IBM DB2 Alphablox

IBM

Relational Reporting Developer's Guide

Version 84

IBM DB2 Alphablox

Relational Reporting Developer's Guide

Version 8.4

Third Edition (March 2006)

This edition applies to version 8, release 4, of IBM DB2 Alphablox for Linux, UNIX and Windows (product number 5724-L14) and to all subsequent releases and modifications until otherwise indicated in new editions.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Chapter 1. Relational Reporting Overview

This chapter provides an overview of the features, functionality, components, and browser support of the Relational Reporting feature in IBM DB2 Alphablox for Linux, UNIX and Windows.

- "Relational Reporting" on page 1
- "Components of Relational Reporting" on page 2
- "Browser Support" on page 5
- "Accessibility Support" on page 5

Relational Reporting

Relational Reporting supports generation of dynamic, interactive reports from relational data sources. You can extract data from any relational data sources defined to DB2 Alphablox into a result set and then perform data formatting, calculations, and report editing tasks. The reports are rendered in Dynamic HTML and can be saved in PDF format.

A key feature of Relational Reporting is the Report Editor user interface that gives your users the power to analyze data and create reports on the fly based on their needs. When a report is rendered in interactive mode, a set of interactive context menus are turned on. When the users move their mouse over the "hot spots" in the report, these menus pop up and users can hide columns, reorder columns, sort the data, add break groups, specify group header and footer text, and change the look and feel of the report via point-and-click.

A set of Relational Reporting Blox is available to provide you the ability to:

- Connect to a relational data source defined via DB2 Alphablox home page
- Extract data from the data source using SQL queries
- Add a member based on some calculation
- Sort and filter the data
- Include or exclude members
- Arrange the column layout
- Add break groups
- Specify formats for different data types and missing values
- Add banding
- Specify the footer and header texts and types of aggregation (`sum`, `average`, `min`, `max`, `count`, and `none`)
- Specify the colors and fonts for various elements in the report
- Enable the interactive context menus for users
- Bookmark a report for later retrieval
- Send a report to PDF
- Save a report in PDF or HTML

Localization

Relational Reporting supports localization. Interactive menus in reports will automatically be displayed in the correct language based on the locale of the client.

However, because Blox components have a session scope, if users change the language in the browser after a Blox is already loaded on the page, the new language will not be reflected in the interactive menus in the same session.

Tip: In order for the report to display correctly in languages other than English, you should specify to use the UTF-8 character set using the JSP `page` directive's `contentType` attribute. See "General Development Tips" on page 21 for more information.

Components of Relational Reporting

At the heart of Relational Reporting is ReportBlox. A set of other Blox (SortBlox, FilterBlox, MembersBlox, OrderBlox, GroupBlox, CalculateBlox, StyleBlox, FormatBlox, SQLDataBlox, DataSourceConnectionBlox, ErrorBlox, and PdfBlox) work with ReportBlox to handle discrete data extraction, manipulation, report formatting, and report rendering functions.

All Relational Reporting Blox are Java™ beans. Each Blox provides a very specific set of functionality as its name suggests. One handles JDBC connectivity to a relational data source; another does nothing but sort the data; and another, nothing but calculation. This clear division of functionality gives you the flexibility to add the appropriate Blox to perform specific tasks only when they are needed. It also gives you better control of the desired outcomes.

To create a relational report with these Blox, you use the provided custom tags in your JSP pages. The order you put these Blox together dictates how these Blox are connected and how the data result set gets transformed. For example, in the following diagram, a calculated member is added based on some calculation, and then some filtering operation is performed. This filtering operation may be based on the values of the calculated member.

```
<bloxreport:report>

    <bloxreport:calculate>

      ...
    </bloxreport:calculate>


    <bloxreport:filter>
      ...
    </bloxreport:filter>


    ...

</bloxreport:report>
```
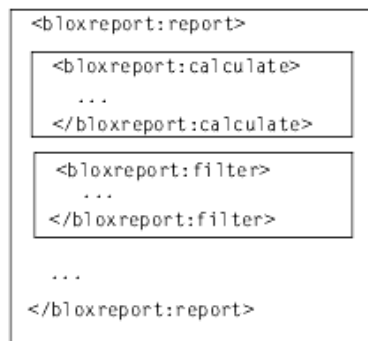
The following is a list of all Blox that support creation of relational reports:

Blox	Description
ReportBlox	Generates a report based on the data result set produced by SQLDataBlox. The report can be rendered either in a static HTML table or a table in Dynamic HTML (DHTML) with interactive report editing functionality.
SQLDataBlox	Executes a SQL query against a data source and stores the result in a result set.

DataSourceConnectionBlox	Represents a connection to a relational data source defined to DB2 Alphablox.
RDBResultSetDataBlox	Represents an RDBResultSet object derived from a DataBlox.
CalculateBlox	Lets you add a calculated member on the Column dimension based on a specified calculation expression.
SortBlox	Lets you sort the data based on specified members.
FilterBlox	Enables filtering of numeric data based on the criterion you specify.
GroupBlox	Supports grouping of data. By adding a GroupBlox to your ReportBlox, you can create break groups in your report and specify the type of aggregation—`sum`, `average`, `count`, `max`, `min`, or `none`— at the end of each break group.
MembersBlox	Lets you specify members to be included or excluded.
OrderBlox	Lets you specify the order from left to right in which the members are returned to the result set
FormatBlox	Lets you specify the display format for numeric and date data types. It also lets you define the text to display for missing data.
StyleBlox	Lets you set styles such as the fonts, colors, and positions of various elements in the report. You can set style for missing or negative values, and styles for individual columns.
TextBlox	Lets you specify the text to display for group headers, footers, totals, or column headers.
PdfBlox	Lets you send a relational report to PDF with the layout you specify.
PersistenceBlox	Lets you save the state of a report for later retrieval.
ErrorBlox	Catches the exception thrown and prints the details in an HTML table with better handling and display of nested exceptions.

Reports as an HTML Tables

Relational reports are rendered as Dynamic HTML tables. Each element in the tables, such as the columns, the rows, break group headers, and break group footers, has a specific cascading style class associated with it. You can customize the font colors, font families, font sizes, font weight, text alignment, and background colors for each element as you normally would with HTML and Cascading Style Sheet (CSS).

Relational Reporting offers an interactive Report Editor user interface for your users to edit the report. Associated with ReportBlox is this `interactive` property that determines if the report should be rendered in static HTML or dynamic HTML. When you set the `interactive` attribute in the `<bloxreport:report>` tag to

`true`, the context menus will be activated when users mouse over "hot spots" in the report where they can make changes. This Report Editor is described next.

Report Editor User Interface

When a relational report is set to render in interactive mode, context menus pop up when users mouse over certain areas in the report, allowing the users to dynamically and interactively edit the report. The "hot spots" in a report are:
- column headers
- break group headers
- break group totals

The context menus popped up corresponding to these hot spots are Column Header Context Menu, Group Header Context Menu, and Group Total Context Menu. The Blox Sampler for Relational Reporting provides a "Simplest Interactive Report" example using a canned data that lets you easily run a report and experiment with the interactive Report Editor User Interface. See Blox Sampler — Relational Reporting under the Assembly tab under the DB2 Alphablox Home Page.

These menus provide the users with dynamic report editing functionality such as sorting, adding break groups, hiding or showing columns, specifying break group aggregation type, and editing column names and break group totals text. The **Style...** menu option brings up the Report Style dialog box that allows the users to set text font face, size, style, color, and alignment for various elements in the report. All these context menus and the Report Style dialog box are implemented in DHTML. You can easily customize the appearances of these menus and dialog box by editing or supplying your own style sheet that specifies the style for each of the style class defined.

Column Header Context Menu

The Column Header Context Menu offers the following menu items:
- Sort
- Hide
- Show All
- Rename
- Group
- Clear groups
- Style...

Via these menu items, users can dynamically sort on a column, hide a column, show all columns (to bring back columns that are hidden via the **Hide** menu item), rename the column heading, group the report based on data values in a column, clear all break groups, and style the report.

Group Header Context Menu

The Group Header Context Menu offers the following menu items:
- Clear Group
- Style...

The Clear Group option allows the users to clear just that break group rather than all break groups.

Group Total Context Menu

The Group Total Context Menu offers the following menu items:
- Sum
- Count

- Average
- Min
- Max
- None
- Edit Text
- Style...

The default aggregation type for columns containing numeric data is sum. The default aggregation type for columns containing strings is count. When a user uses the Column Header Context Menu to dynamically add a break group, these will be the default aggregation types applied unless you have specified otherwise in your JSP file. The detail on adding break groups and specifying aggregation type is described in "Specifying and Styling Break Group Headers, Footers, and Totals" on page 59.

Rendering Reports to PDF

You can render a report or allow your users to save edited reports to PDF by using PdfBlox. You can specify whether the PDF rendering should include a header or a footer and set the margins and orientation. For more information, see the topic on "Saving as PDF" on page 71.

Browser Support

Relational reports can be rendered in non-interactive HTML table or interactive DHTML table. Browsers supported are:

- IE 5.5 and above

- FireFox 1.0.4 and above

- Mozilla 1.7

For non-interactive reports, Netscape 7.1 is also supported. When users try to access an interactive report using Netscape browsers, an alert window will pop up, informing them that Netscape browsers are not supported for interactive reports.

Accessibility Support

The keystrokes used to navigate in an interactive report built using ReportBlox are generally consistent with the standard Windows keyboard shortcuts. Shortcuts for common tasks are described in the *Release Overview* in this information center as well as in the provided user help. Keyboard access is supported for Internet Explorer v6 only.

Users with limited vision can utilize a screen reader program to read the data in a rendered report. For these users, application developers should set the ReportBlox readerMode tag attribute to true. This turns off the **Style...** menu options in the interactive menus because screen readers cannot set their focus on the dialog to read the text. In addition, the styling options offered in the **Report Style** dialog are not useful to users with limited vision. You can achieve this type of personalization using user profile, such as via a custom user property. For example, you can specify a custom user property through the DB2 Alphablox Admin Pages and assign a specific value for that property for users who need screen reader support. You can then access the value using the RepositoryBlox getUserProperty() method, and programmatically set the readerMode tag attribute to true.

Chapter 2. Relational Reporting Concepts

This section discusses the key concepts in Relational Reporting that are essential to the understanding of the overall design and to effective development of relational reports.

- "Concepts for Relational Reporting" on page 7
- "Styling the Relational Reports" on page 9
- "Relational Reporting Custom Tags" on page 14
- "Expression Syntax" on page 18

Concepts for Relational Reporting

Several key concepts essential to the understanding of the overall design and to effective development of relational reports are discussed in the following sections:

- "Componentized Blox Based on Standard Technologies" on page 7
- "Report Rendering" on page 7
- "Report Pipeline" on page 8
- "Accessing Individual Blox" on page 8
- "Columns and Members" on page 9

Componentized Blox Based on Standard Technologies

Blox supporting the Relational Reporting feature are functionally decoupled. Each of them does a distinctive set of data transformation, access, or presentation tasks. Each takes the result set produced by the Blox before them and produces a result set that can be sent to ReportBlox for presentation or passed on to other Blox for further data transformation. Because these Blox are componentized and functionally decoupled, you can flexibly connect them together to produce your desired result. When you add Relational Reporting Blox to your JSP page, the order in which they are added dictates how the result set is transformed along the way.

These Relational Reporting Blox adhere to standard Web technologies, including JavaBeans™, JavaScript™ 1.2 and Cascading Style Sheet 2. This means you can use the standard Web development technologies in your JSP files with Relational Reporting Blox to produce desired outcome and to extend their functionality.

Note: Even though you can have both a PresentBlox and a ReportBlox in the same JSP page, since PresentBlox and all the other supporting user interface Blox (GridBlox, ChartBlox, PageBlox, DataLayoutBlox, and ToolbarBlox) do not process and pass along the result set in the same way, you cannot integrate Relational Reporting Blox into a user interface Blox or vice versa.

Report Rendering

Relational reports are rendered into HTML tables. When the interactive mode is set to `true`, the report is rendered in DHTML. You can specify whether the reports should be rendered in interactive mode or non-interactive mode. This is done through setting the `interactive` attribute of the `<bloxreport:report>` tag. When the `interactive` attribute is set to `true`, users are given the power to format the report based on their needs and wants.

Beside the two rendering modes, another option in which a report that reflects real time data can be presented to users is in PDF. You can generate the report on the fly and directly send the report to PDF using PdfBlox.

Note: Unlike PresentBlox and all the other supporting user interface Blox, you cannot set the visibility of a relational report to `false` nor can you use the `<blox:display>` tag to reference a ReportBlox.

Report Pipeline

The order you put these Relational Reporting Blox together dictates how these Blox are connected and how the data result set gets transformed through this pipeline. Many Blox in Relational Reporting act as both data consumers and data producers. They consume the data returned from the previous Blox, transform it, and produce data for the subsequent Blox, with the ultimate consumer being the ReportBlox.

Because each Blox handles very specific tasks and the order in which they are connected makes a difference, you can manipulate the Blox in various ways to arrive at your desired report. However, this can also result in incorrect data if you are not careful. For instance, sorting the data before or after grouping will produce different results. Adding a calculated column before or after grouping the data also result in different data. These will be discussed in the chapters where the specific tasks are described.

Not all Blox in Relational Reporting are data transformers. Some of the Blox handle data formats, report styling, display text specifications, PDF rendering, or error handling. The order these Blox are specified has no impact on the data transformation process and they are either processed separately (such as ErrorBlox) or after the data is completed processed. The following table shows the transformers and non-transformers:

Data Transformers	Non Data Transformers
• CalculateBlox	• ErrorBlox
• FilterBlox	• FormatBlox
• GroupBlox	• StyleBlox
• MembersBlox	• TextBlox
• OrderBlox	• PersistenceBlox
• SortBlox	• PdfBlox

Note: DataSourceConnectionBlox, JDBCConnectionBlox, RDBResultSetDataBlox and SQLDataBlox are not transformers, but they produce data for the data transformers.

Accessing Individual Blox

A key difference between Relational Reporting Blox and PresentBlox and the other user interface Blox is you can specify an `id` for Blox nested within ReportBlox. With PresentBlox, GridBlox, and ChartBlox, you can only specify the `id` tag attribute for the outmost Blox; nested Blox cannot have an `id`. With Relational Reporting Blox, while the `<bloxreport:report>` tag is the outmost tag, Blox added as a nested tag can still have a unique `id` that you can script to directly. For example,

```
<bloxreport:report id="myReport">
   <bloxreport:sqlData id="mySQLData" ...>
      <bloxreport:dataSourceConnection ... />
   </bloxreport:sqlData>
...
</bloxreport:report>
```

You can then directly script against mySQLData to dynamically change the query.

Columns and Members

Even though Relational Reporting lets you extract data from a relational data source and transform it into reports that are typically in rows and columns, you will find that often times you need to specify the "members" rather than the "columns" in various data transformation operations. "Members" are used to refer to the data fields or data members in the result set. "Columns" are used to refer to the table columns in the rendered report. Generally speaking, when you are styling your report, you are dealing with "columns." When you are performing data transformation tasks, you are dealing with "members." A member may be in a different column position once your users move or re-order the columns through the Report Editor user interface. A member may not be on a column if it becomes a break group.

Styling the Relational Reports

Various elements in the rendered report have associated style classes. The desired appearances of your reports can be customized by specifying the style to use for each of the classes.

A style sheet, report.css, is provided that you can use out of the box. This style sheet resides at:

`<alphablox_dir>/repository/theme/`

where `<alphablox_dir>` is the directory into which DB2 Alphablox is installed.

In this directory, you will find a report.css file. To use the style sheets provided out of the box, in your JSP pages, add

`<link rel="stylesheet" href="/AlphabloxServer/theme/report.css" />`

to the `<head>` section. Since the data is rendered as an HTML table, your report can be displayed even if you do not define the styles either via an external style sheet or inline styles in the JSP page. It is simply displayed as a plain HTML table. When the report is rendered in interactive mode, the style sheets have to exist in order for the context menus and Report Style dialog box to display and work properly.

Style Classes

The classes used in the generated reports include the following:

For the overall report:
- .report
- .loadingmessage

For break groups:
- .groupheader1
- .groupheader2

- .groupheader*N*
- .groupfooter1
- .groupfooter2
- .groupfooter*N*
- .grouptotal1
- .grouptotal2
- .grouptotal*N*

For columns and data style:
- .column
- .columnhover
- .selected
- .data
- .banding

For the interactive context menus:
- .menu
- .choice
- .choicehover
- .separator
- .selected

For the Report Style dialog box:
- .dialog
- .dialogtitle
- .dialogbody
- .dialoggroup
- .dialoggrouptitle
- .dialogradiogroup
- .dialogbutton
- .dialogbuttongroup

Style Classes in the Report

The following two images show the style class each report element uses:

1. .groupheader1
2. .groupheader2
3. .column
4. .data
5. .grouptotal2
6. .groupfooter2
7. .grouptotal1
8. .groupfooter1
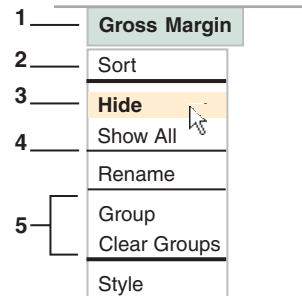
1. .selected
2. .choice
3. .choicehover
4. .separator
5. .menu

With each break group added, the *N* in the classes groupheader*N*, groupfooter*N*, and grouptotal*N* increases by 1.

While the aggregation data at the end of each break group is rendered using the grouptotal*N* style classes, you can add footer text at end of each break group that uses the groupfooter*N* style classes.

You can create your own style sheet that defines how you want each of the classes to look and specify in your JSP file where to find the style sheet. For example, the following is a sample style sheet that demonstrates how you can define the fonts and colors to use for each class.

```
.report {
   background-color: white;
   color: black;
   font-family :Arial, sans-serif;
   border: solid 1 black;
   padding: 5;
   margin : 5;
   width: 0;
}

/* for display of "Report Loading..." message */
.loadingmessage {
   white-space:nowrap;
}

/* break groups */
.groupheader1 {
   background-color: #99CCFF;
   color: black;
```

```
            font-size: 135%;
            text-align: center;
            border: solid white 1;
        }

        .grouptotal1 {
            background-color: white;
            color: black;
            font-size: 120%;
            text-align: right;
            padding : 5;
            border-top: double 3 black;
            border-bottom: double 3 black;
        }

        .groupfooter1 {
            display : none;
        }

        .groupheader2 {
            background-color: #6699CC;
            color: black;
            font-size: 120%;
            text-align: left;
            padding : 2 5;
            border-bottom: solid lightgrey 1;
        }

        .grouptotal2 {
            background-color: white;
            color: black;
            font-size: 100%;
            text-align: right;
            padding 2;
            border-top: solid black thin;
        }

        .groupfooter2 {
            display : none;
        }

        .column {
            color: black;
            font-size: 90%;
            font-weight: bold;
            padding : 2 3;
            text-align: left;
            border: solid white 1;
        }

        .columndragged {
            font-size: 90%;
            font-weight: bold;
            padding : 2 3;
            color: white;
            text-align: left;
            border: solid white 1;
            position: absolute;
            display: none;
            background-color: darkgray;
            cursor: hand;
        }

        .data {
            font-size: 90%;
            text-align: right;
            padding-left:20;
```

```
}

.banding {
   background-color: #CCCCFF;
}

/* Context Menus */
.menu {
   position: absolute;
   background-color: #E3E3E3;
   color: black;
   font-size: 90%;
   font-family: sans-serif;
   text-align: left;
   padding: 1;
   margin: 1;
   cursor: default;
   border: solid white 1;
}

.choice {
   padding : 1 5;
   white-space: nowrap;
   width: 100%;
}

.choicehover {
   background-color: #336699;
   color: white;
   padding : 1 5;
   white-space: nowrap;
}

.separator {
   padding: 1;
   font-size: 0;
   border-top : solid 1 black;
}

.selected {
   cursor: hand;
   border-color: darkgray;
   color: black;
   background-color: lightgrey;
}
```

Style Classes in the Report Style Dialog Box

The Report Style dialog box is rendered in DHTML. By viewing the HTML source from the browser, you will notice how the dialog window is rendered using <DIV> and <SPAN> tags and style classes. All styles associated with the dialog box start with the word "dialog."

Style Classes for ErrorBlox

ErrorBlox displays uncaught errors in a collapsible list using an HTML table. When an error is expanded, detailed error listing (the stack trace) is displayed. For the table and the collapsible list to display appropriately, in your error handling page containing an ErrorBlox, you should also use the report.css style sheet provided.

```
<!--Import the Reporting Blox Tag Library-->
<%@ taglib uri="bloxreporttld" prefix="bloxreport" %>
<%@ page isErrorPage="true" %>

<html>
   <head>
      <title>Error</title>
```

```
    <link rel="stylesheet"
          href="/AlphabloxServer/theme/report.css"
          type="text/css" />
  </head>

  <body>
     <bloxreport:error id="errorBlox" />
  </body>
</html>
```

The names of the style classes for displaying the errors all start with the word
"error":

- `.errorbox`
- `.errorimage` (for the information icon)
- `.errortitlebar`
- `.errormessage` (for the message right next to the information icon)
- `.errornode` (for the expandable/collapsible node for a group of stack traces)
- `td.errorstacktrace` (for each stack trace)
- `.errordetail` (for detailed description for each stack trace)
- `.errorbutton` (for the Hide Details/Show Details button)

Tip: This expand-and-collapse behavior is not supported in Netscape browsers.

Relational Reporting Custom Tags

Custom JSP tags are available for connecting these Relational Reporting Blox.
These tags are packaged in the `bloxreport.tld` file. To use these tags, import the
Blox Report Tag Library as follows:

```
<%@ taglib uri="bloxreporttld" prefix="bloxreport" %>
```

Each Relational Reporting Blox has a corresponding JSP tag with a set of attributes.
The <bloxreport:report> tag is the "wrapper" tag that nest the other tags, as
shown in the following example. You create a relational report by generating a
data result set from a relational data source and then transform the result set into
the format you want through some calculation, sorting, filtering, re-ordering, data
formatting, and report styling.

```
<bloxreport:report>
   <%--Generating a data result set from the data source
   and the SQL query specified--%>
   <bloxreport:sqlData ...>
      <bloxreport:dataSourceConnection ... />
   </bloxreport:sqlData>

   <%--Performing data transformation--%>
   <bloxreport:calculate .../>
   <bloxreport:sort .../>
   <bloxreport:filter .../>
   <bloxreport:group .../>
   <bloxreport:order .../>

   <%--Specifying the text to display for group headers, group
       footers, group totals, column headers, and data cells--%>
   <bloxreport:text>
      <bloxreport:data .../>
      <bloxreport:columnHeader .../>
      <bloxreport:groupHeader .../>
      <bloxreport:groupFooter .../>
      <bloxreport:groupTotal .../>
   </bloxreport:text>
```

```
<%--Formatting the data--%>
<bloxreport:format>
   <bloxreport:numeric .../>
   <bloxreport:date .../>
   <bloxreport:missing .../>
   <bloxreport:html .../>
   <bloxreport:aggregation .../>
</bloxreport:format>

<%--Styling the report using CSS style strings--%>
<bloxreport:style>
   <bloxreport:text ... />
   <bloxreport:numeric ... />
   <bloxreport:date ... />
   <bloxreport:banding ... />
   <bloxreport:missing ... />
   <bloxreport:negative ... />
   <bloxreport:column ... />
   <bloxreport:data .../>
   <bloxreport:columnHeader .../>
   <bloxreport:groupHeader .../>
   <bloxreport:groupFooter .../>
   <bloxreport:groupTotal .../>
</bloxreport:style>
</bloxreport:report>
```

The only tag that can wrap outside the `<bloxreport:report>` tag is the
`<bloxreport:pdf>` tag. You use it when you want your users to view the report
with live data directly in PDF.

```
<bloxreport:pdf>
   <bloxreport:report>
      <bloxreport:sqlData>
         <bloxreport:dataSourceConnection .../>
      </bloxreport:sqlData>
      <bloxreport:calculate .../>
      <bloxreport:sort .../>
      <bloxreport:filter .../>
      <bloxreport:group .../>
      <bloxreport:order .../>
      <bloxreport:format />
      <bloxreport:style />
   </bloxreport:report>
</bloxreport:pdf>
```

All these Blox tags can be assigned an id to uniquely identify the specific instance
of that Blox. Even though in many cases one is not required, it is good practice to
specify a unique id, particularly to every instance of ReportBlox. This allows you
to reference it later to dynamically change its attribute values. When you have
more than one ReportBlox on a page, this allows the DB2 Alphablox to correctly
identify the state of the correct instance of ReportBlox.

Some of the Blox have nested tags for specifications of multiple elements. The
concept of nested tags is discussed next.

Nested Tags

Relational Reporting Blox that have nested tags are FormatBlox, GroupBlox,
SortBlox, StyleBlox, and TextBlox. For example, FormatBlox has nested tags to
define formats for numeric, date, and missing data:

```
<bloxreport:format>
  <bloxreport:numeric format="####.00;(####.00)" />
  <bloxreport:numeric format="$#,###.##;$(#,###.##)"
```

```
    member="Sales" />
  <bloxreport:date format="yyyy.MM.dd G 'at' hh:mm:ss z" />
  <bloxreport:missing format="Value Missing" />
</bloxreport:format>
```

GroupBlox has nested tags to specify the aggregation type for each column member:

```
<bloxreport:group members = "Product, Area">
  <bloxreport:aggregation member = "Sales" type = "sum" />
  <bloxreport:aggregation member = "Cost" type = "average" />
  <bloxreport:aggregation member = "Store" type = "count" />
  <bloxreport:aggregation member = "Units" type = "max" />
</bloxreport:group>
```

SortBlox has one nested tag to define each sorting rule:

```
<bloxreport:sort>
  <bloxreport:rule member="Product" ascending="true" />
  <bloxreport:rule member="Week_Ending" ascending="false" />
</bloxreport:sort>
```

StyleBlox has nested tags to define styles for missing value, negative data, and data columns:

```
<bloxreport:style>
  <bloxreport:missing style="background-color: aqua;" />
  <bloxreport:negative style="color: red;" />
  <bloxreport:column style="background-color: #CCCCCC; color:
white;" columnName="Area" />
  <bloxreport:column style="text-align:center;" columnName="Code" />
</bloxreport:style>
```

It also has nested tags to define styles for group headers, group footers, group totals, column headers, and data cells:

```
<bloxreport:style>
  <bloxreport:groupHeader
    level="1" style="font-size:120%;" />
  <bloxreport:groupFooter
    level="1" style="font-size:120%;" />
  <bloxreport:groupTotal
    columnName="Sales" style="font-size:90%;" />
  <bloxreport:columnHeader
    columnName="Cost" style="font-size:90%;" />
  <bloxreport:data
    columnName="Product" style="color: #FFFFCC;" />
</bloxreport:text>
```

TextBlox has nested tags to define display text for group headers, group footers, group totals, column headers, and data cells:

```
<bloxreport:text>
  <bloxreport:groupHeader
    level="1" text="<b><i>My Group Header Text Here</i></b>" />
  <bloxreport:groupFooter
    level="1" text="My Group Footer Text Here" />
  <bloxreport:groupTotal
    columnName="Sales" text="Total: <value/>" />
  <bloxreport:columnHeader
    columnName="Cost" text="Unit Cost" />
  <bloxreport:data
    columnName="Product" text="<b><value/></b>" />
</bloxreport:text>
```

Standalone Tags

There are three Blox in Relational Reporting that can or have to be added alone in a JSP page: ErrorBlox, PdfBlox, and PersistenceBlox. ErrorBlox is added alone in your error handling page, as described in "Use ErrorBlox for Better Error Reporting" on page 24. PdfBlox can be added outside of the `<bloxreport:report>` tag to directly render a report in PDF. It can also be added alone in a JSP page that takes a ReportBlox ID passed in typically through the HTTP request object and then sends the ReportBlox to PDF, as described in "Rendering a Report Directly in PDF" on page 72 and "Saving Reports as PDF Files" on page 71. PersistenceBlox also takes a ReportBlox passed in and saves the state of the ReportBlox in the repository. For detail, see "Bookmarking Reports and Saving States" on page 68.

The Order of Syntax Evaluation

Within the `<bloxreport:report>` tag, you can flexibly add any other Relational Reporting Blox for formatting and data manipulation to meet your report development needs. Sometimes you may need to have more than one instance of the same tag in order to reach your development goal. Examples include the `<bloxreport:calculate>` for multiple calculated members and `<bloxreport:filter>` for multiple filtering operations. These tags are evaluated in the sequence as they are declared, from top to bottom. Therefore, if you filter out some data using FilterBlox, subsequent operations will no longer contain those data since they no longer exist in the result set. In cases where you have multiple instances of the same tag setting different values for the same attributes, the last value set will be the value used.

Examples of Having Multiple Instances of the Same Tag

Example 1:
```
<bloxreport:format>
  <bloxreport:numeric format="####.00;(####.00)" />
  <bloxreport:numeric format="$#,###.##;$(#,###.##)"
     member="Sales" />
</bloxreport:format>
```

Example 2:
```
<bloxreport:format>
  <bloxreport:numeric format="####.00;(####.00)" />
  <bloxreport:numeric format="$#,###.##;$(#,###.##)"
     member="Sales" />
  <bloxreport:numeric format="$#,###;$(#,###)"
     member="Sales" />
</bloxreport:format>
```

In the first example, all numeric data will be displayed using the `"####.00;(####.00)"` format. The only exception is the data in the Sales column, which will be displayed with in the `"$#,###.##;$(#,###.##)"` format. In the second example, the third `<bloxreport:numeric>` tag overrules the second tag since both applies to member "Sales," resulting in Sales data displayed in the `"$#,###;$(#,###)"` format.

Examples of Different Tags

In the following two examples, the outcomes will be different.

Example 1:

```
<bloxreport:sort
  member="Units"
/>
<bloxreport:members
  excluded = "Units"
/>
```

Example 2:
```
<bloxreport:members
  excluded = "Units"
/>
<bloxreport:sort
  member="Units"
/>
```

In the first example, the result set is first sorted based on the Units member and then the Units member is removed. In the second example, since the Units member is removed from the result set first, the subsequent sort operation will fail.

