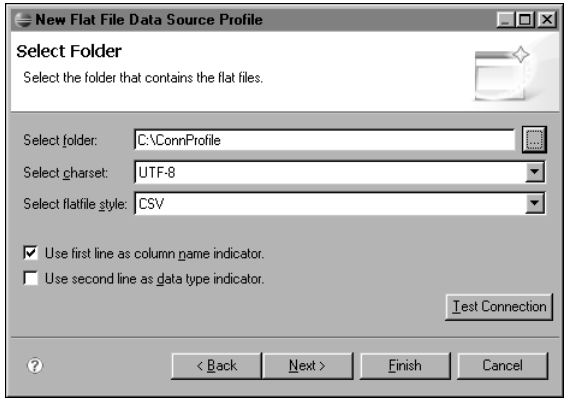


Figure 19-4 Connection profile properties dialog

A data sample from csvTestODA.csv file is shown on Listing 19-5.

Listing 19-5 csvTestODA.csv

```
PRODUCTNAME,QUANTITYINSTOCK,MSRP
1969 Harley Davidson Ultimate Chopper,7933,95.7
1952 Alpine Renault 1300,7305,214.3
1996 Moto Guzzi 1100i,6625,118.94
2003 Harley-Davidson Eagle Drag Bike,5582,193.66
1972 Alfa Romeo GTA,3252,136
```

7 Export the connection profile to C:\ConnProfile\CSVProfile.dat, as shown in Figure 19-5. For simplicity, save the profile to the same location that contains the data source CSV file.

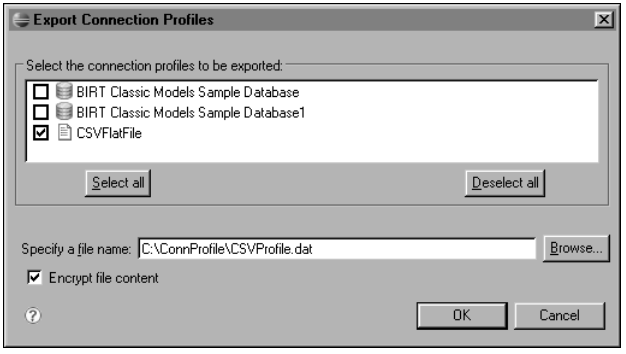


Figure 19-5 Export Connection Profile

8 From the main menu, choose Data➛New Data Source. On New Data Source, select Create from a connection profile in the profile store. Then, choose Next. As shown in Figure 19-6, browse to and select the development connection profile, C:\ConnProfile\CSVProfile.dat.

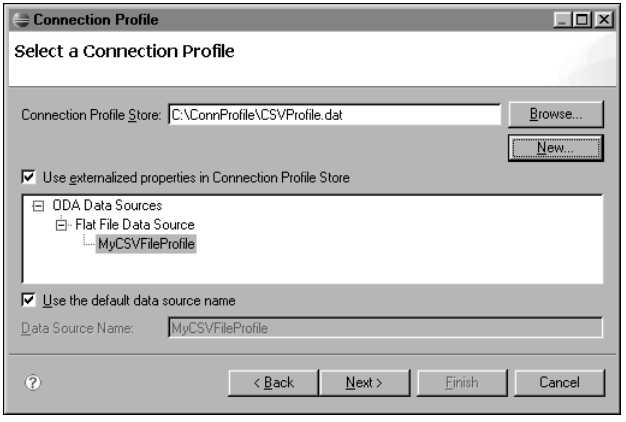


Figure 19-6 Select a connection profile

Choose Next. Then, choose Finish.

9 From the main menu, choose Data➛New Data Set➛New Data Set. Choose Next. Include all the available columns from the csvTestODA.csv file. Choose Finish. Choose OK.

10 Add a table containing all the columns in the data set to the report layout.

11 Preview the report. The report shows only the six data rows from the development CSV file, as shown in Figure 19-7.

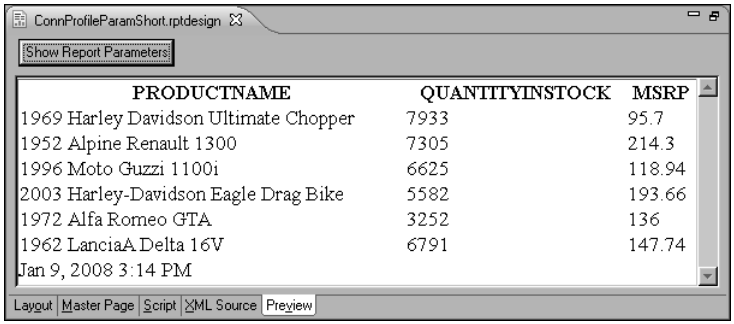


Figure 19-7 Report preview

12 In Data Explorer, right-click Report Parameters and choose New Parameter. Add a report parameter, named ConnProfileURL, using the default options, as shown in Figure 19-8. Choose OK.

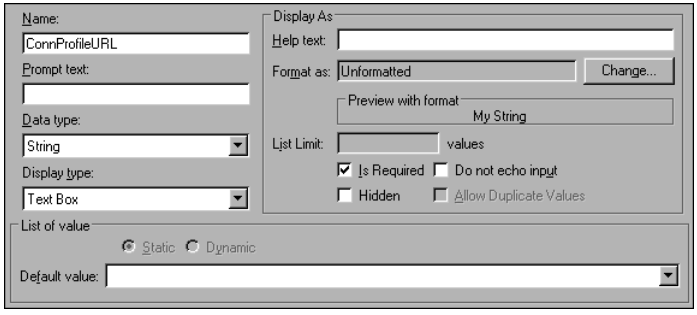


Figure 19-8 Creating a parameter

13 In Data Explorer, double-click the CSVFlatFile data source to open the Properties dialog. In the left frame, select Property Binding. To bind the ConnectionProfileStoreURL property to the report parameter, in Connection Profile Store URL, type the following expression:

```
params["ConnProfileURL"]
```

The property binding page looks like the one on Figure 19-9.

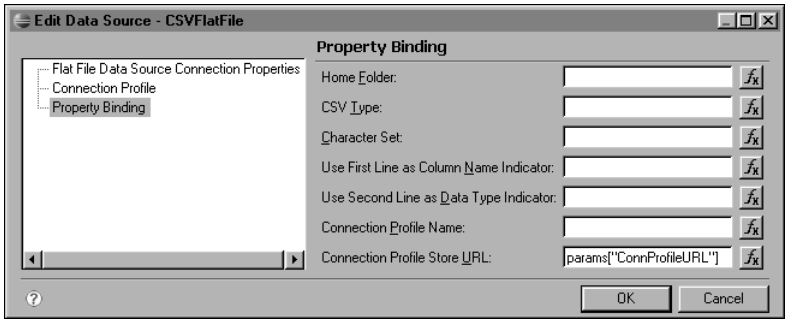


Figure 19-9 Data Source Property Binding

14 In Data Source Explorer, right-click the CSVFlatFile data source type and choose Properties. As shown in Figure 19-10, in Select folder, type the path of the production environment:

```
S:\ConnProfile
```

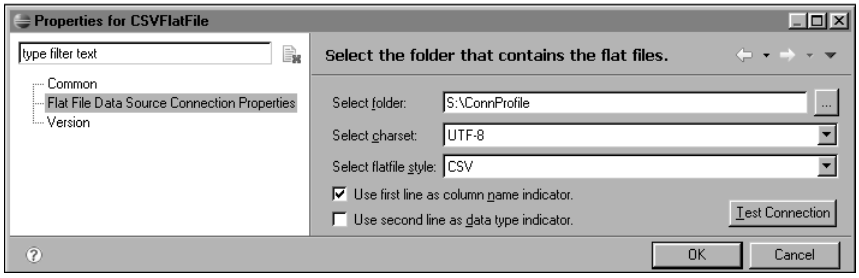


Figure 19-10 Connection profile properties

15 Export this connection profile to the production environment. As shown in Figure 19-11, in Specify a file name, type:

```
S:\ConnProfile\CSVProfileProduction.dat
```

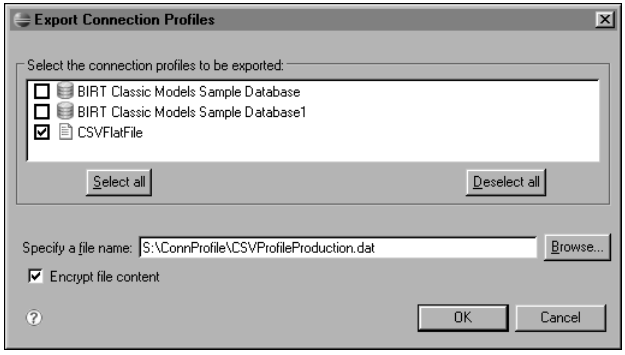


Figure 19-11 Export the production connection profile

16 Preview the report and type the location of the production connection profile as the value for the parameter:

```
S:\ConnProfile\CSVProfileProduction.dat
```

The report shows the full set of data rows from the production profile, similar to Figure 19-12.

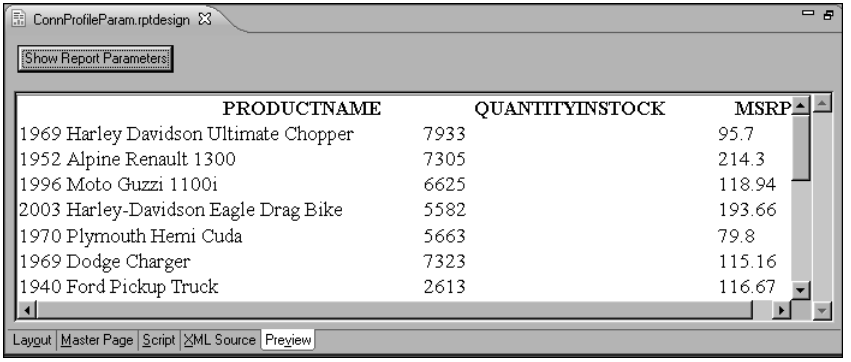


Figure 19-12 Production report preview

Binding a connection profile name to a report parameter

You can externalize the connection profile name for a data source by binding it to a report parameter. For example, you have two JDBC connection profiles to the same database using different user names and passwords. At run time, you can select the profile you want to use to connect to the database. The connection profile name property is OdaConnProfileName, as shown in Listing 19-6.

