

DB2 Alphablox for UNIX and Windows v5.6

Relational Reporting Developer's Guide



Note: Before using this information and the product it supports, read the information in “Notices” on page 9.

First edition (August 2004)

This edition applies to version 5, release 6, of IBM DB2 Alphablox for UNIX and Windows V5.6 (product number 5724-J16) and to all subsequent releases and modifications until otherwise indicated in new editions.

Copyright © 1996 - 2004 Alphablox Corporation. All rights reserved.

© Copyright International Business Machines Corporation 1996, 2004. All rights reserved.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP
Schedule Contract with IBM Corp.

Contents

Relational Reporting Developer's Guide

Notices	9
Trademarks	11
Preface	13
About This Book	14
Related Documents	16
Online Documentation User Interface	17
Document Conventions	18
Icons	18
Typography	18
Contacting IBM	19
Product Information	19
Comments on the Documentation	20
Chapter 1	
Relational Reporting Overview	21
Relational Reporting	22
Localization	23
Components of Relational Reporting	23
Reports as an HTML Tables	26
Report Editor User Interface	26
Column Header Context Menu	28
Group Header Context Menu	28
Group Total Context Menu	29
Rendering Reports to PDF	29
Browser Support	30
Chapter 2	
Relational Reporting Concepts	31
Concepts for Relational Reporting	32
Componentized Blox Based on Standard Technologies	32
Report Rendering	32
Report Pipeline	33
Accessing Individual Blox	34

Columns and Members	34
Styling the Relational Reports	34
Style Classes	35
Style Classes in the Report	37
Style Classes in the Report Style Dialog Box	41
Style Classes for ErrorBlox	41
Relational Reporting Custom Tags	43
Nested Tags	44
Standalone Tags	46
The Order of Syntax Evaluation	46
Session Scope	48
Expression Syntax	48
Member Identifiers vs. Display Names	49

Chapter 3

Relational Report Development	51
Before You Begin	52
General Development Tips	52
General Report Development Steps	54
Define the Application and Data Source	54
Include the Reporting Blox Tag Library	55
Use a stylesheet	55
Use ErrorBlox for Better Error Reporting	56
Add Blox Tags	57
Creating Your First Relational Report	57
The Simplest Report	57
Task: Create a Simplest Report	58
The Simplest Interactive Report	58
Task: Create a Simplest Interactive Report	59
Learning Resources	59

Chapter 4

Accessing and Retrieving Data	61
Using SQLDataBlox and DataSourceConnectionBlox	62
Dynamically Setting the Query	62
Using RDBResultSetDataBlox to Access RDBResultSet from DataBlox	63
Error Handling Against SQLDataBlox	65

Chapter 5

Processing and Manipulating Data	67
Sorting Data	68
Filtering Data	70
Grouping Data	71

Adding Calculated Columns	71
Calculations Involving Missing Data	73
Adding Calculated Members Before Grouping	73
Removing Members	74
Hiding and Showing Members	75
Hiding and Showing Missing Data	76

Chapter 6

Formatting the Report and Data 79

Display Areas in a Rendered Report	80
Report Layout Formatting and Styling Summary Table	81
Styling vs. Formatting vs. Setting Text	83
Processing Sequence for StyleBlox, FormatBlox, and TextBlox	84
StyleBlox vs. CSS Styles	85
Formatting Data	87
Wrapping HTML Code Around Data Values	88
Adding HTML code to Data Returned from a Query	89
Styling Data Displayed in Report	90
Specifying and Styling Column Headers	93
Styling Column Headers	95
Specifying Column Width, Color and Style	96
Special Substitution Variables for Displaying Member Names and Values	97
The <member/> Substitution Variable	97
The <value/> Substitution Variable	98
Using the <member/> and <value/> Variables	99
Setting or Turning Off Cell Banding	100
Setting the Report Display Area	101
Adding Background Images	102

Chapter 7

Grouping Data 103

Overview of Break Groups and Break Group Levels	104
Break Group Aggregations	105
Specifying and Styling Break Group Headers, Footers, and Totals	107
Calculating Group-based Summary Columns	110
Adding Report Title and Column Summary (Aggregations)	115
Using MembersBlox in Conjunction with GroupBlox	115

Chapter 8

Saving and Exporting Data 117

Issues with Saving Interactive Reports Directly from Browser	118
Saving as Static HTML to File System	119
Bookmarking Reports and Saving States	121

Loading Bookmarks	123
Saving as PDF	125
Saving Reports as PDF Files	125
Rendering a Report Directly in PDF	126
Saving to Excel or Other Applications	127
Exporting to Excel	127
Sending a Report Directly to Excel	130

Chapter 9

Styling the Report Editor User Interface	131
Style Classes in the Report Editor User Interface	132
Overriding the Style Classes	135
User Help for Using the Report Editor	135

Chapter 10

Advanced Topics	137
Managing Session Scope	138
The Relational Reporting API	139
Creating an Interactive Report using the API	140
Dynamically Changing the Query	142
Example 1: Directly access the SQLDataBlox and resets its query	143
Example 2: Dynamically setting queries without refreshing the whole page using the global refreshReport() JavaScript method	144
Accessing Data Rows and Cell Values in Rendered Report	147

Chapter 11

Development and Troubleshooting Tips	149
General Tips and Development Steps	150
Design Considerations	150
Providing User Help	151
Localization of Help	151
Impact of Style Setting on Performance	152
Common Reporting Blox Tag Errors	153
Forgetting to include the taglib directive for Reporting Blox Tag Library ..	153
Forgetting to use the correct prefix for Relational Reporting Blox	153
Incorrect case of a tag or tag attribute	153
Forgetting to include the stylesheet	154
Refreshed page doesn't reflect code modification	154
Refer to member or column names incorrectly	154
Troubleshooting Tips	155
Error Handling Using ErrorBlox	156

Chapter 12**Relational Reporting Blox Tag Reference 159**

Using Blox Tags	160
CalculateBlox	161
The <bloxreport:calculate> Tag	162
DataSourceConnectionBlox	163
The <bloxreport:dataSourceConnection> Tag	163
ErrorBlox	165
The <bloxreport:error> Tag	165
FilterBlox	166
The <bloxreport:filter> Tag	166
FormatBlox	168
The <bloxreport:format> Tag	169
GroupBlox	172
The <bloxreport:group> Tag	173
MembersBlox	176
The <bloxreport:members> Tag	176
OrderBlox	178
The <bloxreport:order> Tag	178
PdfBlox	180
The <bloxreport:pdf> Tag	180
PersistenceBlox	182
The <bloxreport:persistence> Tag	182
RDBResultSetDataBlox	184
The <bloxreport:rdbResultSetData> Tag	184
ReportBlox	186
The <bloxreport:report> Tag	186
SortBlox	189
The <bloxreport:sort> Tag	189
SQLDataBlox	191
The <bloxreport:sqlData> Tag	191
StyleBlox	193
<bloxreport:style> Tag and Its Sub Tags	194
TextBlox	197
Nested Tags Inside <bloxreport:text>	198
The <member/> and <value/> Substitution Variables	200

Appendix A**Relational Reporting Tags for Copy-and-Paste205**

All Tags Nested Within <bloxreport:report>	205
All Tag Attributes for PdfBlox	207

Appendix B

Deprecated Tags for Relational Reporting 209

 Deprecated Tags and Attributes in Release 5.5 210

Index 211

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.

Licenses 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

Alphablox, InLine Analytics, Alphablox Analysis Server, Blox, SpreadsheetBlox, and the Alphablox logo are trademarks or registered trademarks of Alphablox Corporation.

IBM, DB2, DB2 Universal Database, WebSphere, and DB2 OLAP Server are trademarks of International Business Machines Corporation in the United States, other countries, or both.

Intel and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java 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.

Preface

This Preface describes the intended audience, organization, and conventions used in the *Relational Reporting Developer's Guide*. It also contains information about the DB2 Alphablox documentation set and information about how to contact IBM for technical problems or comments on the documentation.

Contents

- “About This Book” on page 14
- “Related Documents” on page 16
- “Online Documentation User Interface” on page 17
- “Document Conventions” on page 18
- “Contacting IBM” on page 19

About This Book

The *Relational Reporting Developer's Guide* contains information about creating reports from relational data sources. It includes conceptual information on the report development approach, procedures for performing report creation related tasks, and reference information on Relational Reporting Blox .

The *Relational Reporting Developer's Guide* is primarily designed for the people who develop applications and reports using DB2 Alphablox. The book is organized into the following chapters and appendix:

- Chapter 1, “Relational Reporting Overview” on page 21

This chapter provides an overview of the features and components of Relational Reporting.

- Chapter 2, “Relational Reporting Concepts” on page 31

This chapter describes the major concepts, terms, and expression syntax associated with Relational Reporting.

- Chapter 3, “Relational Report Development” on page 51

This chapter describes the general relational report assembly process as well as specific tasks regarding report generation, formatting, and distribution.

- Chapter 4, “Accessing and Retrieving Data” on page 61

This chapter discusses different ways to retrieve relational data into a ReportBlox.

- Chapter 5, “Processing and Manipulating Data” on page 67

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.

- Chapter 6, “Formatting the Report and Data” on page 79

This chapter provides information on how to format the overall report and the data in the report for desired look and feel.

- Chapter 7, “Grouping Data” on page 103

This chapter discusses how to add break groups to your report, format the various group headings, footers, and break group totals based on the break group levels. In addition, information on how to add a group-based calculated column that provides summary data per group—such as ranking, percent of totals, running totals, and running count—is also provided.

- Chapter 8, “Saving and Exporting Data” on page 117

This chapter describes the general relational report assembly process as well as specific tasks regarding report generation, formatting, and distribution.

- Chapter 9, “Styling the Report Editor User Interface” on page 131

This chapter describes how you can customize the interactive Report Editor user interface for your reports.

- Chapter 10, “Advanced Topics” on page 137

This chapter covers advanced topics such as managing session scope and using Alphablox Analytics Relational Reporting API.

- Chapter 11, “Development and Troubleshooting Tips” on page 149

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

- Chapter 12, “Relational Reporting Blox Tag Reference” on page 159

This chapter contains a detailed listing of custom JSP tags for Blox associated with Relational Reporting. Associated syntax, usage, tag attributes, and a code sample are provided.

- Appendix A, “Relational Reporting Tags for Copy-and-Paste” on page 205

This appendix provides a copy-and-paste template for all tags in the Reporting Blox Tag Library.

- Appendix B, “Deprecated Tags for Relational Reporting” on page 209

This appendix lists the deprecated tags for Relational Reporting.

Related Documents

The DB2 Alphablox documentation set includes books and online help. The books are all available in HTML, PDF, and printed format. Context sensitive help is available for all parts of the Alphablox Analytics home page as well as within Alphablox applications. The DB2 Alphablox documentation set includes the following books:

Title	Description
<i>Administrator's Guide</i>	Contains information about setting up and managing DB2 Alphablox and about DB2 Alphablox in a J2EE environment.
<i>Developer's Guide for the DHTML Client</i>	Provides guidance on designing, developing, and deploying analytical applications using the DHTML client. If you are new to DB2 Alphablox or are developing new applications, it is recommended that you start with this book.
<i>Developer's Reference for the DHTML Client</i>	A complete API reference for developing applications using the DHTML client; contains information on each Blox, including its JSP syntax, properties, methods, and objects.
<i>Relational Reporting Developer's Guide</i>	Contains information about setting up ReportBlox to build a report from relational data.
<i>Cube Server Administrator's Guide</i>	Contains information about setting up Alphablox cubes. Alphablox cubes allow you to present a multidimensional view of data stored in a relational data warehouse or data mart database.
<i>Installation Guide</i>	Contains information on system requirements, installing and configuring Alphablox Analytics, installing sample data, and migrating application from previous versions.