Expression Syntax

The general rules when specifying the expressions for operation or evaluation are:

- The entire expression should be included in double quotes:

```
<bloxreport:calculate
  expression = "Sales = Unit_Cost * Units_Sold" />

<bloxreport:filter
  expression = "Units > 500" />
```

- If a member name contains characters other than a-z, A-Z, 0-9, and _, it should be enclosed in square brackets ([ ]). If you have spaces or special characters in the member names, always enclose them in [ ]. In cases where a member name may be mistaken for a number, also use [ ] to avoid confusion.

```
<%--The following calculated member has "%" in its name--%>
<bloxreport:calculate
  expression = "[Profits%] = Sales/ [Gross Margin]"
/>

<%--Enclose member names with spaces in []--%>
<bloxreport:calculate
  id = "myCalc"
  expression = "CurrentQuarter = [April 01] + [May 01] +
    [June 01]"
/>
```

- If a member name already contains [ ], use an additional closing "]" to indicate the end of the member name. For example, to specify the member name West[CA] in an expression, you should say [West[CA]]].

- Supported operators:
  - For calculation expressions: +, -, *, and /
  - For filter expressions: =, <, >,and !=

- Supported separator for calculation operators are (). For example:

```
<bloxreport:calculate
  expression = "[Profit%] = Sales/(Unit_Cost * Units_Sold)"
/>
```

- No compound expressions. That is, you cannot connect two expression strings with connectors such as AND, OR, &, or |. Instead, you need to use two FilterBlox for multiple filter operations:

```
<bloxreport:filter
    expression = "Code > 240" />
<bloxreport:filter
    expression = "Code < 400" />
```

- Supported operators only work on numeric data. This includes integer, floating point, and currency. You will not be able to perform calculation, sorting, or filtering on string, date, time, or boolean data types.

Member Identifiers VS. Display Names

When specifying rules for sorting, expressions for data filtering, members to exclude, calculation expression to add a calculated member, and the exact order the members should be in, you are using operations that involve data manipulation. In these cases, you should put member names in square brackets when they do not start with a letter or when they contain special characters (characters other than a-z, A-Z, 0-9, or underscores). This indicates to the DB2 Alphablox Relational Reporting engine that these are variables in the expression. Without square brackets, the reporting system may fail to correctly identify the members. For example, a member named "2002" can be mistaken for a number and a member named "Region-East" can be mistaken for a numeric expression. A valid identifier requires that the name starts with a letter, followed by any combination of a-z, A-Z, underscore, and 0-9. Any name that does not follow this requirement should be placed in square brackets.

The following example shows the use of square brackets to enclose the member names that do not meet the requirement:

```
<bloxreport:calculate
    expression="[Profit%] = [Gross Margin]/Sales" />

<bloxreport:filter expression = "[Q1 Sales] <10000" />

<bloxreport:sort member="[Profit%]" />

<bloxreport:order included = "Product, [Profit%], Sales, Cost" />
```

However, when you are dealing with text displayed in the rendered report, square brackets are no longer required since there is no confusion whether the text string is a member name or an expression. For example, when you StyleBlox to specify how the columns and texts should be displayed, or when you use the `<bloxreport:columnHeader>` tag in TextBlox to specify the column header text, you are operating on the display names in the rendered report. In these cases, you do not need to enclose the member names in square brackets:

```
<bloxreport:report id="salesreport1">
...
   <bloxreport:text>
      <bloxreport:columnHeader
         columnName="Profit%">
         text="Profit Pct" />
      <bloxreport:groupTotal
         columnName="Profit%"
         text="Avg.: <value/>" />
   </bloxreport:text>

   <bloxreport:style>
      <bloxreport:columnHeader style="color: blue;"
         columnName="Profit%" />
   </bloxreport:style>
...
</bloxreport:report>
```

Note: As a rule of thumb, always specify the names of the member in the same cases as they are exacted from the data source. Referring to the member "Cost" as "COST" or "cost" may result in errors.

Chapter 3. Relational Report Development

This section describes the general steps to develop a relational report and discusses common report development tasks. Some tips on design considerations and troubleshooting are provided.

Before You Begin

Before you begin your development, check the *Developer's Guide* for general development preparation tips. You should also ask yourself a few questions to decide if Relational Reporting is what you need and how you should design the report so it meets your users' requirements.

- Is the database from which you want to create a report relational? If you have a multidimensional database, use GridBlox instead.
- Do your users need to chart the data? If yes, use PresentBlox instead.
- Do your users need to analyze the data to find answers or trends? If yes, take advantage of DB2 Cube Views and the DB2 Alphablox cubing engine to create a multidimensional representation of your data. This enables quick multidimensional access to relational data stored in a data warehouse or data mart database via the PresentBlox user interface.
- What data do your users want to see? Users usually do not need to have all of the data. A page that shows all of the data takes a long time to load and may not fit into the viewable area on the screen.

General Development Tips

The following are a few general rules about using and testing JSP tags that you should be aware of:

- Import the DB2 Alphablox Tag Libraries. For Relational Reporting, you need to import the Blox Report Tag Library as follows:

  ```
  <%@ taglib uri="bloxreporttld" prefix="bloxreport" %>
  ```

- Blox tags and attribute names start with all lowercase first word, with the first letter of each subsequent word in uppercase. Examples are:
  - `<bloxreport:sqlData />`
  - `<bloxreport:dataSourceConnection />`
  - `<bloxreport:groupHeader />`
  - `<bloxreport:groupFooter />`
  - `<bloxreport:groupTotal />`
  - `<bloxreport:columnHeader />`

- When specifying the value for an attribute, enclose the value in double or single quotes. Even when the value appears to be numeric, JSP tags require all values for attributes to be enclosed in quotes. In the following example:

  ```
  <bloxreport:calculate
     expression = "[Profit%] = Sales/GrossMargin"
     index = "4"
  />
  ```

The calculated member Profit% will be added as the fifth column with the value of the `index` attribute set to 4. Even though 4 is a number, it still needs to enclosed within quotes.

- When you test a JSP page with Blox tags in your browser, you may want to configure your browser to not cache any content and visit the server to fetch the content with every request.

- During development, often times you make changes to the value of an attribute in a Blox tag and want to test the changes. Since the object on the server already exists for that session, it is reused and therefore will not reflect the changes you make. You will need to start a new session by opening a new instance of the browser. With the Netscape browser, this means you need to close up all browser windows before you open a new one.

- If you need to dynamically change the query or certain Blox attributes, use a dynamic `bloxName` for your ReportBlox. For details, check out the sections on "Managing Session Scope" on page 81 and "Dynamically Changing the Query" on page 88.

- When you want to dynamically set the value of an attribute or a property of a Relational Reporting Blox, it is required that an `id` is specified while the instance of the Blox is created. Note that with the user interface Blox you use to present multidimensional data in grids and charts, you cannot assign an `id` to nested Blox. In Relational Reporting, you can. This gives you greater control of the individual Blox supporting ReportBlox.

- When you have two ReportBlox on a page and one (or both) of them is rendered in interactive mode, you should also specify a unique `id` in order for DB2 Alphablox to correctly identify the state of each instance of the ReportBlox. In general, for ReportBlox, an `id` is required if you want to do any of the following:
  - dynamically set the values of its attributes
  - have two ReportBlox on a page, with one or both rendered in interactive mode
  - pass the current state of the ReportBlox to another JSP page (for example, to send the ReportBlox to PDF or Excel)

- Relational reports are displayed based on the locale of the browser. However, in order for the report to display correctly in languages other than English, you should specify to use the UTF-8 character set using the `page` directive's `contentType` attribute. With the `contentType` attribute, you can define the MIME type and character set.

  ```
  <%@ page contentType="text/html; charset=UTF-8" %>
  ```

- Since interactive reports are rendered in DHTML, if a report contains thousands of data cells, it may take the browser several minutes to render the page. This is a limitation with the browser. For best performance and results, prepare your data in the native environment of your relational data sources and extract only the needed data into ReportBlox. This may involve de-normalizing the data or preparing tables containing summary data.

Note: This Guide focuses on the use of relational reporting tags. For Java methods associated with relational reporting Blox, see the Relational Reporting Javadoc. The Javadoc is available for both the server-side API and the ReportBlox API and can be from the following directory:

```
<alphablox_dir>/system/documentation/javadoc
```

where `<alphablox_dir>` is the directory in which DB2 Alphablox is installed.

General Report Development Steps

There are five general steps to develop a relational report using ReportBlox and its supporting Blox. These steps are described next.

Define the Application and Data Source

As with any Alphablox application development, you need to define the application and the data sources for your application to DB2 Alphablox through the DB2 Alphablox Admin Pages. See the *Administrator's Guide* or online help on the admin pages for details on application and data source definition.

Once you define the application, an application folder with the context name you provided on the admin pages will be automatically created under the webapps/ folder, with a WEB-INF folder that contains the deployment descriptor web.xml and the tlds/ folder that contains the DB2 Alphablox Tag Libraries descriptor files. Keep these files intact.

If you do not have immediate access to a relational database, a relational canned data is provided to help you get started. This will be described in the section on "The Simplest Report" on page 25.

Include the Blox Report Tag Library

You need to include the Blox Report Tag Library in order to use the custom tags for Relational Reporting. Include the following line in the beginning of your JSP page:

```
<%@ taglib uri="bloxreporttld" prefix="bloxreport" %>
```

For general information on using the DB2 Alphablox Tag Libraries, see *Developer's Guide*.

Use a style sheet

Since the rendered reports have associated style classes, you should use an external Cascading Style Sheet to define the styles. This is particularly important when the reports are rendered in interactive mode. Without a defined style for each of the classes associated with the display of context menus and Report Style dialog box, the Report Editor user interface will not work properly.

Since style sheets cascade, as a best practice, use the style sheet provided out-of-the-box and then add your own style sheet to modify only the styles for classes you want to change:

```
<link rel="stylesheet" href="/AlphabloxServer/theme/report.css" />
<link rel="stylesheet" href="yourStyleSheet.css" />
```

In *yourStyleSheet.css*, specify your styles to the classes you want to overwrite. For example, if you want the data cells to have yellow background color:

```
.data {
    color: yellow
}
```

An alternative technique is to import the main style sheet from your custom style sheet:

```
// mystyle.css
@import url( /AlphabloxServer/theme/report.css );
.data {
   color: green
}
```

Important: When you upgrade or install new version of DB2 Alphablox, the
default theme file `report.css` in `<alphablox_dir>/repository/theme/`
will be overwritten. Therefore, you should create your own copy of the
style sheet rather than modify the `report.css` directly.

Use ErrorBlox for Better Error Reporting

You should specify an error handling page with every JSP page you create for
better error reporting. In Relational Reporting, ErrorBlox catches the uncaught
exceptions thrown and prints the details in an HTML table using Cascading Style
Sheet. To use ErrorBlox, you can create an error report JSP page with the following
lines:

```
<%@ taglib uri="bloxreporttld" prefix="bloxreport" %>
<%@ page isErrorPage="true" %>

<html>
   <head>
      <link rel="stylesheet" href="/AlphabloxServer/theme/report.css"
      type="text/css" />
   </head>

   <body>
      <bloxreport:error id="errorBlox" />
   </body>
</html>
```

The first line specifies the tag libraries description (TLD) file to load in order to use
the `<bloxreport:error>` tag. The second line identifies itself as an error reporting
page. Inside the `<head>` tag, a style sheet supplied by DB2 Alphablox is specified to
display the errors in an easy-to-read table. Then an ErrorBlox is added inside the
`<body>` tag. This is all you need for a basic custom error handling page.

Then in your JSP pages containing ReportBlox, add the following line to point to
the error reporting page you just created as the error handling page:

```
<%@ page errorPage="yourError.jsp" %>
```

You can customize the styles used to display the errors. Note that when you
upgrade or install new version of DB2 Alphablox, the default theme file
`report.css` in `<alphablox_dir>/repository/theme/` will be overwritten. Therefore,
you should create your own copy of the style sheet rather than modify the
`report.css` directly.

For more information on error handling and the use of ErrorBlox, see "Error
Handling Using ErrorBlox" on page 97. For more information on ErrorBlox style
classes, see "Style Classes for ErrorBlox" on page 13.

Add Blox Tags

To add a Relational Reporting Blox on your JSP page, you can copy the syntax
from examples provided or from the Chapter 12, "Relational Reporting Blox Tag
Reference," on page 99. Start with the `<bloxreport:report>` tag as it wraps outside
of all other Blox for data retrieval, manipulation, and formatting. See "Relational
Reporting Custom Tags" on page 14 for details on how you connect these Blox and
what the general rules are for expression syntax.

Creating Your First Relational Report

After you have defined the application and your relational data source to DB2 Alphablox, you can proceed to create your first relational report.

The Simplest Report

For every relational report you create, there are three essential Blox that you will need:

- ReportBlox
- SQLDataBlox
- DataSourceConnectionBlox

DataSourceConnectionBlox lets you connect to your relational data source. SQLDataBlox lets you specify the SQL command to extract the needed data from your data source. ReportBlox lets you generate the output in an HTML table. With these Blox you can produce your first relational report. The following is all the JSP code needed. This is also the code needed for basically every report you create.

```
<%@ taglib uri="bloxreporttld" prefix="bloxreport" %>

<html>
<head>
</head>
<body>

<bloxreport:report id = "myReport">
   <bloxreport:sqlData query = "SELECT... FROM... WHERE...">
      <bloxreport:dataSourceConnection
         dataSourceName = "yourRDBdataSource">
      </bloxreport:dataSourceConnection>
   </bloxreport:sqlData>
</bloxreport:report>

</body>
</html>
```

Note that in Relational Reporting, member names are case sensitive. In your SQL query statement, if you rename columns in the SELECT list, be sure to enclose the column names in double quotes if you expect the case to be preserved. For example:

```
SELECT FROM myTable total_sq_ft AS "Sq_Ft", sq_ft_pct AS "Pct"
```

In the `<bloxreport:sqlData>` tag, you will need to escape the quotation marks with back-slashes:

```
<bloxreport:sqlData
query = "SELECT FROM myTable total_sq_ft AS \"Sq_Ft\", sq_ft_pct AS \"Pct\"">
```

Task: Create a Simplest Report

1. In you JSP development environment, open a new file and copy and paste the above code into your file.

2. Enter your SQL query and relational data source name.

   Note: If you do not have immediate access to a relational database, a canned relational data is available for use. This canned data is actually a Java class and does not understand SQL. To use it, add the `<bloxreport:cannedData />` tag to your code, replacing `<bloxreport:sqlData>` and `<bloxreport:dataSourceConnection>`:

```
<%@ taglib uri="bloxreporttld" prefix="bloxreport" %>
<html>
<head>
</head>
<body>

<bloxreport:report id="myReport">
    <bloxreport:cannedData/>
</bloxreport:report>

</body>
</html>
```

3. Save your file with `.jsp` as the extension into your application folder. Now you can open a browser window and test this file.

The Simplest Interactive Report

To create your first interactive report involves two additional steps:

1. Add the `interactive` attribute to your `<bloxreport:report>` tag and set the value to `true`.

   ```
   <bloxreport:report id="myReport" interactive="true">
       ...
   </bloxreport:report>
   ```

   The `interactive` attribute is set to false by default.

2. Add a reference to the supplied style sheets in the `<head>` section.

   ```
   <head>
       <link rel="stylesheet" href="/AlphabloxServer/theme/report.css" />
   </head>
   ```

Task: Create a Simplest Interactive Report

1. Open the simplest report you created in "Task: Create a Simplest Report" on page 25.
2. Insert `interactive="true"` inside the `<bloxreport:report>` tag.
3. Insert the reference to the supplied style sheet in the `<head>` section.
4. Save the file.
5. Open a new browser window and test the JSP page you just saved.

Important: During your development, if you make a change to an attribute value of a Blox tag, since the server already instantiates the object and the object exists during the session scope, you will need to close the browser window or test the modified JSP page using a different instance of the browser in order to see the changes you made.

Now that you have created your first report, you can continue to add more Blox to further transform the result set and format the report.

Learning Resources

The subsequent chapters in this *Relational Reporting Developer's Guide* are organized by task. Besides going to the chapters that discuss tasks of interest, make sure you check out Chapter 11, "Development and Troubleshooting Tips," on page 93 for essential tips that will aid you in your report development. The Blox Sampler - Relational Reporting example set in the Application Studio has live examples that demonstrate many of the tasks detailed in this book.

Chapter 4. Accessing and Retrieving Data

Getting data into a ReportBlox is the first step to building your relational report. This chapter discusses different approaches that you can feed data into a ReportBlox.

- "Using SQLDataBlox and DataSourceConnectionBlox" on page 27
- "Using RDBResultSetDataBlox to Access RDBResultSet from DataBlox" on page 28
- "Error Handling Against SQLDataBlox" on page 30

Using SQLDataBlox and DataSourceConnectionBlox

DataSourceConnectionBlox handles data connection, providing access to your relational data sources defined to DB2 Alphablox through the DB2 Alphablox Admin Pages. It handles connection properties such as `dataSourceName`, `username`, and `password`. SQLDataBlox extracts the data based on the SQL query you specified, providing data to the Relational Reporting data pipeline.

As described in "The Simplest Report" on page 25, the general outline of you JSP looks as follows:

```
<%@ taglib uri="bloxreporttld" prefix="bloxreport" %>
<html>
<head>
   <link rel="stylesheet" href="/AlphabloxServer/theme/report.css" />
</head>
<body>

<bloxreport:report id = "myReport">
   <bloxreport:sqlData query = "SELECT... FROM... WHERE...">
      <bloxreport:dataSourceConnection
         dataSourceName = "yourRDBdataSource">
      </bloxreport:dataSourceConnection>
   </bloxreport:sqlData>
</bloxreport:report>

</body>
</html>
```

For details on DataSourceConnectionBlox and its tag attributes, see DataSourceConnectionBlox on page 103. For details on SQLDataBlox and its tag attributes, see SQLDataBlox on page 127. For a list of supported relational data sources, see the System Requirements section in the *Installation Guide*.

Dynamically Setting the Query

If you need to dynamically set or change the query, SQLDataBlox has a `setQuery(`*`queryString`*`)` method and an `execute()` method. You must call the `execute()` method after the new query is set. For a complete example, see "Dynamically Changing the Query" on page 88 and the accompanying example in the Blox Sampler - Relational Reporting example set in the Application Studio.

Using RDBResultSetDataBlox to Access RDBResultSet from DataBlox

Another way data can be fed to a ReportBlox is via a RDBResultSetDataBlox. RDBResultSetDataBlox allows you to create a relational report using the RDBResultSet returned from an existing DataBlox. DataBlox provides data to user interface Blox such as PresentBlox, GridBlox, and ChartBlox (tags for these Blox are in the Blox Tag Library, available with the `<%@ taglib uri="bloxtld" prefix="blox" %>` taglib directive). DataBlox cannot provide data directly to ReportBlox. However, RDBResultSet returned from a DataBlox can be used as the source of data for Relational Reporting via RDBResultSetDataBlox. This is useful for applications where relational details for data in a GridBlox cell from a multidimensional data source can be presented in an attractive, easy-to-read layout.

DB2 Alphablox's drill-through support for Microsoft® Analysis Services data sources in GridBlox and DataBlox, for example, uses a RDBResultSetDataBlox that takes the DataBlox's RDBResultSet to generate a relational report with ReportBlox. You simply set the GridBlox `drillThroughEnable` property to `true` and no custom code is needed. When users choose to drill through from a data cell, relational detail for the cell is displayed in a pre-formatted report using ReportBlox.

For custom report, you can supply your own JSP containing a ReportBlox formatted to you liking. A live example is available in the MSAS version of Blox Sampler, under the Retrieving Data section.

To feed the RDBResultSet of a DataBlox to a ReportBlox, in the calling JSP containing the GridBlox (or PresentBlox), you should call the JSP containing the ReportBlox by passing along three pieces of information:

- The `id` of the DataBlox whose RDBResultSet the ReportBlox will use as the data producer.
- The coordinates (`colIndex` and `rowIndex`) of the cell whose relational details is requested.

The JSP containing the ReportBlox may looks as follows:

```
<%@ taglib uri="bloxreporttld" prefix="bloxreport" %>
<html>
<head>
   <link rel="stylesheet" href="/AlphabloxServer/theme/report.css" />
</head>
<body>

<bloxreport:report id="drillThroughFromDataBlox" ...>

   <bloxreport:rdbResultSetData
      bloxRef="myDataBlox"
      columnCoordinate="<%= request.getParameter(\"colIndex\") %>"
      rowCoordinate="<%= request.getParameter(\"rowIndex\") %>"
   />

   <%--further data manipulation and report formatting--%>
   ...

</bloxreport:report>
</body>
</html>
```

The value for the `bloxRef` attribute should be the `id` of the DataBlox defined in the calling JSP. The `colIndex` and `rowIndex` parameters are passed in from a scriptlet or Java class from the DHTML client in the calling JSP.

For example, assume the calling JSP has a DataBlox as follows:

```
<blox:data id="myDataBlox"
    dataSourceName="QCC-MSAS"
    selectableSlicerDimensions="Measures"
    query="yourQueryString"/>
```

For the DHTML client, you may have a PresentBlox that uses `myDataBlox`:

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri='bloxuitld' prefix='bloxui'%>
...
<body>
<blox:present id="myPresentBlox"
    width="700"
    height="580">
    <blox:data bloxRef="myDataBlox"/>
    <blox:grid drillThroughEnabled="true" />
    <bloxui:actionFilter
        className="<%= myCustomDrillThrough.class.getName() %>"
            componentName="dataAdvancedDrillThrough" />
</blox:present>
...
```

When the user chooses the Advanced... > Drill through option from the right-click menu (componentName = "dataAdvancedDrillThrough"), the myCustomDrillThrough class is called. The class may look as follows:

```
import com.alphablox.blox.uimodel.core.grid.GridCell;
import com.alphablox.blox.uimodel.tags.IActionFilter;
...
public class myCustomDrillThrough implements IActionFilter
{
    public void actionFilter( DataViewBlox blox, Component component )
throws Exception {
        GridBrixModel grid =
((PresentBloxModel)blox.getBloxModel()).getGrid();
        GridCell[] cells = grid.getSelectedCells();

        // Make sure that a single data cell is selected
        if ( cells.length != 1 || cells[0].isRowHeader() ||
cells[0].isColumnHeader() || !(cells[0] instanceof GridBrixCellModel ))
        {
            MessageBox.message( component, "Error", "You must select a
single data cell to drill through" );
            return;
        }

        GridBrixCellModel cell = (GridBrixCellModel)cells[0];
        int rowIndex = cell.getNativeRow();
        int colIndex = cell.getNativeColumn();
        String bloxName = blox.getBloxName();

        String urlStr = "myDrillThrough.jsp?bloxRef="+bloxName;
        urlStr += "&colIndex=";
        urlStr += colIndex;
        urlStr += "&rowIndex=";
        urlStr += rowIndex;
        String timestamp = String.valueOf(System.currentTimeMillis());
        urlStr += "&reportName=";
        urlStr =  urlStr + "reportBlox"+timestamp;

        ClientLink link =  new ClientLink( urlStr,
"reportBlox"+timestamp);
        component.getDispatcher().showBrowserWindow( link );
    }
}
```

For a complete example, see the Retrieving Data section under Blox Sampler.

If you are using the Java client, you can use the `eventTriggerDrillThrough` JavaScript callback property:

```
<blox:data
<blox:present id="myPresentBlox"
   mayscriptEnabled="true"
   width="700"
   height="580">
   <blox:data
      bloxRef="myDataBlox"/>
   <blox:grid
      drillThroughEnabled="true"
      eventTriggerDrillThrough="multipleDTWindows"/>
</blox:present>
```

When the user chooses to drill through, a JavaScript function called multipleDTWindow in the same calling JSP gets triggered :

```
...
<head>
<blox:header/>

<script language="JavaScript">
function multipleDTWindows(var1,var2,var3) {
   var bloxName;
   var colIndex;
   var rowIndex;
   var eventObj = var1;

   bloxName = "myDataBlox";
   colIndex = eventObj.getEventProperty(eventObj, "column");
   rowIndex = eventObj.getEventProperty(eventObj, "row");

   var urlStr = "someReportBlox.jsp?bloxRef="+bloxName;
   urlStr += "&colIndex=";
   urlStr += colIndex;
   urlStr += "&rowIndex=";
   urlStr += rowIndex;
   var currentDate = new Date();
   var timestamp = currentDate.getTime();
   urlStr += "&reportName=";
   urlStr =  urlStr + "reportBlox"+timestamp;
   window.open(urlStr,"reportBlox"+timestamp);
   return false;
}
</script>
</head>
```

For a complete example, see the Retrieving Data section in Blox Sampler.

Error Handling Against SQLDataBlox

When the data query produces no data, the default message displayed in the report is "No data." You can customize this message using the ReportBlox's `noDataMessage` tag attribute. If the query is not valid or the data source happens to be unavailable at the time, your users will see a list of JSP errors. In this case, you may want to set ReportBlox's `errors` property to `false` so exceptions thrown are not intercepted. You can then add a try/catch block to identify the problem and display a more friendly error message. See "Error Handling Using ErrorBlox" on page 97 for more discussions and a code example. See the tag reference for ReportBlox on page 123 for details on the `noDataMessage` and `noDataDueToErrorMessage` tag attributes.

Chapter 5. Processing and Manipulating Data

Once you retrieve data into a ReportBlox, you can add more Blox to process and manipulate the data. This chapter discusses various common data processing and manipulation tasks such as sorting and filtering data, hiding, removing, and reordering columns, and adding calculated columns. These involve data "transformer" Blox, such as SortBlox, FilterBlox, CalculateBlox, GroupBlox, MembersBlox, and OrderBlox.

The "Sorting, Filtering, Hiding, and Calculating Data" example in the Blox Sampler for Relational Reporting demonstrates most of the Blox discussed in this chapter.

Sorting Data

To sort your data, use SortBlox. With a simple, single level of grouping, such as grouping a report by Product, you can specify the following:

```
<bloxreport:sort member="Product" />
```

By default, the report will be sorted in ascending order. You can specify the sort order using the ascending attribute:

```
<bloxreport:sort member="Product"
   ascending="false" />
```

By default, missing data are displayed last. You can specify to have them displayed first:

```
<bloxreport:sort member="Product"
   ascending="false"
   missingLast="false" />
```

Notice the SortBlox's member attribute is in singular form, meaning you can only sort on one member using this syntax. SortBlox supports compound sorting using multiple sort rules. Each rule is a nested tag within a SortBlox, and for each rule you need to specify the member you want to sort on:

```
<bloxreport:sort>
   <bloxreport:rule
      member="Type"
      missingLast="true"
      ascending="true" />
   <bloxreport:rule member="Product" />
</bloxreport:sort>
```

Keep in mind that the order you specify the rules makes a difference. The first rule will be the primary sort; the second, the secondary sort.

Important: Having multiple sort rules is different from having multiple SortBlox. If you add multiple SortBlox, the later one will override the sort operation performed earlier.

SortBlox is the only Blox that handles sorting. If you want a report grouped by product type, and also want the break groups appear in alphabetical order, you should sort the data first before you group them. This is because GroupBlox only handles data grouping and does not sort the data before grouping. For example, if you want a report to be grouped by "Type" and then "Product," with the break groups listed in alphabetical order, then you should sort on "Type" first, then "Product", and then "Sales" before you add the break groups:

```
<bloxreport:sort>
   <bloxreport:rule
      member="Type"
      missingLast="true"
      ascending="true" />
   <bloxreport:rule member="Product" />
   <bloxreport:rule member="Sales" />
</bloxreport:sort>
<bloxreport:group members="Type, Product" />
```

Note that:

- There are three sort rules in this SortBlox. Each sort rule involves only one member. Nesting your sort rules using one SortBlox makes it a compound sorting operation. If you used multiple SortBlox, the later SortBlox would override the previous ones and would not preserve the previous sort results.
- SortBlox is added before the GroupBlox, so the report will be grouped with the different levels of break groups appear in alphabetical order, as shown in the above screen shot.

For details on adding break groups, see "Overview of Break Groups and Break Group Levels" on page 57. For details on SortBlox and SortBlox tag attributes, see SortBlox on page 125. For details on GroupBlox and GroupBlox tag attributes, see GroupBlox on page 111.

Filtering Data

To filter data based on some criterion, use FilterBlox. Each FilterBlox takes one comparison expression. You can add multiple FilterBlox, each with its own comparison expression. For example, if you are only interested in data whose sales numbers fall between the range of 200 and 401, you would use two FilterBlox as follows:

```
<bloxreport:report id="myReport">
   ...
   <bloxreport:filter expression="Sales > 200" />
   <bloxreport:filter expression="Sales < 401" />
   ...
</bloxreport:report>
```

Since the second FilterBlox will filter the data based on the result of the first FilterBlox, if you need to perform operations such as keeping only sales data that are either greater than 400 or smaller than 200, you need to prepare the data in your database environment before retrieving them into ReportBlox.

Note that FilterBlox only works on numeric data. It supports four operators for comparison: >, <, =, and !=. It provides the isMissing() function for filtering out missing data (or for keeping missing data only).

Important: By default, FilterBlox does not filter out missing data unless you specify so:

```
<bloxreport:filter expression="not isMissiing(Sales)" />
```

It is important to keep in mind the different behaviors and tag syntax between FilterBlox and SortBlox:

FilterBlox	SortBlox
You can chain multiple FilterBlox for compound filtering.	There can only be one SortBlox. The one declared last will override the previous ones. For compound sorting use multiple sort rules.
Each FilterBlox takes only one filter expression.	SortBlox can take multiple sort rules, each specified using a nested `rule` tag.
You can only filter number data.	You can sort on numeric data, strings, dates, and time.

See FilterBlox on page 106 for details on FilterBlox usage and "Expression Syntax" on page 18 for more discussion on how to specify member names if they contain spaces or special characters.

Grouping Data

To add grouping to your report, use GroupBlox:

```
<bloxreport:group members="Area, Location" />
```

The GroupBlox's `members` attribute lets you specify the members to group on. The above example will group the report by Area and then by Location. The report is grouped based on the sequence of the members are specified. Once a GroupBlox is added, you can:

- Get the aggregation value for each column in a break group
- Calculate group-bases summary columns to show ranking, running totals, percentage of totals, and running counts.
- Specify texts for group totals based on break group level
- Specify the break group header and group footer text based on break group level

Since these involve multiple closely related tasks, they are discussed in details in "Overview of Break Groups and Break Group Levels" on page 57. For details on GroupBlox tag syntax, see GroupBlox on page 111.

Adding Calculated Columns

You can add calculated columns using CalculateBlox. To specify the new column name and the calculation expression, use the `expression` tag attribute:

```
<bloxreport:calculate
    expression = "Profit = Sales - Cost"
/>
```

This creates a calculated column called Profit, whose values are derived by subtracting Cost from Sales. If the member names involve special characters or spaces, enclose the names in square brackets:

```
<bloxreport:calculate
   expression = "[Profit%] = Sales/ [Gross Margin]"
/>
```

By default, the new member is added to the end of the Column dimension. To specify the exact column position where the new member should be added, use the index attribute. Column counts start with 0, so in the following example, the new member "Profit%" will be added as the first column.