Listing 19-6 Data source definition in a BIRT report

```
<data-sources>
   <oda-data-source
   extensionID="org.eclipse.datatools.connectivity.oda.flatfile"
   name="CSVFlatFile" id="8">
      <property name="HOME">C:\ConnProfile</property>
      <property name="DELIMTYPE">COMMA</property>
      <property name="CHARSET">UTF-8</property>
      <property name="INCLTYPELINE">NO</property>
      <property name="OdaConnProfileName">CSVFlatFile</property>
      <property name="OdaConnProfileStorePath">
         C:\testABIRT\metadata\.plugins\
   org.eclipse.datatools.connectivity\ServerProfiles.dat
      </property>
   </oda-data-source>
</data-sources>
```

Externalizing the connection profile properties on the iServer

The iServer database connection configuration file is used to externalize the data source properties for any data source connection property in a BIRT report design. The data source properties are externalized in the connection configuration file that is made accessible to the iServer.

As the connection profile store URL is the ODA data source property, OdaConnProfileStorePath, the file path to the Connection Profile can itself be externalized. To externalize the report design's Connection Profile Store URL ODA data source property, specify it in the iServer's connection configuration file. When the report is deployed to the iServer and executed, the server reads the connection profile from the file path specified in the iServer's database connection configuration file. The file path specified in the report design is ignored.

Understanding externalization precedence

Data source properties in a report design can be externalized to the connection profile and to the iServer connection configuration file. In addition, the Connection Profile Store URL itself can be externalized. The following precedence rules explain how iServer and Information Console determine the final list of data source properties for report execution:

- Information Console

  Data source properties in the connection profile override the data source properties in the report design.

- iServer

  Data source properties in the iServer connection configuration file override the data source in the connection profile that overrides the data source connection properties in the report. The ascending order of precedence for iServer is as follows:

  - Data source properties in the report design
  - Data source properties in the connection profile
  - Data source properties in the iServer connection configuration file

The following sample connection configuration file externalizes the file path to the connection profile and shows the required structure:

```
<Config>
<Runtime>
<ConnectOptions Type="org.eclipse.birt.report.data.oda.jdbc_SQL
   Server Data Source">
   <Property PropName="OdaConnProfileStorePath">
      C:\SqlServer.profile
```

```
   </Property>
</ConnectOptions>
</Runtime>
</Config>
```

The connection profile referenced by the BIRT report design is read when the report is executed in Information Console and iServer. The path to the connection profile in the design has to be visible to the Information Console and iServer applications.

Referencing the external connection profile

The path to the external connection profile is stored in the BIRT report design. The ODA data source property, Connection Profile Store URL, holds this value. The path can be a relative or an absolute file path, or a URL. File paths, whether relative or absolute, must be accessible by the Information Console web application when the report is deployed to Information Console. Similarly this path must be accessible by the iServer when the report is deployed to the iServer. Actuate does not recommend the use of relative file paths. Typically, the location of the connection profile in all three environments, Actuate BIRT Designer, Information Console, and iServer, resolves to a different path. Absolute paths have the disadvantage that the absolute path used in the Actuate BIRT Designer environment on Windows will not be available when the report is deployed to Information Console or iServer on UNIX. On UNIX, you can use relative paths with the use of soft links, but these links are not available on Windows.

When the file path to the connection profile is different in the design environment compared to the Information Console and iServer deployment environments, there are some options to avoid having to change the report design file before deployment, as described in the following sections.

When specifying network paths in BIRT reports always use the Universal Naming Convention (UNC) to describe the path, instead of a mapped drive letter. Windows XP and later do not allow processes running as services to access network resources through mapped network drives. For this reason, a report that uses a mapped drive letter to access a resource runs in Actuate BIRT Designer Professional, but fails when the report runs on iServer or Information Console, because the iServer or Information Console processes cannot resolve the mapping address.

For example, a BIRT report uses a flat file Production.csv as a data source. The flat file is located on a shared network drive on a machine, named ProductionServer. The UNC network path to the file is \\ProductionServer\e$\Data and it is mapped as X:\ in your system. Using the path X:\ to define the data source HOME folder works only in Actuate BIRT Designer. Using the UNC path \\ProductionServer\e$\Data in the data source definition is the correct way to define network paths.

Accessing BIRT report design and BIRT resources paths in custom ODA plug-ins

ODA providers often need to obtain information about a resource path defined in ODA consumer applications. For example, if you develop an ODA flat file data source, you can implement an option to look up the data files in a path relative to a resource folder managed by its consumer. Such resource identifiers are needed at both design-time and run-time drivers.

ODA consumer applications are able to specify:

- The run-time resource identifiers and pass them to the ODA run-time driver in an application context map
- The design-time resource identifiers in a DataSourceDesign, as defined in an ODA design session model

Accessing resource identifiers in run-time ODA driver

For run time, the BIRT ODA run-time consumer passes its resource location information in a org.eclipse.datatools.connectivity.oda.util.ResourceIdentifiers instance in the appContext map. ODA run-time drivers can get the instance in any one of the setAppContext methods, such as IDriver.setAppContext:

- To get the instance from the appContext map, pass the map key ResourceIdentifiers.ODA_APP_CONTEXT_KEY_CONSUMER_RESOURCE_IDS, defined by the class as a method argument.
- To get the BIRT resource folder URI call getApplResourceBaseURI( ) method.
- To get the URI of the associated report design file folder call getDesignResourceBaseURI( ) method. The URI is application dependant and it can be absolute or relative. If your application maintains relative URLs, call the getDesignResourceURILocator.resolve( ) method to get the absolute URI.

The code snippet on Listing 19-7 shows how to access the resource identifiers through the application context.

Listing 19-7 Accessing resource identifiers at run time

```
URI resourcePath = null;
URI absolutePath = null;

Object obj = this.appContext.get(
   ResourceIdentifiers.ODA_APP_CONTEXT_KEY_CONSUMER_RESOURCE_IDS
   );
if ( obj != null )
{
   ResourceIdentifiers identifier = (ResourceIdentifiers)obj;
```

```
   if ( identifier.getDesignResourceBaseURI( ) != null )
   {  resourcePath = identifier.getDesignResourceBaseURI();

      if ( ! resourcePath.isAbsolute() )
         absolutePath =
            identifier.getDesignResourceURILocator().resolve(
            resourcePath );
   else
         absolutePath = resourcePath;
   }
}
```

Accessing resource identifiers in design ODA driver

The resource identifiers are available to the custom ODA designer UI driver. The designer driver provides the user interface for the custom data source and data set. Typically, to implement a custom behavior, the data source UI driver extends the following class:

```
org.eclipse.datatools.connectivity.oda.design.ui.wizards
   .DataSourceWizardPage
```

The DataSourceWizardPage class has an inherited method getHostResourceIdentifiers( ) that provides access to the resource and report paths. The extended DataSourceWizardPage just needs to call the base method to get the ResourceIdentifiers for its path's information. Similarly, if the custom driver implements a custom data source editor page, it extends:

```
org.eclipse.datatools.connectivity.oda.design.ui.wizards
   .DataSourceEditorPage
```

The DataSourceEditorPage class has an inherited method getHostResourceIdentifiers( ). The extended class just needs to call the base class method to get the ResourceIdentifiers object for the two resource and report paths base URIs. Related primary methods in the org.eclipse.datatools.connectivity.oda.design.ResourceIdentifiers class are:

- getDesignResourceBaseURI();
- getApplResourceBaseURI();

Chapter

20

Configuring fonts in Actuate BIRT iServer

This chapter contains the following topics:

- About configuring fonts
- Understanding font configuration file priorities
- Understanding how BIRT engine locates a font
- Understanding the font configuration file structure

About configuring fonts

Actuate Information Console and iServer support rendering BIRT reports in different formats such as PDF, Microsoft Word, Postscript, and PowerPoint. The processes that do the conversion use the fonts installed on your system to display the report characters.

BIRT uses a flexible mechanism that supports configuring font usage and substitution. This mechanism uses font configuration files for different purposes that control different parts of the rendering process. The configuration files can configure the fonts used in specific operating systems, for rendering to specific formats, in specific locales only, or combinations of these parameters.

The plug-in folder, org.eclipse.birt.report.engine.fonts, contains the font configuration files. Table 20-1 shows the location of this folder in the supported BIRT environments.

Table 20-1 Locations of the font configuration file plug-in folder

Environment	Font configuration file folder location
Actuate BIRT Designer Professional	$Actuate<release>\BRDPro\eclipse\plugins
Information Console	$Information Console\iportal\WEB-INF\platform \plugins
iServer	$Actuate<release>\iServer\Jar\BIRT\platform \plugins

Understanding font configuration file priorities

BIRT reports use five different types of font configuration files. The font configuration file-naming convention includes information about the rendering format, the system platform, and the system locale, as shown in the following general format:

```
fontsConfig_<Format>_<Platform>_<Locale>.xml
```

The platform name is defined by the Java System property, os.name. The current Java Network Launch Protocol (JNLP) specification does not list the values for the os attributes. Instead it states that all values are valid as long as they match the values returned by the system property os.name. Values that only match the beginning of os.name are also valid. If you specify Windows and the os.name is Windows 98, for example, the operating system name is accepted as valid.

The following sample Java class code shows how to check the os.name property for the value on your machine:

```
class WhatOS
{
  public static void main( String args[] )
   {
    System.out.println( System.getProperty("os.name") );
   }
}
```

BIRT supports the following types of font configuration files, with increasing priority:

- For all rendering formats

  These files have no format specifier in their names. These configuration files are divided into three sub-types.

  - The default configuration file:

    `fontsConfig.xml`

  - Configuration files for a specific platform, for example:

    `fontsConfig_Windows_XP.xml`

  - Configuration files for a specific platform and locale, for example:

    `fontsConfig_Windows_XP_zh.xml`
    `fontsConfig_Windows_XP_zh_CN.xml`

- For certain formats only

  These files include the format specifier in their names. These configuration files are divided into two sub-types:

  - The default configuration file for a format, for example:

    `fontsConfig_pdf.xml`

  - Configuration files for a format for a specific platform:

    `fontsConfig_pdf_Windows_XP.xml`

Understanding how BIRT engine locates a font

The PDF layout engine renders the PDF, Postscript, and PowerPoint formats. The engine tries to locate and use the font specified at design time. The PDF layout engine searches for the font files first in the fonts folder of the plug-in, org.eclipse.birt.report.engine.fonts. If the specified font is not in this folder, the BIRT engine searches for the font in the system-defined font folder. You can change the default load order by using the settings in the font configuration file.