Javadoc is available for the server-side API, ReportBlox API, and FastForward API in the following directory:

```
<alphablox_dir>/system/documentation/javadoc/
{blox,report,fastforward}
```

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

Online Documentation User Interface

The DB2 Alphablox documentation is also available online in HTML and PDF formats. To open the Online Documentation, select the **Online Documentation** link on the **Help** menu or from any help page on the Alphablox Analytics home page.

When you select the Online Documentation, it opens in a frameset. The right frame displays documentation pages; the left frame contains the following navigation tabs:




Tab	Description
Contents	<p>The Contents tab presents a tree view of all the online books in the documentation set. Click on a book icon beside a heading to expand or collapse the tree, displaying or hiding the topics within that heading. To view a topic, click on its hyperlinked heading.</p> <p>To access a page containing links to all of the PDF versions of the documentation, click the PDF Documentation book icon and then click the PDF Documentation hyperlink.</p>
Index	<p>The Index tab presents an alphabetical list of all indexed words for every document in the DB2 Alphablox documentation set. To view a topic, click on the indexed item. If multiple pages are available for a topic, click the link with the page title for the first topic, click the link with the number 2 for the second topic, and so on.</p>
Search	<p>The Search tab provides a text search.</p> <p>The search feature provides a simple search on words entered. You can search a single book instead of the entire documentation set by selecting a book from the dropdown list. The search supports the use of asterisks (*) for wildcard searches, but does not use “near” logic or perform partial word search. Entering multiple words implies an and between the words, returning pages that contain all the words entered. The search is not case sensitive.</p> <p>To enter a search, click the Search tab, type one or more words in the search box, and press the Search button. The search presents a list of HTML pages containing the search word(s).</p> <p>To view a page, click on its hyperlinked heading. When the page appears in the right frame, use its hyperlinks or the browser’s Find command to locate the word(s) within the page.</p>

Document Conventions

Icons and typography call attention to or elaborate on areas of interest throughout the DB2 Alphablox documentation set.

Icons

The icons used in the documentation are as follows:

Icons	Description
	Identifies information helpful for the current task.
	Identifies conceptual information on a particular topic or suggestions for usage.
	Identifies important information that the audience should know before proceeding with a task.

Typography

The typography used in the documentation is as follows:

Convention	Description
Bold	Caution statements, labels, headings, and table headers appear in a bold font.
<i>Italics</i>	Italics indicate an emphasized word or phrase as well as book titles.
Monospace type	Code examples, filenames, object names, property names, and method names appear monospace type.
“Quotation marks”	The proper syntax for Blox properties and methods or queries may require single or double quotation marks. In addition, quotation marks surround a cross-reference to another topic.

Contacting IBM

If you have a technical problem, please review and carry out the actions suggested by the product documentation before contacting DB2 Alphablox Customer Support. This guide suggests information that you can gather to help DB2 Alphablox Customer Support to serve you better.

For information or to order any products, contact an IBM representative at a local branch office or contact any authorized IBM software remarketer. If you live in the U.S.A., you can call one of the following numbers:

- 1-800-IBM-SERV for customer support
- 1-888-426-4343 to learn about available service options

Product Information

If you live in the U.S.A., then you can call one of the following numbers:

- 1-800-IBM-CALL (1-800-426-2255) or 1-800-3IBM-OS2 (1-800-342-6672) to order products or get general information.
- 1-800-879-2755 to order publications.

<http://www.ibm.com/software/data/db2/alphablox>

Provides links to information about DB2 Alphablox.

<http://www.ibm.com/software/data/db2/udb>

The DB2 Universal Database Web pages provide current information about news, product descriptions, education schedules, and more.

<http://www.elink.ibm.com/>

Click Publications to open the International Publications ordering Web site that provides information about how to order books.

<http://www.ibm.com/education/certify/>

The Professional Certification Program from the IBM Web site provides certification test information for a variety of IBM products.

Note: In some countries, IBM-authorized dealers should contact their dealer support structure instead of the IBM Support Center.

Comments on the Documentation

Your feedback helps IBM to provide quality information. Please send any comments that you have about this book or other DB2 Alphablox documentation. You can use any of the following methods to provide comments:

- Send your comments using the online readers' comment form at www.ibm.com/software/data/rcf.
- Send your comments by electronic mail (e-mail) to comments@us.ibm.com. Be sure to include the name of the product, the version number of the product, and the name and part number of the book (if applicable). If you are commenting on specific text, please include the location of the text (for example, a title, a table number, or a page number).

Relational Reporting Overview

This chapter provides an overview of the features, functionality, components, and browser support of Relational Reporting.

Contents

- “Relational Reporting” on page 22
- “Components of Relational Reporting” on page 23
- “Browser Support” on page 30

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 Alphablox Analytics 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.

Profitability Report by Week_Ending				
2000-04-01				
Location	Product	Sales	Cost	
Napa	Milk Choco	Sort	\$242.50	\$71.78
Napa	Caram	Hide	\$648.00	\$203.40
Napa	Coffe	Show All	\$495.00	\$155.10
Sonoma	Milk Choco	Rename	\$227.50	\$67.34
Sonoma	Caram	Group	\$612.00	\$191.76
Sonoma	Coffe	Clear Groups	\$450.00	\$141.00
		Style...	total: \$2,675.00	total: \$830.38
2000-04-08				
Location	Product	Sales	Cost	
Beverly Hills	Milk Chocolate	Bunny	\$25.00	\$7.40
Beverly Hills	Caramel	Suckers	\$63.00	\$19.74
Beverly Hills	Coffee	Suckers	Sales Value Missing	
Napa	Milk Chocolate	Bunny	\$277.50	\$82.14
Napa	Caramel	Suckers	\$738.00	\$231.24
Napa	Coffee	Suckers	\$540.00	\$169.20
Sonoma	Milk Chocolate	Bunny	\$255.00	\$75.48
Sonoma	Caramel	Suckers	\$693.00	\$217.14
Sonoma	Coffee	Suckers	\$54.00	\$157.92
			total: \$2,645.50	total: \$960.26
			Total Sales: \$5,320.50	Total Cost: \$1,790.64

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

- Connect to a relational data source defined via Alphablox Analytics 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. Reports will automatically be displayed in the correct language based on the locale of the client.



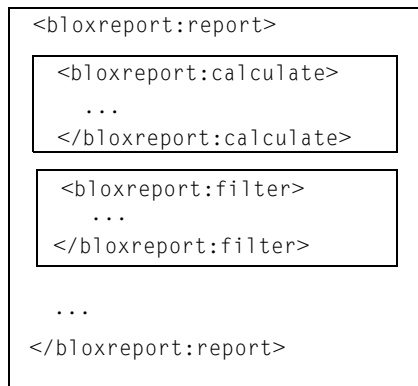
In order for the report to display correctly in languages other than English, you should specify to use the UTF-8 character set using the JSP page directive's `contentType` attribute. See “General Development Tips” on page 52 for more information.

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.



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 Alphablox Analytics.

Blox	Description
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.

Blox	Description
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.

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

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:

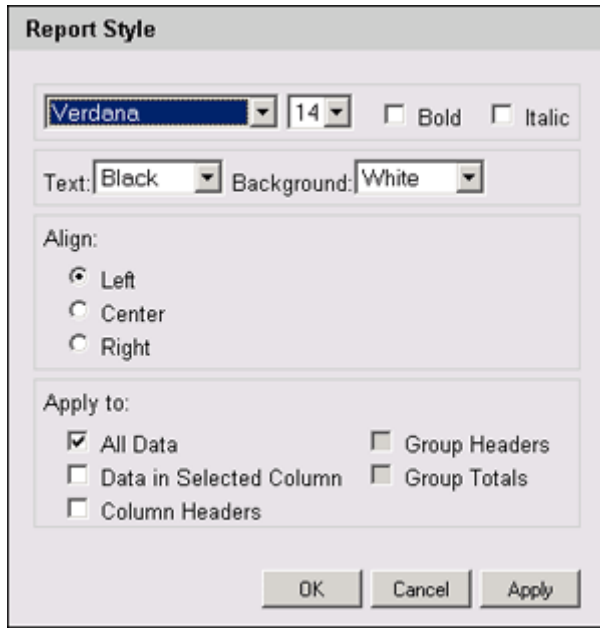
- column headers
- break group headers
- break group totals

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 screenshot shows a report titled "Profitability by Week Ending" with data grouped by week. The first group is for the week ending 2000-04-01, and the second is for 2000-04-08. Each group has a sub-header with columns for Location, Product, Gross Margin, and Profit%. The data rows are highlighted in yellow. A context menu is open over the "Caramel Suckers" row in the second group, showing options like Sort, Hide, Show All, Rename, Group, Clear Groups, and Style... Another context menu is open over the "2000-04-08" group header, showing Clear Group and Style... options. A "group totals" row is at the bottom, showing a total Gross Margin of \$1,659.03 and a total Profit% of 70. The overall total for all groups is \$3,172.77 and 69%.

Profitability by Week Ending			
2000-04-01			
Location	Product	Gross Margin	Profit%
Napa	Caramel Suckers	\$444.60	69
Napa	Coffee Suckers	\$339.90	69
Sonoma	Caramel Suckers	\$420.24	69
Sonoma	Coffee Suckers	\$309.00	69
		\$1,513.74	69
2000-04-08			
Location	Product	Gross Margin	Profit%
Beverly Hills	Coffee Suckers		
Napa	Milk Chocolate Blocks w/ Almonds		73
Napa	Caramel Suckers		69
Napa	Coffee Suckers	\$370.80	69
Sonoma	Caramel Suckers	\$475.86	69
		\$1,659.03	70
		\$3,172.77	69

These menus provide the users with dynamic report editing functionality such as sorting, adding break groups, hide/show columns, specifying break group aggregation type, and edit 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 stylesheet 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...

Profitability Report by Week_Ending			
2000-04-01			
Location	Product	Cost	Sales
Sonoma	Caramel Suckers	\$191.76	\$612.00
Sonoma	Coffee Suckers	\$141.00	\$450.00
		total: \$332.76	total: \$1,062.00
2000-04-08			
Location	Product	Cost	Sales
Napa	Coffee Suckers	.20	\$540.00
Sonoma	Milk Chocolate Bunny	.48	\$255.00
Sonoma	Caramel Suckers	.14	\$693.00
		total: .82	total: \$1,488.00
		Total Cost: \$794.58	Total Sales: \$2,550.00

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 107.

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.

Browser Support

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

- IE 5.5 and above
- Netscape v4.7 and above

Browsers supported for the interactive mode are:

- IE 5.5 and above

When users try to access an interactive report using Netscape browsers, an alert window will pop up, informing them that Internet Explorer is required for interactive reports.