```
<bloxreport:calculate
   expression = "[Profit%] = Sales/ [Gross Margin]"
   index = "0"
/>
```

Note the following when you use CalculateBlox:

- If any member used in the calculation involves missing or null values, the calculation will result in missing data. See "Calculations Involving Missing Data" on page 34 for more information.
- If a member name already contains [ ], use an additional closing "]" to indicate the end of the member name. For example, to specify the member name West[CA] in an expression, you should say [West[CA]]].
- Supported operators for calculation expressions are +, -, *, and /
- Supported separator for calculation operators are (). For example:

```
<bloxreport:calculate
   expression = "[Profit%] = Sales/(Unit_Cost * Units_Sold)"
/>
```

- Supported operators only work on numeric data. This includes integer, floating point, and currency. You will not be able to perform calculations on string, date, time, or boolean data types.
- Four functions are supported in the calculation expressions: rank(), percentOfTotal(), runningTotal(), and runningCount(). These functions are related to how the report is grouped, and are discussed in details in "Calculating Group-based Summary Columns" on page 62.

Calculations Involving Missing Data

When one of the operands involved contains missing value, the result will be considered missing or Not a Number (NaN). This is different in column-wise aggregations, where missing data is ignored. The following table shows how missing data is treated in calculations depending on whether it is group total aggregation or a calculated column using CalculateBlox.

	Product A	Product B	A + B
Store A	5	10	15
Store B	4	10	14
Store C	3	7	10
Store D		3	
Store E	4	5	9
	Total: 16	Total: 35	Total: 48
	Count: 4	Count: 5	Count: 4
	Avg: 4	Avg: 7	Avg: 12

The above table shows the data for two products in five stores. Store D's data for Product A is missing. For Store D, the value for the calculated member A+B will be considered missing. For the aggregation value for each column, the missing data is ignored. Note that by default an empty string is displayed when the value is missing. To specify the display text when the data is missing, see "Formatting Data" on page 45.

Adding Calculated Members Before Grouping

Because of the underlying data pipeline model, the aggregation value for the calculated member can be different depending on where the GroupBlox is added in relation to CalculateBlox. The following example demonstrates the differences.

GroupBlox Before CalculateBlox

```
<bloxreport:group members="Product">
<bloxreport:calculate expression = "[Z] = X*Y" />
```

X	Y	Z
2	5	10
3	5	15
5	10	**50**

CalculateBlox Before GroupBlox:

```
<bloxreport:calculate expression = "[Z] = X*Y" />
<bloxreport:group members="Product">
```

X	Y	Z
2	5	10
3	5	15
5	10	**25**

If a GroupBlox is added first, the aggregation values of X and Y will be used to calculate the aggregation value of the calculated member Z. If a GroupBlox is added afterwards, the aggregation value for Z will be calculated based on the data in the column.

The advanced calculation functions `rank()`, `percentOfTotal()`, `runningTotal()`, and `runningCount()` allow you to specify how you want to calculate the rank (percent of total, running total, and running count) in relation to the break groups. When using these advanced calculation functions, you need to have break groups added first before specifying the calculation functions. Since the usage of these functions are closely related to break groups, the details are discussed in "Calculating Group-based Summary Columns" on page 62 in the Chapter 7, "Grouping Data," on page 57 chapter.

Removing Members

You can specify which members to be included in or excluded from the result set. Members excluded are no longer available for subsequent operations. For example, to remove "Cost" and "Units_Sold" from the data pipeline, you would use:

```
<bloxreport:members excluded="Cost, Units_Sold" />
```

Unlike excluded members in OrderBlox, excluded members in MembersBlox is no longer in the result set. Because they are no longer in the result set, when users choose to Show All from the interactive Column Header Context Menu, they will not be returned.

You can add multiple `<bloxreport:members>` tags in a `<bloxreport:report>` tag. Each MembersBlox takes only either the `excluded` or the `included` attribute. If you specify both attributes within one `<bloxreport:members>` tag, the last attribute will be accepted and the earlier one will be ignored.

MembersBlox does not deal with the ordering of the members. If you have members A, B, C, D, and E in the result set in that order, with

```
<bloxreport:members included="E, D, C" />
```

The members and their order in the result set becomes C, D, and E. To specify the ordering of the members, use OrderBlox. For more information on hiding members without removing them from the data pipeline, see "Hiding and Showing Members" on page 36. For more information on MembersBlox tag attributes, see MembersBlox on page 113.

Hiding and Showing Members

You can hide a member in the result set from users by using OrderBlox. OrderBlox lets you specify which members to temporarily include or exclude in the result set and in what order. The `<bloxreport:order>` tag has two attributes: `excluded` and `included`. `excluded` lets you specify members to hide; `included` lets you specify the members to show, and the order you specify the members is the order they will be in.

For example, assume you have the following six members in the result set:

Product	Location	Sales	Cost	Gross_Margin	Units_Sold

To hide "Cost" and "Units_Sold" from your users, you would use:

```
<bloxreport:order excluded="Cost, Units_Sold" />
```

You can also re-order the columns in the result set using OrderBlox. To specify the columns and their order to appear in the report, set the value of the `included` attribute to the list of members you want to include separated by commas. The order you specify the members is the order they appear in the report from left to right. For example:

```
<bloxreport:order included="Product, Gross_Margin, Sales" />
```

makes Product, Gross_Margin, and Sales the first three columns in the result set. All other members become hidden.

You can add multiple `<bloxreport:order>` tags in a `<bloxreport:report>` tag. Each OrderBlox takes only either the `excluded` or the `included` attribute. If you specify both attributes within one `<bloxreport:order>` tag, the last attribute will be accepted and the earlier one will be ignored.

Members excluded are still is the result set and available for subsequent operations. When the `interactive` attribute in the `<bloxreport:report>` tag is set to `true`, users can still see the excluded members when they choose to **Show All** from the interactive Column Header Context Menu.

To permanently remove a member from the report, see "Removing Members" on page 35. For hiding missing data, see "Hiding and Showing Missing Data" on page 37. For more information on OrderBlox tag attributes and usage, see OrderBlox on page 115. The following table summaries the similarities and differences between MembersBlox and OrderBlox:

MembersBlox	OrderBlox
Allows you to remove members from the result set. Excluded members are no longer in the data pipeline, and therefore not available to subsequent data transformation.	Allows you to hide members from users. Excluded members are still in the data pipeline, available for subsequent data transformation tasks.
Does not set the order of the members with its `included` attribute.	Sets the order of the members with its `included` attribute.
Each MembersBlox tag can only take either the `included` or the `excluded` attribute.	Like MembersBlox, each OrderBlox tag can only take either the `included` or the `excluded` attribute.
Members not in the `included` tag attribute are excluded (permanently removed from the data pipeline).	Members not in the `included` tag attribute are excluded (temporarily hidden from users)

Hiding and Showing Missing Data

If you want to filter out missing data, FilterBlox has an `isMissing(`*memberName*`)` function that allows you do so. In the following example, only rows whose Type (product type) and Gross Margin data are not missing are returned:

```
<bloxreport:filter expression = "not isMissing(Type)"/>
<bloxreport:filter expression = "not isMissing([Gross Margin])"/>
```

Similarly, if you want to show only rows whose sales data is missing, you can specify:

```
<bloxreport:filter expression="isMissiing(Sales)" />
```

FilterBlox only takes one expression at a time. For details on filtering data, see "Filtering Data" on page 32. For FilterBlox syntax and usage, see FilterBlox on page 106.

Note that you can use FormatBlox to display the desired text when the data is missing. The `<bloxreport:missing>` tag nested within `<bloxreport:format>` lets you specify the string to display for the named member when the data is missing:

```
<bloxreport:format>
    <bloxreport:missing format = "Sales value missing" member = "Sales" />
    <bloxreport:missing format = "Units value missing" member = "Units" />
</bloxreport:format>
```

The `format` attribute will only take a string. You cannot assign any calculation expression or variables as the value. To specify the text displayed for missing data, you can use StyleBlox. StyleBlox lets you specifying the styles to use for missing data or data with negative values, among other things. The following example sets the font style and colors for displaying missing data:

```
<bloxreport:style>
    <bloxreport:missing style="font-style: italic;
        color: white;background-color: gray;"/>
</bloxreport:style>
```

For more on setting data display format using FormatBlox, see "Formatting Data" on page 45. For more on setting styles using StyleBlox, see StyleBlox on page 128 for its syntax and usage.

Chapter 6. Formatting the Report and Data

This section describes how you can format the general layout of a report such as the column widths, font colors and styles, text background color, background images, or size of the display area. This involves primarily the use of FormatBlox for data format specification, StyleBlox for styling the font size, text/background colors, and text alignment, and TextBlox for specifying the displayed text in the report. These three Blox are not data transformers, and are typically added at the end of your report JSP after the data transformation tasks are done.

The "Formatting the Report and Data" example in the Blox Sampler for Relational Reporting demonstrates most of the tasks discussed in this chapter.

Display Areas in a Rendered Report

A relational report is rendered into several general areas. Column headers and data are the two display areas available in every report. When a report is grouped, then group headers, group footers, and group totals areas become available.

In the example above, the report is grouped by Product, creating:

- Area 1: level 1 group header area for the overall report
- Area 2: level 2 group header area for each of the Product category
- Area 3: column header area (also available in reports without grouping)
- Area 4: data area (also available in reports without grouping)
- Area 5: level 2 group total area for aggregations of numeric data within each Product category
- Area 6: level 2 group footer area for each Product category
- Area 7: level 1 group total area for aggregations of numeric data for the overall report
- Area 8: level 1 group footer area for the overall report

If the report is grouped by Product and then Week_Ending, then a level 3 is added with a group header area, group footer area, and group total area for each week's data.

For each of the display areas in the report, you can use TextBlox to specify the text to display, rename the column headers, set the group header/footer/total texts, or even modify the displayed data or break group member names or wrap HTML code around them. Likewise, you can use StyleBlox to specify the style for each of the report areas.

Report Layout Formatting and Styling Summary Table

The following table provides a summary of how the different areas in a rendered report can be customized. Attributes in square brackets ([ ]) are optional. When the optional attributes are not specified, the text or style will be applied to all columns or all levels.

To set text using TextBlox	To set styles using StyleBlox	To set styles using CSS style classes
Column Headers		

Use the columnHeader sub tag:
 	
`<bloxreport:text>`
  `<bloxreport:columnHeader`
  `text="`*`new column header`*`"`
  `columnName="`*`columnName`*`"`
  `/>`
`</bloxreport:text>`
	Use the columnHeader sub tag:
	
	`<bloxreport:style>`
	  `<bloxreport:columnHeader`
	  `style="`*`CSS style string`*`"`
	  `[columnName="`*`columnName`*`"]`
	  `/>`
	`</bloxreport:style>`
		`.column`

Data Cells		

Use the data sub tag:
 	
`<bloxreport:text>`
  `<bloxreport:data`
  `text="`*`new data text`*`"`
  `[columnName="`*`columnName`*`"]`
  `/>`
`</bloxreport:text>`
	Use the data sub tag:
	
	`<bloxreport:style>`
	  `<bloxreport:data`
	  `style="`*`CSS style string`*`"`
	  `[columnName="`*`columnName]`*`"`
	  `/>`
	`</bloxreport:style>`
		`.data`

Group Footers		

Use the groupFooter sub tag:
 	
`<bloxreport:text>`
  `<bloxreport:groupFooter`
  `text="group footer text"`
  `[level="`*`N`*`"]`
  `/>`
`</bloxreport:text>`
	Use the groupFooter sub tag:
	
	`<bloxreport:style>`
	  `<bloxreport:groupFooter`
	  `style="`*`CSS style string`*`"`
	  `level="`*`N`*`"`
	  `/>`
	`</bloxreport:style>`
		`.groupfooter1,`
		`.groupfooter2, ...`
		`.groupfooter`*`N`*

Group Headers		

Use the groupHeader sub tag:
 	
`<bloxreport:text>`
  `<bloxreport:groupHeader`
  `text="group header text"`
  `[level="`*`N`*`"]`
  `/>`
`</bloxreport:text>`
	Use the groupHeader sub tag:
	
	`<bloxreport:style>`
	  `<bloxreport:groupHeader`
	  `style="`*`CSS style string`*`"`
	  `level="`*`N`*`"`
	  `/>`
	`</bloxreport:style>`
		`.groupheader1,`
		`.groupheader2, ...`
		`.groupheader`*`N`*

Group Totals		

Use the groupTotal sub tag:
 	
`<bloxreport:text>`
  `<bloxreport:groupTotal`
  `text="group total text"`
  `[level="`*`N`*`"]`
  `[columnName="`*`columnName`*`"]`
  `/>`
`</bloxreport:text>`
	Use the groupTotal sub tag:
	
	`<bloxreport:style>`
	  `<bloxreport:groupTotal`
	  `style="`*`CSS style string`*`"`
	  `level="`*`N`*`"`
	  `[columnName="`*`columnName`*`"]`
	  `/>`
	`</bloxreport:style>`
		`.grouptotal1,`
		`.grouptotal2, ...`
		`.grouptotal`*`N`*

More detailed tasks are discussed throughout this chapter. Also, since the availability of the group headers, footers, and totals depends on whether and how the report is grouped, more details are discussed in the chapter on Chapter 7, "Grouping Data," on page 57.

Styling VS. Formatting VS. Setting Text

The words "styling" and "formatting" are sometimes used interchangeably to refer to everything from adding a title, specifying column widths, setting font sizes, colors, and alignment, to having numeric data formatted in a certain way. Since Relational Reporting uses cascading style classes and renders reports in DHTML, it is important to differentiate styling-related tasks that are based on CSS principles from data formatting tasks that have nothing to do with CSS.

StyleBlox in Relational Reporting provides a way for you to style the data in a report based on the data type or member using CSS style strings. For example, you can set the style for numeric data, text data, negative data values using a CSS style string such as "font-size: 85%; color: white; background-color: blue;."

FormatBlox, on the other hand, lets you specify the data display format for dates and numeric data by following Java's format masks. The following table compares the differences between StyleBlox and FormatBlox:

StyleBlox	FormatBlox
Sets display style (such as text size, font, color, alignment, and background color) for different data types and report areas.	Sets the display format for numeric, date, and missing data.
Styles specification follows CSS principles.	Format specification follows Java format masks.
Nested tags: • `numeric` • `date` • `missing` • `text` • `banding` • `negative` • `data` (data only) • `column` (both data and header for the specified column) • `columnHeader` (column header only) • `groupHeader` • `groupFooter` • `groupTotal` All the above tags have a `style` attribute for specifying the CSS style string.	Nested tags: • `numeric`: specifying the numeric format, for example, `$#,###.00` • `date`: specifying the data format, for example, `yyyy.MM.dd` • `missing`: specifying the text to display if data is missing • `aggregation`: specifying the format for aggregation values (group totals) All the above tags have a `format` attribute for specifying the format string.

TextBlox lets you set the display text or wrap HTML code around the current text in the five areas in a rendered report: group headers, group footers, group totals, column headers, and data. It has five identical sub tags as StyleBlox: `columnHeader`, `data`, `groupHeader`, `groupFooter`, and `groupTotal`. See "Report Layout Formatting and Styling Summary Table" on page 41 for a summary table that compares the tags.

Note that FormatBlox, StyleBlox, and TextBlox are not data transformers, and therefore the order they are added in your JSP does not impact the data pipeline.

Once the data is transformed through sorting, filtering, calculating, hiding/removing members, or grouping, tags for these Blox are then processed to render the final report.

Processing Sequence for StyleBlox, FormatBlox, and TextBlox

After the data is transformed through the pipeline, DB2 Alphablox Relational Reporting engine renders the report in the following sequence:

1. The format mask gets applied to the value first (FormatBlox).
2. The formatted value is wrapped in text (TextBlox).
3. Then the cell is output with the style (StyleBlox).

For the following code:

```
<bloxreport:format>
   <bloxreport:numeric format = "$#,###.00" />
</bloxreport:format>

<bloxreport:text>
  <bloxreport:groupTotal
    text = "Total: <value/>" />
</bloxreport:text>

<bloxreport:style>
  <bloxreport:groupTotal
    style = "color: green" />
</bloxreport:style>
```

The resulting HTML code for a data cell may look as follows:

```
<td style="color: green">Total: $1,000.00</td>
```

The order these tags are added are not important since they will always be processed in the sequence described above. It is important to use the TextBlox only to set texts and leave styling to StyleBlox. Since you can add HTML code to text set through TextBlox, you should only add none-style related code since the styles are likely going to be overridden by StyleBlox or the style sheet.

For details on styling reports, see "Styling Data Displayed in Report" on page 47, and "Specifying and Styling Column Headers" on page 49. The details for StyleBlox usage and tag syntax are described in StyleBlox on page 128.

For details on data formatting, see "Formatting Data" on page 45. The details for FormatBlox usage and tag syntax are described in FormatBlox on page 108.

For details on setting texts, see "Specifying and Styling Column Headers" on page 49 and "Specifying and Styling Break Group Headers, Footers, and Totals" on page 59. The details for TextBlox usage and tag syntax are described in TextBlox on page 131.

StyleBlox VS. CSS Styles

A relational report is rendered using a set of styles defined in `report.css` in `<alphablox_dir>/repository/theme/`. The complete listing is provided in "Style Classes" on page 9. While StyleBlox lets you style the data and column headers based on data type or member, the CSS styles encompass the entire report, including the overall report, break group related areas, all the interactive menus, and the Report Style dialog box.

Overall Report

The outmost wrapping HTML tag for a relational report is a `DIV`. The `.report` class lets you define the entire report display attributes, such as the report background color, border, padding, and margin. The default style for `.report` has a white background with a solid, 1-pixel black border.

Break Groups

When a report is organized in groups by GroupBlox, group headers, footers, and totals become available. Each of these areas has a corresponding style class. Depending the levels of grouping, you have the following style classes:

- `.groupheader1`, `.groupheader2`, ..., `.groupheaderN`
- `.groupfooter1`, `.groupfooter2`, ..., `.groupfooterN`
- `.grouptotal1`, `.grouptotal2`, ..., `.grouptotalN`

For a visual representation of each of these areas, see the example in "Display Areas in a Rendered Report" on page 40. For detailed discussion of grouping and styling a report with break groups, see the chapter on Chapter 7, "Grouping Data," on page 57.

Interactive Context Menus

All the interactive context menus are rendered using style classes and can be customized to match your report's overall look and feel. Classes include `.menu`, `.choice`, `.choicehover`, `.separator`, and `.selected.`

Report Style Dialog Box

The Report Style dialog is rendered using a set of style classes and can be customized to match your application's overall look and feel. For details, see Chapter 9, "Styling the Report Editor User Interface," on page 77.

Columns and Data

There are classes for the actual data and column headers in a report. This is where the StyleBlox and the style classes overlap somewhat. When the two collide, StyleBlox win. The following are the classes for defining the styles for column headers, column headers when mouse over, column headers when they are selected, data, and banding.

- `.column:` for column headers
- `.columnhover:` for column headers when the mouse hovers over
- `.selected:` column headers when they are selected
- `.data:` for data rows
- `.banding:` for alternate data rows

Places where StyleBlox and the style classes overlap are the `.column`, `.data`, and `.banding` styles. The following table shows the overlaps and how they work differently:

Nested Tags in StyleBlox	CSS Style Classes
`column:` style applies to both the header and the data of the specified member.	[none]

Nested Tags in StyleBlox	CSS Style Classes
columnHeader: style applies to all column headers if a member is not specified. If a member is specified, the style is applied only to the header of that column.	.column: style applies to all column headers.
data: style applies to all data if a member is not specified. If a member is specified, the style is applied only to data for that column.	.data: style applies to all data
banding: style applies to alternate data rows	.banding: style applies to alternate data rows

Note: Note that if these styles are specified in both places, styles specified in StyleBlox will be applied.

Formatting Data

To specify the default data format by data type or by member, you can use FormatBlox. FormatBlox supports specification of display format for the following types of data:

- numeric
- date (and time)
- missing data
- aggregation (for aggregation data, also known as the group totals, which are available when a report is grouped and the values are always numeric).

You can specify the data format using Java format masks, as shown in the following example:

```
<bloxreport:format>
   <bloxreport:numeric format="####.00;(####.00)" />
   <bloxreport:numeric format="$#,###.00;$(#,###.00)"
    member="Sales"    />
   <bloxreport:aggregation format="$#,###;$(#,###)"
    member="Sales"    />
   <bloxreport:date format="yyyy.MM.dd G 'at' hh:mm:ss z" />
   <bloxreport:date format="EEE, MMM d, ''yy" member="Date" />
   <bloxreport:missing format="Units Value Missing"
    member="Units" />
   <bloxreport:missing format="Sales Value Missing"
    member="Sales" />
</bloxreport:format>
```

This code example sets:

- the default format for positive numeric data to "####.00"; the default format for negative numeric data to "(####.00)." For example, 1234.5 becomes 1234.50, and -1234.5 becomes (1234.50).
- the numeric data for member "Sales" to "$#,###.00;$(#,###.00)." For example, 1234.5 becomes $1,234.50, and -1234.5 becomes $(1,234.50).
- the group totals for "Sales" to "$#,###;$(#,###)." For example, 1234.5 becomes $1,235, and -1234.5 becomes $(1,235).
- the default format for dates to "yyyy.MM.dd G 'at' hh:mm:ss z". An example of this format is 2001.10.01 AD at 09:27:13 PDT.
- the date format for member "Date Member" to "EEE, MMM d, ''yy". An example of this format is Mon, October 1, '01.

- the text display when member "Units" contains missing data to "Units Value Missing"
- the text display when member "Sales" contains missing data to "Sales Value Missing"

For details on format masks, see
http://java.sun.com/j2se/1.4.2/docs/api/java/text/DecimalFormat.html.

You can add only one `<bloxreport:format>` tag in a `<bloxreport:report>` tag. However, within the `<bloxreport:format>` tag, you can have multiple `<bloxreport:numeric>`, `<bloxreport:date>`, and `<bloxreport:missing>` tags. This allows you to set the data format for different members.

For adding styles or wrapping HTML code around data values, see the next sections on ""Wrapping HTML Code Around Data Values"" and ""Styling Data Displayed in Report" on page 47."

Wrapping HTML Code Around Data Values

You may want to add HTML code around data values in a report to, for example, add additional text, images, styles, or links to other related information. TextBlox provides a way to set the text for five distinct areas in a report, one of which is the data area. TextBlox sends the entire text string to the browser untouched, except for two special substitution variables, `<member/>` and `<value/>`.

The displays areas in a rendered report is described in "Display Areas in a Rendered Report" on page 40. Using TextBlox, you can rename or wrap HTML code around five display areas: column headers, group headers, group footers, group totals, and data. The following example shows how the data values in the column "Location" are replaced with the string "Not to be disclosed" using the TextBlox's nested `data` tag:

```
<bloxreport:text>
  <bloxreport:data
    columnName = "Location"
    text = "Not to be disclosed"
  />
</bloxreport:text>
```

In most cases, however, what you want is to wrap HTML code around the values rather than replace the values. The following example shows how the data values in the column "Location" become hyper links that bring up more information on each location.

```
<bloxreport:report id="myReport">
  ...
  <bloxreport:text>
    <bloxreport:data
      columnName="Location"
      text="<a href=\"javascript:getURL('<value/>')\"><value/></a>" />
  </bloxreport:text>
  ...
</bloxreport:report>
```

`<value/>` is a substitution variable (not a JSP tag) that gets substituted into the actual data values when TextBlox sends the text string to the browser. Inside the TextBlox tag, you can have multiple `data` tags. For each column whose data values you want to modify, add a nested `data` tag.

The getURL() JavaScript function may look as follows:

```
function getURL(location) {
    shortName = location.substr(0,product.indexOf(" "));
    url="http://myserver/sites/" + shortName + ".html";
    open(url);
}
```

When users click on Beverly Hills, the JavaScript getURL("Beverly Hills")is called and url gets the value of http://myserver/sites/Beverly.html.

For more information on TextBlox, its nested tags, and the substitution variables, see TextBlox on page 131 and "Special Substitution Variables for Displaying Member Names and Values" on page 52.

Adding HTML code to Data Returned from a Query

Another way to add HTML code to data returned from a query can be done by instructing the FormatBlox not to encode the HTML using the FormatBlox's <bloxreport:html> nested tag:

```
<%
    // This example always starts with a new ReportBlox. So remove the
    // old one if it exists.
 BloxContext context = BloxContextFactory.getBloxContext(pageContext.getRequest(),
     pageContext.getResponse());
 Blox blox = context.getBlox("htmlLinkReport");
 if(blox != null) {
  context.deleteBlox(blox);
 }

    Query = "Select Week_Ending, Area, '<a href=info.jsp?location='
    + Loc + '>' + Loc + '</a>' as 'Location', Product" +
    "FROM qcc";
%>

<bloxreport:report id="htmlLinkReport" interactive="true">
    <bloxreport:sqlData
        query="<%= Query %>">
    <bloxreport:dataSourceConnection
    dataSourceName="myRDB" />
</bloxreport:sqlData>

<bloxreport:format>
    <bloxreport:html member="Location" />
</bloxreport:format>
```

The returned data for the Location column will preserve the HTML code. When users click on the Sonoma link, a request for page moreInfo.jsp?location=Sonoma is issued. Notice that the member is given an alias "Location" in the query. Since the DB2 Alphablox Relational Reporting engine is instructed to "leave HTML alone" and not to encode it, we need a display name for the member so we can reference it in the <bloxreport:html> tag.

Styling Data Displayed in Report

To specify the data display style such as font face, size, style, color, text alignment, or background color, use StyleBlox or cascading style sheets.

A relational report is displayed based on the styles you specify for each of the style classes used by Relational Reporting. The style classes related to data display are:

- .data
- .banding

The following table shows the use of these classes:

Example	Effect

`.data {  font-size: 11;`
`text-align: right;  padding-left:`
`20;  color: black;`
`background-color: white;`
`}`
	All data cells are displayed in the style specified.

`.banding {  background-color:` `#CCCCFF; }`	Alternative data rows are displayed in the style specified.

For a complete list of classes you can use to style a report, see "Style Classes" on page 9.

To set the display style for missing data, negative data values, or for a specified data type or data column, you can use StyleBlox. StyleBlox lets you specify the display style for the following data type:

Data Type	Nested Tags within `<bloxreport:style>`	Effect
date	`<bloxreport:date` `style="yourStyle"/>`	All columns containing date or timestamp are displayed in the style specified.
numeric	`<bloxreport:numeric` `style="yourStyle" />`	All columns containing numeric data are displayed in the style specified.
text	`<bloxreport:text` `style="yourStyle" />`	All columns containing text are displayed in the style specified.

The default text alignment by data type in the rendered report is as follows:

Data Type	Text Alignment
Date	left
Numeric	right
Text	left

You can overwrite the default using StyleBlox as shown in the following example:

```
<bloxreport:style>
   <bloxreport:text style="text-align: center" />
</bloxreport:style>
```

In addition, StyleBlox lets you set the display style for the following report data and element:
- cell banding
- missing data
- negative data values
- a specified column

The corresponding tags and their effects are described in the following table:

Report Data or Element	Nested Tags within `<bloxreport:style>`	Effect
cell banding	`<bloxreport:banding style="yourStyle"/>`	Alternate data rows are displayed in the specified style. **Note:** The style for alternate data rows can also be specified using the `.banding` style class. However, styles set through StyleBlox win over the styles set in style classes.
missing data	`<bloxreport:missing style="yourStyle"/>`	All missing data (or null data) are displayed in the style specified.
negative data values	`<bloxreport:negative style="yourStyle"/>`	All negative data values in the report are displayed in the style specified.
a specified column	`<bloxreport:column style="yourStyle" member="memberName"/>`	Both data and column headers in the named column is displayed in the style specified.

In the following example:

```
<bloxreport:style>
   <bloxreport:banding style="background-color: #FFCCFF" />
   <bloxreport:missing style="background-color: aqua" />
   <bloxreport:negative style="background-color: red" />
   <bloxreport:column style="color: yellow" member="Country"/>
   <bloxreport:column style="color: blue" member="State"/>
</bloxreport:style>
```

- the background color for alternate data rows are #FFCCFF
- the background color for the cell containing missing data is set to aqua
- the background color for the cell containing negative data is set to red
- the text color for the Country member is set to yellow
- the text color for the State member is set to blue

Note: You can have only one StyleBlox in a report. If you have multiple StyleBlox, only the last StyleBlox has any effect on the report. However, for its nested tags, you can have multiple `column`, `missing`, `negative` or `banding` tags, and styles set through these nested tags will cascade.

Specifying and Styling Column Headers

To specify column headers that are different from the field names in the database, you can do so in your SQL statement that extracts the data, such as using a statement like ″`SELECT FROM myTable column1 AS newColumnName1, column2 AS newColumnName2....`″ In this case, the members are known to ReportBlox as newColumnName1 and newColumnName2. Note that member names are case-sensitive, so in your later references to these members, their cases should be respect.

Note: Oracle returns the new names in all uppercases. To preserve the cases, quote the member names:

```
"SELECT FROM myTable total_sq_ft AS \"Sq_Ft\", sq_ft_pct AS \"Pct\""
```

Another approach is to use the TextBlox. TextBlox lets you specify the display texts for the five areas in a rendered report: column headers, data, and, if the report is

grouped, group headers, group footers, and group totals (see "Display Areas in a Rendered Report" on page 40 for these areas on a report). The `<bloxreport:text>` tag has nested tags for you to specify the text or add HTML code around the value/member for each of the five areas.

To specify a column header, use the TextBlox's nested `columnHeader` tag. This tag has two tag attributes:

- `columnName`— required; the name of the member on this column
- `text`— required; the text to display for the column header. Everything you specify here will be sent to the browser for processing, including any HTML code added.

The following example sets the column header for member "Sales" to "Product Sales" and the column header for member "Units" to "Units Sold:"

```
<%@ taglib uri="bloxreporttld" prefix="bloxreport" %>
<bloxreport:report id = "MyReport">
...
   <bloxreport:text>
     <bloxreport:columnHeader
        columnName="Sales">
        text="Product Sales" />
     <bloxreport:columnHeader
        columnName="Units"
        text="Units Sold" />
   </bloxreport:text>
...
</bloxreport:report>
```

You can wrap HTML code around column headers to add formatting, links, or images. The following example:

- Renames the column header for member Cost to Unit Cost.
- Adds a link from the header to another URL in the `myApp` application for additional information on the cost.
- Adds an image `info.gif` that resides in the same directory as this report JSP.