When the required font for a character is not available in the search path or is incorrectly installed, the engine uses the fonts defined in the UNICODE block for that character. If the UNICODE definition also fails, the engine replaces the character with a question mark (?) to denote a missing character. The font used for the ? character is the default font, Times-Roman.

The engine maps the generic family fonts to a PDF embedded Type1 font, as shown in the following list:

- cursive font styles to Times-Roman
- fantasy font styles to Times-Roman
- monospace font styles to Courier
- sans-serif font styles to Helvetica
- serif font styles to Times-Roman

Understanding the font configuration file structure

The font configuration file, fontsConfig.xml, consists of the following major sections:

- <font-aliases>
- <composite-font>
- <font-paths>

<font-aliases> section

In the <font-aliases> section, you can:

- Define a mapping from a generic family to a font family. For example the following code defines a mapping from the generic type "serif" to a Type1 font family Times-Roman:

  ```
  <mapping name="serif" font-family="Times-Roman"/>
  ```

- Define a mapping from a font family to another font family. This definition is useful if you want to use a font for PDF rendering which differs from the font used at design time. For example, the following code shows how to replace simsun with Arial Unicode MS:

  ```
  <mapping name="simsun" font-family="Arial Unicode MS"/>
  ```

Previous versions of Actuate BIRT Designer use the XML element <font-mapping> instead of <font-aliases>. In the current release, a <font-mapping> element works in the same way as the new <font-aliases> element. When a font configuration file uses both <font-mapping> and

<font-aliases>, the engine merges the different mappings from the two sections. If the same entries exist in both sections, the settings in <font-aliases> override those in <font-mapping>.

<composite-font> section

The <composite-font> section is used to define a composite font, which is a font consisting of many physical fonts used for different characters. For example, to define a new font for currency symbols, you change font-family in the following <block> entry to the Times Roman font-family:

```
<composite-font>
…
<block name="Currency Symbols" range-start="20a0"
   range- end="20cf" index="58" font-family="Times Roman" />
…
</composite-font>
```

The composite fonts are defined by <block> entries. Each <block> entry defines a mapping from a UNICODE range to a font family name, which means the font family is applied for the UNICODE characters in that range. You cannot change the block name or range or index as it is defined by the UNICODE standard. The only item you can change in the block element is the font-family name. You can find information about all the possible blocks at http://www.unicode.org/charts/index.html.

In cases when the Times Roman font does not support all the currency symbols, you can define the substitution character by character using the <character> tag, as shown in the following example:

```
<composite-font>
    …
      <character value="?" font-family="Angsana New"/>
      <character value="\u0068" font-family="Times Roman"/>
    …
</composite-font>
```

Note that characters are represented by the attribute, value, which can be presented two ways, the character itself or its UNICODE code.

You can find information about all the currency symbols from http://www.unicode.org/charts/symbols.html.

A composite font named all-fonts is applied as a default font. When a character is not defined in the desired font, the font defined in all-fonts is used.

<font-paths> section

If the section <font-paths> is set in fontsConfig.xml, the engine ignores the system-defined font folder, and loads the font files specified in the section,

<font-paths>. You can add a single font path or multiple paths, ranging from one font path to a whole font folder, as shown in the following example:

```
<path path="c:/windows/fonts"/>
<path path="/usr/X11R6/lib/X11/fonts/TTF/arial.ttf"/>
```

If this section is set, the PDF layout engine will only load the font files in these paths and ignore the system-defined font folder. If you want to use the system font folder as well, you must include it in this section.

On some systems, the PDF layout engine does not recognize the system-defined font folder. If you encounter this issue, add the font path to the <font-paths> section.

Chapter

21

Working with BIRT encryption in Actuate BIRT iServer

This chapter contains the following topics:

- About BIRT encryption
- About the BIRT default encryption plug-in
- Creating a BIRT report that uses the default encryption
- Deploying multiple encryption plug-ins
- Deploying encryption plug-ins to iServer
- Generating encryption keys
- Creating a custom encryption plug-in
- Using encryption API methods

About BIRT encryption

BIRT provides an extension framework to support users registering their own encryption strategy with BIRT. The model implements the JCE (Java™ Cryptography Extension). The Java encryption extension framework provides multiple popular encryption algorithms, so the user can just specify the algorithm and key to have a high security level encryption. The default encryption extension plug-in supports customizing the encryption implementation by copying the BIRT default plug-in, and giving it different key and algorithm settings.

JCE provides a framework and implementations for encryption, key generation and key agreement, and Message Authentication Code (MAC) algorithms. Support for encryption includes symmetric, asymmetric, block, and stream ciphers. The software also supports secure streams and sealed objects.

A conventional encryption scheme has the following five major parts:

- Plaintext, the text to which an algorithm is applied.
- Encryption algorithm, the mathematical operations to conduct substitutions on and transformations to the plaintext. A block cipher is an algorithm that operates on plaintext in groups of bits, called blocks.
- Secret key, the input for the algorithm that dictates the encrypted outcome.
- Ciphertext, the encrypted or scrambled content produced by applying the algorithm to the plaintext using the secret key.
- Decryption algorithm, the encryption algorithm in reverse, using the ciphertext and the secret key to derive the plaintext content.

About the BIRT default encryption plug-in

BIRT's default encryption algorithm is implemented as a plug-in named:

`com.actuate.birt.model.defaultsecurity_<Release>`

Table 21-1 shows the location of this plug-in folder in the supported BIRT environments.

Table 21-1 Locations of the default encryption plug-in folder

Environment	Font configuration file folder location
Actuate BIRT Designer Professional	$Actuate<Release>\BRDPro\eclipse\plugins

Table 21-1 Locations of the default encryption plug-in folder

Environment	Font configuration file folder location
Information Console	$Information Console\iportal\webapps\iportal \WEB-INF\platform\plugins
iServer	$Actuate<Release>\iServer\Jar\BIRT\platform\ plugins

About supported encryption algorithms

Two different cryptographic methods, private-key and public-key encryptions, solve computer security problems. Private-key encryption is also known as symmetric encryption. In private-key encryption, the sender and receiver of information share a key that is used for both encryption and decryption. In public-key encryption, two different mathematically related keys, known as a key pair, are used to encrypt and decrypt data. Information encrypted using one key can only be decrypted by using the other member of the key pair. BIRT's default encryption plug-in supports the following algorithms within these two methods:

- Private-key encryption
  - DES is the Digital Encryption Standard as described in FIPS PUB 46-2 at http://www.itl.nist.gov/fipspubs/fip46-2.htm. The DES algorithm is the most widely used encryption algorithm in the world. This algorithm is the default encryption that BIRT uses.
  - DESede, triple DES encryption

    Triple-DES or DESede is an improvement over DES. This algorithm uses three DES keys k1, k2, and k3. A message is encrypted using k1 first, then decrypted using k2 and encrypted again using k3. This technique is called DESencryptiondecryptionencryption. Two or three keys can be used in DESede. This algorithm increases security as the key length effectively increases from 56 to 112 or 168.
- Public-key encryption supports the RSA algorithm

  RSA uses both a public key and a private key. The public key can be known to everyone and is used for encrypting messages. Messages encrypted with the public key can only be decrypted using the private key.

About the components of the BIRT default encryption plug-in

The BIRT default encryption plug-in consists of the following main modules:

- acdefaultsecurity.jar
- encryption.properties file

- META-INF/MANIFEST.MF
- plugin.xml

About acdefaultsecurity.jar

This JAR file contains the encryption classes. The default encryption plug-in also provides key generator classes that can be used to create different encryption keys.

About encryption.properties

This file specifies the encryption settings. BIRT loads the encryption type, encryption algorithm, and encryption keys from the encryption.properties file to do the encryption. The file contains pre-generated default keys for each of the supported algorithms.

You define the following properties in the encryption.properties file:

- Encryption type

  Type of algorithm. Specify one of the two values, symmetric encryption or public encryption. The default type is symmetric encryption.

- Encryption algorithm

  The name of the algorithm. You must specify the correct encryption type for each algorithm. For the symmetric encryption type, BIRT supports DES and DESede. For public encryption type, BIRT supports RSA.

- Encryption mode

  In cryptography, a block cipher algorithm operates on blocks of fixed length, which are typically 64 or 128 bits. Because messages can be of any length, and because encrypting the same plaintext with the same key always produces the same output, block ciphers support several modes of operation to provide confidentiality for messages of arbitrary length. Table 21-2 shows all supported modes.

Table 21-2 Supported encryption modes

Mode	Description
None	No mode
CBC	Cipher Block Chaining Mode, as defined in the National Institute of Standards and Technology (NIST) Federal Information Processing Standard (FIPS) PUB 81, "DES Modes of Operation," U.S. Department of Commerce, Dec 1980
CFB	Cipher Feedback Mode, as defined in FIPS PUB 81

Table 21-2 Supported encryption modes

Mode	Description
ECB	Electronic Codebook Mode, as defined in FIPS PUB 81
OFB	Output Feedback Mode, as defined in FIPS PUB 81
PCBC	Propagating Cipher Block Chaining

- Encryption padding

  Because a block cipher works on units of a fixed size, but messages come in a variety of lengths, some modes, for example CBC, require that the final block be padded before encryption. Several padding schemes exist. The supported paddings are shown in Table 21-3. All padding settings are applicable to all algorithms.

Table 21-3 Supported encryption paddings

Mode	Description
NoPadding	No padding.
OAEP	Optimal Asymmetric Encryption Padding (OAEP) is a padding scheme that is often used with RSA encryption.
PKCS5Padding	The padding scheme described in RSA Laboratories, "PKCS #5: Password-Based Encryption Standard," version 1.5, November 1993. This encryption padding is the default.
SSL3Padding	The padding scheme defined in the SSL Protocol Version 3.0, November 18, 1996, section 5.2.3.2.

- Encryption keys

  Actuate provides pre-generated keys for all algorithms.

Listing 21-1 shows the default contents of encryption.properties.