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.

Contents

- “Concepts for Relational Reporting” on page 32
- “Styling the Relational Reports” on page 34
- “Relational Reporting Custom Tags” on page 43
- “Expression Syntax” on page 48

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 32
- “Report Rendering” on page 32
- “Report Pipeline” on page 33
- “Accessing Individual Blox” on page 34
- “Columns and Members” on page 34

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.



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 do not have the options to render reports in different rendering modes (such as DHTML or Java). Instead, 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.



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



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.

Two stylesheets `report.css` and `colemans.css` are provided that you can use out of the box. These stylesheets reside at:

```
<alphanblox_dir>/system/AlphanbloxPlatform/AlphanbloxServer/report/
```

where `<alphanblox_dir>` is the directory into which Alphanblox Analytics is installed.

In this directory, you will actually find several `.css` files: `report.css`, `coleman.css`, `styles.css`, `dialog.css`, and `error.css`.

- `styles.css`: contains the complete definition of style classes used to display a report.
- `dialog.css`: contains the definition for styles used by the Report Style dialog box in the Report Editor user interface.
- `error.css`: contains definition of styles used by ErrorBlox for error reporting.

The `report.css` file imports the above three style sheets. In your JSP pages, add

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

to the `<head>` section to use this stylesheet. 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 stylesheet or in-line styles in the JSP page. It is simply displayed as a plain HTML table. When the report is rendered in interactive mode, the stylesheets have to exist in order for the context menus and Report Style dialog box to display and work properly.

The `coleman.css` stylesheet mimics the Coleman theme used to render PresentBlox and GridBlox. It actually imports `report.css` and redefines only some of the style classes.

Style Classes

The classes used in the generated reports include the following:

For the overall report:

- `.report`
- `.loadingmessage`

For break groups:

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

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:

Profitability Report by Week_Ending				
2000-04-01				
Location	Product		Units Sold	Sales
Napa	Milk Chocolate	Sort	97	\$242.50
Napa	Caramel Su	Hide	72	\$648.00
Napa	Coffee Su	Show All	55	\$495.00
Sonoma	Milk Chocolate	Rename	91	\$227.50
Sonoma	Caramel Su	Group	68	\$612.00
Sonoma	Coffee Su	Clear Groups	50	\$450.00
		Style...	avg: 72	total: \$2,675.00
2000-04-08				
Location	Product		Units Sold	Sales
Beverly Hills	Milk Chocolate	Bunny	10	\$25.00
Beverly Hills	Caramel Suckers		7	\$63.00
Beverly Hills	Coffee Suckers		Units Value Missing	Sales Value Missing
Napa	Milk Chocolate	Bunny	111	\$277.50
Napa	Caramel Suckers		82	\$738.00
Napa	Coffee Suckers		60	\$540.00
Sonoma	Milk Chocolate	Bunny	102	\$255.00
Sonoma	Caramel Suckers		77	\$693.00
Sonoma	Coffee Suckers		56	\$54.00
			avg: 63	total: \$2,645.50
			Avg. Units: 67	Total Sales: \$5,320.50

With each break group added, the *N* in the classes `groupheaderN`, `groupfooterN`, and `grouptotalN` increases by 1.

While the aggregation data at the end of each break group is rendered using the `grouptotalN` style classes, you can add footer text at end of each break group that uses the `groupfooterN` style classes.

The Sales column is moved by dragging the moving the column to a new column position. The font, style, and color are specified in the `columndragged` style class.

Break group footers, using the `groupfooterN` style classes.

Caramel Suckers

Location	Sales	Units Sold	Sales
Napa		72	\$648.00
Sonoma		68	\$612.00
Napa		82	\$738.00
Sonoma		77	\$693.00
total: 299			total: \$2,691.00

--End of Caramel Suckers

Coffee Suckers

Location	Units Sold	Sales
Napa	55	\$495.00
Sonoma	50	\$450.00
Beverly Hills	Units Value Missing	Sales Value Missing
Napa	60	\$540.00
total: 165		total: \$1,485.00

--End of Coffee Suckers

You can create your own stylesheet that defines how you want each of the classes to look and specify in your JSP file where to find the stylesheet. For example, the following is a sample stylesheet 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;
    }

```



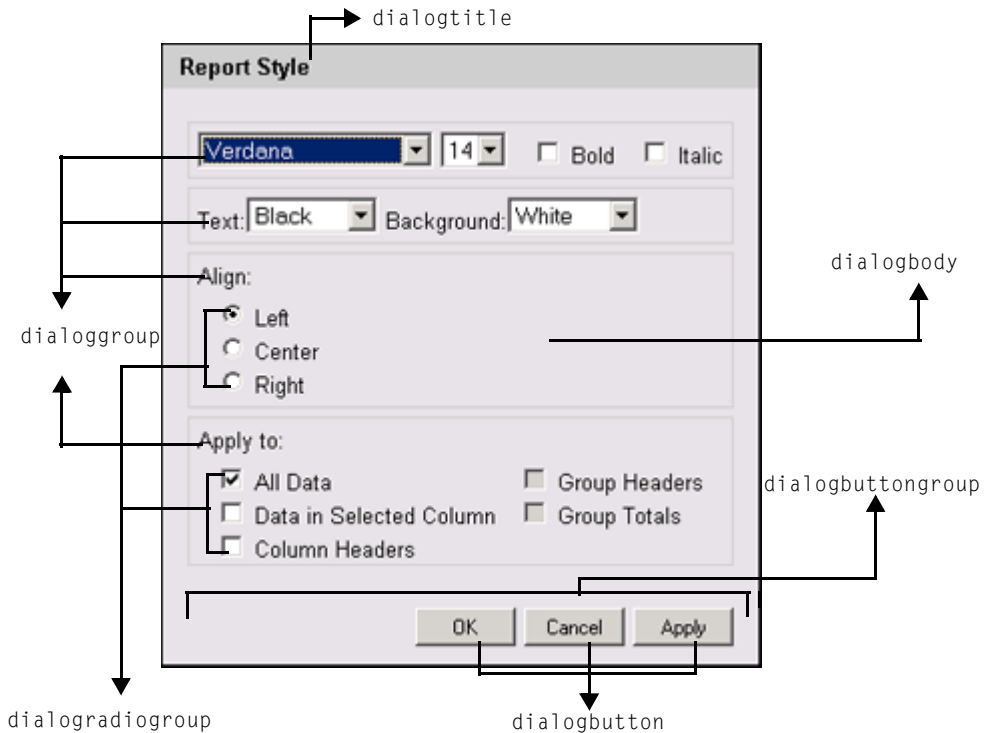
```

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 `` tags and style classes.



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 `error.css` stylesheet provided.

```

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

```

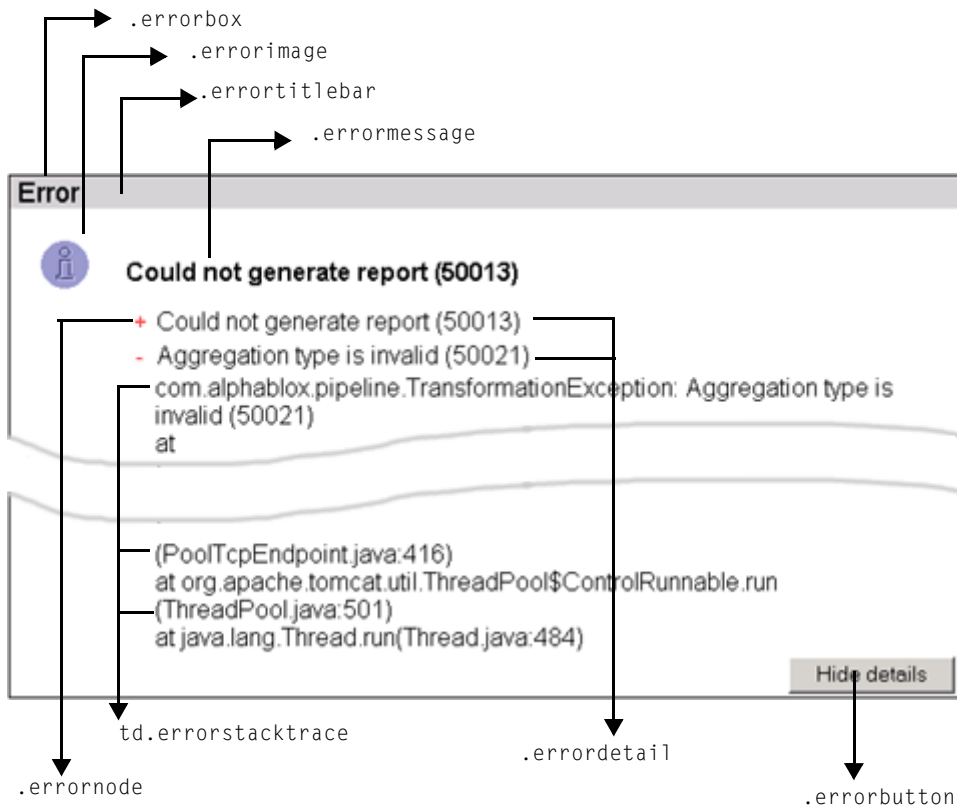
```

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

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

```

The style classes used to display the error list is shown as follows:



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 Reporting Blox 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 Alphas Analytics 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:style>

```

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 56. PdfBlox can be added outside of the `<bloxreport:report>` tag to directly render a report in PDF, or it can 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 126 and “Saving Reports as PDF Files” on page 125. 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 121.

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.

Session Scope

By default, all tags have a session scope. Once the object is created on the server and bound to a session, changes made to the value tag attributes in your JSP files during development will not apply unless you start a new session. This is why you need to start a new browser session to test changes you make. For your users, this means the changes they make through the Report Editor user interface are preserved only till the end of the session, not across sessions. Also, once your users make changes to the report, they will not be able to return to the “default” report as you created them in your JSP file through page reload from the browser. To provide a way for your users to retrieve the default report, you will need to explicitly provide a link or a button that unbinds the object from the session scope so new objects will be created that reflect the values you set in your JSP pages. See “Managing Session Scope” on page 138 for more details on session scope.

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 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>
```

 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.

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.

Contents

- “Before You Begin” on page 52
- “General Report Development Steps” on page 54
- “Creating Your First Relational Report” on page 57
- “Learning Resources” on page 59

Before You Begin

Before you begin your development, check the *Developer's Guide for the DHTML Client* 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 GridBlox instead.
- 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 long loading 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 Alphablox Tag Libraries. For Relational Reporting, you need to import the Reporting Blox 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 be 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 Netscape, this means you need to close up all browser windows before you open a new one.

There are two useful techniques you can use to avoid having to close and reopen a new browser window. One technique is to use the `removeAttribute()`, `setAttribute()` and `getAttribute()` methods associated with the `session` object. For example:

```
<% session.removeAttribute("myReportBlox"); %>
<bloxreport:report id="myReportBlox" .../>
```

This `removeAttribute()` method unbounds the object from the session. As the JSP engine goes to the next line, since no object of that name is found associated with the session, a new instance will be created. However, before you deploy the application, remember to remove this scriptlet code. The `setAttribute()` and `getAttribute()` methods allow you to dynamically manipulate the objects without deleting them. Another technique is use a dynamic `bloxName` for your `ReportBlox`. For details, check out the sections on “Managing Session Scope” on page 138 and “Dynamically Changing the Query” on page 142.