```
<bloxreport:text>
  <bloxreport:columnHeader
    columnName="Cost"
    text="<a href=\"/myApp/products/CostList.html\">
        <img src="info.gif">Unit Cost</a>" />
</bloxreport:text>
```

Note: The text string for the `text` attribute should be on one line with no line breaks. It is broken into multiple lines due to the limited width of the printed page. When copying and pasting the code from the online version, you should remove the line breaks.

See the Saving and Exporting Data with Dynamic Query example in Blox Sampler— Relational Reporting for an live example.

TextBlox preserves your HTML code for the column header so you can use the standard HTML to format the headers. The URL and image path can be relative or absolute:

- For absolute URLs, the string should begin with "`http://`".
- For relative URLs:
  - Starting the string with a slash (/) indicates that the URL is relative to the server root. The application context needs to be included in the URL.

– Starting the string without a slash indicates that the URL is relative to the current document.

To add HTML code around the member name without renaming it, use the <member/> substitution variable, as shown in the following example:

```
<bloxreport:text>
   <bloxreport:columnHeader
      columnName="Cost"
      text="<a href=\"/myApp/products/CostList.html\">
         <img src="info.gif"><member/></a>"/>
</bloxreport:text>
```

is a substitution variable that gets substituted into the member name when the relational report HTML is sent to the browser. It is not a JSP tag.

Important: If the report is rendered in interactive mode, when users choose to rename the column header using the Column Header Context Menu, all the HTML code will appear. This could be undesirable. In addition, your users can easily overwrite your HTML code and formatting. It is recommended that you set the `interactive` attribute of your ReportBlox to `false` if you have HTML code wrapped around your column headers or footers.

Important: The text string for the `text` attribute should be on one line with no line breaks. It is broken into multiple lines due to the limited width of the printed page. When copying and pasting the code from the online version, you should remove the line breaks.

Tip: Avoid adding styling strings using TextBlox. Styles are output last by the DB2 Alphablox Relational Reporting engine after the data has been formatted and wrapped in text. Setting styles through the `text` attribute in TextBlox's nested tags is not as efficient and can cause you confusion as the styles are overridden by styles set in style sheets and StyleBlox.

Tip: In an interactive report, if you wrap an anchor tag outside the column header text (or group headers, footers, and totals), when you mouse over the column header, the context menu will not pop up. You will need to hover somewhere else in the cell (but not the link) in order for the menu to pop up. This is the browser's behavior. Keep this in mind as you design your report.

Styling Column Headers

To style a column header, use the StyleBlox's nested `columnHeader` tag. This tag has two tag attributes:

- `columnName`— optional; the name of the member on this column
- `style`— required; the style to apply to the column header.

The following example sets the column header style for "Sales" to center-aligned:

```
<%@ taglib uri="bloxreporttld" prefix="bloxreport" %>
<bloxreport:report id = "MyReport">
...
   <bloxreport:style>
      <bloxreport:columnHeader
         columnName="Sales">
         style="text-align: center;" />
   </bloxreport:style>
...
</bloxreport:report>
```

Recall that in TextBlox's nested `columnHeader` tag, you can wrap HTML code around the column header text. Avoid adding styling strings using TextBlox since styles are output last by the DB2 Alphablox Relational Reporting engine after the data has been formatted and wrapped in text. Setting styles through the `text` attribute in TextBlox's nested tags is not as efficient and can cause you confusion since the styles are overridden by styles set in style sheets and StyleBlox.

Specifying Column Width, Color and Style

You can use StyleBlox's `<bloxreport:column>` tag to specify the width, background color and font style for both data and the column headers in the named column. Specify the style as you normally would to an HTML table cell, with each style attribute and value pair separated with a semicolon (;).

```
<bloxreport:style>
   <bloxreport:column style="font-weight: bold; color: white;
      background-color: gray" member="Type" />
   <bloxreport:column style="width:20px; background-color:
      #99ffcc; font-weight: bold" member="Product" />
</bloxreport:style>
```

Typical attributes you can specify for a table cell include:

- background color
- font face, size, style, weight, and color
- text alignment: left, right, center
- border color

Use an a style sheet editor or an HTML editor with a graphical user interface to help you achieve your desired style.

To specify the style for column headers, you can:

- Specify the style for the `.column` class, which is applied to all column headers.
- Specify your own style class and apply that to specified column headers using the nested `<bloxreport:columnHeader>` tag inside TextBlox.

See "Specifying and Styling Column Headers" on page 49 for more details and code examples.

Special Substitution Variables for Displaying Member Names and Values

Two special substitution variables are available for you to get the name of a member on a column, the break group member names, or the value of a member. These are useful when you need to add HTML code to around column headers, dynamically display break group member names, get the value of a specific member, or display data from another column. The following sections describe the two variables.

The <member/> Substitution Variable

The `<member/>` variable is used to substitute the name of the current member into the current location. It is valid inside all of TextBlox's nested tags. It has one optional `level` attribute for referencing current or higher level break group members (*level* <= *currentLevel*).

Examples:

- Used inside TextBlox's nested `groupHeader` tag to extract the break group member name of the current or higher level:

```
<bloxreport:report ...>
  <bloxreport:text>
    <bloxreport:groupHeader level="1"
      text="Sales Report by <member/>" />
    <bloxreport:groupHeader level="2"
      text="<i><member/></i>" />
  </bloxreport:text>
</bloxreport:report>
```

- Used inside TextBlox's nested `columnHeader` tag to extract the name of the column header:

```
<bloxreport:report ...>
  <bloxreport:text>
    <bloxreport:columnHeader columnName="Product"
      text="<span class="myStyleclass"><member/></span> />
  </bloxreport:text>
</bloxreport:report>
```

- Used inside TextBlox's nested `groupFooter` tag to extract the name of the current or higher level break group member:

```
<bloxreport:report ...>
  <bloxreport:text>
    <bloxreport:groupFooter level="2"
      text="Group footer for Level 2: by <member level="\1\"/>,
<member/>" />
  </bloxreport:text>
</bloxreport:report>
```

You can only reference a break group member at the current or a higher level. If a lower level is specified, the entire tag string will be treated as texts and no variable substitution will occur. If no level is specified, the current level is implied. For example, a level 2 break group footer can only reference level 2 and level 1 break group members, not level 3.

Note: Double quotes in the text string should be escaped (″\″).

The <value/> Substitution Variable

The `<value/>` variable is used to substitute the value of current member or a specified member. It is valid inside the `data`, `groupTotal`, `groupFooter`, and `groupHeader` tags. It can be used to extract either values of a different column into data in the current column, or the group aggregation data (group totals) of a different member into the group aggregation data in the current column.

Examples:

- Used inside TextBlox's nested `groupTotal` tag to extract the aggregation value for the break group:

```
<bloxreport:report ...>
  <bloxreport:text>
    <bloxreport:groupTotal level="1"
      columnName="products"
      text="<i>Total:</i> <value />" />
  </bloxreport:text>
</bloxreport:report>
```

- Inside the TextBlox's nested `groupHeader` tag for extracting the group total of a named member:

```
<bloxreport:report ...>
  <bloxreport:text>
    <bloxreport:groupHeader level="1"
```

```
      columnName="products"
      text="<div align=\"left\"><member/></div>
      <div align=\"right\">Rank:<value member=\"RankbyProduct\" /
></div>" />
   </bloxreport:text>
</bloxreport:report>
```

- Used inside TextBlox's nested `data` tag to extract the data value for the column:

```
<bloxreport:report ...>
  <bloxreport:text>
    <bloxreport:data
      columnName ="products"
      text="<a href=\"javascript:getURL();\"><value/></a>" />
  </bloxreport:text>
</bloxreport:report>
```

To get the value of another member, specify the member name. For example:

```
<value member="Sales" />
```

If the member name is not recognized or does not exist, the entire tag string will be treated as texts and no variable substitution will occur.

Note: Double quotes in the text string should be escaped (″\″).

Using the <member/> and <value/> Variables

The following is an example of a level 2 group header referencing a level 1 break group member:
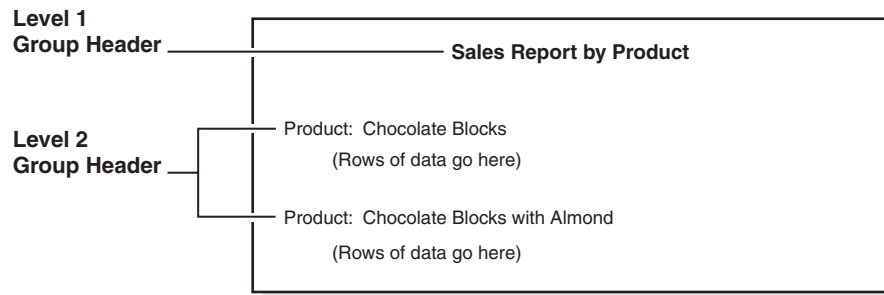
This is done with the following code:

```
<bloxreport:report ...>
  <bloxreport:text>
    <bloxreport:groupHeader level="1"
      text="Sales Report by <member/>" />
    <bloxreport:groupHeader level="2"
      text="<member level="1"/>: <member/>" />
  </bloxreport:text>
</bloxreport:report>
```

The <member/> variable has a `level` attribute. If `level` is not specified, the current grouping level as specified in the `groupHeader`, `groupFooter` or `groupTotal` tag is implied. You can only reference the member of the current grouping level or at a higher level. For example, a level 2 group header can only reference level 2 and level 1 group members, not level 3. If the level is set to 3, the entire tag string will be treated as texts and no variable substitution will occur.

The <value/> variable has a `member` attribute. If `member` is not specified, the current member in the column is implied. For the `groupHeader` tag, you should specify the `member` whose aggregation value for the grouping level you want to extract.

Without specification of a member, the `<value/>` substitution variable is ignored since the DB2 Alphablox Relational Reporting engine cannot determine which aggregation value should be used.

Since group headers, footers, and totals are only available when a report is grouped, more details are discussed in the Chapter 7, "Grouping Data," on page 57 chapter and the section on "Specifying and Styling Break Group Headers, Footers, and Totals" on page 59.

For live examples that demonstrate how these variables can be useful, see the examples in "Formatting the Report and Data" and the "Grouping and Adding Group-based Summary Columns" sections in Relational Reporting Blox Sampler.

Setting or Turning Off Cell Banding

The style classes related to data display are `.data` and `.banding.` The default style sheet has the two classes set to different background colors so data rows appear in alternate colors. To set the banding colors, either specify the style for the `.banding` style class or use the StyleBlox's nested banding tag:

```
<style>
    .banding { background-color: #ffffcc; }
</style>
```

or

```
<bloxreport:style>
    <bloxreport:banding style="background-color: #ffffcc;" />
</bloxreport:style>
```

To turn off cell banding, set the same style for both classes. For example:

```
.data, .banding {
    background-color: white;
}
```

Note that:

- Styles set through StyleBlox win over the styles set in style classes.
- In an interactive report, users can use the Report Style dialog (via the context menus' Style... option) to set styles for data cells in a column or all data cells in the report. This will override the cell banding setting specified in your JSP. Since the Report Style dialog is not aware of existing styles in the rendered report, existing cell banding will be replaced by the background color specified in the dialog for data in the column (or all columns if styles are set to apply to all data cells).

Setting the Report Display Area

As relational reports are displayed using Cascading Style Sheet, you can specify the display area on the page using the `.report` class. Sometimes you may not want the report to take up the entire browser window and only want the report to be displayed inside an area on a specified position in a page. You can do so to the style sheet your JSP page referenced, or you can specify that in-line right within your JSP page. For example:

```
<head>
...
   <style>
      .report { height: 400; width: 500 }
   </style>
</head>
```

This will cause the report to display as a scrollable 400 X 500 box inside the browser window. This gives you greater control of the page layout. For example, you may have a ReportBlox followed by a GridBlox and some text on the same page.

Important: Not all browsers provide full support for Cascading Style Sheet. Some DHTML related techniques may not work in Netscape browsers. Many resources are available on the Web that provide tips for using CSS across multiple browsers.

Adding Background Images

To add a background image, add the image to the .report class in the style sheet:

```
.report { height: 300; width: 450;
 background: white url(background.gif) no-repeat
 fixed center
}
```

This will add an image at the center of the report, relative to the display area.

Important: Not all browsers provide full support for Cascading Style Sheet. Some DHTML related techniques may not work in Netscape browsers. Many resources are available on the Web that provide tips for using CSS across multiple browsers.

Chapter 7. Grouping Data

This section discusses tasks that are closely related to grouping of data. These include creating break groups, adding aggregation data per break group, adding calculated columns that provide group-based summary data, and formatting the report for break group headers, footers, and totals for each break group level. These involve primarily the use of GroupBlox for adding break groups, TextBlox for specifying text for various display areas in the report, and CalculateBlox for adding group-based summary data columns.

The "Grouping and Adding Group-based Summary Columns" section in the Blox Sampler for Relational Reporting has examples demonstrating most of the tasks discussed in this chapter.

Overview of Break Groups and Break Group Levels

Often times a report is grouped for ease of reading, data interpretation and data comparison. Once a report is grouped:

- Group headers, footers, and totals/aggregations areas become available. You can specify the text and display style for each of these areas
- You can add group-based calculations (ranking, percent of total, running total, and running count) based on specified break group level.

GroupBlox lets you group a report by specifying the members whose values you want to use as break groups. For example, the following code lets you group the data by product.

```
<bloxreport:group members="Product" />
```

When a break group is added, a level is automatically added, with level 1 being the overall level that includes all data, and level 2 being the level for the individual break groups. In the above example, the title of the report becomes level 1 group heading, and each Product group becomes level 2 group headers. See "Display Areas in a Rendered Report" on page 40 for these areas on a report.

A report without any grouping has only the column headers and data areas. Once a report is grouped in some way, group headers and group footers become available. In addition, an aggregation value, or group total, is available for each numeric column per break group. The default aggregation type is sum. You can specify the aggregation type using the nested `<bloxreport:aggregation>` tag. The following example shows a report grouped by "Product," and for each product, we want a count of the locations that sell that product, and an average of units sold.

```
<bloxreport:group members="Product">
   <bloxreport:aggregation member="Location" type="count" />
   <bloxreport:aggregation member="Units" type="average" />
</bloxreport:group>
```

Tip: To create a report with aggregation values for each column, or to create a report "title" without any grouping, add a GroupBlox with the group members set to an empty string:

```
<bloxreport:group members="" />
```

It is important to note that:

* The order the grouping members are specified is important. With each member specified in the members attribute, another level of grouping is added.

* *GroupBlox handles only grouping and does not imply sorting.* In the above example, the product break groups do not appear in alphabetical order unless you sort the data on product first. See "Sorting Data" on page 31 for information on sorting.

Break Group Aggregations

Aggregation values for numeric data are automatically available in a grouped reported. They appear in a report's group totals area. Supported aggregation types are:

* average
* count
* max
* min
* none
* sum

By default, the aggregation type for numeric data is sum unless specified otherwise. If you do not want any aggregations, set the type to none. For none-numeric columns, the default aggregation value is treated as missing data. The only valid aggregation type for none-numeric data is count.

When aggregations are calculated, missing values are ignored. The following example shows the aggregation values for the give data:

A column with the following data	The aggregation values will be:
1	• sum: 15
2	• average: 3
3	• count: 5
4	• max: 5
5	• min: 1
Missing	

The following table shows the Blox and tags to use for various tasks related to group totals:

Task	Solution
Setting style	CSS style class: grouptotal1, .grouptotal2, ..., .grouptotal*N*.

	StyleBlox's groupTotal sub tag:
	
	 `<bloxreport:style>`
	`  <bloxreport:groupTotal level="N"`
	`    style="color: red" />`
	`</bloxreport:style>`

Task	Solution
Specifying data format	FormatBlox's aggregation sub tag: ``` <bloxreport:format> <bloxreport:aggregation format="$#,###" /> </bloxreport:format> ```
Modifying displayed group totals	TextBlox's nested `groupTotal` tag: ``` <bloxreport:text> <bloxreport:groupTotal columnName="Cost" text="Total: <value/>" /> </bloxreport:text> ```

See "Specifying and Styling Break Group Headers, Footers, and Totals" on page 59 and "Formatting Data" on page 45 for more information.

Specifying and Styling Break Group Headers, Footers, and Totals

When you group a report using GroupBlox, three report display areas become available: group headers, group footers, and group totals. With each level of grouping added, another level of group header, footer, and total areas become available. For a diagram that shows the different areas, see "Overview of Break Groups and Break Group Levels" on page 57.

In a report grouped by Product, for example, you may want the report title to be "Profitability by Product." For each product group, you get a group header, a group total for each numeric data column, and a group footer. The group header and footer are specified using the TextBlox's nested `groupHeader` and `groupFooter` tags:

```
<bloxreport:report id="anotherSampleReport" ...>
  ...
  <bloxreport:text>
    <bloxreport:groupHeader level="1"
      text="Profitability by <member/>" />

    <bloxreport:groupFooter level="1"
      text="--End of Report" />

    <bloxreport:groupTotal columnName="Cost"
      text="Grand Total Cost: <value/>" />

  </bloxreport:text>
</bloxreport:report>
```

Level 1 is the outmost level. With each level of grouping added, a level is added. If a report is grouped by Product and then by Country, for example, level 1 represents the overall report; level 2, each Product break group; and level 3, each Country break group. This same rule applies to the `groupHeader`, `groupFooter`, and `groupTotal` tags.

TextBlox sends the entire specified text string to the browser untouched, except that it looks for two special substitution variables, `<member/>` and `<value/>`, replacing the variables with the actual member names or data values.

Note the following:
- If the text string includes any double quotes, they should be escaped ("\").

- You should avoid adding styling strings using TextBlox. Setting styles through the `text` attribute in TextBlox's nested tags is not as efficient and can cause you confusion since the styles are overridden by styles set in style sheets and StyleBlox.
- Group footers for levels 1 through 3 are disabled by default in the supplied style sheet. This means:
  - If you want the group footers for levels 1 to 3 to display, you need to overwrite the style class in your page or style sheet: `.groupfooterN { display: inline; }`
  - If you have 4 or more levels of break groups, group footers for level 4 and up will be displayed while the higher level footers don't. To hide the footers for level 4 and up, set the display to none: `.groupfooterN { display: none; }`

It is important to know that adding groupings, specifying group headers/footers/totals, and setting the styles used to display them involves different Blox, multiple tags, and several style sheet classes, as described in the following table:

Add Grouping	

Use GroupBlox:
 	
`<bloxreport:group`
`  members="Product,Week_Ending" />`
	Product is the first level grouping; Week_Ending, the second level.

Specify text for group headers, footers, and totals	

Use TextBlox:
 	
`<bloxreport:text>`
`  <bloxreport:groupHeader`
`    level="1"`
`    text="Profitability by <member/>" />`
`  <bloxreport:groupHeader`
`    level="2"`
`    text="<member level="1" />: <member/>" />`
`  <bloxreport:groupHeader`
`    level="2"`
`    text="<member/> (Ranking:<value member=\"SalesRank\"/>"`
`  />`
`  <bloxreport:groupFooter`
`    level="2"`
`    text="--End of <member/>" />`
`  <bloxreport:groupFooter`
`    level="1"`
`    text="--End of Report" />`
`  <bloxreport:groupTotal`
`    level="1"`
`    columnName="Cost">`
`    text="Grand Total: <value/>" />`
`  <bloxreport:groupTotal`
`    level="2"`
`    columnName="Cost">`
`    text="subtotal: <value/>" />`
`</bloxreport:text>`
	`groupHeader`, `groupFooter`, and `groupTotal` are nested tags within the `<bloxreport:text>` tag. However, they are only in effect when the report is grouped. If `level` is not specified, the text will be applied to group headers, footers, and totals of all levels.
	
	The `<member/>` variable will be substituted by the member name of the specified grouping level. If `level` is not specified, the current grouping level is implied. You can only reference the current or higher levels of break group members.
	
	The `<value/>` variable will be substituted by the aggregation value for the specified member for the specified level of grouping. If `level` is not specified, the current grouping level is implied. You can only reference the current or higher levels of break group members.
	
	See "Special Substitution Variables for Displaying Member Names and Values" on page 52 for details on the `<member/>` and `<value/>` variables.

Specify display style for group headers, footers, and totals	

Use CSS style sheet:
 	
``css
.groupheader1 {
   font-size: 110%;
   background-color: #9999FF;
   font-weight: bold; }
 	
.groupfooter1 {
   font-size: 80%;
   background-color: #9999FF; }
.grouptotal1 {
   font-size: 100%;
   text-align: right; }
``
	Use the style classes groupheader*N*, groupfooter*N*, and grouptotal*N* to specify the display styles.

Use StyleBlox:
 	
``xml
<bloxreport:style>
  <bloxreport:groupHeader
    level="1"
    style="font-size: 110%;" />
  <bloxreport:groupHeader
    level="2"
    style="font-size: 90%;" />
  <bloxreport:groupFooter
    level="1"
    style="font-weight: bold;" />
  <bloxreport:groupTotals
    style="font-size:90%;font-style:italic;" />
</bloxreport:style>
``
	`groupHeader`, `groupFooter`, and `groupTotal` are nested tags within the `<bloxreport:style>` tag. These tags have a `style` attribute for specifying CSS style strings. They are only in effect when the report is grouped. If `level` is not specified, the style will be applied to group headers, footers, or totals of all levels.

Specify aggregation type for group totals	

Use GroupBlox:
 	
``xml
<bloxreport:group
  members = "Product, Week_Ending">
  <bloxreport:aggregation
    member="Units"
    type="sum"/>
</bloxreport:group>
``
	Use the `<bloxreport:aggregation>` tag nested within the `<bloxreport:group>` tag to specify the aggregation type for each numeric data column. The default aggregation type for numeric data is sum. Valid values are `sum`, `average`, `count`, `min`, `max`, and `none`.

Again, group footers for levels 1 through 3 are disabled in the supplied style sheet:

```css
.groupfooter1, .groupfooter2, .groupfooter3 { display: none; }
```

To enable group footers for a particular level, add the following CSS to your style sheet:

```css
.groupfooterN { display: block; }
```

or

```css
.groupfooterN { display: inline; }
```

Note: Keep in mind that when the report is rendered in interactive mode, users can change the column name, group totals text, and display styles for each element in the report via the interactive context menus. Therefore, you should use the `<member/>` and `<value/>` substitution variables to dynamically set your header and footer texts.

Note: In an interactive report, if you wrap an anchor tag around a column header, group header, group footer, or group total to make it a link, when you mouse over them, the context menu will not pop up. You will need to hover somewhere else in the cell (but not the link) in order for the menu to pop up. This is the browser's behavior. Keep this in mind as you design your report.

Calculating Group-based Summary Columns

Using CalculateBlox, you can add a calculated column based on a calculation expression. CalculateBlox also has four calculation functions for adding group-based summary columns that rank the data, calculate the running totals, percent of totals, and running counts in each break group. Since these are group-based calculations, their working relies on an already grouped report. That is, before the CalculateBlox is added to the pipeline, the report should have been grouped using a GroupBlox.

For a live example, see the Saving and Exporting Data with Dynamic Queries example in Blox Sampler — Relational Reporting. This example shows a report grouped by Area, Location, and then Week_Ending, with added columns based on the values in Units.

For each calculation, you can specify a grouping level. By default, it is set to the lowest grouping level. That is, if a report is grouped by:

- Area
  - Location
    - Week_Ending

then by default the calculation is based on level 4 (for each week) grouping, the lowest level available. You can optionally specify to have the summary columns calculated based on level 3 (for each location), 2 (for each area), or 1 (for all areas in the report with no grouping). You can get to the aggregation values for each grouping level using the `<value/>` substitution variable. Set the grouping level in the example and experiment with how the summary data is calculated.

The following table shows the syntax for each of the functions. All function names are case-sensitive.

Function	Syntax

rank()	rank(memberName [, ASC|DESC] [, *level*])
	
	Defaults to the lowest level of grouping if `level` is not specified or the specified level does not exist. The default sort direction is `DESC`. The directions are case-sensitive. Examples:
	
	`<bloxreport:calculate expression = "Rank = rank(Units)" />`
	`<bloxreport:calculate expression = "[Rank of Sales] = rank(Sales, 2)" />`
	`<bloxreport:calculate expression="Rank = rank(Cost), ASC" />`
	`<bloxreport:calculate expression="Rank = rank(Cost, ASC, 1)" />`

percentOfTotal()	percentOfTotal(memberName [, *level*])
	
	Defaults to the lowest level of grouping if `level` is not specified or the specified level does not exist.
	
	`<bloxreport:calculate expression = "[% Total] = percentOfTotal(Units)" />`
	`<bloxreport:calculate expression = "[% Total] = percentOfTotal(Units, 2)" />`
	
	The calculated values will be decimals (for example, 0.245). To format the data as 24.5%, use FormatBlox on page 108.

runningCount()	runningCount(memberName [, *level*])
	
	Defaults to the lowest level of grouping if `level` is not specified or the specified level does not exist. Examples:
	
	`<bloxreport:calculate expression = "RunningCount = runningCount(Units)" />`
	`<bloxreport:calculate expression = "RunningCount = runningCount(Units, 3)" />`

Function	Syntax
runningTotal()	runningTotal(memberName [, *level*])
	Defaults to the lowest level of grouping if level is not specified or the specified level does not exist. Examples:
	<bloxreport:calculate expression = "RunningTotal = runningTotal(Units)" />

Keep in mind that *these summary columns are calculated based on groups. You should have a GroupBlox before the CalculateBlox in order for the summary data to be meaningful and accurate.* If the GroupBlox is added after the CalculateBlox (either in the JSP or as users change the break groups using the interactive user interface), the calculated columns are treated as regular numeric data columns and the group aggregation values will be calculated as such. As a result, you will get a report that contains meaningless data.

Adding Report Title and Column Summary (Aggregations)

To add a report title, you can do that in HTML outside of the ReportBlox. Or, if the report is grouped, level 1 group header will appear as the title of the report. See "Specifying and Styling Break Group Headers, Footers, and Totals" on page 59.

If you want to provide summary data at the end of your report or add a title without adding any break group, you can use the <bloxreport:group> tag and set the value of the members attribute to an empty string (members = ""). You cannot provide summary data (group totals) without adding a <bloxreport:group> tag.

The following example shows how to add aggregated data without break groups:

```
<bloxreport:group members = "" >
   <bloxreport:aggregation member = "Sales" type = "sum" />
   <bloxreport:aggregation member = "Units" type = "average" />
</bloxreport:group>
```

Using MembersBlox in Conjunction with GroupBlox

If you are using MembersBlox in conjunction with GroupBlox, you should use MembersBlox before GroupBlox. This is because in interactive reports, users may run into problems when they choose to **Clear Group**. For example, if you have members A, B, C, D, and E in the result set, and you have a GroupBlox that makes member A the break group member:

```
<bloxreport:group
   members="A" />
```

Member A is now moved to a different "dimension," similar to the page dimension in a PresentBlox when you work with multidimensional data sources.

After the GroupBlox, in you choose to include only members B and C on the Column dimension with MembersBlox:

```
<bloxreport:members
   included="B, C" />
```

This keeps only members B and C on the Column dimension in the result set. If your report is rendered in interactive mode, and your users choose to **Clear Group** from the Column Header Context Menu, an error will occur since member A no longer exists in the Column dimension as specified by the MembersBlox. To avoid this potential problem, use MembersBlox before GroupBlox.

Chapter 8. Saving and Exporting Data

This section discusses different ways your relational report can be saved or exported for the purpose of sharing, printing, or offline reviewing. Options include saving it as a static HTML or PDF file, sending it to Excel or other applications, or bookmarking the reports.

The Saving and Exporting Data with Dynamic Queries example in Blox Sampler - Relational Reporting demonstrates the different options discussed in this section.

Issues with Saving Interactive Reports Directly from Browser

Since a relational report is rendered as HTML tables, your users may want to use the browser's File -> Save As... option to save a copy on their local system. The result, however, may not be desirable. If you save a report using a Netscape browser, only the HTML is saved. Images, style sheets, and other external resources referenced are not downloaded. As a result, the saved report will look different from the original format and layout. In Internet Explorer, one can choose to save a page as a complete Web page, as a Web archive file, or as HTML only. Each option will produce different results. The following table summarizes the results of trying to save a relational report through the browser:

Browser	File Saving Option	Result of File --> Save As...
Non-interactive Reports		
Internet Explorer	Save as complete Web page	An error message saying the Web page cannot be saved.
	Save as HTML only	The report is saved without the style sheets and therefore may look different.
	Save as Web Archive (.mht)	The report is saved correctly with all images and styles stored in one .mht file. Can be viewed in Internet Explorer.
Mozilla, Firefox	Save as complete Web page	The report is saved with the style sheets.
Netscape		The report is saved without the style sheets and therefore may look different.
Interactive Reports		

Browser	File Saving Option	Result of File --> Save As...
Internet Explorer	Save as complete Web page	Either an error message saying the Web page cannot be saved, or the page is saved, but when loaded, the page only displays "Report loading..." message.
	Save as HTML only	The report is saved but it contains no data.
	Save as Web Archive (.mht)	The report is saved correctly with all images and styles stored in one .mht file. The DHTML associated with the Report Editor interface is also saved; however, it does not function when the page is loaded.
Mozilla, Firefox	Save as complete Web page	The report is saved with the style sheets. The DHTML associated with the Report Editor interface is also saved; however, it does not function when the page is loaded.
Netscape		Not supported. Report is rendered as a static HTML table and the layout and format may be incorrect.

The "Web Archive, single file" file saving feature in Internet Explorer stores a Web page as a single document (with an .mht file extension), embedding the graphics and styles rather than storing them in a separate folder. An MHT file can be viewed directly in Internet Explorer. No links or references to external resources are needed. When a relational report is saved as an MHT file, the report format and layout are correctly preserved. However, for an interactive report, the context menus in DHTML are saved as well. When a user views the saved MHT file on his local machine, the Report Editor user interface will appear to be available while in reality it does not function at all.

To provide a reliable way for your users to save a report with the correct layout, you can provide a "bookmarking" feature that stores the state of the report in the DB2 Alphablox repository. You can also render the report in PDF for your users to print or save it onto their local system. Or you can offer to save an interactive report in static HTML without the Report Editor interface to avoid confusion caused by the non-functioning Report Editor user interface. All these options are described next.

Saving as Static HTML to File System

To offer your users the option to save an interactive report as a static HTML page onto their local system for later reviewing, the key is to set the interactive property of the ReportBlox to false. This ensures the saved report will not include the non-functioning context menus. Otherwise, the DHTML associated with the context menus are saved with the report, resulting in potential confusion as these context menus do not function offline.

Below is a sample report saver JSP file that demonstrates how a report is saved onto a specified location, with all the users' modifications to the report through the Report Editor interface preserved.

In your JSP file containing ReportBlox, assuming you have code like the following to call the report saver JSP page and pass in the report's bloxName:

```
<head>
<%
  String reportName = "myReport";
%>
<script>
  var REPORT_NAME = "<%= reportName %>";
  function toFile() {
    window.open( "file.jsp?reportname=" + REPORT_NAME, REPORT_NAME +
      "_file", "height=400, width=600, scrollbars=yes, resizable=yes");
  }
</script>
</head>

<body>
...
<a href="javascript:toFile()" >Save File</a>

<bloxreport:report id="report" bloxName="<%=reportName%>"
  interactive="true" errors="true" >
  ...
</bloxreport:report>
```

The report saver JSP page `file.jsp` looks like the following:

```
<!--Importing the associated java classes in order to use
the APIs -->
<%@ page import="com.alphablox.blox.*" %>
<%@ page errorPage="error.jsp" %>
<%@ page import="java.io.*" %>

<!---Getting the reportName via the URL-->
<% String reportName = (String)request.getParameter( "reportname" );%>

<html>
<head>
  <title><%= reportName %> Saving Report As File</title>
</head>
<body>
<%
  // set the report to non-interactive mode so the menu items do not get
  // rendered
  if( reportName != null ){
    BloxContext context = BloxContextFactory.getBloxContext(request, response);
    ReportBlox reportBlox = (ReportBlox)context.getBlox( reportName );
    boolean isInteractive  = reportBlox.isInteractive();
    if( reportBlox != null ){
      reportBlox.setInteractive(false);
      try {
        /* Specify a location to save the file. Modify this
        for your application. */
        String filePath =  "C:\\temp\\" + reportName + ".html";
        FileWriter file = new FileWriter( filePath );
        file.write("<html>");
        file.write("<head>");
        file.write("<title>" + reportName + " Report</title>" );
        file.write( "<style>" );

        // Write the contents of the style sheet into the page
        String appPath = application.getRealPath("/");
        String styleSheetPath = appPath + "\\style\\reportstyles.css";
        // Create input stream object.
        FileInputStream fis = new FileInputStream( styleSheetPath );
        // Set variable for looping through bytes.
        int c;
        while( (c = fis.read() ) != -1) {
          file.write(c); // Loop to read and write bytes.
        }
        fis.close(); // Close output and input resources.
```

```
                    // Done writing out style sheet

                    file.write( "</style>" );
                    file.write( "</head>" );
                    file.write( "<body>" );
                    reportBlox.writeUpdate( file ); // Generate the report table.
                    file.write( "</body>" );
                    file.write( "</html>" );
                    file.close( );

                    out.println( "Your file was successfully saved at " + filePath +"." );
                }
                catch( Exception e ) {
                    out.println( "The page not saved :\n" + e.getMessage() );
                }
                finally {
                    // Set back to initial state
                    reportBlox.setInteractive( isInteractive );       }
                } else {
                    out.println( "The report " + reportName + " does not exist." );
                }
            } else {
            out.println( "This page was called without a report name." );
        }
    }
%>
</body>
</html>
```

The same approach can be used to provide a page that is more suitable for printing from the browser window.

Bookmarking Reports and Saving States

Bookmarking a report allows a user to save the "states" of a report in the DB2 Alphablox repository. Using PersistenceBlox, you can save a report's format and data layout in the repository's reportingpersistence/ directory. When the bookmarked report is loaded, a connection to the data source is instantiated and the up-to-date data is represented with the preserved format and layout. Because live data is fetched when a bookmark is loaded from the repository, the underlying data source needs to be available at load time.

To save a bookmark on a relational report, you need to specify the following in the <bloxreport:persistence> tag:

- the location under the repository where the bookmark is to be stored

- the name of the bookmark

- the ID of the ReportBlox that you want to bookmark

- the "save" operation

It is important to know that before you bookmark a report, if the report is rendered in interactive mode, you should set it to non-interactive. If you don't, the report will not display correctly when the bookmark is loaded. This also means that bookmarked reports can only be loaded as non-interactive reports.

Once a bookmark is saved in the repository, to continue to allow your users to interact with the report on the screen, you should also reset the interactive mode to true. The following example assumes a report.jsp file that provides a bookmark saving function via an HTML form:

```
<%@ taglib uri="bloxreporttld" prefix="bloxreport" %>
<%@ page errorPage="error.jsp" %>

<html>
<head>
   <title>Bookmark Example</title>
   <link rel="stylesheet" href="/AlphabloxServer/theme/report.css">
</head>
<body>
   <h1>Bookmark Example</h1>
   <form method="GET" action="save.jsp" target="_blank">
     <input type="text" name="bookmark" value="bookmark name"/>
     <input type="submit" value="Save this bookmark"/>
   </form>

   <bloxreport:report id="MySalesReport1" interactive="true">
      <bloxreport:cannedData/>
   </bloxreport:report>

</body>
</html>
```

As the user enters a name for the bookmark and clicks the Save this bookmark button, save.jsp is invoked and displayed in a separate browser window. The following code snippet demonstrates the sequence of setting a report to non-interactive mode, saving a bookmark of the report, and then resetting the interactive to true:

```
//save.jsp
<%@ taglib uri="bloxreporttld" prefix="bloxreport" %>
<%@ page errorPage="error.jsp" %>
<%@ page import="com.alphablox.blox.*" %>

<html>
<head></head>
<body>
<h1>Report Bookmark Saved</h1>
<%
   String bookmark = request.getParameter("bookmark");

   // "MySalesReport1" is the ID of a ReportBlox
   // page that calls this SaveBookmark.jsp page.
   BloxContext context = BloxContextFactory.getBloxContext(request, response);
   ReportBlox report = (ReportBlox) context.getBlox("MySalesReport1" );

   // Set the interactive mode to false
   report.setInteractive( false );
%>
<%--Add a PersistenceBlox, providing the ReportBlox ID
    (targetBloxID), location, bookmark name (persistedName),
    and operation to perform--%>
<bloxreport:persistence id="p1"
   targetBloxId="MySalesReport1"
   location="sales/east"
   persistedName="<%= bookmark %>"
   operation="save" />

// Set the report back to interactive mode
<%
   report.setInteractive( true );
%>

<p>The bookmark <b><%= bookmark %></b> has been saved.</p>
</body>
</html>
```

If your DB2 Alphablox repository is file based, the above example will save a bookmark named "SavesApr02" in the following location:

```
<alphablox_dir>/repository/reportingpersistence/sales/east/
```

If however, your DB2 Alphablox repository is database based, the above example will save a bookmark in the repository with `"reportingpersistence/sales/east"` in the `CONTEXT` column and `"SavesApr02"` in the `NAME` Column.

Loading Bookmarks

To retrieve a bookmark for relational reports, specify the location, the name of the bookmark, and the "load" operation. In order for the bookmark to be loaded successfully, the underlying data source needs to be available for live data to be retrieved. The following example demonstrates how to generate a list of bookmarks saved in the repository and make each bookmark a link:

```
<%@ taglib uri="bloxreporttld" prefix="bloxreport" %>
<%@ page errorPage="error.jsp" %>

<html>
<head>
   <title>List Bookmarks</title>
</head>
<body>
   <p>Your bookmarks:</p>
   <bloxreport:persistence id="repository" />
   <table>
   <%
      String[] files = repository.list("sales/east",
        repository.LIST_TYPE_STATES );
      for( int i = 0; i < files.length; i++){
      %>
        <tr><td><a href="load.jsp?bookmark=<%= files[i] %>">
        <%= files[i] %></td></tr>
      <%
      }
   %>
   </table>
</body>
</html>
```

When the user clicks a bookmark link, `load.jsp` is invoked with the bookmark name passed in:

```
<%@ taglib uri="bloxreporttld" prefix="bloxreport" %>
<%@ page errorPage="error.jsp" %>

<html>
<head>
   <title>Loading Bookmark</title>
   <link rel="stylesheet" href="/AlphabloxServer/theme/report.css">
</head>
<%
   String bookmark = request.getParameter( "bookmark" );
%>
<h1>Bookmark: <%= bookmark %></h1>
<%
   if( bookmark != null ) {
      %>
      <bloxreport:persistence
         location="sales/east"
         persistedName="<%= bookmark %>"
         operation="load" />
```

```
        <%
    }
%>
</html>
```

It's important to know that when a bookmarked report is loaded, PersistenceBlox creates an instance of the ReportBlox object on the server. If an instance already exists in the session, PersistenceBlox will overwrite the old instance. The Saving and Exporting Data with Dynamic Queries example in Blox Sampler - Relational Reporting demonstrates how you can manage this issue using `bloxName`.

Saving as PDF

A relational report can be rendered in PDF format using PdfBlox. This section discusses two scenarios where you may offer a PDF version of a relational report. The first scenario is the more common scenario where you provide a link or button on the page to render the report to PDF. The other scenario discussed is when you want users to view reports only in PDF; that is, instead of displaying a report in DHTML (interactive mode) or static HTML (non-interactive mode), you can have a report displayed directly in PDF.

Saving Reports as PDF Files

To offer users an option to save a report as a PDF file, you may offer a button or a link on the page, that when clicked, saves the report as a PDF file. This involves a separate JSP file as shown in the following diagram:
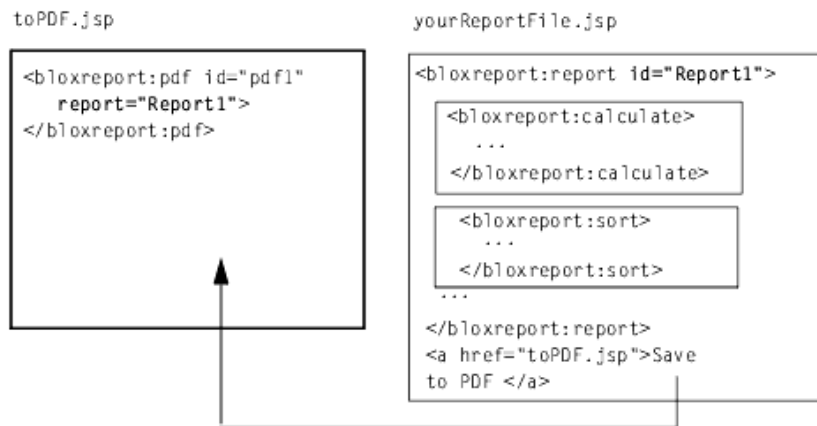
The `yourReport.jsp` page has a ReportBlox with an `id` of Report1. When users click the Save to PDF button, the second page `toPDF.jsp` is called. This page has a PdfBlox that takes the Report1 ReportBlox and saves it as a PDF file.

To dynamically render any report with only one copy of the `toPDF.jsp` file, use the following code to get the parameter passed with the request:

```
<%@ taglib uri="bloxreporttld" prefix="bloxreport" %>
<bloxreport:pdf id="mypdf"
    report='<%= request.getParameter("report") %>' />
```

This is all you need in `toPDF.jsp`. The first line is to include the needed tag library and the second line adds a PdfBlox with its `report` attribute set to the value of the `report` parameter passed in via the request. Note that a unique `id` for the

`<bloxreport:pdf>` tag is required. In your JSP file containing the report, the link should be modified to pass the report `id` via the `report` parameter:

```
<a href="toPDF.jsp?report=reportId">Send to PDF</a>
```

You have to specify an `id` to the ReportBlox in order for PdfBlox to correctly reflect the state of the report to produce the PDF. That is, the PDF rendering engine will preserve the changes the user makes through the interactive context menus.

Note the following:

- You can have only one PdfBlox on a page; if you have more than one, the first one will be the one displayed in the browser.
- PdfBlox will preserve the style set through the style sheet, the interactive context menus, or StyleBlox in the JSP file.
- The rendered PDF will not include the other HTML texts in the JSP. PdfBlox only renders the ReportBlox in the JSP page and ignores all other HTML texts on the page.
- The header includes the specified logo on the upper left corner. The header text you specify using the `headerText` tag attribute always appears in the center. You can specify the width of the top border and the header height. You can also set style on the header text using standard XHTML. For example, `headerText="<span style='color:blue'>Product Sales</span>"`.
- The footer always has the timestamp at the lower left corner and the current page number out of the total page count at the lower right corner. The footer text you specify using the `footerText` tag attribute always appears in the center. You can specify the width of the bottom border and the footer height. You can also set style on the footer text using standard XHTML.

Rendering a Report Directly in PDF

To directly display a report reflecting live data in PDF, use the `<bloxreport:pdf>` tag to wrap outside the `<bloxreport:report>` tag, as shown in the following diagram.

```
yourReportFile.jsp
<bloxreport:pdf id="pdf1">
  <bloxreport:report id="report1">
    <bloxreport:calculate>
    ...
    </bloxreport:calculate>

    <bloxreport:sort>
    ...
    </bloxreport:sort>
  ...
  </bloxreport:report>

</bloxreport:pdf>
```

Note:

- A unique `id` for the `<bloxreport:pdf>` tag is required.
- You can have only one PdfBlox on a page. If you add more than one PdfBlox and each has its own unique `id`, even though both objects will be created on the server, the browser will display only the first PdfBlox.
- Each PdfBlox will take only one ReportBlox. If you have two or more ReportBlox within the `<bloxreport:pdf>` tag, the ReportBlox declared the last will be used.

Saving to Excel or Other Applications

This section discusses two scenarios where you may offer users a way to see and manipulate the data in Excel or other applications. The first scenario is the more common scenario where you provide a link or button on the page to send the currently displayed report to Excel. The other scenario discussed is when you want users to view report data directly in Excel; that is, instead of displaying a report in DHTML (interactive mode) or static HTML (non-interactive mode), the data is directly displayed in Excel.

Exporting to Excel

The JSP `page` directive has this `contentType` attribute that tells the browser what content type to expect of the requested page. By setting this attribute, you can instruct the browser what application to invoke to handle the page. For sending the currently displayed report to Excel, you need to:

- Set the `contentType`:

```
<%@ page contentType="application/vnd.ms-excel;
    charset=UTF-8" %>
```

- Set the `interactive` property of the ReportBlox to `false`. By setting the report to be non-interactive, the resulting file will not include menu items from the context menus. This is done in similar way as in "Saving as Static HTML to File System" on page 66.

In your JSP file containing ReportBlox, assuming you have code like the following to call a separate JSP page for exporting the report to Excel and pass in the report's `bloxName`:

```
<head>
<%
  String reportName = "myReport";
%>
<script>
  var REPORT_NAME = "<%= reportName %>";
  function toExcel() {
    window.open( "excel.jsp?reportname=" + REPORT_NAME, REPORT_NAME +
      "_excel", "height=400, width=600, scrollbars=yes, resizable=yes");
  }
</script>
</head>

<body>
...
<a href="javascript:toExcel()" >Export to Excel</a>

<bloxreport:report id="report" bloxName="<%=reportName%>"
  interactive="true" errors="true" >
  ...
</bloxreport:report>
```

`excel.jsp` looks as follows:

```
<!--Using the taglibs -->
<%@ taglib uri="bloxreporttld" prefix="bloxreport" %>
<!--Importing the associated java classes in order to use the APIs -->
<%@ page import="com.alphablox.blox.*" %>
<!-- Setting the contentType -->
<%@ page contentType="application/vnd.ms-excel; charset=UTF-8" %>

<%
  //Set report to non-interactive mode so the menus don't get rendered
    String reportName = (String)request.getParameter( "reportname" );
    if( reportName != null ) {
```

```
        BloxContext context = BloxContextFactory.getBloxContext(request, response);
        ReportBlox reportBlox = (ReportBlox)context.getBlox( reportName );
        if( reportBlox != null ) {
          reportBlox.setInteractive(false);
        } else {
          out.println( "The report " + reportName + " does not exist." );
        }
      } else {
        out.println( "This page was called without a report name." );
      }
%>
<html>
<head>
  <title><%= reportName %> Excel Report</title>
  <%--In line the styles so they'll be saved if the page is saved and
      you are not asked to be authenticated--%>
  <style>
    <jsp:include page="style/reportstyles.css" flush="true"/>
  </style>
</head>
<body>

<%--The reportblox tag that follows causes the html of the report to be
rendered --%>
<bloxreport:report id="reportBlox" bloxName="<%=reportName%>" />
</body>
</html>
```

Notice that we use a JSP include statement to include an inline style sheet. Depending on the version of Excel and your operating system, using imported style sheet may result in the need to authenticate the user when Excel needs to access the style sheet. If you use the JSP include technique to inline a style sheet, note the following:

- This style sheet should not import other style sheets as Excel cannot resolve the imports.

- If you want to use the provided report.css style sheet at /AlphabloxServer/theme/, since this style sheet actually imports three other style sheets (styles.css for the report, dialog.css for the context menus, and errors.css for ErrorBlox), you should change the JSP include statement to use styles.css:

```
<head>
  <style>
    <jsp:include page="/AlphabloxServer/theme/styles.css" flush="true"/>
  </style>
</head>
```

This ensures the report format and layout is preserved in Excel, no additional authentication is required, and Excel does not hang due to failure to resolve the style sheet import statements.

Sending a Report Directly to Excel

The JSP page directive has this contentType attribute that tells the browser what content type to expect of the requested page. You can send a report directly to Excel by setting the appropriate contentType using the following code in the beginning of your JSP page:

```
<%@ page contentType="application/vnd.ms-excel;
charset=UTF-8" %>
```

The browser, upon receiving the returned response from the server, will launch the application specified to open files of this particular content type to display the returned page.

Sending a Report to Comma-separated or Tab-delimited Formats

Often times we need to export data in comma- or tab-delimited formats for use in Excel or other applications. This involves the use of Relational Reporting API to walk through the result set to insert the delimiter between data columns and a newline character between data rows. See the topic on "The Relational Reporting API" on page 82 for a discussion on the API. See the topic on "Navigating the result set: Rendering a Report to Comma-separated or Tab-delimited Formats" on page 85 for the complete example.

Chapter 9. Styling the Report Editor User Interface

You may want to customize the look and feel of the Report Editor user interface to adhere to your corporate color scheme or the overall look and feel of your application. This section describes the style classes used in the Report Editor so you can specify your own styles for those classes.

- "Style Classes in the Report Editor User Interface" on page 77
- "User Help for Using the Report Editor" on page 80

Style Classes in the Report Editor User Interface

The Report Editor user interface includes three context menus and a Report Style dialog box. The fonts and colors displayed are all based on a set of style classes.

The style classes for the context menus are:
- `.menu`
- `.choice`
- `.choicehover`
- `.separator`
- `.selected`

The style classes related for the Report Style dialog box are:
- `.dialog`
- `.dialogtitle`
- `.dialogbody`
- `.dialoggroup`
- `.dialoggrouptitle`
- `.dialogradiogroup`
- `.dialogbutton`
- `.dialogbuttongroup`

The following image shows the styles for context menus:
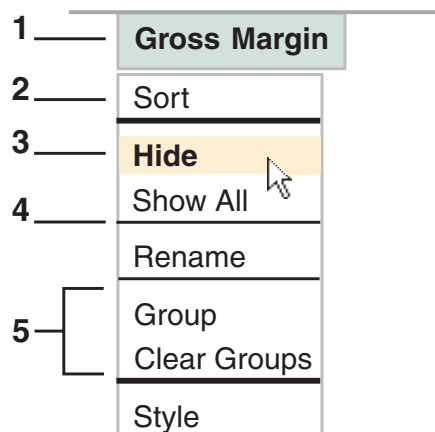
The following diagram shows the classes for the Report Style dialog box:
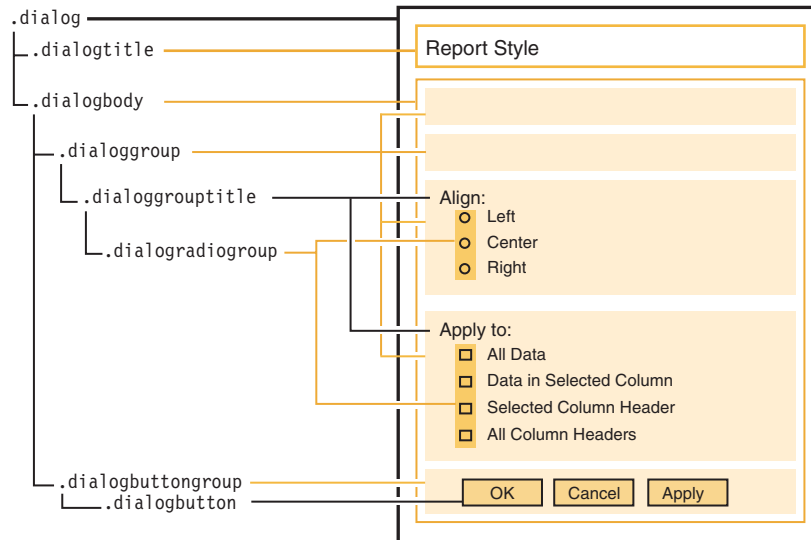
The following is the default style provided in `dialog.css`. It demonstrates how you can define the fonts, colors, and styles for the Report Style dialog window:

```
//for overall background color, fonts, position, borders...
.dialog {
   background-color: #E3E3E3;
   font-size: 80%;
   font-family: sans-serif;
   position: absolute;
   cursor: default;
   border: solid 2 black;
   border-top-width: 1;
   border-left-width: 1;
}

//for the dialog window title area
.dialogtitle {
   background-color: #CCCCCC;
   text-align: left;
   font-weight: bold;
   padding : 5 10 5 10;
   margin-bottom: 5;
   cursor: hand;
}

//for the body of the dialog window
.dialogbody {
   padding : 10;
   vertical-align: middle;
}

/* for each group of selections; there are four groups in the
dialog window, each with a thin border (1px) in color
#CCCCCC */
.dialoggroup {
   text-align: left;
   padding : 5;
   border : solid 1 #CCCCCC;
   margin: 5;
}

//for the title in each group
.dialoggrouptitle {
   margin-bottom: 5;
}
```

```
//for each button, either radio or check box
.dialogradiogroup {
   padding-left: 10;
}

//for each of the three action buttons--OK, Cancel, and
//Apply
.dialogbutton {
   margin-right: 5;
   font-size: 80%;
   padding: 2 0 1 0;
   width: 60;
}

//for the group of three action buttons--OK, Cancel, and
//Apply
.dialogbuttongroup {
   text-align: right;
   padding-top :20;
}
```

Note: When setting font faces, styles, and sizes in the style sheets, watch out particularly for the consistency of the font faces and sizes. Typically you want to use the same font faces and sizes for the following classes so the columns do not change in width and the report does not appear jiggling when users move their cursors over the hot spots:

- `.column`
- `.choicehover`
- `.selected`

Overriding the Style Classes

Having an external style sheet makes your JSP page cleaner and allows you to reuse the same style sheet for your entire application. A good technique to use is to create your external style sheet that imports the style sheets supplied by DB2 Alphablox and then overrides the style classes you want to customize. This means you will not miss any classes that need to be defined, and you only need to add one link to your external style sheet.

For example, your JSP page containing a ReportBlox will reference your style sheet:

```
<html>
<head>
   <link rel="stylesheet" href="myreportbuilder.css" type="text/css" />
   ...
```

In the beginning of your `myreportbuilder.css`, import the style sheets that come with DB2 Alphablox:

```
@import url( /AlphabloxServer/theme/report.css );

.report {
   background-color: white;
   color: black;
   font-family: Trebuchet MS,Verdana, Arial, Helvetica, sans-serif;
   border: solid 1 black;
   padding: 5;
   margin : 5;
   width: 0;
}
```

This way your styles will override the ones in the supplied style sheets. Keep in mind that you should never modify the style sheet in the repository's theme directory since it gets replaced during DB2 Alphablox upgrades.

User Help for Using the Report Editor

If your report is interactive, you may want to provide user help that explains how the user interface works. There is no link to the end user help files from within the Report Editor user interface. You will need to provide your own link. The help files are simply a collection of HTML files and images, so you can easily customize them for your application. For details on where the files are and how to link to them, see "Providing User Help" on page 94.

Chapter 10. Advanced Topics

This section covers advanced topics related to relational report development. In particular, the topics of managing session scope, using the Alphablox Relational Reporting API, and accessing data rows and cell values in the rendered report will be discussed.

- "Managing Session Scope" on page 81
- "The Relational Reporting API" on page 82
- "Dynamically Changing the Query" on page 88
- "Accessing Data Rows and Cell Values in Rendered Report" on page 91

Managing Session Scope

When you use Blox Report tags to add Relational Reporting Blox to your JSP page, these Blox have a session scope by default. Imagine a report builder application that lets your users dynamically create a relational report based on some criteria they specify. Once they select how they want to group or sort the data, they click the "Generate Report" button to create the report. The generated report is set to render in interactive mode, and therefore the users can edit the report using the Report Editor user interface.

Each time a new set of criteria is specified and a new report is requested, you will need to reuse the server object that already exists since no new object will be instantiated. This can be done through the use of the `bloxName` attribute. The optional `bloxName` tag attribute lets you reuse an object as its value can be dynamically assigned. While the required `id` uniquely identifies a Blox on the page, `bloxName` can be dynamically assigned using a variable. The value of `id` is used as the scripting variable name in your JSP. The value of `bloxName` is the name the server knows the object as. If `bloxName` is not specified, `id` is used as both the server object name and the scripting variable name. If `bloxName` is specified, you can change it dynamically in your JSP.