Listing 21-1 Default encryption.properties

```
#message symmetric encryption , public encryption.
type=symmetric encryption

#private encryption: DES(default), DESede
#public encryption:  RSA
algorithm=DES
```

(continues)

```
# NONE , CBC , CFB , ECB( default ) , OFB , PCBC
mode=ECB
# NoPadding , OAEP , PKCS5Padding( default ) , SSL3Padding
padding=PKCS5Padding

#For key , support default key value for algorithm
#For DESede ,DES we only need to support private key
#private key value of DESede algorithm : 20b0020…
#private key value of DES algorithm: 527c2qI

#for RSA algorithm , there is key pair. you should support
   private-public key pair
#private key value of RSA algorithm: 30820…

#public key value of RSA algorithm: 30819…

#private key
symmetric-key=527c23…

#public key
public-key=
```

About META-INF/MANIFEST.MF

META-INF/MANIFEST.MF is a text file that is included inside a JAR to specify metadata about the file. Java's default ClassLoader reads the attributes defined in MANIFEST.MF and appends the specified dependencies to its internal classpath. The encryption plug-in ID is the value of the Bundle-SymbolicName property in the manifest file for the encryption plug-in. You need to change this property when you deploy multiple instances of the default encryption plug-in, as described later in this chapter. Listing 21-2 shows the contents of the default MANIFEST.MF.

Listing 21-2 Default MANIFEST.MF

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Actuate Default Security Plug-in
Bundle-SymbolicName:
   com.actuate.birt.model.defaultsecurity;singleton:=true
Bundle-Version: <release><version>
Require-Bundle: org.eclipse.birt.report.model,
 org.eclipse.core.runtime
Export-Package: com.actuate.birt.model.defaultsecurity.api
Bundle-ClassPath: acdefaultsecurity.jar
Bundle-Vendor: Actuate Corporation
Eclipse-LazyStart: true
Bundle-Activator:
   com.actuate.birt.model.defaultsecurity.properties.
   SecurityPlugin
```

About plugin.xml

plugin.xml is the plug-in descriptor file. This file describes the plug-in to the Eclipse platform. The platform reads this file and uses the information to populate and update, as necessary, the registry of information that configures the whole platform. The <plugin> tag defines the root element of the plug-in descriptor file. The <extension> element within the <plugin> element specifies the Eclipse extension point that this plug-in uses, org.eclipse.birt.report.model.encryptionHelper. This extension point requires a sub-element, <encryptionHelper>. This element uses the following attributes:

- class

  The qualified name of the class that implements the interface IEncryptionHelper. The default class name is com.actuate.birt.model.defaultsecurity.api.DefaultEncryptionHelper.

- extensionName

  The unique internal name of the extension. The default extension name is jce.

- isDefault

  The field indicating whether this encryption extension is the default for all encryptable properties. This property is valid only in a BIRT Report Designer environment. When an encryption plug-in sets the value of this attribute to true, the BIRT Report Designer uses this encryption method as the default to encrypt data. There is no default encryption mode in iServer and Information Console. The encryption model that BIRT uses supports implementing and using several encryption algorithms. The default encryption plug-in is set as default using this isDefault attribute. If you implement several encryptionHelpers, set this attribute to true for only one of the implementations. If you implement multiple encryption algorithms and set isDefault to true to more than one instance, BIRT treats the first loaded encryption plug-in as the default algorithm.

Listing 21-3 shows the contents of the default encryption plug-in's plugin.xml.

Listing 21-3 Default plugin.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.2"?>
<plugin>
   <extension
      id="encryption"
      name="default encryption helper"
      point="org.eclipse.birt.report.model.encryptionHelper">
```

(continues)

```
      <encryptionHelper
         class="com.actuate.birt.model.defaultsecurity.api
         .DefaultEncryptionHelper"
         extensionName="jce" isDefault="true" />
   </extension>
</plugin>
```

Creating a BIRT report that uses the default encryption

This section describes an example that shows how the entire mechanism works. This example uses Actuate BIRT Designer to create a report design. The report design connects to a MySQL Enterprise database server using the user, root, and password, root, as shown in Figure 21-1.

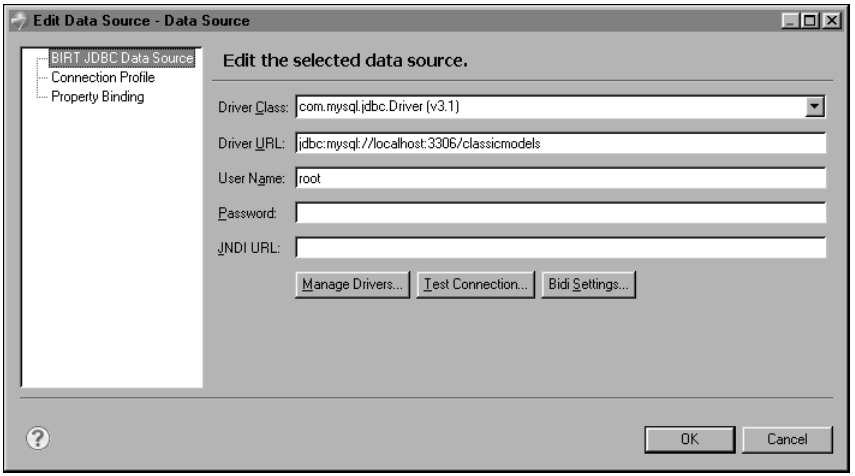


Figure 21-1 Data Source properties for the encryption example

The encryption model stores the encrypted value of the database password in the report design file. Along with the value, the model stores the encryptionID. In this way, it identifies the encryption mechanism used to encrypt the password, as shown in the <encrypted-property> element in the following code:

```
<data-sources>
<oda-data-source
   extensionID="org.eclipse.birt.report.data.oda.jdbc" name="Data
   Source" id="6">
   <property name="odaDriverClass">
      com.mysql.jdbc.Driver
   </property>
   <property name="odaURL">
      jdbc:mysql://localhost:3306/classicmodels
   </property>
```

```
   <property name="odaUser">root</property>
   <encrypted-property name="odaPassword" encryptionID="jce">
      10e52…
   </encrypted-property>
</oda-data-source>
</data-sources>
```

iServer uses the encryptionID attribute of the <encrypted-property> element to identify the algorithm to decrypt the password. After using the algorithm on the value of <encrypted-property>, iServer connects to the database and generates the report.

Deploying multiple encryption plug-ins

In some cases, you need to use an encryption mechanism other than the Data Source Explorer default in your report application. For example, some applications need to create an encryption mechanism using the RSA algorithm that the default encryption plug-in supports. In this case, you must create an additional encryption plug-in instance. For use within Actuate BIRT Designer, you can set this plug-in as the default encryption mechanism. If you change the default encryption mechanism, you must take care when you work with old report designs. For example, if you change an existing password field in the designer, the designer re-encrypts the password with the current default encryption algorithm regardless of the original algorithm that the field used.

How to create a new instance of the default encryption plug-in

1. Make a copy of the default encryption plug-in:
   1. Copy the folder:

      ```
      $ACTUATE_HOME\BRDPro\eclipse\plugins
         \com.actuate.birt.model.defaultsecurity_<Release>
      ```

   2. Paste the copied folder in the same folder:

      ```
      $ACTUATE_HOME\BRDPro\eclipse\plugins
      ```

   3. Rename:

      ```
      $ACTUATE_HOME\BRDPro\eclipse\plugins\Copy of
         com.actuate.birt.model.defaultsecurity_<Release>
      ```

      to a new name, such as:

      ```
      $ACTUATE_HOME\BRDPro\eclipse\plugins
         \com.actuate.birt.model.defaultsecurity_<Release>_rsa
      ```

2. Modify the new plug-in's manifest file:

1 Open:

```
$ACTUATE_HOME\BRDPro\eclipse\plugins
   \com.actuate.birt.model.defaultsecurity_2.3.2_rsa\
   META-INF\MANIFEST.MF
```

2 Change:

```
Bundle-SymbolicName:
   com.actuate.birt.model.defaultsecurity
```

to:

```
Bundle-SymbolicName:
   com.actuate.birt.model.defaultsecurity.rsa
```

MANIFEST.MF now looks similar to the one in Listing 21-4.

Listing 21-4 Modified MANIFEST.MF for the new encryption plug-in

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Actuate Default Security Plug-in
Bundle-SymbolicName:
   com.actuate.birt.model.defaultsecurity.rsa;singleton:=true
Bundle-Version: <Release>.<Version>
Require-Bundle: org.eclipse.birt.report.model,
 org.eclipse.core.runtime
Export-Package: com.actuate.birt.model.defaultsecurity.api
Bundle-ClassPath: acdefaultsecurity.jar
Bundle-Vendor: Actuate Corporation
Eclipse-LazyStart: true
Bundle-Activator: com.actuate.birt.model.defaultsecurity
   .properties.SecurityPlugin
```

3 Save and close MANIFEST.MF.

3 Modify the new plug-in's descriptor file to be the default encryption plug-in:

1 Open:

```
$ACTUATE_HOME\BRDPro\eclipse\plugins
   \com.actuate.birt.model.defaultsecurity_<Release>_rsa
   \plugin.xml
```

2 Change:

```
extensionName="jce"
```

to:

```
extensionName="rsa"
```

plugin.xml now looks similar to the one in Listing 21-5.

3 Save and close plugin.xml.

Listing 21-5 Modified plugin.xml for the new encryption plug-in

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="<Version>"?>
<plugin>
<extension
   id="encryption"
   name="default encryption helper"
   point="org.eclipse.birt.report.model.encryptionHelper">
   <encryptionHelper
   class="com.actuate.birt.model.defaultsecurity.api
   .DefaultEncryptionHelper"
   extensionName="rsa" isDefault="true" />
   </extension>
</plugin>
```

4 Modify the original plug-in's descriptor file, so that it is no longer the default encryption plug-in:

   1 Open:

   ```
   $ACTUATE_HOME\BRDPro\eclipse\plugins
      \com.actuate.birt.model.defaultsecurity_<Release>\
      plugin.xml
   ```

   2 Change:

   ```
   isDefault="true"
   ```

   to:

   ```
   isDefault="false"
   ```

   3 Save and close plugin.xml.

5 Set the encryption type in the new plug-in to RSA:

   1 Open:

   ```
   $ACTUATE_HOME\BRDPro\eclipse\plugins
      \com.actuate.birt.model.defaultsecurity_<Release>_rsa
      \encryption.properties
   ```

   2 Change the encryption type to public encryption:

   ```
   type=public encryption
   ```

   3 Change the algorithm type to RSA:

   ```
   algorithm=RSA
   ```

   4 Copy the pre-generated private and public keys for RSA to the symmetric-key and public-key properties. encryption.properties now looks similar to the one in Listing 21-6.