- 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 Alphablox Analytics 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.



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 Alphablox Analytics 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 Alphablox Analytics through the Alphablox Analytics 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 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 57.

Include the Reporting Blox Tag Library

You need to include the Reporting Blox 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 Alphablox Tag Libraries, see *Developer's Guide for the DHTML Client*.

Use a stylesheet

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 stylesheet to modify only the styles for classes you want to change:


```
<link rel="stylesheet" href="/AlphabloxServer/report/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 stylesheet from your custom stylesheet:

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

 When you upgrade or install new version of Alphablox Analytics, the entire folder at `<alphanblox_dir>/system/AlphabloxPlatform/AlphabloxServer/report/` will be overwritten. Therefore, you should not modify the default stylesheets in there.

Finally, you can also make a copy of the supplied style sheets into your own application folder and modify it to arrive at your desired look and feel. The supplied style sheets are at:

```
<alphanblox_dir>/system/AlphabloxPlatform/AlphabloxServer/report/
```

where `<alphanblox_dir>` is the directory where Alphablox Analytics is installed. When you copy the stylesheets, make sure you copy all stylesheets (except `colemans.css` unless you are using it) since `report.css` imports styles from the other stylesheets.

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/report/
error.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 stylesheet supplied by Alphablox Analytics 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 isErrorPage="yourError.jsp" %>
```

You can customize the styles used to display the errors. Note that when you upgrade or install new version of Alphablox Analytics, the entire folder at `<alphanblox_dir>/system/AlphabloxPlatform/AlphabloxServer/report/` will be overwritten. Therefore, you should not modify the default `error.css` stylesheet in there. Instead, create your own stylesheet in your application directory. Then import the default stylesheet into your own error style sheet:

```
// myErrorStyle.css
@import url( /AlphabloxServer/report/error.css );
.errortitlebar {
  background-color: yellow;
  font-size: 120%;
}
```


For more information on error handling and the use of `ErrorBlox`, see “Error Handling Using `ErrorBlox`” on page 156. For more information on `ErrorBlox` style classes, see “Style Classes for `ErrorBlox`” on page 41.

Add Blox Tags

To add a Relational Reporting Blox on your JSP page, you can copy the syntax from examples provided or from the “Relational Reporting Blox Tag Reference” on page 159. 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 43 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 Alphablox Analytics, 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 your 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.



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/report/
report.css" />
</head>
```

Task: Create a Simplest Interactive Report

- 1 Open the simplest report you created in “Task: Create a Simplest Report” on page 58.
- 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.



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. An alternative is to use the `removeAttribute()` method associated with the `session` object. See the “General Development Tips” on page 52 for details on how to use this technique.

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 the “Development and Troubleshooting Tips” on page 149 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.

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.

Contents

- “Using SQLDataBlox and DataSourceConnectionBlox” on page 62
- “Using RDBResultSetDataBlox to Access RDBResultSet from DataBlox” on page 63
- “Error Handling Against SQLDataBlox” on page 65

Using SQLDataBlox and DataSourceConnectionBlox

DataSourceConnectionBlox handles data connection, providing access to your relational data sources defined to Alphablox Analytics through the Alphablox Analytics 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 57, the general outline of you JSP looks as follows:

```
<%@ taglib uri="bloxreporttld" prefix="bloxreport" %>
<html>
<head>
  <link rel="stylesheet" href="/AlphabloxServer/report/
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 163. For details on SQLDataBlox and its tag attributes, see “SQLDataBlox” on page 191. 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 142 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.

Alphablox Analytics's drillthrough 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 look as follows:

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

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 156 for more discussions and a code example. See “The <bloxreport:report> Tag” on page 186 for details on `noDataMessage` and `noDataDueToErrorMessage` tag attributes.

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.

Contents

- “Sorting Data” on page 68
- “Filtering Data” on page 70
- “Grouping Data” on page 71
- “Adding Calculated Columns” on page 71
- “Removing Members” on page 74
- “Hiding and Showing Members” on page 75
- “Hiding and Showing Missing Data” on page 76

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.



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 grouping data (using GroupBlox) does not imply sorting. GroupBlox only handles data grouping and does not sort the data before grouping. For example, the following is a report grouped by "Type" and then "Product," with the break groups listed in alphabetical order. Within each break group, the "Sales" column is sorted in ascending order:

Dark			
Dark Chocolate Blocks			
Location	Cost	Units	Sales
Beverly Hills	\$3.00	5	\$10.00
Sonoma	\$27.00	45	\$90.00
Napa	\$29.40	49	\$98.00
	\$59.40	99	\$198.00
	\$59.40	99	\$198.00
Milk			
Milk Chocolate Blocks			
Location	Cost	Units	Sales
Beverly Hills	\$7.20	12	\$24.00
Sonoma	\$67.80	113	\$226.00
Napa	\$72.00	120	\$240.00
	\$147.00	245	\$490.00
Milk Chocolate Blocks w Almonds			
Location	Cost	Units	Sales
Beverly Hills	\$10.05	15	\$37.50
Sonoma	\$89.78	134	\$335.00
Napa	\$101.17	151	\$377.50
	\$201.00	300	\$750.00
Milk Chocolate Bunny			
Location	Cost	Units	Sales
Beverly Hills	\$7.40	10	\$25.00
Sonoma	\$67.34	91	\$227.50

To create such a report, 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 screenshot.

For details on adding break groups, see “Overview of Break Groups and Break Group Levels” on page 104. For details on SortBlox and SortBlox tag attributes, see “SortBlox” on page 189. For details on GroupBlox and GroupBlox tag attributes, see “GroupBlox” on page 172.

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).



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

```
<bloxreport:filter expression="not isMissing(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.

FilterBlox	SortBlox
Each FilterBlox takes only one filter expression.	SortBlox can take multiple sort runles, each specified using a nested <code>rule</code> tag.
You can only filter number data.	You can sort on numeric data, strings, dates, and time.

See “FilterBlox” on page 166 for details on FilterBlox usage and “Expression Syntax” on page 48 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 104. For details on GroupBlox tag syntax, see “GroupBlox” on page 172.

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 73 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 110.

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	10
Store B	4	10	14
Store C	3	7	10
Store D		3	
Store E	4	5	7
	Total: 16	Total: 35	Total: 41
	Count: 4	Count: 5	Count: 4
	Ave: 4	Ave: 7	Avg: 10.25

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 87.

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 110 in the Grouping Data 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 75. For more information on `MembersBlox` tag attributes, see “`MembersBlox`” on page 176.

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 in 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 74. For hiding missing data, see “Hiding and Showing Missing Data” on page 76. For more information on OrderBlox tag attributes and usage, see “OrderBlox” on page 178. The following table summarizes 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 <code>included</code> attribute.	Sets the order of the members with its <code>included</code> attribute.
Each MembersBlox tag can only take either the <code>included</code> or the <code>excluded</code> attribute.	Like MembersBlox, each OrderBlox tag can only take either the <code>included</code> or the <code>excluded</code> attribute.
Members not in the <code>included</code> tag attribute are excluded (permanently removed from the data pipeline).	Members not in the <code>included</code> 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 to 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="isMissing(Sales)" />
```

FilterBlox only takes one expression at a time. For details on filtering data, see “Filtering Data” on page 70. For FilterBlox syntax and usage, see “FilterBlox” on page 166.

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 87. For more on setting styles using `StyleBlox`, see “StyleBlox” on page 193 for its syntax and usage.

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.

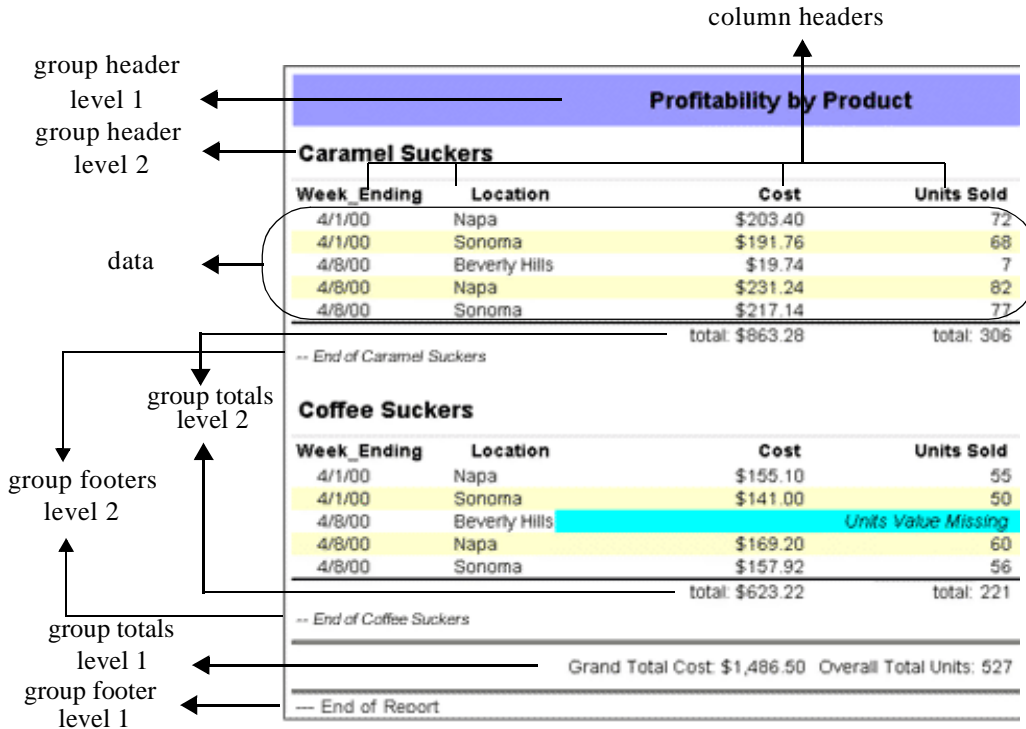
The “Formatting the Report and Data” example in the Blox Sampler for Relational Reporting demonstrates most of the tasks discussed in this chapter.