```
<% String bloxName="report1"; %>

<bloxreport:report id="myReportBlox"
   bloxName="<%= bloxName %>" >
...
```

Changing the `bloxName` value is a useful development technique so you do not need to close and restart the browser windows every time you make a change to your JSP. More importantly, it allows you to dynamically set Blox properties based on user selection. The Saving and Exporting Data with Dynamic Queries example in Blox Sampler - Relational Reporting also demonstrates the use of `bloxName` to render a report to different formats in a separate browser window.

Another phenomenon with session scope is that users' changes to the report will not be preserved when they access the report in a different session. You will want to provide report saving options such as sending the report to PDF or Excel so your users can save their report onto their local system or somewhere on the server. For providing report saving functionality, see "Saving as Static HTML to File System" on page 66, "Saving Reports as PDF Files" on page 71, and "Exporting to Excel" on page 73.

The Relational Reporting API

As described earlier in "Report Pipeline" on page 8, a relational report is created by processing the Relational Reporting Blox in the sequence they are added. The ultimate data producers are SQLDataBlox and RDBResultSetDataBlox. The ultimate consumer is ReportBlox. CalculateBlox, FilterBlox, GroupBlox, MembersBlox, OrderBlox, and SortBlox in the middle are "transformers" that serve as both data consumers and data producers that take the data from the previous Blox, transform it in some way, and pass it on to the next Blox.

All these "transformers" inherit from the *IConsumer* and *IProducer* interfaces. *IConsumer* has a setInput() method that sets the input for the Blox. IProducer has a getData() method that gives you access to the *IDataSet* interface for further access to *IDimension* and *IMember*. For example, by calling the SQLDataBlox getData() method, you gain access to *IDataSet* and the ability to walk through the result set.

The following example demonstrates how a relational report is created using the API:

```
<%@ page import="com.alphablox.blox.*" %>
<html>
<head>
   <link rel="stylesheet" href="/AlphabloxServer/theme/report.css" />
</head>
<body>
<%
  String query="SELECT location, product_name, sales, units, cost
FROM qcc WHERE week_ending = '2000-04-08'";

  try {
    ReportBlox rBlox = new ReportBlox();
    rBlox.setErrors(true);
    rBlox.setId("myRBlox");

    DataSourceConnectionBlox dConn = new DataSourceConnectionBlox();
    dConn.setDataSourceName("qcc-rdb");
    dConn.connect();

    SQLDataBlox dBlox = new SQLDataBlox();
    dBlox.setInput(dConn);
    dBlox.setQuery(query);
    dBlox.execute();

    // Create the grouping
    GroupBlox myGroup = new GroupBlox();
    myGroup.setMembers( new String [] {"location"});
    myGroup.setAggregationType("units", "none");

    // Set up the input for the group
    myGroup.setInput(dBlox);
    rBlox.setInput(myGroup);

    // Finally call ReportBlox's write() method to write it out
    rBlox.write(out);

  }
  catch ( Exception e ) {
    ErrorBlox eBlox = new ErrorBlox();
    Throwable msg = eBlox.getRootCause(e);
    out.println("<br><b> This Exception was captured</b><br> "+
msg.getMessage());
```

```
  }
%>
</body>
</html>
```

Note that this example creates a non-interactive report as this is the default behavior.

Creating an Interactive Report using the API

To create an interactive report using the API, there are three additional things you need to specify.

1. Set the ReportBlox to interactive using the `setInteractive` method.

    ```
    rBlox.setInteractive(true);
    ```

2. Set the URL prefix. This information is needed by the servlet handling interaction. Typically this is handled automatically for you when you use the Blox Report Tag Library to create your ReportBlox. When using the API, you have to explicitly specify the URL prefix as follows:

    ```
    BloxContext context = BloxContextFactory.getBloxContext(request, response);
    rBlox.setUrlPrefix(context.getContextPath() + "/" +
        URLFactory.ALPHABLOX_SERVER_PREFIX);
    ```

3. Place the ReportBlox into BloxContext. When you use the Blox Report Tag Library, this is also handled automatically for you. When using the API, you have to explicitly add the ReportBlox to the BloxContext and call the `init()` method.

    ```
     String scriptId = rBlox.getBloxName();
     if(scriptId == null)
     {
        scriptId = rBlox.getId();
     }
     rBlox.init(context,scriptId);
    ```

    This piece of code registers the ReportBlox with the BloxContext using the optional `bloxName` if it is specified. If not, it uses the Blox `id`.

The complete example is now as follows:

```
<%@ page import="com.alphablox.blox.*,
                 com.alphablox.net.URLFactory"%>
<html>
<head>
   <link rel="stylesheet" href="/AlphabloxServer/theme/report.css" />
</head>
<body>
<%
  String query="SELECT Week_Ending, location, product_name, sales, units, cost
     FROM qcc WHERE Week_Ending_Text = '04-08-2000'";

  try {
    ReportBlox rBlox = new ReportBlox();
    rBlox.setErrors(true);
    rBlox.setId("myRBlox");

    // 1. Set the ReportBlox to interactive
    rBlox.setInteractive(true);

    DataSourceConnectionBlox dConn = new DataSourceConnectionBlox();
    dConn.setDataSourceName("qcc2003-rdb");
    dConn.connect();

    SQLDataBlox dBlox = new SQLDataBlox();
    dBlox.setInput(dConn);
```

```
                dBlox.setQuery(query);
                dBlox.execute();

                // Create the grouping
                GroupBlox myGroup = new GroupBlox();
                myGroup.setMembers( new String [] {"location"});
                myGroup.setAggregationType("units", "none");

                // Set up the input for the group
                myGroup.setInput(dBlox);
                rBlox.setInput(myGroup);

                // 2. Set the URL prefix
                BloxContext context = BloxContextFactory.getBloxContext(request, response);
                rBlox.setUrlPrefix(context.getContextPath() + "/" +
                    URLFactory.ALPHABLOX_SERVER_PREFIX);


                // 3. Add the ReportBlox to the Context
                String scriptId = rBlox.getBloxName();
                if(scriptId == null)
                {
                    scriptId = rBlox.getId();
                }
                rBlox.init(context,scriptId);

                // Finally call ReportBlox's write() method to write it out
                rBlox.write(out);

            }
        catch ( Exception e ) {
            ErrorBlox eBlox = new ErrorBlox();
            Throwable msg = eBlox.getRootCause(e);
            out.println("<br><b> This Exception was captured</b><br> "+
    msg.getMessage());
            }
    %>
    </body>
    </html>
```

You probably have found that it is a lot more convenient to use tags to create your relational report. You can always script to the objects using their id when you need to.

Walking through the pipeline: clearing the sort order

This example demonstrates how to walk through a ReportBlox pipeline to find instances of a Blox. In this example, we find any sorting done on our report and clear the sorting.

1. Import the IConsumer and IProducer packages.

   ```
   <%@ page import="com.alphablox.blox.*,
                    com.alphablox.pipeline.IConsumer,
                    com.alphablox.pipeline.IProducer" %>
   ```

2. The query is dynamically set using the SQLDataBlox setQuery() and execute() methods. For more information, see the topic on "Dynamically Changing the Query" on page 88.

3. Use a WHILE loop to check, for each IConsumer, whether the input is a SortBlox. If it is a SortBlox, set the rules to null. See "The Relational Reporting API" on page 82 for more information on IConsumer and IProducer.

The complete example is as follows:

```
<%@ taglib uri="bloxreporttld" prefix="bloxreport" %>
<%@ page import="com.alphablox.blox.*,
                 com.alphablox.pipeline.IConsumer,
                 com.alphablox.pipeline.IProducer" %>
<html>
<head>
   <link rel="stylesheet" href="/AlphabloxServer/theme/report.css" />
</head>
<body>
<%
   String DATASOURCE = "qcc-rdb";
   String query = "select * from qcc where sales > 8000";

   // This block of code is placed before the ReportBlox in case the report
   // is set to non-interactive mode
   BloxContext context = BloxContextFactory.getBloxContext(request, response);
   ReportBlox reportBlox = (ReportBlox)context.getBlox("salesReport");
   if( reportBlox != null ) {
      Object input = reportBlox.getInput();

      while( input instanceof IConsumer )
      {
         if( input instanceof SortBlox )
         {
            ((SortBlox)input).setRules( null );
            out.println( "Sort rules cleared" );
         }

         input = ((IConsumer)input).getInput();
      }
      // You could check if input is SQLDataBlox to be safe.
      // Dynamicaly execute the query each time the page is loaded.
      ((SQLDataBlox)input).setQuery( query );
      ((SQLDataBlox)input).execute();
   }
%>

<bloxreport:report id = "assetReport" interactive="true">

   <bloxreport:sqlData id="assetReportSql"
      query = "<%= query %>" >
      <bloxreport:dataSourceConnection
         dataSourceName = "<%=DATASOURCE%>" >
      </bloxreport:dataSourceConnection>
   </bloxreport:sqlData>
   <bloxreport:style>
      <bloxreport:banding style="background-color: white;" />
      <bloxreport:numeric style="text-align: center" />
   </bloxreport:style>
</bloxreport:report>
</body>
</html>
```

Navigating the result set: Rendering a Report to Comma-separated or Tab-delimited Formats

This is an advanced example demonstrating how to export data in tab-delimited format to Excel or other applications by walking through the SQLDataBlox result set. It also demonstrates how to access the ReportBlox or SQLDataBlox for a relational report from BloxContext.

The following is a JSP file where a ReportBlox is created and retrieved from BloxContext:

- A link is offered for users to export the data. When the link is clicked, another JSP file is called (toExcelWithTab.jsp) with the report name passed along.
- The report name is used to identify the SQLDataBlox for the ReportBlox for this report from BloxContext.

```
<%@ taglib uri="bloxreporttld" prefix="bloxreport" %>
<%@ page import="com.alphablox.blox.*"  %>
<%@ page errorPage="error.jsp" %>
<%
  String reportName = "qcc";
  String reportDataName = reportName + "_data" ;
  String dataSourceName = "qcc-rdb";
  String query = "SELECT * from QCC where sales > 5000";

  //Get the ReportBlox and SQLDataBlox for this report from BloxContext
  BloxContext context = BloxContextFactory.getBloxContext(request, response);
  ReportBlox reportBlox = (ReportBlox)context.getBlox(reportName);
  SQLDataBlox report_dataBlox = (SQLDataBlox)context.getBlox(reportDataName);

  if( report_dataBlox != null )
  {
    report_dataBlox.setQuery( query );
    report_dataBlox.execute();
  }
%>

<html>
<head>
  <link href="/AlphabloxServer/theme/coleman.css" rel="stylesheet">
</head>
<body>
<a href="toExcelWithTab.jsp?reportname=<%= reportName %>"
target="<%= reportName %>_csv" >Send to CSV </a>
<hr>
<bloxreport:report id="report" bloxName="<%=reportName%>"
  interactive="false" errors="true" >
  <bloxreport:sqlData id="data" bloxName="<%= reportDataName %>"
    query="<%=query%>" >
    <bloxreport:dataSourceConnection
      dataSourceName="<%= dataSourceName %>" />
  </bloxreport:sqlData>
</bloxreport:report>
</body>
</html>
```

The toExcelWithTab.jsp file then performs the following actions:

- The MIME type is set to application/vnd.ms-excel. This triggers the browser to load this page as an Excel worksheet.
- Retrieve the data result set from the SQLDataBlox for this report.
- Walk through the result set to add a tab as a column delimiter and a newline as a line delimiter. This requires importing the com.alphablox.pipeline.* package for the Relational Reporting API.

Sets the MIME type to application/vnd.ms-excel

```
<%@ taglib uri="bloxreporttld" prefix="bloxreport" %>
<%@ page import="com.alphablox.blox.*,
                 com.alphablox.pipeline.*"  %>
<%@ page errorPage="error.jsp" %>
<%@ page import="java.io.*" %>
<%@ page contentType="application/vnd.ms-excel; charset=UTF-8" %

<%
String reportName = request.getParameter( "reportname" );
String delimiterName = "tab";
```

```
String reportDataName = reportName + "_data" ;
String title = "Render to tab-delimited format";
String lineDelimiter = "\n";
String columnDelimiter = "\t";

// Output to strings used to indicate the beginning and end of the report data
String dataSetBegin = "Data for " + reportName;
String dataSetEnd  = "\nEnd " + reportName + " data";

%>

<bloxreport:sqlData id="data"
 bloxName="<%= reportDataName %>"

<%
// Get the results from SQLDataBlox using getData().
// IDataSet represents the relational data in the pipeline, with a column
// dimension and a row dimension-- IDimension.COLUMN_DIMENSION_NAME and
// IDimension.ROW_DIMENSION_NAME.
IDataSet dataSet = data.getData();
IDimension columnDimension = dataSet.getDimension(IDimension.COLUMN_DIMENSION_NAME);
int rowIdx=dataSet.getDimensionIndex(IDimension.ROW_DIMENSION_NAME);
int colIdx=dataSet.getDimensionIndex(IDimension.COLUMN_DIMENSION_NAME);
int rowCnt=dataSet.getDimension(IDimension.ROW_DIMENSION_NAME).getMemberCount();
int colCnt=dataSet.getDimension(IDimension.COLUMN_DIMENSION_NAME).getMemberCount();
int[] coordinates = new int[2];

out.println( "There are " + rowCnt + " rows and " + colCnt + " columns." ) ;

if( dataSetBegin != null )
  out.println( dataSetBegin );

// itterate over the column headers
for( int column = 0; column < colCnt; column++ )
{
  if( column != 0 )
    out.print( columnDelimiter );
    out.print( columnDimension.getMember( column ).getName() );
}
out.print( lineDelimiter );

// itterate over the actual data
for( int row = 0; row < rowCnt; row++ )
{
  if( lineDelimiter != null && row != 0 )
    out.print( lineDelimiter );

  for( int column = 0; column < colCnt; column++ )
  {
    if( column != 0 )
      out.print( columnDelimiter );
      coordinates[rowIdx]     = row;
      coordinates[colIdx] = column;
      out.print( dataSet.getValue( coordinates ) );
  } // end columns

} //end rows

if( dataSetBegin != null )
  out.print( dataSetEnd );
%>
```

Dynamically Changing the Query

All the relational reporting Blox are instantiated in the BloxContext as beans. These beans can be individual accessed and their methods invoked in a JSP page. For methods associated with these Blox, see the ReportBlox Javadoc at:

```
<alphablox_dir>\system\documentation\javadoc\report\index.html
```

Since changes to tag attribute values are not re-evaluated when the page reloads, when you want to dynamically change the query after the report has been created and rendered, you should reuse the object on the server. To reuse the object:

- Make sure you set the `bloxName` attribute on your Blox.
- Get the Blox from the Blox context:

```
<%
   BlxoContext context = BloxContextFactory.getBloxContext(request, response);
   SQLDataBlox dataBlox  = (SQLDataBlox)context.getBlox( "myReportData" );
   dataBlox.setQuery( query );
   dataBlox.execute();
%>
```

where "myReportData" is the `bloxName` assigned to the SQLDataBlox.

The following examples demonstrate different ways to perform the task more efficiently without removing the objects from the session. One technique is to access the Blox you need to modify directly from the session attribute (for example, accessing the SQLDataBlox to set and execute a new query) .

Example 1: Directly access the SQLDataBlox and resets its query

- The example has two radio buttons for user to select. One for February and the other for March.
- The `com.alphablox.blox.*` import statement is needed to use the ReportBlox API.
- The SQLDataBlox in the ReportBlox has an `id` of `sqlDataBlox`. Upon initial page load, its query is set to `query2`, which shows the data for February.
- `monthlyData` is typecast to be of type SQLDataBlox.
- When the user selects a month, the page reloads and the requested month is checked. The query for the existing SQLDataBlox is reset and executed using the `setQuery()` and `execute()` methods.

```
<%@ taglib uri="bloxreporttld" prefix="bloxreport"%>
<%@ page errorPage="error.jsp" %>
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ page import="com.alphablox.blox.*" %>
<%
   String query2 = "SELECT month, sales FROM qcc WHERE year=2000 AND
month=2";
   String query3 = "SELECT month, sales FROM qcc WHERE year=2000 AND
month=3";
   String query  = query2; //defaults to February
   String monthNum = null;
   String thisURL  = "http://" + request.getServerName() +
      (request.getServerPort()==80?"":":" +
      request.getServerPort()) + request.getRequestURI();
   SQLDataBlox monthlyData = null;

   // if monthlyData already exists, there will be a sesson
   // object of same name. This will never happen in the first
   // pass through this page
```

```
        BloxContext context = BloxContactFactory.getBloxContext(request, response);
        monthlyData = (SQLDataBlox) context.getBlox( "sqlDataBlox" );
        monthNum = request.getParameter( "monthNum" );

        // User did request a monthnum
        if( (monthNum != null) && ! ("".equals( monthNum )) )     {
            if( "3".equals( monthNum ) )
            {
                query = query3;
            }
            if( monthlyData != null )
            { // if monthlyData exists, the query is executed through
              // a method on the Blox
                monthlyData.setQuery( query );
                monthlyData.execute();
            }
        }
%>
<html>
<head>
    <title>ReportBlox with Changing Query</title>
    <!--Uses the default style sheet; required for interactive
        mode-->
    <link rel="stylesheet"
    href="/AlphabloxServer/theme/report.css" />
</head>

<body bgcolor="#FFFFFF">
<form name="monthForm" action="<%=thisURL%>" method=POST  >
    <input type="radio" name="monthNum" value="2"
        OnClick="document.monthForm.submit()"
        <%=(! "3".equals( monthNum ))?"checked":""%> >February
    <input type="radio" name="monthNum" value="3"
        OnClick="document.monthForm.submit()"
        <%=("3".equals( monthNum ) )?"checked":""%> >March
</form>

<b>The current Query is:</b>
<pre><%=query%></pre>
<bloxreport:report id="profitReport" interactive="false">
    <bloxreport:sqlData id="sqlDataBlox"
        query="<%=query%>" >
        <bloxreport:dataSourceConnection
            dataSourceName="qcc-rdb" />
    </bloxreport:sqlData>
</bloxreport:report>
```

The Sales Report with HTML Links example in the Blox Sampler - Relational Reporting example set uses a similar technique to dynamically set the query.

Example 2: Dynamically setting queries without refreshing the whole page using the global refreshReport() JavaScript method

This example demonstrates how to dynamically setting the query without reloading the page by calling another JSP that sets the query on the server and then refresh the ReportBlox in the current page.

- When users request a different query such as through a form selection, instead of loading in a new page or refreshing the whole page, this example uses an iframe as the target for the form post action. This allows you to execute some server-side code without reloading the current page.
- The JSP page that is called resets the query for the underlying SQLDataBlox.

- In order to refresh just the ReportBlox on the current page to reflect the change to the underlying query, we use the global refreshReport(*ReportBloxName*) JavaScript method:

```
<script>
   function refresh( reportName ) {
      refreshReport( reportName );
   }
</script>
```

Note: The refreshReport() JavaScript method only works in interactive reports.

- This example also demonstrates the use of bloxName. Unlike the id tag attribute, which cannot be dynamically set, the optional bloxName allows you to dynamically create Blox names with tags. If you specify the value of bloxName for a Blox, then:
  - this bloxName will be the name of the Blox the DB2 Alphablox server knows this object as
  - this bloxName will be the name of the rendered JavaScript object (to be used in the global JavaScript function refreshReport() when referencing the ReportBlox)
  - id will now only serve as the Java scripting variable you use in your JSP page.

For the complete example, see the Saving and Exporting Data with Dynamic Queries example in the Blox Sampler - Relational Reporting example set.

When users specify new criteria for the report, the called page reuses the ReportBlox and SQLDataBlox objects in BloxContext to set a new query:

```
<%@ taglib uri="bloxreporttld" prefix="bloxreport" %>
<%@ page import="com.alphablox.blox.*" %>

<%
   // Some code omitted here that dynamically sets the new query.
   // The variables reportName and reportDataName are set to the values of bloxName
   // for ReportBlox and SQLDataBlox.


   BloxContext context = BloxContextFactory.getBloxContext(request, response);
   ReportBlox reportBlox = (ReportBlox)context.getBlox(reportName);
   SQLDataBlox dataBlox = (SQLDataBlox)context.getBlox(reportDataName);
   dataBlox.setQuery( query );
   dataBlox.execute();
%>

<html>
<head>
</head>

<body>
<script>
   <% if( isError )
      {//pop the error message in an alert dialog if there was one
         out.println( "alert( \"" + errorMessage + "\")" );
      }
      else
      {//there is a js function in the parent page called refresh( reportName )
         out.println("parent.refresh( \"" + reportName + "\");");
      }
       %>
</script>
</body>
</html>
```

Accessing Data Rows and Cell Values in Rendered Report

Each data row in a report is rendered in an HTML `<TR>` tag and each data cell in a `<TD>` tag. Using JavaScript, you can gain access to a data row or its cell values. For example, using the TextBlox's `data` tag, you can add HTML code to data cells that, when a data cell is clicked, calls a JavaScript function to get the value of the cell. From the data cell, you can also get to its parent, the data row. The following example demonstrates how to get the cell values in an entire row when the user clicks on an information icon on a row.

In this example, we add an information icon using TextBlox's `data` tag to a column whose column header is replaced by an empty space and data is replaced by an info icon:

```
<bloxreport:text>
  <bloxreport:columnHeader columnName="ADummyColumn"
    text=" " >
    </bloxreport:columnHeader>
  <bloxreport:data columnName="ADummyColumn"
    text="<img src=\"i.gif\" width=\"16\" height=\"16\"
      border=\"0\" onclick=\"getRowValues(this)\" >" >
  </bloxreport:data>
</bloxreport:text>
```

When users click on the `i.gif` icon, the `getRowValues()` function is called with the current anchor object passed in. The `getRowValues()` function then gets the parent object (the row object) of the clicked object and iterates through the cells in the table row to get the cell values:

```
function getRowValues( anchorObj )           {
  var currObj = anchorObj;
  while( (currObj.tagName != "TR") && (currObj.tagName != null) ){
    currObj = currObj.parentElement;
  }

  var rowObj   = currObj;
  var cellCount = rowObj.cells.length;
  var tdObj     = rowObj.firstChild;
  var colValues = new Array( cellCount );

  for( i = 0; i < cellCount; i++ ) {
    currObj = rowObj.cells[i];
    while( (currObj != null) && (currObj.innerText == null) ) {
      currObj = currObj.firstChild;
        }

    if( currObj != null )
      colValues[i] = currObj.innerText;
  }

  alert( "ColValues is: " + colValues );
}
```

The Saving and Exporting Data with Dynamic Queries example in Blox Sampler - Relational Reporting demonstrates this and other JavaScript techniques.

Chapter 11. Development and Troubleshooting Tips

This section discusses general design considerations and troubleshooting tips helpful to your relational report development tasks.

General Tips and Development Steps

Make sure you check out the "General Report Development Steps" on page 23 for the essential steps to create any relational report. Then go over the "General Development Tips" on page 21 that provides important notes and insight for successful development of a relational report.

Design Considerations

The following are some design tips that you should take into consideration as you develop your relational report:

- Even though you can use OrderBlox to limit the data displayed in the report, keep in mind that when the report is rendered in interactive mode (with the `interactive` attribute of the `<bloxreport:report>` tag set to `true`), your users may bring back all data if they choose to **Show All** or **Clear Groups** if there is no further data manipulation (such as data filtering or sorting) after the OrderBlox is added.

- If you turn on the interactive Report Editor to allow your users to dynamically change the layout and break groups in the report, keep in mind that they can clear the break groups you have specified in your JSP file and create their own break groups. In this case, when you design the group header using the `<bloxreport:groupHeader>` tag, make sure you use the `<member/>` substitution variable to extract the name of the break group. Make sure you use the `<value/>` variable in your `<bloxreport:groupTotal>` tag to extract the aggregation value for the specified column member for the break group.

- Interactive reports generally take longer for the browsers to render. The difference may become noticeable when a report contains hundreds of data rows.

- If you have users using Netscape browsers, keep in mind that:
  - interactive reports will not display or work properly
  - use of StyleBlox may slow down the rendering (see "Impact of Style Setting on Performance" on page 94)

- If your report is interactive, you may want to provide user help that explains how the user interface works. User help files are provided for you if you want to offer them to your users. You will need to provide your own link to the help files. More details are provided in the next section on "Providing User Help" on page 94.

Providing User Help

User help files that explain how the Report Editor user interface works are provided in multiple languages. There is no link to the end user help files from the Report Editor. To offer these online help files to your users, make a copy of the help files and add a link or a button from your application to point to your copy of help files.

The help files are located at:

```
<db2alphablox_dir\system\documentation\help\ReportBlox\<locale>
```

where `<db2alphablox_dir>` is the directory in which DB2 Alphablox is installed.

The end user help consists of a set of HTML pages, a style sheet, and a set of images, and can be easily customized. In the following example, a copy of the help files is located under a `help/` subdirectory in the application folder, and the user help is loaded into a separate browser window using JavaScript:

```
<script>
function openHelp(url) {
    window.open(url,'reportHelp','width=500,height=560,
    scrollbars=yes,toolbar=no,menubar=no,directories=no,
    status=no');
}
</script>
...
<a href="javascript:openHelp('help/index.html');">
<img src="images/help.gif" alt="Instructions on using interactive reports"
border=0></a>
```

Note: To use the help files without modification, you can also link directly to the documentation directory using:

```
<a href="javascript:openHelp('/AlphabloxServer/documentation/help/
ReportBlox/{<locale>/index.html');">
```

The benefit of using the original copy of help files in the documentation directory is that the help files will always be up to date when you upgrade DB2 Alphablox. For the same reason, if you need to customize the help files, always modify your own copy. Otherwise, your changes may be lost when you upgrade.

Localization of Help

In order for the report to display correctly in languages other than English, you should specify to use the UTF-8 character set using the `page` directive's `contentType` attribute. With the `contentType` attribute, you can define the MIME type and character set.

```
<%@ page contentType="text/html; charset=UTF-8" %>
```

If you have users using different languages, you can detect the locale of the browser through the HTTP `request` object. In this case, copy all the versions of help files you need, use the `request.getLocale()` method to get the locale, and then dynamically set link to the right version of help.

Impact of Style Setting on Performance

When you use StyleBlox to set styles, in the rendered report, the styles are specified for each data cell. For example, if you have the following code that sets the text color for numeric data to "blue":

```
<bloxreport:style>
    <bloxreport:numeric style="color: blue"/>
</bloxreport:style>
```

For each data cell containing numeric data, the generated HTML code will look as follows:

```
<td class='data' style='text-align: right;color: blue;'>
145</td>
<td class='data' style='text-align: right;color: blue;'>
12.55</td>
```

Note that `'text-align: right'` is at the beginning of the style list because ReportBlox has default text alignment based on data type. In Netscape browsers, when you have a large report involving thousands of data cells, the time it takes to render the report will be noticeably slower. To improve the performance in Netscape browsers with large reports, limit your use of StyleBlox to avoid a long list of styles for each table cell. You can also overwrite the default text alignment by setting it to none:

```
<bloxreport:style>
  <bloxreport:text style=""/>
  <bloxreport:column style="text-align: center" columnName="Area"/>
  <bloxreport:column style="text-align: left" columnName="Product"/>
</bloxreport:style>
```

This removes the default text alignment and reduces the style list from:

```
<td class='data'
    style='text-align: left;text-align: center;'> N. Cal</td>
```

to:

```
<td class='data' style=';text-align: center;'> N. Cal</td>
```

Generally speaking, using style classes is better than using StyleBlox as styles set in StyleBlox will be added to each component (such as each data cell) the style is for, creating a long CSS style string to be wrapped around each occurrence of the element.

In Internet Explorer, the CSS is usually not an issue.

Common Reporting Blox Tag Errors

Included in the following list are some of the most frequently encountered errors that you are likely to come across when working with the Blox Report Tag Library.

Forgetting to include the taglib directive for Blox Report Tag Library

If you forget to place the taglib directive for the Blox Report Tag Library at the top of your JSP page, no report will be rendered since none of the tags will be recognized. Make sure you have the following taglib directive in all of your JSP pages containing Relational Reporting Blox:

```
<%@ taglib uri="bloxreporttld" prefix="bloxreport" %>
```

Forgetting to use the correct prefix for Relational Reporting Blox

If you have the taglib directive for the Blox Tag Library (`blox.tld`) and try to use the `blox` prefix for your relational report, you will get an JSP error. For example, with the `<blox:report>` tag, you will get an error complaining that there is no such tag in the tag library imported.

Since the Blox Report Tag Library is separate from the Blox Tag Library, make sure you have the correct taglib directives at the top of your JSP page and use different prefixes for the two tag libraries.

Incorrect case of a tag or tag attribute

Case and spelling errors will result in compilation errors. In these cases, the JSP compiler will throw an exception indicating that an invalid tag or tag attribute has been used. Keep in mind that tags and attribute names start with all lowercase first word, with the first letter of each subsequent word in uppercase. Examples are:
`<bloxreport:sqlData />`, `<bloxreport:dataSourceConnection />`, `<bloxreport:groupHeader />`, `<bloxreport:groupFooter />`, or `<bloxreport:groupTotal />`.

Forgetting to include the style sheet

Without a defined style for each of the classes associated with the display of context menus and Report Style dialog box, the Report Editor user interface will not work properly. The Report Editor user interface will be chaotic, with the context menus popping up in the wrong places with incorrect font sizes.

Without a defined style for each of the classes associated with the display of the column headers, data, or break group footers, a report will not be as attractive or easy to read as it will be rendered as a plain HTML table.

Since style sheets cascade, as a best practice, use the style sheet provided out-of-the-box and then add your own style sheet (or use inline styles) to modify only the styles for classes you want to change:
```
<link rel="stylesheet" href="/AlphabloxServer/theme/report.css" />
<link rel="stylesheet" href="yourStyleSheet.css" />
```

Refreshed page doesn't reflect code modification

Blox tags and JSP statements within these tags are interpreted only the first time a page is loaded. Since the object on the server already exists for that session, it is reused and therefore will not reflect the changes you make. You will need to start a new session by opening a new instance of the browser.

Refer to member or column names incorrectly

When performing data manipulation tasks such as sorting, calculating, grouping, or filtering data, you should put member names in square brackets when they do not start with a letter or when they contain special characters (characters other than a-z, A-Z, 0-9, or underscores). This indicates to the DB2 Alphablox Relational Reporting engine that these are variables in the expression. Without square brackets, the reporting system may fail to correctly identify the members. When performing report layout formatting and styling tasks such as setting group headers and footers, or styling the data and column headers, you are dealing with the rendered report and square brackets are no longer required. This is because there is no confusion whether the text string is a member name or an expression.

Because of this difference in data manipulation versus report layout, you will notice that some Relational Reporting Blox have a `member` (or `members`) attribute while others have a `columnName` attribute. For example, GroupBlox has a members attribute, while sub tags inside TextBlox and StyleBlox have a `columnName` attribute. For more information, see "Columns and Members" on page 9, "Member Identifiers VS. Display Names" on page 19

Troubleshooting Tips

Messages from ReportBlox are sent as DEBUG level messages to the DB2 Alphablox server log. To examine the activities from ReportBlox and its supporting Blox, you can set the message level to DEBUG. However, keep in mind that the log file can grow large fairly quickly. For catching uncaught errors, you should create an error catching page and specify to have it handle error reporting in every JSP page you create. See "Use ErrorBlox for Better Error Reporting" on page 24.

The following are some general troubleshooting tips:

- Make sure the style sheet referenced actually exists and is in the correct location. Different browsers have different tolerance for missing style sheets.
- When you have the `interactive` attribute of the `<bloxreport:report>` tag set to `true`, if the interactive context menus in the Report Editor user interface do not show up or show up as lines of plain text in the beginning of your report, this is often because you do not have a style sheet associated with the report, or the style sheet referenced is not found. Add a link to your style sheet or to the style sheet supplied:

  `<link rel="stylesheet" href="/AlphabloxServer/theme/report.css" />`

- If the browser seems to hang while displaying a report, it is probably due to the size of the rendered report. Keep in mind that the report is rendered as an HTML table. A table with 10,000 rows and 10 columns can take several minutes for any browser to display.
- Member names are case sensitive. Referring to the member "Cost" as "COST" or "cost" may result in errors in some cases (such as CalculateBlox, OrderBlox, FilterBlox, SortBlox, and GroupBlox, which need to identify the exact members to act on) and may be ignored in others (such as FormatBlox or StyleBlox).

  Note: If you are using IBM® DB2 Universal Database™, Oracle, or Microsoft SQL Server, and you rename columns in the SELECT list, be sure to enclose the column names in double quotes if you expect the case to be preserved. For example:

  `SELECT FROM myTable total_sq_ft AS "Sq_Ft", sq_ft_pct AS "Pct"`

  Be sure to escape the quotation marks with back-slashes when you include them in your JSP page:

  `"SELECT FROM myTable total_sq_ft AS \"Sq_Ft\", sq_ft_pct AS \"Pct\""`

- Make sure that the error handling page specified in the `page` directive actually exists in the correct location. Otherwise, you will encounter an error complaining that the page cannot be found.

Error Handling Using ErrorBlox

By default, when a query returns no data, the message "No data" is displayed. If no data occurs further down the pipeline during data transformation due to an error, the default message displayed is "No data: An error occurred while generating report data." This can happen, for example, if you try to group or sort the data based on a non-existent member.

When ReportBlox's errors attribute is set to `false` (the default), exceptions are intercepted. You should supply an error handling page and use a try/catch block to catch errors or your users may see unfriendly error messages in cases such as bad queries or unavailability of the data source.

Sometimes you may need to check the exceptions in order to track down where the problem is. Exceptions thrown by Relational Reporting Blox are nested, and it can be difficult to identify the root cause. To iterate through the nested exceptions and get to the root cause:

1. Set the ReportBlox's errors tag attribute to `true` so the exceptions are not intercepted and you can catch them with a try/catch block.
2. Use the ErrorBlox's `getRootCause(Throwable` *exception*`)` method that will iterate through the nested exceptions of the specified exception and return the root cause exception. ErrorBlox also has a `getNextException(Throwable` *exception*`)` method that will return the next nested exception for the specified exception.

The following example shows how an exception is caught and the root cause is identified:

```
<%@ page import="com.alphablox.blox.*" %>
<%@ taglib uri="bloxreporttld" prefix="bloxreport"%>

<html>
<head>
  <title>Exception Test</title>
  <link rel="stylesheet" href="/AlphabloxServer/theme/report.css" />
</head>

<%
  String query="SELECT product_name, area, location From qcc WHERE
week_ending = '2000-04-08'";
%>

<body>
<%
  try { %>
    <bloxreport:report id="testTryCatch" errors="true">
      <bloxreport:sqlData query="<%= query%>" >
        <bloxreport:dataSourceConnection dataSourceName="qcc-rdb" />
      </bloxreport:sqlData>
    </bloxreport:report>
<% }
  catch ( Exception e ) {
    ErrorBlox eblox = new ErrorBlox();
    Throwable msg = eblox.getRootCause(e);
    out.println("<br><b> This Exception was captured</b><br> "+
msg.getMessage());
  }
%>
</body>
</html>
```

Chapter 12. Relational Reporting Blox Tag Reference

This section lists the tags associated with each Relational Reporting Blox. For each Blox, its associated tags, tag attributes, properties, methods, method syntax and usage are provided. For Java methods available to Relational Reporting Blox, see the Relational Reporting Javadoc, available from the Help link in DB2 Alphablox Admin Pages or from the DB2 Alphablox library page at http://www.ibm.com/software/data/db2/alphablox/library.html.

Using Blox Tags

The custom Blox tags for Relational Reporting work similarly to the tags for the other Blox in DB2 Alphablox:

- To use the custom tags for Relational Reporting Blox, the following line should be included in the beginning of your JSP files:

  `<%@ taglib uri="bloxreporttld" prefix="bloxreport" %>`

- The tags start with `<bloxreport:...>`, and ends with a corresponding ending tag `</bloxreport:...>`. If the tag does not involve nested tags, it can end with a slash (`<bloxreport:calculate ... />`) without a separate ending tag.

- The `<bloxreport:report>` tag is similar to the `<blox:present>` tag in that it is the wrapping tag for most all the other Relational Reporting Blox. The only exceptions are ErrorBlox, PdfBlox, and PersistenceBlox.

- Tags and attribute names always start with lowercase first word and an uppercase first letter for subsequent words (for example, `dataSourceConnection`).

A major different between the `<bloxreport:report>` tag and the `<blox:present>` tag is that Blox nested within the `<bloxreport:report>` tag can have their own `id` that you can script to, whereas Blox nested within the `<blox:present>` tag cannot. This is because Blox supporting the Relational Reporting feature are functionally distinctively decoupled.

For a list of all tags associated with ReportBlox., see "Relational Reporting Tags for Copy-and-Paste," on page 135.

CalculateBlox

CalculateBlox lets you add a calculated member to a report. For the calculated member, you specify a calculation expression and the position this new member should appear. The JSP tag for CalculateBlox is `<bloxreport:calculate>` .

Syntax

```
<bloxreport:report id="idName">
  ...
  <bloxreport:calculate id = "idName"
     expression="calculatedMemberName= calculationExpression"
     index = "int">
  </bloxreport:calculate>
  ...
</bloxreport:report>
```

Usage

You can use multiple `<bloxreport:calculate>` tags in a `<bloxreport:report>` tag. ReportBlox performs the calculations in the order they are specified. The expression consists of two parts: the name of the calculated member and the calculation expression. In the following example:

```
"[Total Sales] = [Unit Price] * [Units_Sold]"
```

"Total Sales" is the name of the calculated member, and its value is the product of `[Unit Price] * [Units_Sold]`. The square brackets (`[ ]`) are needed in the expression since there are spaces in the member name.

Note the following when you use CalculateBlox:

- If a member name already contains [ ], use an additional closing "]" to indicate the end of the member name. For example, to specify the member name `West[CA]` in an expression, you should say `[West[CA]]]`.
- Supported operators for calculation expressions are `+, -, *,` and `/`
- Supported separator for calculation operators are (). For example:

```
<bloxreport:calculate
    expression = "[Profit%] = Sales/(Unit_Cost * Units_Sold)"
/>
```

- Supported calculation functions are: `rank()`, `percentOfTotal()`, `runningTotal()`, and `runningCount()`. For example:

```
<bloxreport:calculate
    expression = "[% Total] = percentOfTotal(Units)"
/>
```

These functions are related to how the report is grouped, and are discussed in details in "Calculating Group-based Summary Columns" on page 62.

- Supported operators only work on numeric data. This includes integer, floating point, and currency. You will not be able to perform calculations on string, date, time, or boolean data types.

If any member used in the calculation involves missing or null values, the calculation will result in missing data.

Tag Description

The **`<bloxreport:calculate>`** Tag

Tag Attribute	Required	Default	Description
id	No		The unique identifier for this instance of CalculateBlox.
bloxName	No		The unique identifier for this instance of CalculateBlox on the server that allows you to dynamically set its name. See "Managing Session Scope" on page 81.
expression	No	Sum of all numeric columns in a new column named "Total"	The expression CalculateBlox evaluates to add the named calculated member.
			An invalid or empty expression results in an error.
			If a <bloxreport:calculate> tag is added without the expression attribute, CalculateBlox will automatically add a calculated member called "Total" with the sum of all column members as the value.
			For valid expression syntax, see "Expression Syntax" on page 18.
index	No	member count	The position in the Column dimension where the calculated member appears, with 0 being the first member.
			By default calculated members are added to the end.

Examples

```
<bloxreport:calculate
    expression = "[Profit%] = Sales/GrossMargin"
    index = "4"
/>
```

This tag adds a calculated member named Profit% as the fifth member, with Profit% deriving from Sales divided by GrossMargin.

See Also

"Adding Calculated Columns" on page 33, "Calculating Group-based Summary Columns" on page 62.

DataSourceConnectionBlox

DataSourceConnectionBlox represents a connection to a relational data source defined to DB2 Alphablox. The JSP tag for DataSourceConnectionBlox is `<bloxreport:dataSourceConnection>` . The tag for DataSourceConnectionBlox needs to be nested under `<bloxreport:sqlData>`. Nonetheless, it is a Blox and can be assigned an id for later references in your code.

Syntax

```
<bloxreport:report id = "idName">
   <bloxreport:sqlData
      <bloxreport:dataSourceConnection
         id = "dataSourceIdName"
         dataSourceName = "dataSourceName" >
      </bloxreport:dataSourceConnection>
   </bloxreport:sqlData>
</bloxreport:report>
```

Usage

Only one `<bloxreport:dataSourceConnection>` tag can be added in a `<bloxreport:sqlData>` tag. Only one `<bloxreport:sqlData>` tag can be added in a `<bloxreport:report>` tag.

Tag Description

The `<bloxreport:dataSourceConnection>` Tag

Tag Attribute	Required	Default	Description
id	No		The unique identifier for this instance of DataSourceConnectionBlox.
bloxName	No		The unique identifier for this instance of DataSourceConnectionBlox on the server that allows you to dynamically set its name. See "Managing Session Scope" on page 81.
dataSourceName	No		The name of the relational data source as defined via the DB2 Alphablox home pages. A non-relational data source results in an error.
userName	No		The username for accessing the data source specified.
password	No		The password for the username specified to access the data source. If password is an empty string, then `password=""`. If password is blank or null, do not specify a password.

Examples

```
<bloxreport:report id="profitReport">
   <bloxreport:sqlData>
      <bloxreport:dataSourceConnection
         id = "myDataSource"
         dataSourceName = "chocoblocks"
         userName = "sa"
         password = "allmighty">
      </bloxreport:dataSourceConnection>
   </bloxreport:sqlData>
</bloxreport:report>
```

The above code specifies a defined data source named "chocoblocks" with username "sa" and password "allmighty" as the SQL data source for an instance of ReportBlox named "profitReport."

ErrorBlox

ErrorBlox prints the details of errors in an HTML table. The JSP tag for ErrorBlox is `<bloxreport:error>`.

Syntax

```
<bloxreport:error id = "idName">
</bloxreport:error>
```

Usage

Use the `<bloxreport:error>` tag in your custom JSP error reporting page. To identify a JSP file as the error reporting page, add `<%@ page isErrorPage="true" %>` in the beginning of the file. Then in your regular JSP pages, use the following directive to point to your custom JSP error reporting page:

```
<%@ page errorPage="yourErrorPage.jsp" %>
```

See "Use ErrorBlox for Better Error Reporting" on page 24 for details on how to create an error reporting JSP. See "Error Handling Using ErrorBlox" on page 97 for more details on using ErrorBlox to catch exceptions.

Tag Description

The `<bloxreport:error>` Tag

Tag Attribute	Required	Default	Description
id	No		The unique identifier for this instance of ErrorBlox.
bloxName	No		The unique identifier for this instance of ErrorBlox on the server that allows you to dynamically set its name. See "Managing Session Scope" on page 81.

Examples

A custom JSP error page using ErrorBlox:

```
<%@ taglib uri="bloxreporttld" prefix="bloxreport" %>
<%@ page isErrorPage="true" %>

<html xmlns="http://www.w3.org/1999/xhtml">
   <head>
      <title>My JSP Error Reporting Page</title>
      <link rel="stylesheet" href="/AlphabloxServer/theme/report.css"
 type="text/css" />
   </head>

   <body>
      <H1>Error Reporting Page</H1>
      <bloxreport:error id="errorBlox" />
   </body>
</html>
```

FilterBlox

FilterBlox allows you to filter a numeric result set based on a specified expression. The JSP tag for FilterBlox is `<bloxreport:filter>`. Data filtered out are removed from the result set.

Syntax

```
<bloxreport:report id="idName">
  ...
  <bloxreport:filter id = "filterIdName"
      expression = "filterExpression" >
  </bloxreport:filter>
  ...
</bloxreport:report>
```

Usage

You can only filter on numeric data. You can have multiple `<bloxreport:filter>` tags in a `<bloxreport:report>` tag. Each FilterBlox takes exactly one filter expression. You cannot specify compound filters using AND or OR. Instead, you can chain multiple FilterBlox to achieve your desired outcome. Since the subsequent FilterBlox will filter the data based on the result of the previous FilterBlox, if you need to perform operations such as keeping only sales data that are either greater than 400 or smaller than 100, you need to prepare the data in your database environment before retrieving them into ReportBlox.

By default, missing data are not filtered out. To exclude missing data from the filtered result, add a separate filter using the `isMissing()` function (specify `not isMissing()` for negation). See examples below for more details.

For member names that contain characters other than a-z, A-Z, 0-9, and _, they should be enclosed in square brackets ([ ]). In cases where a member name may be mistaken for a number, also use [ ] to avoid confusion. If a member name already contains [or], use an additional closing "]" to indicate the end of the member name. For example, to specify the member name West[CA] in an expression, you should say [West[CA]]].

Tag Description

The `<bloxreport:filter>` Tag

Tag Attribute	Required	Default	Description
id	No		The unique identifier for this instance of FilterBlox.
bloxName	No		The unique identifier for this instance of FilterBlox on the server that allows you to dynamically set its name. See "Managing Session Scope" on page 81.

Tag Attribute	Required	Default	Description
expression	Yes		The criteria for filtering.
			If a value does not meet the condition specified in the expression, it is filtered out. Valid operators for filter expressions are =, <, >,and !=. For example:
			`expression = "Sales > 3000"` `expression = "[Product Code] != 200"`
			You cannot have calculation operators inside the filter expression (+, -, /, or * is not allowed), nor can you add methods or Java scriptlets inside the expression. An invalid expression results in an error.
			One function is available for filter expressions: isMissing(*memberName*).
			See "Expression Syntax" on page 18 for discussions on expression syntax and how to specify member names if the names contain special characters or spaces.

Examples

```
<bloxreport:report id="salesReport">
   ...
   <bloxreport:filter
      id = "filter1"
      expression = "Sales < 10000"
   />
   <bloxreport:filter
      id = "filter2"
      expression = "Sales > 0"
   />
   ...
</bloxreport:report>
```

The above block of code keeps rows of data where Sales is between 0 and 10000. Note that missing data will be returned. To exclude missing data, add another filter:

```
<bloxreport:filter
   id = "filter3"
   expression = "not isMissing(Sales)"
/>
```

FormatBlox

FormatBlox allows setting the display format for numeric, date, and missing data. Null data in the database is treated as missing data. If no format is specified:

- the default format type for the client locale is applied
- the default display text for missing or null data is an empty string

  The JSP tag for FormatBlox is `<bloxreport:format>`. Since FormatBlox handles data formatting for multiple data types and each data type has multiple attributes, a separate tag is required for each data type. In addition, it has a tag that allows you to specify whether HTML code can be added around the data returned from a data query. These tags are:

- `<bloxreport:numeric>`
- `<bloxreport:date>`
- `<bloxreport:missing>`
- `<bloxreport:html>`
- `<bloxreport:aggregation>`

The `<bloxreport:numeric>` tag applies to all numeric data types, including integer, floating points, and currency. The `<bloxreport:date>` tag applies to all data types inheriting from `java.util.Date`, and these include the `Date`, `Time`, and `Timestamp` subclasses. All formats conform to Java's format masks.

Syntax

```
<bloxreport:report id="SalesReport">
  ...
  <bloxreport:format>
    <bloxreport:numeric format = "formatExpression1" />
    <bloxreport:numeric format = "formatExpression2"
      member = "memberName" />
    <bloxreport:date format = "formatExpression1" />
    <bloxreport:date format = "formatExpression2"
      member = "memberName" />
    <bloxreport:missing format = "stringToDisplay" member =
      "memberName" />
    <bloxreport:html member = "memberName1" />
    <bloxreport:html member = "memberName2" />
    <bloxreport:aggregation member = "memberName2"
format="formatExpression" />
  </bloxreport:format>
 ...
</bloxreport:report>
```

Usage

You can add only one `<bloxreport:format>` tag in a `<bloxreport:report>` tag. Within the `<bloxreport:format>` tag, you can have multiple `<bloxreport:numeric>`, `<bloxreport:date>`, and `<bloxreport:missing>` tags. Note the following:

- Use Java format masks for the format expression.
- If a member is not specified, the format is applied to all data of that data type.
- If a member is specified, the specified format is applied only to that member.
- If formats are specified twice for the same member, the format specified last is applied.

For Java decimal format, see http://java.sun.com/j2se/1.4.2/docs/api/java/text/DecimalFormat.html. For Java

date and time format, see
http://java.sun.com/j2se/1.3/docs/api/java/text/SimpleDateFormat.html

FormatBlox does not handle report layout formatting. See Chapter 6, "Formatting the Report and Data," on page 39 for tasks related to report layout formatting. See StyleBlox on page 128 for tags that allow you to specify styles such as font size, background color, and text alignment.

Tag Description

The `<bloxreport:format>` Tag

Tag Attribute	Required	Default	Description
`<bloxreport:format>`			
`id`	No		The unique identifier for this instance of FormatBlox
`bloxName`	No		The unique identifier for this instance of FormatBlox on the server that allows you to dynamically set its name. See "Managing Session Scope" on page 81.
`<bloxreport:numeric>`, `<bloxreport:date>`, `<bloxreport:missing>`			
`format`	Yes		The format for the data type.
`member`	No		The member to apply the specified format.
`<bloxreport:html>`			
`member`	No		A member's display name. The data for the specified member, when returned from the SQL query, will preserve the added HTML code. By default, any HTML code added is encoded unless you specify not to by specifying the member whose data you want the HTML to be left alone. See "Adding HTML code to Data Returned from a Query" on page 47 for an example and more details on using a member's display name.
			You can have multiple `<bloxreport:html>` tags, but each can only have one member specified. To specify that all HTML code should be left alone for data returned for all members, set the value to an empty string as follows:
			`member=""`
`<bloxreport:aggregation>`			
`member`	No		A member's display name.
`format`	Yes		The format to apply to aggregation values (group totals) for the specified member. If `member` is not specified, the format is applied to all aggregation values.

Examples

```
<bloxreport:report id="SalesReport">
  ...
  <bloxreport:format>
    <bloxreport:numeric format="####.00;(####.00)" />
    <bloxreport:numeric format="$#,###.00;$(#,###.00)"     member="Sales"     />
    <bloxreport:aggregation member="Sales" format="$#,###;$(#,###)" />
    <bloxreport:date format="yyyy.MM.dd G 'at' hh:mm:ss z" />
    <bloxreport:date format="EEE, MMM d, ''yy" member="Date" />
    <bloxreport:missing format="Units Value Missing"
      member="Units" />
    <bloxreport:missing format="Sales Value Missing"
```

```
      member="Sales" />
  </bloxreport:format>
 ...
</bloxreport:report>
```

This code example sets:

- the default format for positive numeric data to "####.00"; the default format for negative numeric data to "(####.00)." For example, 1234.5 becomes 1234.50, and -1234.5 becomes (1234.50).
- the numeric data for member "Sales" to "$#,###.00;$(#,###.00)." For example, 1234.5 becomes $1, 234.50.
- The aggregation values for member "Sales" to "$#,###;$(#,###)." For example, 1234.5 becomes $1,235.
- the default format for dates to "yyyy.MM.dd G 'at' hh:mm:ss z". An example of this format is 2001.10.01 AD at 09:27:13 PDT.
- the date format for member "Date Member" to "EEE, MMM d, ''yy". An example of this format is Mon, October 1, '01.
- the text display when member "Units" contains missing data to "Units Value Missing"
- the text display when member "Sales" contains missing data to "Sales Value Missing"

GroupBlox

GroupBlox allows creation of break groups. The JSP tag for GroupBlox is
`<bloxreport:group>`. Since the aggregation summary for each member can be
different, a separate `<bloxreport:aggregation>` tag is needed to set the member
name and aggregation type for that member.

Syntax

```
<bloxreport:report id="idName">
   ...
   <bloxreport:group members="breakmember1, breakmember2, ...">
      <bloxreport:aggregation
         member="memberName"
         type="aggregationType">
      </bloxreport:aggregation>
   </bloxreport:group>
   ...
</bloxreport:report>
```

Usage

You can have multiple `<bloxreport:group>` tags within a `<bloxreport:report>` tag.
Within each `<bloxreport:group>` tag , you can have multiple
`<bloxreport:aggregation>` tags. The order of the break members you specify in the
`members` attribute determines the break group level. For example, the following tag:

```
<bloxreport:group
   members="Product, Country">
</bloxreport:group>
```

groups the report by Product first and then by Country, making the overall report
(for all products) level 1, individual product group at level 2, and individual
country group at level 3. This group level corresponds to the level you specify in
TextBlox's nested tags:

```
<bloxreport:text>
   <bloxreport:groupHeader level = "int"
      text="Some group header text here" />
</bloxreport:text>
```

or

```
<bloxreport:text>
   <bloxreport:groupFooter level = "int"
      text="Some group footer text here" />
</bloxreport:text>
```

To specify the styles such as font size, colors, and background colors for
breakgroup headers and footers, see "Styling the Relational Reports" on page 9
and "Styling Data Displayed in Report" on page 47.

Tag Description

The `<bloxreport:group>` Tag

Tag Attribute	Required	Default	Description
`<bloxreport:group>`			
`id`	No		The unique identifier for this instance of GroupBlox.

Tag Attribute	Required	Default	Description
bloxName	No		The unique identifier for this instance of GroupBlox on the server that allows you to dynamically set its name. See "Managing Session Scope" on page 81.
members	No		Members for grouping. To add aggregation data (`sum`, `average`, `count`, `max`, and `min`) for columns at the end of your report without break groups, set the value of `members` to an empty string (members = "").
`<bloxreport:aggregation>`			
member	Yes		The member to provide an aggregation value.
type	No	sum	Type of aggregation. Valid values are: • `sum` • `average` • `count` • `max` • `min` • `none` Note that: • The default aggregation type for all numeric data columns is `sum`. • If no type is specified, the default is `sum`. • Type `count` works on both string and numeric values. All others only work on numeric values. • If an aggregation is specified on an invalid data type, it is ignored. • If the member to provide an aggregation value is not found, it is ignored. • If there are missing data, they are excluded from the aggregation. That is, they are not included in the data count or any other types of aggregation.

Examples

```
<bloxreport:group members = "Product">
   <bloxreport:aggregation member = "Sales" type = "sum" />
   <bloxreport:aggregation member = "Units" type = "average" />
</bloxreport:group>
```

The above code creates a break group by Product, and for aggregation values in the summary row, shows the count of number of States and the average for Units. For all other members whose aggregation type is not specified, their totals will be displayed.

MembersBlox

MembersBlox allows you to specify which members to include in or exclude from a report. Excluded members no longer exist in the result set. The JSP tag for MembersBlox is `<bloxreport:members>`.

Syntax

```
<bloxreport:report id = "idName">
  ...
   <bloxreport:members id = "membersIdName"
      excluded = "member1, member2,..."
      included = "member1, member2,..." >
   </bloxreport:members>
  ...
</bloxreport:report>
```

Usage

You can add multiple `<bloxreport:members>` tags in a `<bloxreport:report>` tag. Each MembersBlox takes only either the `excluded` or the `included` attribute. If you specify both attributes within one `<bloxreport:members>` tag, the last attribute will be accepted and the earlier one will be ignored.

Excluded members are permanently removed from the result set. To temporarily hide a member or members, use OrderBlox on page 115.

Tag Description

The `<bloxreport:members>` Tag

Tag Attribute	Required	Default	Description
excluded	No		Members to be excluded. Excluded members are no longer in the result set. If the member names do not start with a letter or contain spaces or special characters, they need to be enclosed in square brackets. See "Member Identifiers VS. Display Names" on page 19 for detail.
id	No		The unique identifier for this instance of MembersBlox.
bloxName	No		The unique identifier for this instance of MembersBlox on the server that allows you to dynamically set its name. See "Managing Session Scope" on page 81.
included	No		Members to be included. Members not in the list are excluded. If the member names do not start with a letter or contain spaces or special characters, they need to be enclosed in square brackets. See "Member Identifiers VS. Display Names" on page 19 for detail.

Examples

```
<bloxreport:report id = "ProfitReport">
   <bloxreport:members
      id = "members1"
      excluded = "[Unit Cost], [Unit Price]"
   />
</bloxreport:report>
```

The above example specifies that the Unit Cost and Unit Price members are not to be included in the report. Both are enclosed in square brackets as the names contain spaces.

See Also

OrderBlox on page 115

OrderBlox

OrderBlox allows you to specify the order from left to right in which the members are returned. It also allows you to temporarily hide members. The JSP tag for OrderBlox is `<bloxreport:order>`.

Syntax

```
<bloxreport:report id = "idName">
  ...
  <bloxreport:order id = "orderIdName"
     excluded = "member1, member2,..."
     included = "member1, member2,..." >
  </bloxreport:order>
  ...
</bloxreport:report>
```

Usage

You can add multiple `<bloxreport:order>` tags in a `<bloxreport:report>` tag. Each OrderBlox takes only either the `excluded` or the `included` attribute. If you specify both attributes within one `<bloxreport:order>` tag, the last one will be accepted and the earlier one will be ignored.

The `excluded` attribute allows you to temporarily hide members. Excluded members do not display in the rendered report but still exist in the result set. In an interactive report (`interactive = "true"`), when users choose to **Show All** from the interactive context menu, hidden members will show. To permanently hide a member or members, use MembersBlox on page 113.

Tag Description

The `<bloxreport:order>` Tag

Tag Attribute	Required	Default	Description
excluded	No		Members to be excluded in the ordering.
			Unlike excluded members via MembersBlox, excluded members are still in the result set, available for subsequent data transformation actions in the pipeline.
			If the member names do not start with a letter or contain spaces or special characters, they need to be enclosed in square brackets. See "Member Identifiers VS. Display Names" on page 19 for detail.
id	No		The unique identifier for this instance of OrderBlox.
bloxName	No		The unique identifier for this instance of OrderBlox on the server that allows you to dynamically set its name. See "Managing Session Scope" on page 81.
included	No		Members to be included in the order from left to right.
			If the member names do not start with a letter or contain spaces or special characters, they need to be enclosed in square brackets. See "Member Identifiers VS. Display Names" on page 19 for detail.

Examples

```
<bloxreport:report id = "ProfitReport">
   <bloxreport:order
      id = "order1"
      excluded = "[Unit Cost], [Unit Price]"
   />
</bloxreport:report>
```

The above example specifies that the Unit Cost and Unit Price members are not to be displayed in the report. Both are enclosed in square brackets as the names contain spaces. In an interactive report, users can see these members by choosing to **Show All** via the Column Header Context Menu.

See Also

MembersBlox on page 113

PdfBlox

PdfBlox allows you to send a relational report to PDF . The JSP tag for PdfBlox is `<bloxreport:pdf>`.

Syntax

```
<bloxreport:pdf id = "idName"
    [attributeN = "valueN"] >
</bloxreport:pdf>
```

Usage

You can have only one `<bloxreport:pdf>` tag in a JSP page. Unlike other Blox, an id is required. The `<bloxreport:pdf>` tag can be added outside the `<bloxreport:report>` tag to directly send a report to PDF with live data. It can also stand alone in a JSP page to send an instance of a ReportBlox to PDF.

Tag Description

The `<bloxreport:pdf>` Tag

Tag Attribute	Required	Default	Description
id	Yes		The unique identifier for this instance of PdfBlox.
bloxName	No		The unique identifier for this instance of PdfBlox on the server that allows you to dynamically set its name. See "Managing Session Scope" on page 81.
bottom	No	1 in	Specifies the bottom margin. Valid units include pixels (px), points (pt), inches (in), millimeters (mm), and centimeters (cm). If not specified, pixels (px) will be used.
footerHeight	No	30 px	Specifies the height of the footer. Valid units include pixels (px), points (pt), inches (in), millimeters (mm), and centimeters (cm). If not specified, pixels (px) will be used.
footerText	No		The footer, including text and layout, defined using XHTML. For example: `headerText="<span style='color:blue'>Company confidential</span>"`
footerVisible	No	true	Specifies if the footer (page numbers) should be visible. The default is true.
headerHeight	No	50 px	Specifies the height of the header. Valid units include pixels (px), points (pt), inches (in), millimeters (mm), and centimeters (cm). If not specified, pixels (px) will be used.
headerText	No		The header, including text and layout, defined using XHTML. For example, `headerText="<span style='color:blue'>Annual Report</span>"`
headerVisible	No	true	Specifies if the header (logo, time, and date) should be visible. The default is true.
height	No	1 1in	Specifies the height of the page. Valid units include pixels (px), points (pt), inches (in), millimeters (mm), and centimeters (cm). If not specified, pixels (px) will be used.
left	No	1.25 in	Specifies the left margin. Valid units include pixels (px), points (pt), inches (in), millimeters (mm), and centimeters (cm). If not specified, pixels (px) will be used.

Tag Attribute	Required	Default	Description
logoSource	No		Specifies the logo to use when the report is rendered to PDF. Images added in the JSP file using the HTML tag `<img src="...">` are not included in the PDF render. You must use this attribute to explicitly specify the source image URL. The URL must be one of the following formats: • A complete URL using the file protocol with an absolute path. For example, on a Windows system: `file:///C:\\alphablox\\webapps\\MyApp\\background.gif` • A relative URL to your DB2 Alphablox installation. For example, if the image `background.gif` is stored in your DB2 Alphablox Repository inside the *images/* directory, `logoSource` should be set to `/repository/images/background.gif`.
portrait	No	true	Specifies if the page should be rendered in portrait. `False` will result in the page rendered in landscape. The default is `true`, with the report rendered in portrait.
report	No		Specifies the instance of ReportBlox (the ReportBlox `id`) to render to PDF.
right	No	1.25 in	Specifies the right margin. Valid units include pixels (px), points (pt), inches (in), millimeters (mm), and centimeters (cm). If not specified, pixels (px) will be used.
top	No	1 in	Specifies the top margin. Valid units include pixels (px), points (pt), inches (in), millimeters (mm), and centimeters (cm). If not specified, pixels (px) will be used.
width	No	8.5 in	Specifies the width of the page. Valid units include pixels (px), points (pt), inches (in), millimeters (mm), and centimeters (cm). If not specified, pixels (px) will be used.

Examples

```
<!---Content of toPDF.jsp---->
<%@ taglib uri="bloxreporttld" prefix="bloxreport" %>
<bloxreport:pdf id="myPDF"
    report='<%= request.getParameter("report") %>' />
```

The above example shows a simple yet complete JSP page that is called when a user clicks on a button or link on another JSP page containing a ReportBlox This JSP page may look like the following:

```
<!---Content of myReport.jsp---->
<%@ taglib uri="bloxreporttld" prefix="bloxreport" %>
<html>
<body>
<a href="toPDF.jsp?report=myReport1">Send to PDF</a>
<bloxreport:report id="myReport1" interactive="true">

    ...
</bloxreport:report>
</body>
</html>
```

PersistenceBlox

PersistenceBlox allows you to save the state of a ReportBlox to the Alphablox Repository for later retrieval under the location and name specified. The JSP tag for PersistenceBlox is `<bloxreport:persistence>`.

Syntax

```
<bloxreport:persistence id = "idName"
    targetBloxId="ReportBloxId"
    location="location"
    persistedName="bookmarkName"
    operation="operation"
/>
```

Usage

There can be multiple `<bloxreport:persistence>` tags on a JSP page. Before a bookmark is saved for a specified ReportBlox, set the ReportBlox' `interactive` attribute to `false`. If a bookmark is saved on an interactive ReportBlox, the report will not display properly when the bookmark is loaded from the Repository. When the bookmark is retrieved, the required data source needs to be accessible. All bookmarks are saved under the `reportingpersistence/` folder under the Repository.

Tag Description

The `<bloxreport:persistence>` Tag

Tag Attribute	Required	Default	Description
`id`	No		The unique identifier for this instance of PersistenceBlox.
`bloxName`	No		The unique identifier for this instance of PersistenceBlox on the server that allows you to dynamically set its name. See "Managing Session Scope" on page 81.
`location`	Yes		The folder name under the repository's `reportingpersistence/` folder. You can specify sub-folders such as "sales/east", which creates an `east/` folder under `sales/` under the `reportingpersistence/` folder in the repository if they do not already exist.
`operation`	Yes		The operation to perform on the bookmark specified (via the `persistedName` attribute). Valid values are `save` and `load`.
`persistedName`	Yes		The name of the bookmark.
`targetBloxId`			The `id` of the ReportBlox to bookmark. This attribute is required when the operation is `save`.

Examples