   5 Save and close encryption.properties.

Listing 21-6 Modified encryption.properties file for the new encryption plug-in

```
#message symmetric encryption , public encryption
   type=public encryption
#private encryption: DES(default), DESede
#public encryption:  RSA
   algorithm=RSA
# NONE , CBC , CFB , ECB( default ) , OFB , PCBC
   mode=ECB
#NoPadding , OAEP , PKCS5Padding( default ) , SSL3Padding
padding=PKCS5Padding
#For key , support default key value for algorithm
#For DESede ,DES we only need to support private key
#private key value of DESede algorithm : 20b0020e918..
#private key value of DES algorithm: 527c23ea...
# RSA algorithm uses a key pair. You should support
#private-public key pair
#private key value of RSA algorithm: 308202760201003....
#public key value of RSA algorithm: 30819f300d0....
#private key
symmetric-key=308202760....
#public key
public-key=30819f300d0.....
```

6 To test the new default RSA encryption, open Actuate BIRT Designer and create a new report design. Create a data source and type the password.

7 View the XML source of the report design file. Locate the data source definition code. The encryptionID is rsa, as shown in the following sample:

```
<data-sources>
   <oda-data-source name="Data Source" id="6"
        extensionID="org.eclipse.birt.report.data.oda.jdbc" >
      <text-property name="displayName"></text-property>
      <property name="odaDriverClass">
         com.mysql.jdbc.Driver
      </property>
      <property name="odaURL">
         jdbc:mysql://192.168.218.225:3306/classicmodels
      </property>
      <property name="odaUser">root</property>
      <encrypted-property name="odaPassword"
         encryptionID="rsa">
         36582dc88.....
      </encrypted-property>
   </oda-data-source>
</data-sources>
```

8 Create a data set and a simple report design. Preview the report to validate that BIRT connects successfully to the database server using the encrypted password. Before trying to connect to the data source the report engine decrypts the password stored in the report design using the default RSA encryption plug-in. Then the engine submits the decrypted value to the database server.

Deploying encryption plug-ins to iServer

If you deploy your report designs to iServer, you need to deploy the report and the new encryption plug-in to iServer. iServer loads all encryption plug-ins at start up. During report execution, iServer reads the encryptionID property from the report design file and uses the corresponding encryption plug-in to decrypt the encrypted property. Every time you create reports using a new encryption plug-in, make sure you deploy the plug-in to iServer, otherwise the report execution on the server will fail.

When using iServer, you do not need to deploy the encryption plug-ins to Information Console.

How to deploy a new encryption plug-in instance to iServer

1 Copy:

```
$ACTUATE_HOME\BRDPro\eclipse\plugins
   \com.actuate.birt.model.defaultsecurity_2.3.2_rsa
```

to:

```
$ACTUATE_HOME\iServer\Jar\BIRT\platform\plugins
```

2 Publish your report design to iServer.

3 Restart iServer to load the new encryption plug-in.

4 Log in to iServer using Information Console and run the report. iServer now uses the new encryption plug-in to decrypt the password.

Generating encryption keys

The default encryption plug-in provides classes that can be used to generate different encryption keys. The classes names are SymmetricKeyGenerator and PublicKeyPairGenerator. SymmetricKeyGenerator generates private keys, which are also known as symmetric keys. PublicKeyPairGenerator generates public keys. Both classes require acdefaultsecurity.jar in the classpath.

Both classes take two parameters, the encryption algorithm and the output file, where the generated encrypted key is written. The encryption algorithm is a

required parameter. The output file is an optional parameter. If you do not provide the second parameter, the output file is named key.properties and is saved in the current folder. The encryption algorithm values are shown in Table 21-4.

Table 21-4 Key generation classes and parameters

Class name	Encryption algorithm parameter
com.actute.birt.model.defaultsecurity.api.keygenerator.SymmetricKeyGenerator	des
com.actute.birt.model.defaultsecurity.api.keygenerator.SymmetricKeyGenerator	desede
com.actute.birt.model.defaultsecurity.api.keygenerator.PublicKeyPairGenerator	rsa

How to generate a symmetric encryption key

Run the main function of SymmetricKeyGenerator.

1 To navigate to the default security folder, open a command prompt window and type:

```
cd C:\Program Files\Actuate11\BRDPro\eclipse\plugins
   \com.actuate.birt.model.defaultsecurity_<Release>
```

2 To generate the key, as shown in Figure 21-2, type:

```
java -cp acdefaultsecurity.jar
   com.actuate.birt.model.defaultsecurity.api.keygenerator.
   SymmetricKeyGenerator des
```

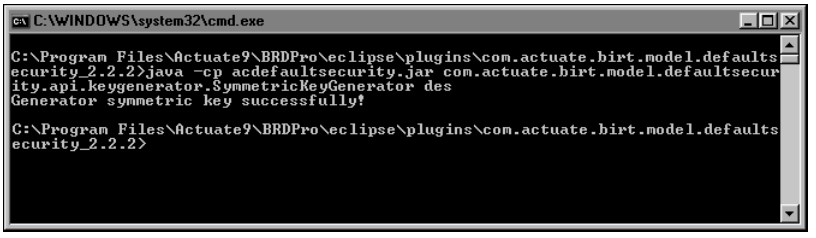


Figure 21-2 Symmetric key generation

3 The generated key is saved in the file, key.properties. The content of the file looks like this one:

```
#Key Generator
#Wed Nov 18 16:17:06 PST 2008
symmetric-key=73c76d5…
```

4 Copy the key from the generated key file to encryption.properties file.

How to generate a public key using RSA encryption

Run the main function of PublicPairGenerator.

1. To navigate to the default security folder, open a command prompt window and type:

```
cd C:\Program Files\Actuate11\BRDPro\eclipse\plugins
   \com.actuate.birt.model.defaultsecurity_<Release>
```

2. In the command prompt window, type:

```
java -cp acdefaultsecurity.jar
   com.actuate.birt.model.defaultsecurity.api.keygenerator.
PublicPairGenerator rsa
```

   The class generates a pair of keys saved in key.properties file:

```
#Key Generator
#Wed Nov 18 15:58:31 PST 2008
public-key=30819f300.....
symmetric-key=3082027502010......
```

3. Copy the key from the generated key file to encryption.properties file.

Creating a custom encryption plug-in

To create a custom encryption plug-in, you need to extend the following extension point:

```
org.eclipse.birt.report.model.encryptionHelper
```

The interface IEncryptionHelper defines two methods, as shown in the following code:

```
public interface IEncryptionHelper
{
   public String encrypt( String string );

   public String decrypt( String string );
}
```

You need to implement these methods and program your encryption and decryption logic there.

To install the custom encryption plug-in, copy the plug-in to the product's plugins folder, where the default plug-in resides. Change the isDefault property in plugin.xml to true. Change the isDefault properties of the rest of the encryption plug-ins to false.

Using encryption API methods

You can call the API methods in the default encryption plug-in when you have to set the encryptionID property, or encrypt data programmatically. The following list describes these methods and shows their signatures:

- IEncryptionHelper::encrypt encrypts the given string, and returns the encrypted string:

  ```
  String IEncryptionHelper::encrypt( String value )
  ```

- IEncryptionHelper::decrypt decrypts the given encrypted string, and returns the original string:

  ```
  public String IEncryptionHelper::decrypt( String string )
  ```

- MetaDataDictionary::getEncryptionHelper returns the encryption helper with the extension ID:

  ```
  public IEncryptionHelper
     MetaDataDictionary::getEncryptionHelper( String id )
  ```

- MetaDataDictionary::getEncryptionHelpers gets all the encryption helpers:

  ```
  public List MetaDataDictionary::getEncryptionHelpers( )
  ```

Chapter

22

Using custom emitters in Actuate BIRT iServer

This chapter contains the following topics:

- About custom emitters
- Deploying custom emitters to iServer
- Rendering in custom formats

About custom emitters

In Actuate BIRT Designer Professional or Interactive Viewer, you can choose to render BIRT reports in several different formats, as shown in Figure 22-1.

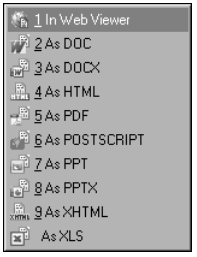


Figure 22-1 Rendering formats

Actuate provides out-of-the-box report rendering for the following file formats:

- DOC - Microsoft Word document
- DOCX - Microsoft Word document, introduced in Windows 7
- HTML - HyperText Markup Language document, a standard format used for creating and publishing documents on the World Wide Web
- PDF - Created by Adobe, a PDF is a portable file format intended to facilitate document exchange.
- POSTSCRIPT - A page description language document for medium to high-resolution printing devices
- PPT - Microsoft PowerPoint document
- PPTX - Microsoft PowerPoint document for Windows 7
- XHTML -Extensible Hypertext Markup Language document, the next generation of HTML, compliant with XML standards
- XLS - MS-Excel Document

If you need to export your document to a format not directly supported by Actuate, such as CSV and XML, you need to develop a custom emitter. Actuate supports using custom emitters to export reports to custom formats. After a system administrator places custom emitters in the designated folder in Information Console or iServer, users are able to use them as output formats when scheduling BIRT report jobs in iServer or exporting BIRT reports in Information Console. Custom emitters are also supported as e-mail attachment formats.

iServer uses the custom emitter format type as a file extension for the output file when doing the conversion. When you develop custom emitters, always use the

same name for a format type and an output file extension type. Management Console and Actuate Information Console for iServer display the options of each emitter for the user to choose when exporting a report.

The *Integrating and Extending BIRT* book, published by Addison-Wesley, provides detailed information about how to develop custom emitters in BIRT.

Deploying custom emitters to iServer

The custom emitters in BIRT are implemented as plug-ins and packaged as JAR files. To make them available to the Actuate products that support them, copy the emitters to the following folder:

```
Actuate<release>/iServer/MyClasses/eclipse/plugins
```

The MyClasses folder appears at different levels on different platforms but it is always available at the product's installation folder.