Contents

- “Display Areas in a Rendered Report” on page 80
- “Styling vs. Formatting vs. Setting Text” on page 83
- “StyleBlox vs. CSS Styles” on page 85
- “Formatting Data” on page 87
- “Wrapping HTML Code Around Data Values” on page 88
- “Styling Data Displayed in Report” on page 90
- “Specifying and Styling Column Headers” on page 93
- “Specifying Column Width, Color and Style” on page 96
- “Special Substitution Variables for Displaying Member Names and Values” on page 97
- “Setting or Turning Off Cell Banding” on page 100
- “Setting the Report Display Area” on page 101
- “Adding Background Images” on page 102

Display Areas in a Rendered Report

A relational report is rendered into several general areas: group headers, group footers, group totals, column headers, and data:



A report without any grouping has only the column headers and data areas. Once a grouping is added, group headers, group footers, and group totals areas become available. If the example above, the report is grouped by Product, creating:

- A level 1 group header area for the overall report
- A level 2 group header area for each of the Product category
- A level 2 group total area for aggregations of numeric data within each Product category
- A level 1 group total area for aggregations of numeric data for the overall report
- A level 2 group footer area for each Product category
- A 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 <code>columnHeader</code> sub tag:	Use the <code>columnHeader</code> sub tag:	.column
<pre><bloxreport:text> <bloxreport:columnHeader text="new column header" columnName="columnName" /> </bloxreport:text></pre>	<pre><bloxreport:style> <bloxreport:columnHeader style="CSS style string" [columnName="columnName"] /> </bloxreport:style></pre>	
Data Cells		
Use the <code>data</code> sub tag:	Use the <code>data</code> sub tag:	.data
<pre><bloxreport:text> <bloxreport:data text="new data text" [columnName="columnName"] /> </bloxreport:text></pre>	<pre><bloxreport:style> <bloxreport:data style="CSS style string" [columnName="columnName"] /> </bloxreport:style></pre>	

To set text using TextBlox	To set styles using StyleBlox	To set styles using CSS style classes
Group Footers		
<p>Use the <code>groupFooter</code> sub tag:</p> <pre><bloxreport:text> <bloxreport:groupFooter text="group footer text" [level="N"] /> </bloxreport:text></pre>	<p>Use the <code>groupFooter</code> sub tag:</p> <pre><bloxreport:style> <bloxreport:groupFooter style="CSS style string" level="N" /> </bloxreport:style></pre>	<pre>.groupfooter1, .groupfooter2,groupfooterN</pre>
Group Headers		
<p>Use the <code>groupHeader</code> sub tag:</p> <pre><bloxreport:text> <bloxreport:groupHeader text="group header text" [level="N"] /> </bloxreport:text></pre>	<p>Use the <code>groupHeader</code> sub tag:</p> <pre><bloxreport:style> <bloxreport:groupHeader style="CSS style string" level="N" /> </bloxreport:style></pre>	<pre>.groupheader1, .groupheader2,groupheaderN</pre>
Group Totals		
<p>Use the <code>groupTotal</code> sub tag:</p> <pre><bloxreport:text> <bloxreport:groupTotal text="group total text" [level="N"] [columnName="columnName"] /> </bloxreport:text></pre>	<p>Use the <code>groupTotal</code> sub tag:</p> <pre><bloxreport:style> <bloxreport:groupTotal style="CSS style string" level="N" [columnName="columnName"] /> </bloxreport:style></pre>	<pre>.grouptotal1, .grouptotal2,grouptotalN</pre>

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 “Grouping Data” on page 103.

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: <ul style="list-style-type: none"> • 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 <code>style</code> attribute for specifying the CSS style string.	Nested tags: <ul style="list-style-type: none"> • numeric: specifying the numeric format, for example, <code>\$.###.00</code> • date: specifying the data format, for example, <code>yyyy.MM.dd</code> • 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 <code>format</code> 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 81 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, 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 stylesheet.

For details on styling reports, see “Styling Data Displayed in Report” on page 90, and “Specifying and Styling Column Headers” on page 93. The details for StyleBlox usage and tag syntax are described in “StyleBlox” on page 193.

For details on data formatting, see “Formatting Data” on page 87. The details for FormatBlox usage and tag syntax are described in “FormatBlox” on page 168.

For details on setting texts, see “Specifying and Styling Column Headers” on page 93 and “Specifying and Styling Break Group Headers, Footers, and Totals” on page 107. The details for TextBlox usage and tag syntax are described in “TextBlox” on page 197.

StyleBlox vs. CSS Styles

A relational report is rendered using a set of styles defined in `report.css` in `<alphablox_dir>/system/AlphabloxPlatform/AlphabloxServer/report/`. The complete listing is provided in “Style Classes” on page 35. 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 80. For detailed discussion of grouping and styling a report with break groups, see the chapter on “Grouping Data” on page 103.

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`, `.choiceover`, `.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 “Styling the Report Editor User Interface” on page 131.

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 mouseover
- `.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
<code>column</code> : style applies to both the header and the data of the specified member.	[none]
<code>columnHeader</code> : 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.	<code>.column</code> : style applies to all column headers.
<code>data</code> : 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.	<code>.data</code> : style applies to all data
<code>banding</code> : style applies to alternate data rows	<code>.banding</code> : style applies to alternate data rows



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.”

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 five areas in a render report is described in “Display Areas in a Rendered Report” on page 80. Using TextBlox, you can rename or wrap HTML code around 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 hyperlinks that bring up more information on each location.

Week_Ending	Area	Location	Product
4/1/00	N. Cal.	Sonoma	Milk Chocolate Bunny
4/1/00	N. Cal.	Sonoma	Caramel Suckers
4/1/00	N. Cal.	Sonoma	Coffee Suckers
4/8/00	S. Cal.	Beverly Hills	Milk Chocolate Blocks
4/8/00	S. Cal.	Beverly Hills	Dark Chocolate Blocks
4/8/00	S. Cal.	Beverly Hills	White Chocolate Blocks

This is done using the following code:

```
<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 197 and “Special Substitution Variables for Displaying Member Names and Values” on page 97.

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:

```
<%
  session.removeAttribute("htmlLinkReport");
```

```

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 AlphasBlox 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
<pre> .data { font-size: 11; text-align: right; padding-left: 20; color: black; background-color: white; } </pre>	All data cells are displayed in the style specified.
<pre> .banding { background-color: #CCCCFF; } </pre>	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 35.

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" />	<p>Alternate data rows are displayed in the specified style.</p> <p> 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.</p>
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



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` tages, 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.



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 80 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\">Unit Cost</a>" />
</bloxreport:text>

```

The rendered report looks as follows:

 Unit Cost	Units	Sales
72	120	240
67.8	113	226
7.2	12	24
84.6	141	282
76.8	128	256
308.4	514	1,028

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\"><member/></a>"/>
</bloxreport:text>
```

`<member/>` 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.



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.



Avoid adding styling strings using `TextBlox`. Styles are output last by the 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 stylesheets and `StyleBlox`.



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 Alfablox 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 stylesheets 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 93 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.



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" />
```

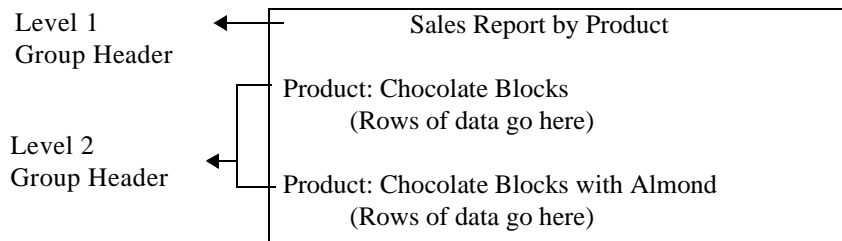
If the member name is not recognized or does not exist, the entire tag string will be treated as texts and no variable substitution will occur.



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 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 Grouping Data chapter and the section on “Specifying and Styling Break Group Headers, Footers, and Totals” on page 107.

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 stylesheet 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:


Week_Ending
2000-04-01

Product	Type	Cost	Units	Sales
Milk Chocolate Blocks	Milk	72	120	240
Dark Chocolate Blocks	Dark	29.4	49	98
White Chocolate Blocks	White	43.8	73	146
Milk Chocolate Blocks w Almonds	Milk	101.17	151	377.5
Milk Chocolate Bunny	Milk	71.78	97	242.5
Caramel Suckers		203.4	72	648
Coffee Suckers		155.1	55	495
Milk Chocolate Blocks	Milk	67.8	113	226
Dark Chocolate Blocks	Dark	27	45	90
White Chocolate Blocks	White	41.4	69	138
Milk Chocolate Blocks w Almonds	Milk	89.78	134	335
Milk Chocolate Bunny	Milk	67.34	91	227.5
Caramel Suckers		191.76	68	612
Coffee Suckers		141	50	450

Monthly Income Statement [CSSOL](#)
October Sales Figures

Sales	37,714
COGS	14,542
Marketing	5,288
Payroll	4,056
Misc	89
Profit	13,739

Rapid market growth is anticipated. According to the Gourmet Food Resource Group, the average consumption of chocolate and related food per person per year will increase by 8% next year. The areas of growth will be different depending on geographical areas. The East is anticipated to have the greatest increase of consumption while the West is not far behind.

 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. The following example shows a watermark background image added to the report, and as the users scroll up and down, the image stays in the center.

Chocolate Report						
Location	Product	Code	Type	Cost	Units	Sales
Napa	Milk Chocolate Blocks	210	Milk	72	120	240
Napa	Dark Chocolate Blocks	220	Dark	29.4	49	98
Napa	White Chocolate Blocks	230	White	43.8	73	146
Napa	Milk Chocolate Blocks w Almonds	240	Milk	101.17	151	377.5
Napa	Milk Chocolate Bunny	252	Milk	71.78	97	242.5
Napa	Caramel Suckers	431		203.4	72	648
Napa	Coffee Suckers	432		155.1	55	495
Sonoma	Milk Chocolate Blocks	210	Milk	67.8	113	226
Sonoma	Dark Chocolate Blocks	220	Dark	27	45	90
Sonoma	White Chocolate Blocks	230	White	41.4	69	138
Sonoma	Milk Chocolate Blocks w Almonds	240	Milk	89.78	134	335
Sonoma	Milk Chocolate Bunny	252	Milk	67.34	91	227.5
Sonoma	Caramel Suckers	431		191.76	68	612



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.

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.

Contents

- “Overview of Break Groups and Break Group Levels” on page 104
- “Specifying and Styling Break Group Headers, Footers, and Totals” on page 107
- “Calculating Group-based Summary Columns” on page 110
- “Adding Report Title and Column Summary (Aggregations)” on page 115
- “Using MembersBlox in Conjunction with GroupBlox” on page 115

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 is at level 2, as shown below:

The diagram illustrates a report titled "Profitability by Product" with two main product groups: "Caramel Suckers" and "Coffee Suckers". The report is structured as follows:

Profitability by Product				
Caramel Suckers				
Week_Ending	Location	Cost	Units Sold	
4/1/00	Napa	\$203.40	72	
4/1/00	Sonoma	\$191.76	68	
4/8/00	Beverly Hills	\$19.74	7	
4/8/00	Napa	\$231.24	82	
4/8/00	Sonoma	\$217.14	71	
		total: \$863.28	total: 306	
-- End of Caramel Suckers				
Coffee Suckers				
Week_Ending	Location	Cost	Units Sold	
4/1/00	Napa	\$155.10	55	
4/1/00	Sonoma	\$141.00	50	
4/8/00	Beverly Hills		Units Value Missing	
4/8/00	Napa	\$169.20	60	
4/8/00	Sonoma	\$157.92	56	
		total: \$623.22	total: 221	
-- End of Coffee Suckers				
		Grand Total Cost: \$1,486.50	Overall Total Units: 527	
-- End of Report				

Annotations in the diagram identify the following components:

- group header level 1:** "Profitability by Product"
- group header level 2:** "Caramel Suckers" and "Coffee Suckers"
- data:** The rows of data within each product group.
- group totals level 2:** "total: \$863.28" and "total: 306" for Caramel Suckers; "total: \$623.22" and "total: 221" for Coffee Suckers.
- group footers level 2:** "-- End of Caramel Suckers" and "-- End of Coffee Suckers"
- group totals level 1:** "Grand Total Cost: \$1,486.50" and "Overall Total Units: 527"
- group footer level 1:** "-- End of Report"
- column headers:** "Week_Ending", "Location", "Cost", "Units Sold"

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>
```



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 68 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 non-numeric columns, the default aggregation value is treated as missing data. The only valid aggregation type for non-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	<ul style="list-style-type: none"> • sum: 15 • average: 3 • count: 5 • max: 5 • min: 1
2	
3	
4	
5	
Missing	

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

Task	Solution
Setting style	<p>CSS style class: <code>grouptotal1, .grouptotal2, ..., .grouptotalN.</code></p> <p>StyleBlox's groupTotal sub tag:</p> <pre><bloxreport:style> <bloxreport:groupTotal level="N" style="color: red" /> </bloxreport:style></pre>
Specifying data format	<p>FormatBlox's aggregation sub tag:</p> <pre><bloxreport:format> <bloxreport:aggregation format="\$#,###" /> </bloxreport:format></pre>
Modifying displayed group totals	<p>TextBlox's nested groupTotal tag:</p> <pre><bloxreport:text> <bloxreport:groupTotal columnName="Cost" text="Total: <value/>" /> </bloxreport:text></pre>

See “Specifying and Styling Break Group Headers, Footers, and Totals” on page 107 and “Formatting Data” on page 87 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 104.