```
<bloxreport:persistence id = "myBookmark"
    targetBloxId="MyReport"
    location="sales/east"
    persistedName="salesApr02"
    operation="save"
/>
```

The above example saves the state of a previous defined ReportBlox whose `id` is "MyReport" into the Repository:

`<alphablox_dir>/repository/reportingpersistence/sales/east/salesApr02`

where `<alphablox_dir>` is the Alphablox installation directory.

RDBResultSetDataBlox

The RDBResultSetDataBlox allows you to use the RDBResultSet object from a DataBlox and generate a relational report using Relational Reporting Blox.

Syntax

```
<bloxreport:rdbResultSetData id="id"
    bloxName="bloxName"
    bloxRef= "DataBloxName"
    columnCoordinate="col"
    drillThroughReportName="reportName"
    rowCoordinate="row"
/>
```

Usage

Only one `<bloxreport:rdbResultSetData>` tag can be added in a `<bloxreport:report>` tag. You have to specify an existing DataBlox using its `bloxRef` attribute. If column and row coordinates are not specified, the `getResultSet()` method on the DataBlox referenced will be called. If the column and row coordinates are specified, the `drillThrough()` method on the DataBlox referenced is called. For DB2 OLAP Server or Hyperion Essbase drill-through, since there may be multiple reports defined for a cell, use `drillThroughReportName` to specify the report of interest.

Tag Description

The **`<bloxreport:rdbResultSetData>`** Tag

Tag Attribute	Required	Default	Description
id	No		The unique identifier for this instance of RDBResultSetDataBlox.
bloxName	No		The unique identifier for this instance of RDBResultSetDataBlox on the server that allows you to dynamically set its name. See "Managing Session Scope" on page 81.
bloxRef	Yes		The name of an existing DataBlox.
columnCoordinate	No		The column coordinate of the specified cell.
drillThroughReportName	No		The name of the DB2 OLAP Server or Essbase drill-through report.
rowCoordinate	No		The row coordinate of the specified cell.

Examples

In the following example, another JSP containing a DataBlox called "myDataBlox" is referenced in this JSP:

```
<bloxreport:rdbResultSetData
    bloxRef="myDataBlox"
    columnCoordinate="<%= request.getParameter(\"colIndex\") %>"
    rowCoordinate="<%= request.getParameter(\"rowIndex\") %>"
/>
```

The `colIndex` and `rowIndex` parameters are passed in from a JavaScript function or a Java scriptlet, depending on the rendering mode of the PresentBlox or GridBlox. For more details, see "Using RDBResultSetDataBlox to Access RDBResultSet from DataBlox" on page 28.

See Also

The drill-through example in Blox Sampler's Retrieving Data section (MSAS version) and the Retrieving Data chapter in the *Developer's Guide*.

ReportBlox

ReportBlox generates a report in an HTML table. The JSP tag for ReportBlox is `<bloxreport:report>`.

When the `interactive` attribute is set to `true`, the Report Editor user interface is turned on. With mouse over a column header, a break group header, or a break group footer, a context menu automatically shows up and allows users to dynamically editing the reports. See "Styling the Relational Reports" on page 9 for descriptions on these context menus.

Syntax

```
<bloxreport:report id="reportID">
   ...
</bloxreport:report>
```

Usage

You can have multiple `<bloxreport:report>` tags within a JSP file. Each report can have only one SQLDataBlox on page 127 and one DataSourceConnectionBlox on page 103. Or it can take an RDBResultSetDataBlox on page 121, which allows you to access an RDBResultSet object from a DataBlox. It is a good practice to specify an `id` for each instance of ReportBlox.

Tag Description

The `<bloxreport:report>` Tag

Tag Attribute	Required	Default	Description
`<bloxreport:report>`			
id	No		The unique identifier for this instance of ReportBlox if `bloxName` is not specified. If `bloxName` is specified, `id` is the local Java scripting variable name. For details, see the `bloxName` entry in *Developer's Reference*.
bloxName	No		The unique identifier for this instance of ReportBlox on the server that allows you to dynamically set its name. See "Managing Session Scope" on page 81 and the `bloxName` entry in *Developer's Reference*.
errors	No	false	Intercepts exceptions thrown. By default, when the data query returns no data, the text "No data" is displayed. If no data occurs as a result of data transformation such as grouping or calculating data on a non existent member, the text "No data: An error occurred while generating report data" is displayed. These messages can be customized using the `noDataMessage` and `noDataDueToErrorMessage` attributes. With this default behavior, your users see a more graceful message rather than an exception. However, in some cases you may want to check the exception in order to track the problem. In this case, set this attribute is set to `true`. See "Error Handling Using ErrorBlox" on page 97 for more information on tracking the root cause of an exception.

Tag Attribute	Required	Default	Description
interactive	No	false	true to render the report in DHTML table for interactive report editing. The default value is false, that is, there will not be interactive context menus for users. When interactive is set to true, a style sheet defining the styles for classes associated with the interactive menu and columns needs to be provided in order for the Report Editor user interface to work appropriately. See "Styling the Relational Reports" on page 9 for details on styles.
noDataMessage	No	No data	The message displayed in the rendered report when the ReportBlox returns no data. Note that errors attribute needs to be set to false (the default) or exceptions will be displayed.
noDataDueToErrorMessage	No	An error occurred while generating report data	The message displayed in the rendered report when errors occur during data transformation in the pipeline. When the ReportBlox returns no data due to such an exception, users will see both noDataMessage and noDataDueToErrorMessage in one string: "No data: An error occurred while generating report data."
noDataDueToErrorMessage	No	An error occurred while generating report data	The message displayed in the rendered report when errors occur during data transformation in the pipeline. When the ReportBlox returns no data due to such an exception, users will see both noDataMessage and noDataDueToErrorMessage in one string: "No data: An error occurred while generating report data."
noDataDueToErrorMessage	No	An error occurred while generating report data	The message displayed in the rendered report when errors occur during data transformation in the pipeline. When the ReportBlox returns no data due to such an exception, users will see both noDataMessage and noDataDueToErrorMessage in one string: "No data: An error occurred while generating report data."
readerMode	No	false	true to turn off the **Style...** menu options in the interactive menus. This reader mode is more suitable for use with screen reader programs because some screen readers may not be able to set their focus on the dialog to read the text. In addition, the styling options offered in the **Report Style** dialog are not useful to users with limited vision.

SortBlox

SortBlox allows sorting on the specified members in either ascending or descending order. The tag for SortBlox is `<bloxreport:sort>`. SortBlox supports compound sorting with nested `<bloxreport:rule>` tag for specification of each sort rule.

Syntax

```
<bloxreport:report id="reportID">
   ...
   <bloxreport:sort member="memberName"
      ascending="boolean"
      missingLast="boolean" />
   ...
</bloxreport:report>
```

For compound sorting with multiple rules:

```
<bloxreport:report id="reportID">
   ...
   <bloxreport:sort>
      <bloxreport:rule
         member="sortMemberName1"
         ascending="boolean"
         missingLast="boolean" />
      <bloxreport:rule member="sortMemberName2" />
   </bloxreport:sort>
   ...
</bloxreport:report>
```

Usage

You can sort on numeric data, strings, dates, and time. SortBlox will sort based on the sequence the rules are specified. If multiple SortBlox are added, the later operation will not retain the earlier sort operations. By default, the data will be sorted in ascending order, with missing data displayed last.

Tag Description

The `<bloxreport:sort>` Tag

Tag Attribute	Required	Default	Description
`<bloxreport:sort>`			
id	No		The unique identifier for this instance of SortBlox.
bloxName	No		The unique identifier for this instance of SortBlox ton the server that allows you to dynamically set its name. See "Managing Session Scope" on page 81.
`<bloxreport:sort>`, `<bloxreport:rule>`			
ascending	No	true	Whether to sort in ascending order.
member	Yes		Name of the member to sort on. You can specify only one member name with each rule.
			If the member name does not start with a letter or contain spaces or special characters, it needs to be enclosed in square brackets. See "Member Identifiers VS. Display Names" on page 19 for detail.
missingLast	No	true	Whether missing values will be displayed last.

Examples

```
<bloxreport:report id="MarketReport">
   ...
   <bloxreport:sort id="sort1">
      <bloxreport:rule
         member="Area"
         missingLast = "false"
         ascending = "false" />
      <bloxreport:rule
         member="Location"
         missingLast = "false"
         ascending = "false" />
   </bloxreport:sort>
   ...
</bloxreport:report>
```

The above example will sort the data based on Area and then Location, generating a result like the following:

Area	Location	Product	Units	Cost	Sales
S. Cal	Beverly Hills	Truffles	#Missing	#Missing	#Missing
S. Cal	Beverly Hills	Brittles	72	120	240
N. Cal	Sonoma	Truffles	#Missing	#Missing	#Missing
N. Cal	Sonoma	Brittles	27	45	90
N. Cal	Napa	Brittles	48	80	160

SQLDataBlox

SQLDataBlox represents a SQL query that can be executed against DataSourceConnectionBlox.

Syntax

```
<bloxreport:report id="profitReport">
   <bloxreport:sqlData
    id = "idName"
    query = "sqlCommand" >
       <bloxreport:dataSourceConnection >
          ...
       </bloxreport:dataSourceConnection>
   </bloxreport:sqlData>
</bloxreport:report>
```

Usage

Only one <bloxreport:sqlData> tag can be added within a <bloxreport:report> tag.

Tag Description

The **<bloxreport:sqlData>** Tag

Tag Attribute	Required	Default	Description
id	No		The unique identifier of this instance of SQLDataBlox.
bloxName	No		The unique identifier for this instance of SQLDataBlox on the server that allows you to dynamically set its name. See "Managing Session Scope" on page 81.
query	Yes		The SQL command to extract data from the relational data source. The data source should be specified using the <bloxreport:dataSourceConnection> tag, a tag that nests within <bloxreport:sqlData>.

Examples

```
<bloxreport:report id="profitReport">
   <bloxreport:sqlData
    id = "dataquery1"
    query = "select Product, UnitSold, UnitCost, Sales from
       chocoblocks" >

       <bloxreport:dataSourceConnection
          dataSourceName="chocoblocks" />
   </bloxreport:sqlData>
</bloxreport:report>
```

Below is a sample output:

Product	UnitSold	UnitCost	Sales
Brittles	x	x	651,345
Fudge	x	x	454,450
Truffles	x	x	450,000
Seasonal	x	x	1,004,045

StyleBlox

StyleBlox lets you set the styles for the displayed data and various areas in the report. You can set styles for data based on data type, for missing or negative data, and for the different areas in a report—column headers, data, group headers, group footers, and group totals. The tag for StyleBlox is `<bloxreport:style>`. It has the following nested tags:

- `<bloxreport:text>`: styles will be applied to data in all text columns
- `<bloxreport:numeric>`: styles will be applied to data in all numeric columns
- `<bloxreport:date>`: styles will be applied to data in all date columns
- `<bloxreport:banding>`: styles will be applied to alternate data rows
- `<bloxreport:missing>`: styles will be applied to missing data
- `<bloxreport:negative>`: styles will be applied to all negative values
- `<bloxreport:column>`: styles will be applied to both the column header and the data for the specified member; member specification is required.
- `<bloxreport:data>`: styles will be applied to all data in the report, unless a column is specified.
- `<bloxreport:columnHeader>`: styles will be applied to all column headers in the report, unless a column is specified
- `<bloxreport:groupHeader>`: styles will be applied to all group headers in the report, unless a level is specified
- `<bloxreport:groupFooter>`: styles will be applied to all group footers in the report, unless a level is specified
- `<bloxreport:groupTotal>`: styles will be applied to all group totals in the report, unless a level is specified

Syntax

```
<bloxreport:style>
   <bloxreport:text style="yourStyle" />
   <bloxreport:numeric style="yourStyle" />
   <bloxreport:date style="yourStyle" />
   <bloxreport:banding style="yourStyle" />
   <bloxreport:missing style="yourStyle" />
   <bloxreport:negative style="yourStyle" />
   <bloxreport:column style="yourStyle" columnName="member" />
   <bloxreport:data style="yourStyle" columnName="member" />
   <bloxreport:columnHeader style="yourStyle" columnName="member" /
   <bloxreport:groupHeader style="yourStyle" level="level" />
   <bloxreport:groupFooter style="yourStyle" level="level" />
   <bloxreport:groupTotal style="yourStyle" level="level" />
</bloxreport:style>
```

Usage

Only one StyleBlox can be added to a `<bloxreport:report>` tag. If you have multiple `<bloxreport:style>` tags, only the last one will be applied. If different styles are specified to the same element within the same `<bloxreport:style>` tag, the style declared last will be applied.

Styles set through StyleBlox win over the styles set in style classes. For example, if you specify the font color for the `.data` style class to be blue in the style sheet, and use the StyleBlox to set all data in text columns to be black, then the data will be in black.

Tag Description

<bloxreport:style> Tag and Its Sub Tags

Tag Attribute	Required	Default	Description
<bloxreport:style>			
id	No		The unique identifier for this instance of StyleBlox.
bloxName	No		The unique identifier for this instance of StyleBlox on the server that allows you to dynamically set its name. See "Managing Session Scope" on page 81.
Nested text, negative, banding Tags			
style	Yes		The style to apply. Specify the style using CSS attributes in the format of attribute: value pair within the quotes, separated with ";".
Nested column, numeric, date, missing Tags			
style	Yes		The style to apply. Specify the style using CSS attributes in the format of attribute: value pair within the quotes, separated with ";".
columnName	Yes for the column tag; No for date, missing, and numeric;		The name of the column to which this style should be applied. For the <bloxreport:column> tag, a columnName has to be specified. The specified style will be applied to both the data and the column header for the specified column. For the other tags, if a columnName is not specified, the specified style will be applied to all columns.
Nested columnHeader, data, groupHeader, groupFooter, groupTotal Tags			
style	Yes		The style to apply. Specify the style using CSS attributes in the format of attribute: value pair within the quotes, separated with ";".
columnName	No		Applies to the data and columnHeader tags. If column name is not specified, the style will be applied to all data or all column headers.
level	Yes		Applies only to the groupHeader, groupFooter, and groupTotal tags.

Tip: The five sub tags inside StyleBlox can also be used within the TextBlox for setting the display text for each of the report areas. When nested inside the TextBlox, the text attribute should be used to set the display text. When nested inside the StyleBlox, the style attribute should be used to set the display style. Use of the style attribute inside TextBlox will be ignored, and use of the text attribute in the StyleBlox will be ignored.

Examples

```
<bloxreport:style>
  <bloxreport:text style="text-align: right;" />
  <bloxreport:numeric style="text-align: right;" />
  <bloxreport:date style="text-align: left;" />
  <bloxreport:banding style="background-color: #CCCCCC;" />
  <bloxreport:missing style="background-color: aqua;" />
  <bloxreport:negative style="background-color: red;" />
  <bloxreport:column style="font-size: 85%; color: white;
```

```
      background-color: gray;" member="Type" />
  <bloxreport:column style="color: purple; " member="Country"/>
  <bloxreport:column style="color: blue;" member="State"/>
</bloxreport:style>
```

The above example sets:
- all text columns to be right-aligned
- all numeric columns to be right-aligned
- all date columns to be left-aligned
- the background color for alternate data rows to #CCCCCC (pale grey)
- the background color for the cell containing missing data to aqua
- the background color for the cell containing negative data to red
- the background color for the Type column to gray, with text in white, at 85% of the regular text size
- the text color for the Country column (both column header and data) to purple
- the text color for the State member (both column header and data) to red

TextBlox

TextBlox lets you specify the display text for the five areas in a rendered report: group headers, group footers, group totals, column headers, and data. It has the following nested tags:

- `<bloxreport:groupHeader>`: text is applied to all group headers, or the group header of a specified level
- `<bloxreport:groupFooter>`: text is applied to all group footers, or the group footer of a specified level
- `<bloxreport:groupTotal>`: text is applied to all group totals, or the group totals of a specified level
- `<bloxreport:columnHeader>`: text is applied to all column headers, or the column header of a named column
- `<bloxreport:data>`: text is applied to all data cells, or data cells of a named column

Within each break group, two substitution variables are available for extracting the name of the column or break group and the aggregation value for the column:

- `<member/>`: This is valid inside the `<bloxreport:groupHeader>` tag for specifying break group headers, the `<bloxreport:groupFooter>` tag for specify break group footers, and the `<bloxreport:columnHeader>` tag in the for specifying column headers.
- `<value/>`: This is valid inside the nested `groupHeader` and `groupFooter` tag for extracting the group total of a named member, the nested `groupTotal` tag for specifying break group totals, and the nested `data` tag for specifying data values.

Syntax

```
<bloxreport:text>
  <bloxreport:groupHeader
    level="level"
    text="text for group header" />
  <bloxreport:groupFooter
    level="level"
    text="text for group footer" />
  <bloxreport:columnHeader
    columnName="columnName"
    text="text for the column header" />
  <bloxreport:data
    columnName="columnName"
    text="text for data in the column" />
  <bloxreport:groupTotal level="level"
    text="text for group totals" />
<bloxreport:text>
```

Usage

Only one TextBlox can be added inside of a `<bloxreport:report>` tag. If you have multiple `<bloxreport:text>` tags, only the last one will be applied. If different texts are specified to the same element within the same `<bloxreport:text>` tag, the text declared last will be applied.

Tag Description

Nested Tags Inside `<bloxreport:text>`

The following table describes TextBlox's sub tags and their attributes.

Attribute	Required	Default	Description
Nested `columnHeader` and `data` Tags			
`columnName`	Yes for `columnHeader`; No for `data`		The name of the column this column header text or data text should be applied to.
`text`	Yes	`<member/>` for `columnHeader`; `<value/>` for `data`	The text to display for the named column, or for all columns if `columnName` is not specified. If `text` is not specified, the entire `columnHeader` and `data` tags will be ignored. The entire value of this attribute is sent to the browser, allowing you to add custom HTML code to the column headers and data values. The only things processed by the DB2 Alphablox Relational Reporting engine are the two substitution variables—`<member/>` and `<value/>`. See "Special Substitution Variables for Displaying Member Names and Values" on page 52.
Nested `groupFooter`, `groupHeader`, and `groupTotal` Tags			
`columnName`	No		Applies to the `groupTotal` tag only. The name of the column this column header text should be applied to.
`level`	No		The level this group footer, header, or total text or style should be applied to, with level 1 being the overall report level (grouping is added using GroupBlox). With each grouping added, a level is added. In a report grouped by product and then by country, for example, level 1 represents the overall report for all products; level 2, each product break group; level 3, each country break group. If `level` is not specified, the specified footer, header, or total text and style will be applied to all grouping levels.
`text`	Yes	`<member/>` for `groupHeader` and `groupFooter`; `<value/>` for `groupTotal`	The text to display for the group footer, header, or total. If `level` is not specified, the specified text will be applied to all grouping levels. The entire value of this attribute is sent to the browser, allowing you to add custom HTML code to the column headers and data values. The only things processed by the DB2 Alphablox Relational Reporting engine are the two substitution variables—`<member/>` and `<value/>`. See "Special Substitution Variables for Displaying Member Names and Values" on page 52.

Tip: The five sub tags inside TextBlox can also be used within the StyleBlox for setting the styles for each of the report areas. When nested inside the TextBlox, the `text` attribute should be used to set the display text. When nested inside the StyleBlox, the `style` attribute should be used to set the

display style. Use of the `style` attribute inside TextBlox will be ignored, and use of the `text` attribute in the StyleBlox will be ignored.

The `<member/>` and `<value/>` Substitution Variables

The DB2 Alphablox Relational Reporting engine, when encountering the `<member/>` variable, will replace the variable with the name of the current member, or the grouping member of a specified, higher level. This variable allows you to extract the member name in a report's group headers, group footers, and column headers area and wrap HTML code around the member name. When used inside the `groupHeader` and `groupFooter` tags, it is replaced by the break group member name. When used inside the `columnHeader` tag, it is replaced by the column name.

The `<member/>` variable has one attribute:

Tag Attribute	Required	Default	Description
`level`	No	The current grouping level, as specified in the `groupHeader`, `groupFooter`, or `groupTotal` tag	You can only reference the member of the current grouping level or at a higher level. For example, a level 2 group header can only reference level 2 and level 1 group members, not level 3. If the level is set to 3, the entire tag string will be treated as texts and no variable substitution will occur. If no level is specified, the current level is implied

When the DB2 Alphablox Relational Reporting engine encounters the `<value/>` variable, it substitutes the variable with the value of the current column or a named column. This variable is valid inside the TextBlox's `data`, `groupTotal`, `groupFooter` and `groupHeader` tags, allowing you to wrap HTML code around the values. When used inside the `groupTotal` tag, it is replaced by the aggregation value for the current column for the break group, or the aggregation value for the named column for the break group. When used inside the `groupHeader` tag, it is replaced by the aggregation value of the named column for the same grouping level.

The `<value/>` variable has one attribute:

Tag Attribute	Required	Default	Description
`member`	No for `groupTotal` and `data`; yes for `groupHeader`	The current member in the column	If the member name is not recognized or does not exist, the entire tag string will be treated as texts and no variable substitution will occur.

For more discussions and examples, see "Special Substitution Variables for Displaying Member Names and Values" on page 52.

Examples

```
<bloxreport:report id="SummaryReport">
  ...
  <bloxreport:text>
    <bloxreport:groupHeader level="1"
      text="Profitability by <member/>" />
    <bloxreport:groupHeader level="2"
      text="<member level=\"1\"/>: <member/>" />
    <bloxreport:columnHeader
      columnName="Units"
```

```
                            text="Units Sold" />
          <bloxreport:groupTotal level="1"
            columnName="Cost"
            text="Grand Total Cost: <value/>" />
          <bloxreport:groupTotal level="1"
            columnName="Sales"
            text="Grand Total Sales: <value/>" />
          <bloxreport:groupTotal level="1"
            columnName="Units"
            text="Overall Total Units: <value/>" />

          <bloxreport:groupTotal level="2"
            columnName="Cost"
            text="total: <value/>" />
          <bloxreport:groupTotal level="2"
            columnName="Sales"
            text="total: <value/>" />
          <bloxreport:groupTotal level="2"
            columnName="Units"
            text="total: <value/>" />
      </bloxreport:text>
    </bloxreport:report>
```

Appendix. Relational Reporting Tags for Copy-and-Paste

This section includes two lists of all tags associated with ReportBlox. The first list include all tags nested within the `<bloxreport:report>` tag. The second list include the tags for PdfBlox. Tag names that represent Blox are bold, indicating you can assign an `id` to that instance of Blox.

- "All Tags Nested Within <bloxreport:report>" on page 135
- "All Tag Attributes for PdfBlox" on page 137

All Tags Nested Within <bloxreport:report>

```
<bloxreport:report id="reportId"
   bloxName="bloxName"
   errors="true|false"
   interactive="true|false"
   noDataMessage="messageText"
   noDataDueToErrorMessage="messageText"
   readerMode="true|false" >

   <bloxreport:sqlData
      query = "sqlCommand" >
      <bloxreport:dataSourceConnection
         id="uniqueID"
         dataSourceName = "dataSourceName" >
      </bloxreport:dataSourceConnection>
   </bloxreport:sqlData>

   <bloxreport:calculate
      expression="calculatedMemberName= calculationExpression"
      index = "int">
   </bloxreport:calculate>

   <bloxreport:sort>
      <bloxreport:rule
         member="sortMember1"
         ascending="boolean"
         missingLast="true|false" />
   </bloxreport:sort>

   <bloxreport:filter
      expression = "filterExpression" >
   </bloxreport:filter>

   <bloxreport:members
      excluded = "member1, member2,..."
      included = "member1, member2,..." >
   </bloxreport:members>

   <bloxreport:order
      excluded = "member1, member2,..."
      included = "member1, member2,..." >
   </bloxreport:order>

   <bloxreport:group members="member1, member2, ...">
      <bloxreport:aggregation
         member="memberName"
         type="aggregationType">
      </bloxreport:aggregation>
   </bloxreport:group>

   <bloxreport:format>
```

```
        <bloxreport:numeric format="numericFormat1" />
        <bloxreport:numeric format="numericFormat2"
          member="member"/>
        <bloxreport:date format="dateFormat1" />
        <bloxreport:date format="dateFormat2" member="member"/>
        <bloxreport:missing format="missingDataFormat" />
        <bloxreport:html member = "member"/>
        <bloxreport:aggregation format="numericFormat1"
          member = "member1" />
        <bloxreport:aggregation format="numericFormat2"
          member = "member2" />
    </bloxreport:format>

    <bloxreport:style>
        <bloxreport:text style="textStyle" />
        <bloxreport:numeric style="numericStyle" />
        <bloxreport:date style="dateStyle" />
        <bloxreport:banding style="dateStyle" />
        <bloxreport:missing style="missingDataStyle" />
        <bloxreport:negative style="negativeStyle" />
        <bloxreport:column style="colStyle1" columnName="member1" />
        <bloxreport:column style="colStyle2" columnName="member2" />
        <bloxreport:data style="yourStyle" />
        <bloxreport:data style="yourStyle" columnName="member" />
        <bloxreport:columnHeader style="yourStyle"/>
        <bloxreport:columnHeader style="yourStyle"
columnName="member" />
        <bloxreport:columnHeader style="yourStyle"
columnName="member" level="n" />
        <bloxreport:groupHeader style="yourStyle" />
        <bloxreport:groupHeader style="yourStyle" level="n" />
        <bloxreport:groupFooter style="yourStyle" />
        <bloxreport:groupFooter style="yourStyle" level="n" />
        <bloxreport:groupTotal style="yourStyle" />
        <bloxreport:groupTotal style="yourStyle" level="n" />
    </bloxreport:style>

    <bloxreport:text>
        <bloxreport:data text="yourText" />
        <bloxreport:data text="yourText"" columnName="column" />
        <bloxreport:columnHeader text="yourText" />
        <bloxreport:columnHeader text="yourText" columnName="column"
level="n" />
        <bloxreport:columnHeader text="yourText" columnName="n" />
        <bloxreport:groupHeader text="yourText" />
        <bloxreport:groupHeader text="yourText" level="n" />
        <bloxreport:groupFooter text="yourText" />
        <bloxreport:groupFooter text="yourText" level="n" />
        <bloxreport:groupTotal text="yourText" />
        <bloxreport:groupTotal text="yourText" level="n" />
    </bloxreport:text>

</bloxreport:report>
```

ReportBlox can also take an RDBResultSetDataBlox, which lets you build a
relational report based on an RDBResultSet object from a DataBlox.

```
<bloxreport:report id="reportId">
    <bloxreport:rdbResultSetData
        bloxRef= "DataBloxName"
        columnCoordinate="col"
        drillThroughReportName="reportName"
        rowCoordinate="row"
    </bloxreport>
    ...
</bloxreport:report>
```

All Tag Attributes for PdfBlox

```
<bloxreport:pdf id = "idName"
   bloxName="bloxName"
   logoSource = "urlToImage"
   portrait = "true"
   bottom = "1in"
   top = "1in"
   height = "11in"
   width = "8.5in"
   left = "1.25in"
   right = "1.25in"
   footerText="text"
   footerHeight="30px"
   footerVisible = "true"
   headerText="text"
   headerHeight="50px"
   headerVisible = "true" >
   report="ReportBlox_id"
</bloxreport:pdf>
```

Important: The attribute values shown are the default unless they are in italic.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation, Licensing, 2-31 Roppongi 3-chome, Minato-ku, Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation, J46A/G4, 555 Bailey Avenue, San Jose, CA 95141-1003 U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

AIX	IBM	DB2
DB2 OLAP Server	DB2 Universal Database	WebSphere

Alphablox and Blox are trademarks or registered trademarks of Alphablox Corporation in the United States, other countries, or both.

Microsoft is a trademark of Microsoft Corporation in the United States, other countries, or both.

Java, JavaBeans, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product or service names may be trademarks or service marks of others.

Index

A

accessibility
  Relational Reporting   5

B

Blox tags
  Relational Reporting Blox, listing   135
Blox tags,
  See tags
Blox tags, using   99
bookmark
  relational reports   68
break group
  totals, specifying   59
break groups
  footers, specifying   59
  headers, specifying   59
  style classes, listing   9
browsers
  Relational Reporting   5

C

CalculateBlox
  definition   3
  percentOfTotal() function   62
  rank() function   62
  runningCount() function   62
  runningTotal() function   63
  tag reference   101
calculated members
  adding, in relational reports   33
calculating data
  in relational reports   33
cell banding
  setting colors   55
  turning off   55
cell banding, Relational Reporting   47
cell values, accessing   91
column header
  Context Menu   4
column headers
  Context Menu   4
column headers, renaming and formatting   49
components
  Relational Reporting   2
concepts
  Relational Reporting   7
custom tags
  See Blox Reporting Tag Library

D

data
  accessing, Relational Reporting   23
  display style   47
  filtering   32
  formatting   45

data *(continued)*
  hiding and showing members, in relational reports   36
  removing members, in relational reports   35
  sorting, in relational reports   31
data columns
  formatting   52
  style classes in Relational Reporting, listing   10
  width, color, and style, setting   52
data columns, renaming and formatting column headers   49
data row, accessing   91
data sources
  Relational Reporting, defining access to   23
DataSourceConnectionBlox
  definition   3
  tag reference   103
design considerations
  Relational Reporting   93
display areas   40
display names   19

E

error reporting
  ErrorBlox, using   24
ErrorBlox
  definition   3
  getNextException() method   98
  style classes   13
  tag reference   105
  using   24
ErrorBlox()
  getRootCause() method   98
Excel, exporting to   73
exporting, to Excel   73
exporting, to static HTML   66

F

FilterBlox
  definition   3
  tag reference   106
filtering
  data, in relational reports   32
footer, for break groups   59
FormatBlox
  definition   3
  HTML formatting, adding   47
  tag reference   108
functions
  percentOfTotal()   62
  rank()   62
  runningCount()   62
  runningTotal()   63

G

getData() method   82
group header
  Context Menu   4

IBM®

Program Number: 5724-L14

Printed in USA

Spine information:

IBM DB2 Alphablox

Relational Reporting Developer's Guide

Version 8.4

IBM