When you install InformationConsole.war file to your own J2EE application server, the shared folder MyClasses is not available. In this case, custom emitter plug-ins should be copied to the following folder:

```
<context-root>/WEB-INF/platform/plugins
```

Every time you deploy a custom emitter you need to restart the product. This ensures the emitter JAR is added to the classpath and the product can discover the new rendering format.

The following tools and products support custom emitters:

- Actuate BIRT DesignerProfessional
- Actuate BIRT Studio
- BIRT Interactive Viewer for iServer
- Information Console for iServer
- iServer

Rendering in custom formats

After deploying the custom emitter, you can see the new rendering formats displayed along with built-in emitters in the following places:

- Preview report in Web Viewer in Actuate BIRT Designer
- Output page of schedule job in Management Console and Information Console for iServer

- Attachment Notification page of schedule job in Management Console or Information Console for iServer
- Export Content in Actuate BIRT Viewer and Actuate BIRT Interactive Viewer

The following examples show the deployment and usage of a custom CSV emitter. The CSV emitter renders a report as a comma separated file. The JAR file name is org.eclipse.birt.report.engine.emitter.csv.jar. The custom format type is MyCSV.

To test the emitter functionality with Management and Information Consoles, you schedule any BIRT report design or report document from the examples in the Public folder. The examples that follow use the report from the sample Encyclopedia volume for an iServer:

```
Public/BIRT and BIRT Studio Examples
   /CustomerList.rptdesign
```

How to deploy a custom emitter

This example assumes that the Actuate products are installed in C:\Program Files\Actuate<Release> folder on Windows.

1 Copy org.eclipse.birt.report.engine.emitter.csv.jar to:

```
C:\Program Files\Actuate<Release>\iServer\MyClasses\eclipse\
   plugins
```

2 Restart the product to make it load the new plug-in in its classpath:

- If using Actuate BIRT Designer, reopen the designer.
- If using iServer, restart Actuate iServer <Release> from Start➤Settings➤Control Panel➤Administrative Tools➤Services, as shown in Figure 22-2.
- If you use a separately deployed Information Console, you must also restart Apache Tomcat for Actuate Information Console <Release>.

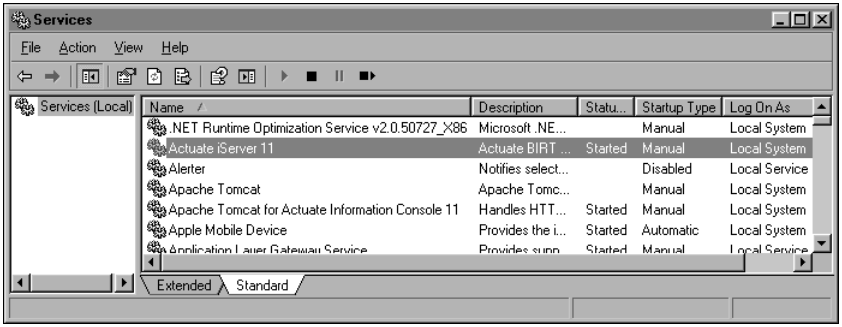


Figure 22-2 Services

The following procedures show how to export a BIRT report to the new MyCSV format in different products.

How to export a BIRT report from Actuate BIRT Designer

1. Open a BIRT report in Actuate BIRT Designer Professional.
2. Preview the report in Web Viewer. The new MYCSV format appears in the list of formats, as shown in Figure 22-3.

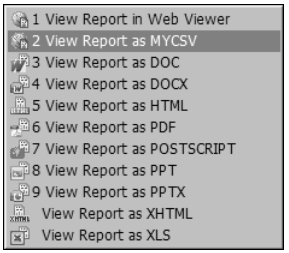


Figure 22-3 List of available formats in Web Viewer

3. Select the MYCSV option. A new window opens and displays the report in MYCSV format, as shown in Figure 22-4.

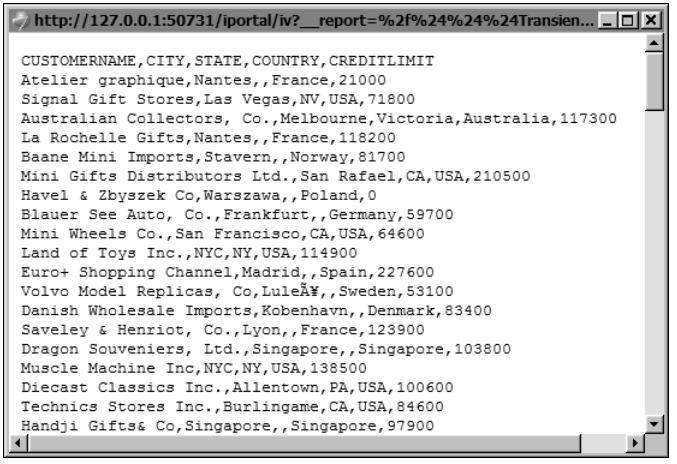


Figure 22-4 Exported content

How to export a BIRT report in iServer Management Console

1. Open iServer Management Console.
2. Navigate to the Public/BIRT and BIRT Studio Examples folder.
3. Click the blue arrow next to CustomerList.rptdesign and choose the Schedule option from the menu.

4 In the Schedule page, select Output tab. The new MYCSV format appears in the list of the available formats, as shown in Figure 22-5.

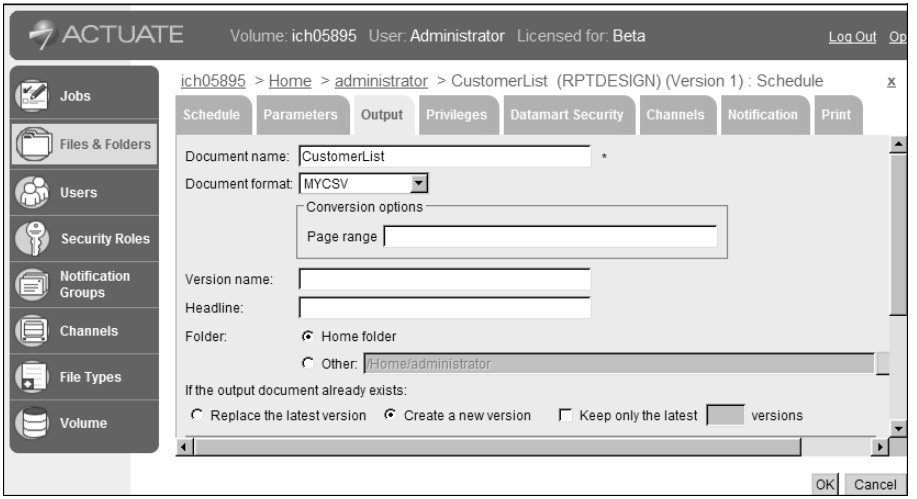


Figure 22-5 Output format in Management Console

5 Choose the Notification Tab in the same Schedule Job page. Select MYCSV format from the Format for the attached report's drop-down list, as shown in Figure 22-6.

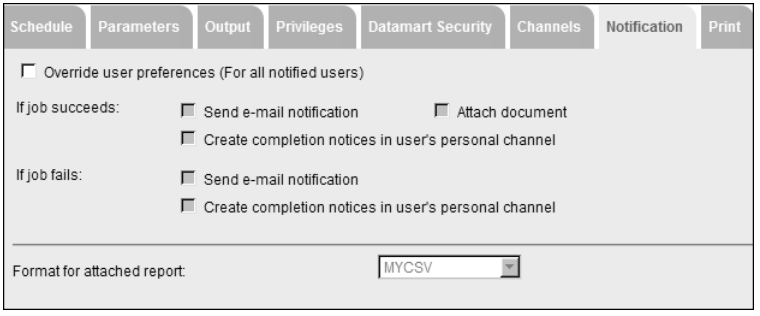


Figure 22-6 Notification tab in the Schedule Job page

6 Choose OK. The generated report is saved as CustomerList.MYCSV in the Encyclopedia volume. The report is also attached to the e-mail notification.

How to export a BIRT report from Information Console or iServer

Schedule a BIRT report to run by choosing Save As on the schedule page. The new MYCSV format appears in the Document Format list. You can also select to attach the output report to an e-mail notification, as shown in Figure 22-7.

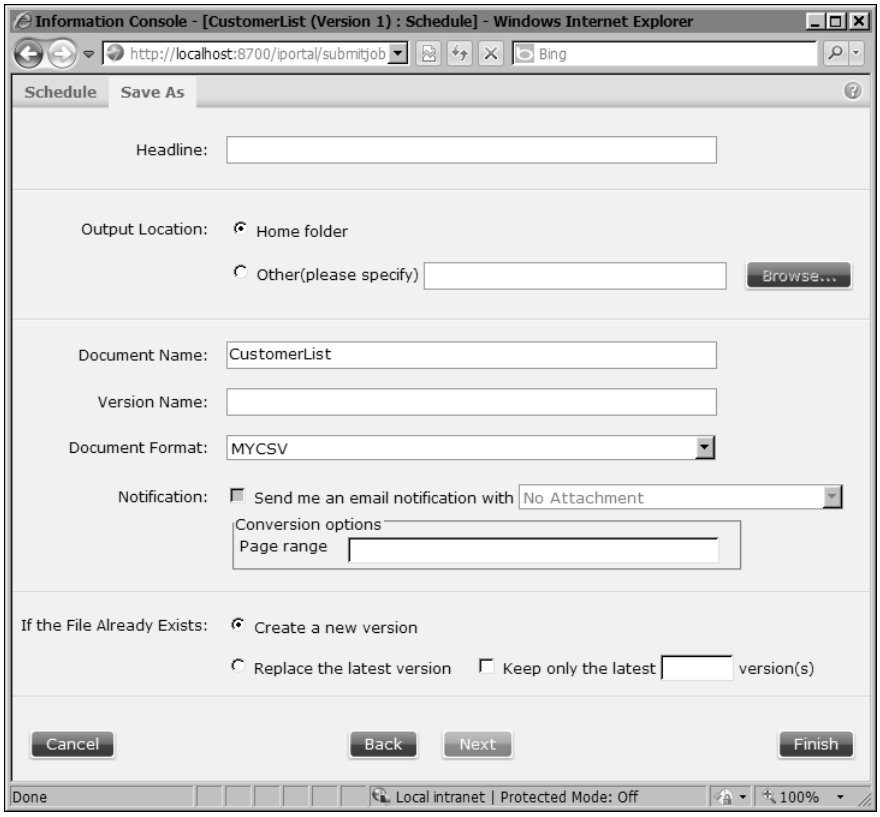


Figure 22-7 Save As tab in the Schedule Jobs page in Information Console

How to export a BIRT report from Actuate BIRT Viewer or Actuate BIRT Interactive Viewer

1. Open a BIRT report in Actuate BIRT Viewer or Interactive Viewer.
2. Select Export Content from the viewer menu. The new MyCSV format shows up in the Export Formats as shown in Figure 22-8.