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 stylesheets and StyleBlox.
- Group footers for levels 1 through 3 are disabled by default in the supplied stylesheet. 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 stylesheet:

```
.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 stylesheet classes, as described in the following table:

Add Grouping	
Use GroupBlox: <pre><bloxreport:group members="Product,Week_Ending" /></pre>	Product is the first level grouping; Week_Ending, the second level.
Specify text for group headers, footers, and totals	
Use TextBlox: <pre><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="\RankOfSales\"/>" /> <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></pre>	<p>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.</p> <p>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.</p> <p>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.</p> <p>See “Special Substitution Variables for Displaying Member Names and Values” on page 97 for details on the <member/> and <value/> variables.</p>

Specify display style for group headers, footers, and totals	
<p>Use CSS stylesheet:</p> <pre>.groupheader1 { font-size: 110%; background-color: #9999FF; font- weight: bold; } .groupfooter1 { font-size: 80%; background-color: #9999FF; } .grouptotal1 { font-size: 100%; text-align: right; }</pre>	<p>Use the style classes <code>groupheaderN</code>, <code>groupfooterN</code>, and <code>grouptotalN</code> to specify the display styles.</p>
<p>Use StyleBlox:</p> <pre><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></pre>	<p><code>groupHeader</code>, <code>groupFooter</code>, and <code>groupTotal</code> are nested tags within the <code><bloxreport:style></code> tag. These tags have a <code>style</code> attribute for specifying CSS style strings. They are only in effect when the report is grouped. If <code>level</code> is not specified, the style will be applied to group headers, footers, or totals of all levels.</p>
Specify aggregation type for group totals	
<p>Use GroupBlox:</p> <pre><bloxreport:group members = "Product, Week_Ending"> <bloxreport:aggregation member="Units" type="sum"/> </bloxreport:group></pre>	<p>Use the <code><bloxreport:aggregation></code> tag nested within the <code><bloxreport:group></code> tag to specify the aggregation type for each numeric data column. The default aggregation type for numeric data is <code>sum</code>. Valid values are <code>sum</code>, <code>average</code>, <code>count</code>, <code>min</code>, <code>max</code>, and <code>none</code>.</p>

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

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

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

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

or

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



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.



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`.

The following shows a report grouped by `Area`, `Location`, and then `Week_Ending`, with these added columns based on the values in `Units`:

(Level 1) For All Areas						
(Level 2) Area: N. Cal.						
(Level 3) Location: Napa						
(Level 4) Week_Ending: 2000-04-01						
Product	Units	Rank	% Total	RunningTotal	RunningCount	
Caramel Suckers	72	5	11.7%	72	1	
Coffee Suckers	55	6	8.9%	127	2	
Dark Chocolate Blocks	49	7	7.9%	176	3	
Milk Chocolate Blocks	120	2	19.4%	296	4	
Milk Chocolate Blocks w Almonds	151	1	24.5%	447	5	
Milk Chocolate Bunny	97	3	15.7%	544	6	
White Chocolate Blocks	73	4	11.8%	617	7	
	617		100.0%	617	7	
(Level 4) Week_Ending: 2000-04-08						
Product	Units	Rank	% Total	RunningTotal	RunningCount	
Caramel Suckers	82	4	11.8%	82	1	
Coffee Suckers	60	6	8.6%	142	2	
Dark Chocolate Blocks	54	7	7.8%	196	3	
Milk Chocolate Blocks	141	2	20.3%	337	4	
Milk Chocolate Blocks w Almonds	167	1	24.0%	504	5	
Milk Chocolate Bunny	111	3	16.0%	615	6	
White Chocolate Blocks	80	5	11.5%	695	7	
	695		100.0%	695	7	
	1,312			1,312	14	

----End of Location: Napa

Four columns—Rank, % Total, RunningTotal, and RunningCount—are calculated based on the value in Units, with grouping level set to 4 (the lowest level, which is the default).

Notice that for the week of 2000-04-01, Milk Chocolate Blocks with Almonds was the number one seller in Napa of the Northern California area, accounting for 24.5% of the units sold. This calculation is based on level 4 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).

For ranking, notice there is no aggregation value within the level the calculation is based on (since this would not be meaningful). With the other calculations, you get their aggregation values for each grouping level, and these values are accessible using the `<value/>` substitution variable.

The following shows the same report with the level for each of the calculated summary columns set to level 3. Compare the output with the previous one and notice that:

- The Rank column now ranks the Units for each level 3 group (in this case, it's Napa), regardless of the week. A group total for Rank is available for each sub-group in Napa.
- The other three summary columns are calculated based on each level 3 grouping.

(Level 1) For All Areas						
(Level 2) Area: N. Cal.						
(Level 3) Location: Napa						
(Level 4) Week_Ending: 2000-04-01						
Product	Units	Rank	% Total	RunningTotal	RunningCount	
Caramel Suckers	72	10	5.5%	72	1	
Coffee Suckers	55	12	4.2%	127	2	
Dark Chocolate Blocks	49	14	3.7%	176	3	
Milk Chocolate Blocks	120	4	9.1%	296	4	
Milk Chocolate Blocks w Almonds	151	2	11.5%	447	5	
Milk Chocolate Bunny	97	6	7.4%	544	6	
White Chocolate Blocks	73	9	5.6%	617	7	
	617	2	47.0%	617	7	
(Level 4) Week_Ending: 2000-04-08						
Product	Units	Rank	% Total	RunningTotal	RunningCount	
Caramel Suckers	82	7	6.2%	699	8	
Coffee Suckers	60	11	4.6%	759	9	
Dark Chocolate Blocks	54	13	4.1%	813	10	
Milk Chocolate Blocks	141	3	10.7%	954	11	
Milk Chocolate Blocks w Almonds	167	1	12.7%	1,121	12	
Milk Chocolate Bunny	111	5	8.5%	1,232	13	
White Chocolate Blocks	80	8	6.1%	1,312	14	
	695	1	53.0%	1,312	14	
	1,312		100.0%	1,312	14	

---End of Location: Napa

When grouping level is set to 3, notice the ranking are now for the Napa location across the two weeks. During these two weeks, the Napa location sold a total of 1312 units, with the first week accounts for 47% of the sale, while the second week sold 53%.

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

Function	Syntax
rank()	<p>rank(memberName [, ASC DESC] [, level])</p> <p>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:</p> <pre><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)" /></pre>
percentOfTotal()	<p>percentOfTotal(memberName [, level])</p> <p>Defaults to the lowest level of grouping if level is not specified or the specified level does not exist.</p> <pre><bloxreport:calculate expression = "[% Total] = percentOfTotal(Units)" /> <bloxreport:calculate expression = "[% Total] = percentOfTotal(Units, 2)" /></pre> <p>The calculated values will be decimals (for example, 0.245). To format the data as 24.5%, use FormatBlox.</p>
runningCount()	<p>runningCount(memberName [, level])</p> <p>Defaults to the lowest level of grouping if level is not specified or the specified level does not exist. Examples:</p> <pre><bloxreport:calculate expression = "RunningCount = runningCount(Units)" /> <bloxreport:calculate expression = "RunningCount = runningCount(Units, 3)" /></pre>
runningTotal()	<p>runningTotal(memberName [, level])</p> <p>Defaults to the lowest level of grouping if level is not specified or the specified level does not exist. Examples:</p> <pre><bloxreport:calculate expression = "RunningTotal = runningTotal(Units)" /></pre>

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 as show in the following screenshot:

(Level 1) An Incorrect Example

(Level 2) Area: N. Cal.

(Level 3) Location: Napa

(Level 4) Week_Ending: 2000-04-01

Product	Units	Rank	% Total	RunningTotal	RunningCount
Caramel Suckers	72	18	2.8%	0	1
Coffee Suckers	55	23	2.1%	306	6
Dark Chocolate Blocks	49	27	1.9%	527	10
Milk Chocolate Blocks	120	7	4.7%	731	15
Milk Chocolate Blocks w Almonds	151	2	5.9%	1,245	20
Milk Chocolate Bunny	97	11	3.8%	1,863	25
White Chocolate Blocks	73	17	2.8%	2,274	30
	617	105	23.9%	6,946	107

(Level 4) Week_Ending: 2000-04-08

Product	Units	Rank	% Total	RunningTotal	RunningCount
Caramel Suckers	82	13	3.2%	147	4
Coffee Suckers	60	21	2.3%	411	8
Dark Chocolate Blocks	54	24	2.1%	626	13
Milk Chocolate Blocks	141	4	5.5%	976	18
Milk Chocolate Blocks w Almonds	167	1	6.5%	1,545	23
Milk Chocolate Bunny	111	9	4.3%	2,061	28
White Chocolate Blocks	80	14	3.1%	2,423	33
	695	86	26.9%	8,189	127
	1,312	191	50.9%	15,135	234

---End of Location: Napa

If GroupBlox is added AFTER the CalculateBlox, these summary columns are treated as regular numeric columns with fixed values. The Rank column now has a total of 191, and data in the other three columns are meaningless as the values are based on the entire report without any break group.



The above example uses FormatBlox to format the data in the % Total column, generated using the `percentOfTotal()` function.

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 107.

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>
```

all_columns				
Product	Country	State	Sales	Units
Piano	USA	NY	1,000	10
Violin	USA	NY	2,000	20
Piano	USA	CA	3,000	30
Violin	USA	CA	4,000	40
Piano	Canada	BC	5,000	50
Piano	USA	CA	6,000	60
Violin	Canada	Victoria	-1,000	-10
Harp	USA	TX	#Missing	#Missing
			20,000	28.571

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.

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.

Contents

- “Issues with Saving Interactive Reports Directly from Browser” on page 118
- “Saving as Static HTML to File System” on page 119
- “Bookmarking Reports and Saving States” on page 121
- “Saving as PDF” on page 125
- “Saving to Excel or Other Applications” on page 127

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		
IE	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 IE.
Netscape		The report is saved without the style sheets and therefore may look different.
Interactive Reports		
IE	Save as complete Web page	An error message saying the Web page cannot be saved.
	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.
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 v5 or later. 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 Alphablox Analytics 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 ){
    ReportBlox reportBlox = (ReportBlox)session.getAttribute(
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 stylesheet 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 Alphablox Analytics 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/report/
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.
        ReportBlox report = (ReportBlox)
        session.getAttribute("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 Alphablox Analytics 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 Alphablox Analytics 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/report/
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>

```

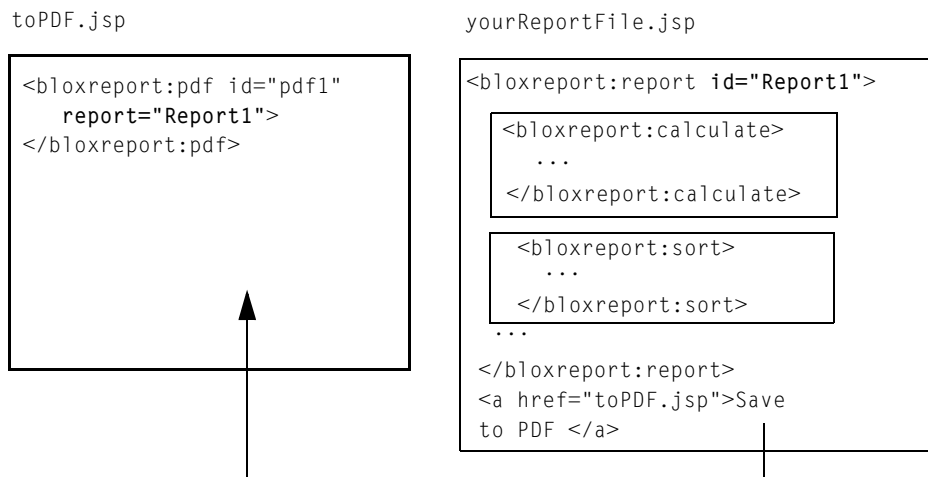
It's important to know that when a bookmarked report is loaded, `PersistenceBlox` creates an instance of the `ReportBlox` object on the server. If an instance already exists in the session, `PersistenceBlox` will overwrite the old instance. The `Saving and Exporting Data with Dynamic Queries` example 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 not 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 rendered PDF has fixed font faces, styles, colors, and sizes. They cannot be set programmatically.
- The content and position of the rendered footer and header cannot be customized. The footer is always at the lower right corner and shows the current page number and the total page number. The header includes the specified logo and a date. The logo will appear at the upper left corner and the date will appear at the upper right corner.

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 the following:

- 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 119.

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 ) {
    ReportBlox reportBlox = (ReportBlox)session.getAttribute(
```



```

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

- This stylesheet should not import other stylesheets as Excel cannot resolve the imports.
- If you want to use the provided `report.css` stylesheet at `/AlphabloxServer/report/`, since this stylesheet actually imports three other stylesheets (`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/report/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 stylesheet import statements.

- If you are using the provided `coleman.css` stylesheet in `/AlphabloxServer/report/`, since it imports styles from `report.css`, you should make a custom version of `coleman.css` to eliminate the need to import other stylesheets. This may involve copying style definitions from `styles.css` into your custom version and modifying them.

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.

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.

Contents

- “Style Classes in the Report Editor User Interface” on page 132
- “User Help for Using the Report Editor” on page 135

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.

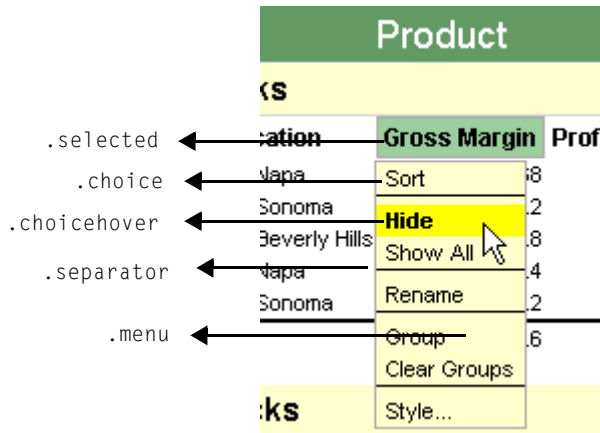
The style classes for the context menus are:

- .menu
- .choice
- .choicehover
- .separator
- .selected

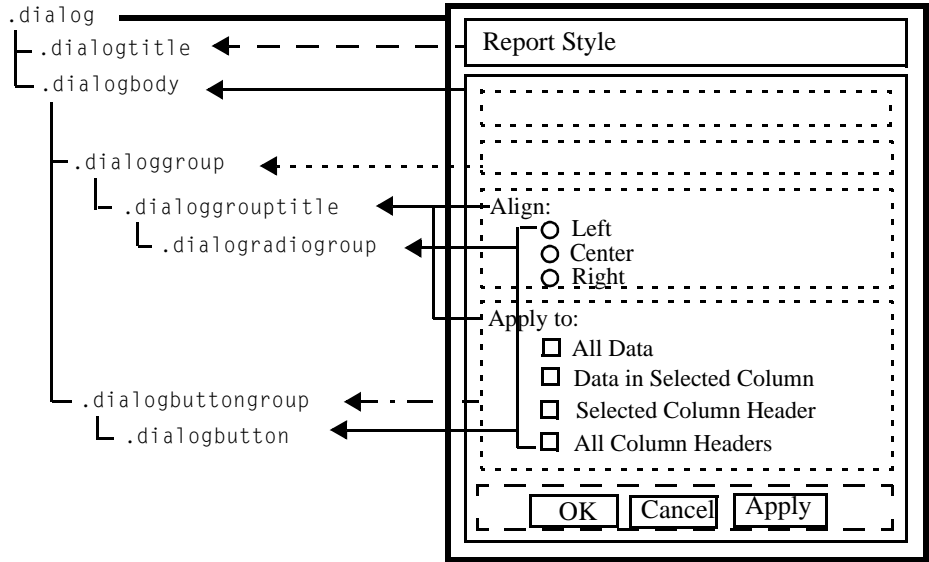
The style classes related for the Report Style dialog box are:

- .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 checkbox
.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;
}

```



When setting font faces, styles, and sizes in the stylesheets, 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 stylesheet makes your JSP page cleaner and allows you to reuse the same stylesheet for your entire application. A good technique to use is to create your external stylesheet that imports the stylesheets supplied by Alphablox Analytics 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 stylesheet.

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

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

In the beginning of your `myreportbuilder.css`, import the stylesheets that come with Alphablox Analytics:

```
@import url( /AlphabloxServer/report/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 stylesheets. Keep in mind that you should never modify the stylesheets in the `/AlphabloxServer/report/` directory since they get wiped out during Alphablox Analytics 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. User help files are provided for you in four languages—English, French, German, and Japanese—if you want to offer them to your users. 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 151.

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.

Contents

- “Managing Session Scope” on page 138
- “The Relational Reporting API” on page 139
- “Dynamically Changing the Query” on page 142
- “Accessing Data Rows and Cell Values in Rendered Report” on page 147

Managing Session Scope

When you use Blox tags to add Relational Reporting Blox to your JSP page, these Blox have a session scope by default. As demonstrated in the following example, 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.

The screenshot shows the 'Product Sales Report Builder' interface. On the left, there is a control panel with the following settings:

- Query: Sales by Product
- Calculate: Profit%
- Sort by: Location
- Order: ascending
- Interactive:
- Generate Report button

The main report area displays the following data:

Sales by Product						
Milk Chocolate Blocks						
Week_Ending	Area	Location	Cost	Units Sold	Sales	Profit%
4/8/00	S. Cal.	Beverly Hills	\$7.20	12	\$24.00	.70
4/1/00	N. Cal.	Napa	\$72.00	120	\$240.00	.70
4/8/00	N. Cal.	Napa	\$84.60	141	\$282.00	.70
4/1/00	N. Cal.	Sonoma	\$67.80	113	\$226.00	.70
4/8/00	N. Cal.	Sonoma	\$76.80	128	\$256.00	.70
total:			\$308.40	total: 514	total: \$1,028.00	.70
Dark Chocolate Blocks						
Week_Ending	Area	Location	Cost	Units Sold	Sales	Profit%
4/8/00	S. Cal.	Beverly Hills	\$3.00	5	\$10.00	.70

Each time a new set of criteria is specified and a new report is requested, you will need to remove the server object that already exists since no new object will be instantiated. This can be done through the standard `removeAttribute()` method:

```
<% session.removeAttribute( "myReportBlox" ); %>
<bloxreport:report id="myReportBlox" >
...

```

Since deleting server objects and re-instantiating them could be a memory intensive operation, you should use this cautiously and only when it is needed in a production environment. A more efficient approach is to reuse, rather than remove, the objects in the session.

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 is used as the scripting variable name in your JSP, `bloxName` can be dynamically assigned and 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. The following example shows how a `bloxName` will be different every time a page is loaded or refreshed:

```
<% String bloxName="report"+System.currentTimeMillis(); %>
<bloxreport:report id="myReportBlox"
  bloxName="<%= bloxName %>" >
...

```

`bloxName` is useful as a development tool so you do not need to close and restart the browser windows everytime you make a change to your JSP. 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 119, "Saving Reports as PDF Files" on page 125, and "Exporting to Excel" on page 127.

The Relational Reporting API

As described earlier in "Report Pipeline" on page 33, 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 *ICustomer* and *IProducer* interfaces. *ICustomer* 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*.

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/report/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 [] {"locatioin"});
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 interactivity. 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.

If, for example, the `salesReport.jsp` is located at `http://myServer/myApp/Sales/East/salesReport.jsp`, you should set the URL prefix as follows:

```
rBlox.setUrlPrefix("/myApp/Sales/East");
```

- 3 Place the ReportBlox into the session. 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 session.

```
session.setAttribute(bloxName, rBlox);
```

The complete example is now as follows:

```
<%@ page import="com.alphablox.blox.*" %>
<html>
<head>
  <link rel="stylesheet" href="/AlphabloxServer/report/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");

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

    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 [] {"locatioin"});
    myGroup.setAggregationType("units", "none");

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

```

        rBlox.setInput(myGroup);

        // 2. Set the URL prefix
        rBlox.setUrlPrefix("/myApp/Sales/East");

        // 3. Add the ReportBlox to the session
        session.setAttribute(bloxName, rBlox);

        // 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.

Dynamically Changing the Query

All the relational reporting Blox are instantiated in the session 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 will need to remove or reuse the objects from the server. To remove an object from the session, you can use the standard `removeAttribute()` method of the session object:

```
<% session.removeAttribute( "yourReportBlox" ); %>
```

This, however, is a memory intensive and expensive operation. 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 session
    // object of same name. This will never happen in the first
    // pass through this page
    monthlyData = (SQLDataBlox) session.getAttribute( "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 stylesheet; required for interactive
  mode-->
  <link rel="stylesheet"
  href="/AlphabloxServer/report/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>

```



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 Alphablox Analytics 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 a live example, see the Dynamic Queries example in the Blox Sampler - Relational Reporting example set. Below is the code:

```

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

<%
String reportName = "qcc";
String weekEnding = "'2000-04-08'";
String query = "SELECT scenario, location, product_name, sales,
units, cost " +
"FROM qcc " +
"WHERE week_ending = " + weekEnding + " " +
" AND scenario = 'Actual' " +
"ORDER BY location, product_name";
session.setAttribute( "weekEnding", weekEnding );
%>

<html>
<head>
  <link rel="stylesheet" href="/AlphabloxServer/report/report.css"
/>
  <script>
    function refresh( reportName ) {
      refreshReport( reportName );
    }
  </script>
  <!-- remove the style tag below to have the report not be
  contained in a scrolling iframe -->
  <style> .report { height:400; width: 600 } </style>
</head>
<body>
<h3>QCC data for week of <%=weekEnding%> </h3>

```



```

        catch (Exception e)
        {
            isError = true;
            errorMessage = e.toString();
            System.out.println("[execQuery.jsp] Error: " + errorMessage);
        }
    %>

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

```

The displayed report looks as follows:

	Product	Units	Sales	% Total	Rank
?	210	\$120.00	\$240.00	11%	5
?	220	\$49.00	\$98.00	4%	7
?	230	\$73.00	\$146.00	7%	6
?	240	\$151.00	\$377.50	17%	3
?	252	\$97.00	\$242.50	11%	4
?	431	\$72.00	\$648.00	29%	1
?	432	\$55.00	\$495.00	22%	2
		\$617.00	\$2,247.00	100%	1

When users click on the info 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 example in Blox Sampler - Relational Reporting demonstrates this and other JavaScript techniques.