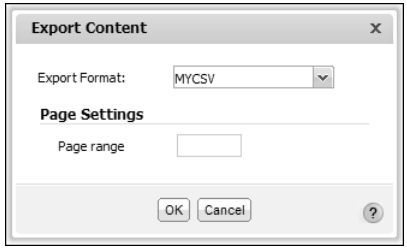


Figure 22-8 Export Content in Actuate BIRT Viewers

3 Choose OK.

A file download window appears, as shown in Figure 22-9. You can choose to open or save the file. The suggested file name is CistomerList.mycsv.

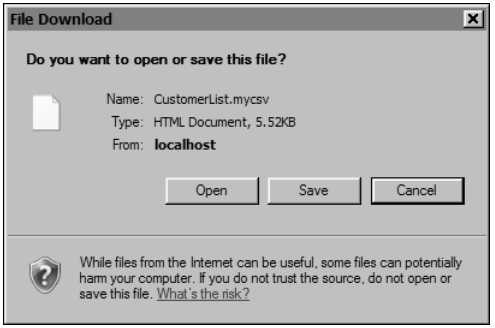


Figure 22-9 File Download

Part Four

Using Actuate BIRT APIs

Chapter

23

Using the BIRT data object API

This chapter contains the following topics:

- About generating data objects from an application
- Generating data object elements for BIRT report designs
- Tutorial 5: Creating a data element using the Design Engine API

About generating data objects from an application

Actuate BIRT iServer includes a Design Engine API extension to create and alter data objects programmatically. This Actuate extension provides Java classes to automate data object generation, retrieving important information required regularly for any application. The classes that support BIRT data objects, DataMartCubeHandle, DataMartDataSetHandle, and DataMartDataSource Handle, are contained in the com.actuate.birt.report.model.api package.

Like the BIRT data objects implemented in the BIRT data explorer, BIRT data objects generated by the Design Engine API generate data sources and data sets from a .datadesign or .data file. Using the extension requires programming in Java. Knowledge of XML is also helpful.

Handling data objects for BIRT reports requires knowledge of programming using the BIRT reporting API and the report object model. For information about the BIRT reporting API and the report object model, see *Integrating and Extending BIRT.* This chapter describes the additional requirements for generating data objects for reports.

Generating data object elements for BIRT report designs

To generate data-object data sources, data sets, and cubes for a BIRT report design, first configure BIRT_HOME to access the Actuate commercial model API Java Archive (JAR) files from Actuate iServer. To accomplish this task, generate a DesignConfig object with a custom BIRT_HOME path, as shown in the following code:

```
// Create an DesignConfig object.
DesignConfig config = new DesignConfig( );
// Set up the path to your BIRT Home Directory.
config.setBIRTHome("C:/Program Files/Actuate11/iServer/Jar/BIRT/
   platform");
```

Use the path to the iServer installation specific to your system.

Using this design configuration object, create and configure a Design Engine object, open a new session, and generate or open a report design object, as shown in the following code:

```
// Create the engine.
DesignEngine engine = new DesignEngine( config );
SessionHandle sessionHandle = engine.newSessionHandle(
   ULocale.ENGLISH );
ReportDesignHandle designHandle = sessionHandle.createDesign( );
```

These objects are contained in the model API package org.eclipse.birt.report.model.api.

The ElementFactory class supports access to all the elements in a report. The following code generates an Element Factory object:

```
ElementFactory factory = designHandle.getElementFactory( );
```

To generate data sources, data sets, and cubes, use the datamart methods of an ElementFactory object: newDataMartCube( ) for a new cube, newDataMartDataSet( ) for a data set, and newDataMartSource( ) for a new data source. For example, to instantiate a new data source, use the following code:

```
DataMartDataSourceHandle dataSource =
   factory.newDataMartDataSource("Data Object Data Source");
```

Associate a handle for a data-object data source with an actual data source from the contents of a data or data design file. For example, to associate a data source handle with a data source from test.datadesign, use the following code:

```
dataSource.setDataMartURL( "test" );
dataSource.setAccessType(
   DesignChoiceConstants.ACCESS_TYPE_TRANSIENT );
```

Finally, add the data element to the report design, as shown in the following code:

```
designHandle.getDataSources( ).add( dataSource );
```

To complete the data source assignment, output the report design into a file and close the design handle object, using code similar to the following:

```
FileOutputStream fos = new FileOutputStream( "output.rptdesign" );
designHandle.serialize( fos );
// Close the document.
fos.close( );
designHandle.close( );
```

The resulting output file, output.rptdesign, contains the new data source, retrieved from test.datadesign. This data source appears in Data Sources in Data Explorer and establishes a link to the .datadesign file, test.datadesign. The XML source for output.rptdesign includes markup similar to the following lines:

```
<datamart-node location="file:/MyProject/test.datadesign">
...
<data-sources>
   <data-mart-data-source name="Data Object Data Source" id="7">
      <property name="datamartURL">test</property>
      <property name="accessType">transient</property>
   </data-mart-data-source>
</data-sources>
```

When exporting this report design to an Encyclopedia volume, also export test.datadesign to maintain the reference to the data source.

Creating data-object data sets for BIRT report designs

To create a data-object data set, use the newDataMartDataSet( ) method from ElementFactory. For example, to instantiate a new data set, use the following code:

```
DataMartDataSetHandle dataSet =
   factory.newDataMartDataSet("Data Set");
```

Associate the data-object data cube with a DataMartDataSourceHandle object and then add the name of a data set from the data or data design file. For example, to access a data set called "SetName," use the following code:

```
dataSet.setDataSource( dataSource.getName( ) );
dataSet.setDataObject( "SetName" );
```

DataMartDataSetHandle inherits the setDataSource( ) method from DataSetHandle.

Finally, add the data element to the report design, as shown in the following code:

```
designHandle.getDataSets( ).add( dataSet );
```

Creating data-object data cubes for BIRT report designs

To create a data-object data cube, use the newDataMartDataCube( ) method from ElementFactory. For example, to instantiate a new data cube, use the following code:

```
DataMartDataCubeHandle dataCube =
   factory.newDataMartDataCube("Data Cube");
```

Associate the data-object data cube with a DataMartDataSourceHandle object and assign a data cube from the data or data design file. For example, to access a data cube called "CubeName," use the following code:

```
dataCube.setDataSource( dataSource.getName( ) );
dataCube.setDataObject("CubeName");
```

Finally, add the data element to the report design, as shown in the following code:

```
designHandle.getDataCubes( ).add( dataCube );
```

Tutorial 5: Creating a data element using the Design Engine API

This tutorial provides step-by-step instructions for creating a Java class that generates a BIRT report design with a BIRT data source generated from a BIRT Data Design file. You perform the following tasks:

- Set up a project.
- Create a GenerateDataObject Java class.
- Create the main( ) method to test the code.
- Run the code.

Task 1: Set up a project

To compile a Design Engine API application, the design engine Java archive (JAR) files from Actuate iServer must be in your class path. You can find the design engine JAR files in the <Actuate home>/iServer/Jar/BIRT/lib directory folder. The main JAR files that contain the design engine classes are coreapi.jar and modelapi.jar files. In addition, you need a data design file from which to generate the data objects. For this tutorial, the data design file is include.datadesign.

1 In Java perspective, select File➤New➤Java Project. New Java Project appears as shown in Figure 23-1.

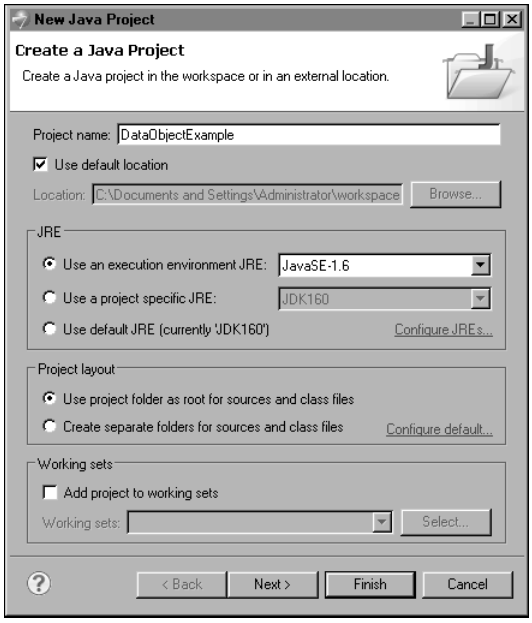


Figure 23-1 Creating the DataObjectExample project

2 In Project Name type:

```
DataObjectExample
```

3 In Project layout, select:

```
Use project folder as root for sources and class files
```

4 Choose Next. Java Settings appears.

5 Set the project build path.

1 Select the Libraries tab.

2 Choose Add External JARs.

3 In JAR Selection, navigate to the iServer\Jar\BIRT\lib directory. For the default installation of BIRT on Windows XP, this directory is:

```
C:\Program Files\Actuate11\iServer\Jar\BIRT\lib
```

4 In JAR Selection, select all of the JAR files in the directory.

5 Choose Open. The libraries are added to the classpath as shown in Figure 23-2.

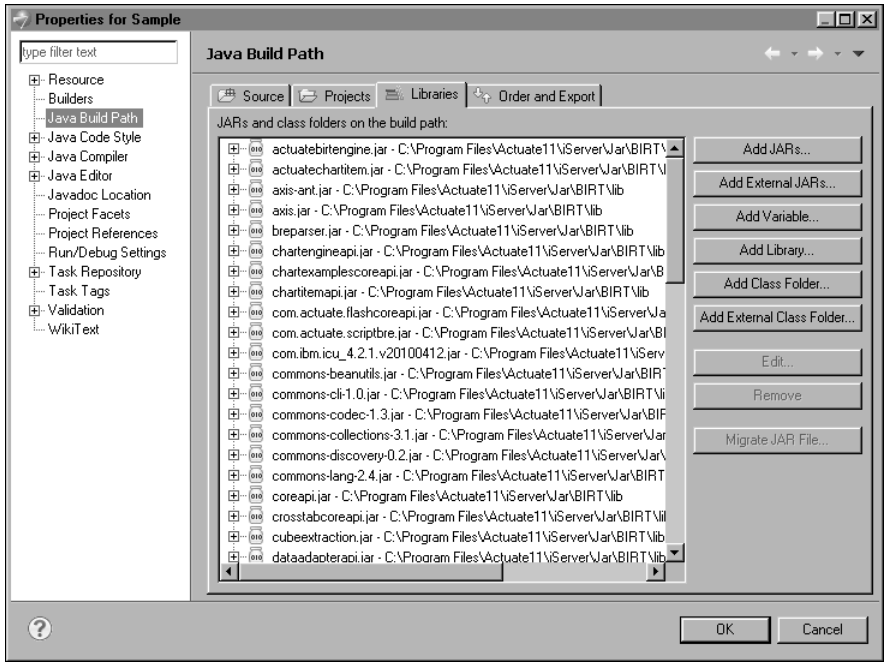


Figure 23-2 DataObjectsAPI project build path

6 Choose Finish.

6 Import the data design file.

1 In the Package Explorer, right-click the DataObjectExample project.

2 Choose Import from the context menu.

3 In Import, choose General➤FIle System and then choose Next.

4 In File System, choose Browse.

5 Navigate to and select a data design file. Then choose Finish. The data design file appears in the project as shown in Figure 23-3.

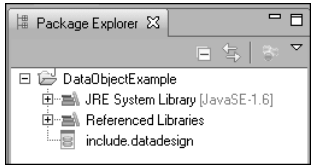


Figure 23-3 DataObjectExample project showing the data design file

Task 2: Create a GenerateDataObject Java class

This Java class creates a simple report design, with table, list, and image elements.

1 Choose File➛New➛Class. New Java Class appears.

2 In Name type:

```
GenerateDataObject
```

3 In Package, as shown in Figure 23-4, type:

```
myPackage
```

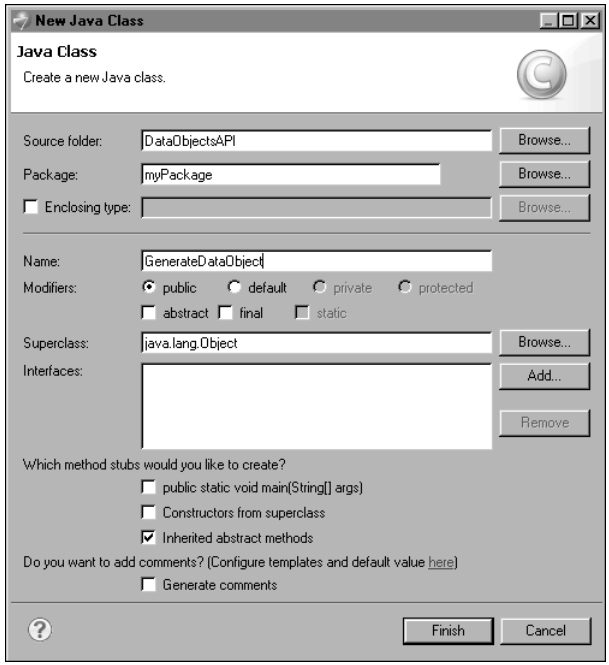


Figure 23-4 Creating GenerateDataObject class

4 Choose Finish. GenerateDataObject.java opens in the Java editor.

5 Add a BIRT_HOME static variable to the class. For the default installation of iServer on a 32-bit Windows system, use the following line in the body of the GenerateDataObject class body:

```
private static final String BIRT_HOME =
   "C:/Program Files/Actuate11/iServer/Jar/BIRT/platform";
```

Task 3: Create the main( ) method to test the code

Add a main( ) method to run the class.

1 Type the following main method:

```
public static void main( String[] args ) throws Exception
{
}
```

An error indicating that the BirtException class is not defined appears.

2 Use Quick Fix (Ctrl+1) to import the BirtException class definition.

3 Add the main method body shown in Listing 23-1 to your main( ) method.

Listing 23-1 main() method code

```
DesignConfig config = new DesignConfig( );
config.setBIRTHome( BIRT_HOME );

DesignEngine engine = new DesignEngine( config );
SessionHandle sessionHandle = engine.newSessionHandle(
   ULocale.ENGLISH );
ReportDesignHandle designHandle = sessionHandle.createDesign();
ElementFactory factory = designHandle.getElementFactory( );

DataMartDataSourceHandle dataSource =
   factory.newDataMartDataSource( "Data Source" );
dataSource.setDataMartURL( "include" );
dataSource.setAccessType(
   DesignChoiceConstants.ACCESS_TYPE_TRANSIENT );
designHandle.getDataSources( ).add( dataSource );

FileOutputStream fos = new FileOutputStream("test.rptdesign");
designHandle.serialize( fos );
fos.close( );

designHandle.close( );
System.out.println("Done");
```

Read the code explanations:

- To access a data source and its contents, the application must first generate and configure a design engine object.

- After creating the engine object, the code instantiates a new session. The SessionHandle object manages the state of all open data and report designs. Use SessionHandle to open, close, and create data designs, and to set global properties, such as the locale and the units of measure for data elements. Create the session handle only once. BIRT supports only a single SessionHandle.
- Generate a new design handle using the SessionHandle object. Create a design engine element factory using the DesignHandle object.
- Create a new instance of DataMartDataSourceHandle and set the datamart URL to the name of a datamart file, include, which corresponds to the include.datadesign file added to the project. Then, configure the access type and add the data source handle to the design handle object.
- Finally, open a file output stream to a report design, test.rptdesign, that uses the data object. Export the data design element to the report design.

4 Add the import statements shown in Listing 23-2 to the beginning of the file.

Listing 23-2 import statement code

```
import java.io.FileOutputStream;
import org.eclipse.birt.core.exception.BirtException;
import org.eclipse.birt.report.model.api.DesignConfig;
import org.eclipse.birt.report.model.api.DesignEngine;
import org.eclipse.birt.report.model.api.ElementFactory;
import org.eclipse.birt.report.model.api.ReportDesignHandle;
import org.eclipse.birt.report.model.api.SessionHandle;
import org.eclipse.birt.report.model.api.elements
   .DesignChoiceConstants;
import com.actuate.birt.report.model.api
   .DataMartDataSourceHandle;
import com.ibm.icu.util.ULocale;
```

Task 4: Run the code

5 Create a Run configuration for GenerateDataObject.java class.

1. In Package Explorer, select:

   `GenerateDataObject.java`

2. From the main menu, choose Run➛ Run Configurations.
3. Double-click the Java Application link in the left frame of the Run Configurations. The GenerateDataObjects configuration gets created.
4. Choose Run. Save and Launch appears. Choose OK.

6 After the execution completes, refresh the contents of the DataObjectExample project. test.rptdesign appears.

7 Open the report design and view the XML Source. The XML contains a datamart element that points to include.datadesign and a data source called include, as shown in the following code:

```
<datamart-node
   location="file:/DataObjectExample/include.datadesign">
...
<data-sources>
   <data-mart-data-source name="Data Source" id="4">
      <property name="datamartURL">include</property>
      <property name="accessType">transient</property>
   </data-mart-data-source>
</data-sources>
```

8 In Data Explorer, expand Data Sources to view the new data source, as shown in Figure 23-5.

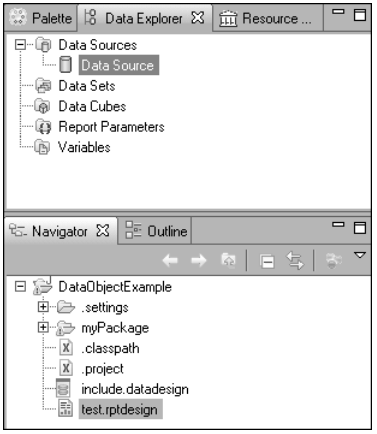


Figure 23-5 Data Source in test.rptdesign

The final code for GenerateDataObject is shown in Listing 23-3.

Listing 23-3 GenerateDataObject.java

```
package myPackage;
import java.io.FileOutputStream;
import org.eclipse.birt.core.exception.BirtException;
import org.eclipse.birt.report.model.api.DesignConfig;
import org.eclipse.birt.report.model.api.DesignEngine;
import org.eclipse.birt.report.model.api.ElementFactory;
import org.eclipse.birt.report.model.api.ReportDesignHandle;
import org.eclipse.birt.report.model.api.SessionHandle;
import org.eclipse.birt.report.model.api.elements
   .DesignChoiceConstants;
import com.actuate.birt.report.model.api.DataMartCubeHandle;
```

```
import com.actuate.birt.report.model.api.DataMartDataSourceHandle;
import com.ibm.icu.util.ULocale;

public class GenerateDataObject {

private static final String BIRT_HOME =
   "C:/Program Files/Actuate11/iServer/Jar/BIRT/platform";

public static void main( String[] args ) throws Exception
{
   DesignConfig config = new DesignConfig( );
   config.setBIRTHome( BIRT_HOME );

   DesignEngine engine = new DesignEngine( config );
   SessionHandle sessionHandle = engine.newSessionHandle(
   ULocale.ENGLISH );
   ReportDesignHandle designHandle = sessionHandle.createDesign();
   ElementFactory factory = designHandle.getElementFactory( );

   DataMartDataSourceHandle dataSource =
   factory.newDataMartDataSource( "Data Source" );
   dataSource.setDataMartURL( "include" );
   dataSource.setAccessType(
   DesignChoiceConstants.ACCESS_TYPE_TRANSIENT );
   designHandle.getDataSources( ).add( dataSource );

   FileOutputStream fos = new FileOutputStream("test.rptdesign");
   designHandle.serialize( fos );
   fos.close( );

   designHandle.close( );
   System.out.println("Done");
   }
}
```

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y