Development and Troubleshooting Tips

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

Contents

- “General Tips and Development Steps” on page 150
- “Design Considerations” on page 150
- “Providing User Help” on page 151
- “Impact of Style Setting on Performance” on page 152
- “Common Reporting Blox Tag Errors” on page 153
- “Troubleshooting Tips” on page 155
- “Error Handling Using ErrorBlox” on page 156

General Tips and Development Steps

Make sure you check out the “General Report Development Steps” on page 54 for the essential steps to create any relational report. Then go over the “General Development Tips” on page 52 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 152)
- 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 151.

Providing User Help

User help files that explain how the Report Editor user interface works are available in four languages: English, French, German, and Japanese. 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:

```
<alphanblox_dir>\system\documentation\help\ReportBlox\{en,
fr,de,ja}
```

where `<alphanblox_dir>` is the directory in which Alphablox Analytics is installed.

The end user help consists of a set of HTML pages, a stylesheet, 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');">
</a>
```



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

```
<a href="javascript:openHelp('/AlphanbloxServer/documentation/help/
ReportBlox/{en,fr,de,ja}/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 Alphablox Analytics. 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, 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 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 Reporting Blox Tag Library.

Forgetting to include the taglib directive for Reporting Blox Tag Library

If you forget to place the taglib directive for the Reporting Blox 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 directory for the Blox Tag Library (`blox.tld`) and try to use the `blox` prefix for your relational report, you will get a JSP error. For example, with the `<blox:report>` tag, you will get:

```
No such tag report in the tag library imported with prefix blox
```

Since the Reporting Blox 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 stylesheet

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 stylesheets cascade, as a best practice, use the stylesheet provided out-of-the-box and then add your own stylesheet (or use inline styles) to modify only the styles for classes you want to change:

```
<link rel="stylesheet" href="/AlphabloxServer/report/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. To test page modifications within Blox tags, one technique is to use the `removeAttribute()` method associated with the `session` object. Another technique is use a dynamic `bloxName` for your `ReportBlox`. All Blox in Relational Reporting have a `bloxName` attribute that allows you to dynamically assign a name to the Blox to effectively reuse the object on the server. For details, check out the sections on “Managing Session Scope” on page 138.

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 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 34, “Member Identifiers vs. Display Names” on page 49

Troubleshooting Tips

Messages from `ReportBlox` are sent as `DEBUG` level messages to the `Alphablox Analytics` 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 56.

The following are some general troubleshooting tips:

- Make sure the stylesheet referenced actually exists and is in the correct location. Different browsers have different tolerance for missing stylesheets.
- 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 stylesheet associated with the report, or the stylesheet referenced is not found. Add a link to your stylesheet or to the stylesheet supplied:

```
<link rel="stylesheet" href="/AlphabloxServer/report/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`).



If you are using IBM DB2 Universal Database, Oracle, or MSSQL 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 a “404—Page Not Found” error.

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-existing 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/report/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>

```


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 Javadoc at:

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

Contents

- “Using Blox Tags” on page 160
- “CalculateBlox” on page 161
- “DataSourceConnectionBlox” on page 163
- “ErrorBlox” on page 165
- “FilterBlox” on page 166
- “FormatBlox” on page 168
- “GroupBlox” on page 172
- “MembersBlox” on page 176
- “OrderBlox” on page 178
- “PdfBlox” on page 180
- “PersistenceBlox” on page 182
- “RDBResultSetDataBlox” on page 184
- “ReportBlox” on page 186
- “SortBlox” on page 189
- “SQLDataBlox” on page 191
- “StyleBlox” on page 193
- “TextBlox” on page 197

Using Blox Tags

The custom Blox tags for Relational Reporting work similarly to the tags for the other Blox in Alphablox Analytics:

- 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 205.

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 110.

- 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.

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 138.
expression	No	Sum of all numeric columns in a new column named “Total”	<p>The expression CalculateBlox evaluates to add the named calculated member.</p> <p>An invalid or empty expression results in an error.</p> <p>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.</p> <p>For valid expression syntax, see “Expression Syntax” on page 48.</p>
index	No	member count	<p>The position in the Column dimension where the calculated member appears, with 0 being the first member.</p> <p>By default calculated members are added to the end.</p>

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 71, “Calculating Group-based Summary Columns” on page 110.

DataSourceConnectionBlox

DataSourceConnectionBlox represents a connection to a relational data source defined to Alphablox Analytics. 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.

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 138.
dataSourceName	No		The name of the relational data source as defined via the Alphablox Analytics home pages. A non-relational data source results in an error.
userName	No		The username for accessing the data source specified.

Tag Attribute	Required	Default	Description
password	No		<p>The password for the username specified to access the data source.</p> <p>If password is an empty string, then password="".</p> <p>If password is blank or null, do not specify a password.</p>

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 56 for details on how to create an error reporting JSP. See “Error Handling Using ErrorBlox” on page 156 for more details on using ErrorBlox to catch exceptions.

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 138.

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/report/
error.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]]`.

The `<bloxreport:filter>` Tag

Tag Attribute	Required	Default	Description
<code>id</code>	No		The unique identifier for this instance of FilterBlox.
<code>bloxName</code>	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 138.

Tag Attribute	Required	Default	Description
expression	Yes		<p>The criteria for filtering.</p> <p>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:</p> <pre>expression = "Sales > 3000" expression = "[Product Code] != 200"</pre> <p>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.</p> <p>One function is available for filter expressions: <code>isMissing(memberName)</code>.</p> <p>See “Expression Syntax” on page 48 for discussions on expression syntax and how to specify member names if the names contain special characters or spaces.</p>

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 “Formatting the Report and Data” on page 79 for tasks related to report layout formatting. See “StyleBlox” on page 193 for tags that allow you to specify styles such as font size, background color, and text alignment.

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 138.
<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>			

Tag Attribute	Required	Default	Description
member	No		<p>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 89 for an example and more details on using a member's display name.</p> <p>You can have multiple <code><bloxreport:html></code> 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:</p> <pre>member=""</pre>
<code><bloxreport:aggregation></code>			
member	No		A member's display name.
format	Yes		The format to apply to aggregation values (group totals) for the specified member. If <code>member</code> 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 34 and “Styling Data Displayed in Report” on page 90.

The <bloxreport:group> Tag

Tag Attribute	Required	Default	Description
<bloxreport:group>			
id	No		The unique identifier for this instance of GroupBlox.
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 138.
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.

Tag Attribute	Required	Default	Description
type	No	sum	<p>Type of aggregation. Valid values are:</p> <ul style="list-style-type: none"> • sum • average • count • max • min • none <p>Note that:</p> <ul style="list-style-type: none"> • The default aggregation type for all numeric data columns is <code>sum</code>. • If no type is specified, the default is <code>sum</code>. • Type <code>count</code> 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. The following is a sample output.

all_Product			
Piano			
Country	State	Sales	Units
USA	NY	1,000	10
USA	CA	3,000	30
Canada	BC	5,000	50
USA	CA	6,000	60
		15,000	37.5
Violin			
Country	State	Sales	Units
USA	NY	2,000	20
USA	CA	4,000	40
Canada	Victoria	-1,000	-10
		5,000	16.667

MembersBlox

MembersBlox allows you to specify which members to include in or exclude from a report. Excluded members no longer exist in the resultset. 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 resultset. To temporarily hide a member or members, use OrderBlox.

The `<bloxreport:members>` Tag

Tag Attribute	Required	Default	Description
<code>excluded</code>	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 49 for detail.
<code>id</code>	No		The unique identifier for this instance of MembersBlox.
<code>bloxName</code>	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 138.

Tag Attribute	Required	Default	Description
included	No		<p>Members to be included. Members not in the list are excluded.</p> <p>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 49 for detail.</p>

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 178

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 resultset. 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.

The `<bloxreport:order>` Tag

Tag Attribute	Required	Default	Description
<code>excluded</code>	No		<p>Members to be excluded in the ordering.</p> <p>Unlike excluded members via MembersBlox, excluded members are still in the result set, available for subsequent data transformation actions in the pipeline.</p> <p>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 49 for detail.</p>
<code>id</code>	No		The unique identifier for this instance of OrderBlox.

Tag Attribute	Required	Default	Description
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 138.
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 49 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 176

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.

The `<bloxreport:pdf>` Tag

Tag Attribute	Required	Default	Description
<code>id</code>	Yes		The unique identifier for this instance of PdfBlox.
<code>bloxName</code>	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 138.
<code>bottom</code>	No	<code>1in</code>	Specifies the bottom margin. Units can be <code>in</code> , <code>cm</code> , <code>px</code> , or <code>pts</code> . Note: there should be no space in between the number and the unit in your specification.
<code>footerVisible</code>	No	<code>true</code>	Specifies if the footer (page numbers) should be visible. The default is <code>true</code> .
<code>headerVisible</code>	No	<code>true</code>	Specifies if the header (logo, time, and date) should be visible. The default is <code>true</code> .
<code>height</code>	No	<code>11in</code>	Specifies the height of the page. Units can be <code>in</code> , <code>cm</code> , <code>px</code> , or <code>pts</code> . The default is <code>11in</code> .
<code>left</code>	No	<code>1.25in</code>	Specifies the left margin. Units can be <code>in</code> , <code>cm</code> , <code>px</code> , or <code>pts</code> . The default is <code>1.25in</code> .

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 <code></code> are not included in the PDF render. You must use this attribute to explicitly specify the source image URL. The URL must be a complete URL using the HTTP protocol: <code>http://<servername>/<path>/mylogo.gif</code>
portrait	No	true	Specifies if the page should be rendered in portrait. <code>False</code> will result in the page rendered in landscape. The default is <code>true</code> , with the report rendered in portrait.
report	No		Specifies the instance of ReportBlox (the ReportBlox id) to render to PDF.
right	No	1.25in	Specifies the right margin. Units can be <code>in</code> , <code>cm</code> , <code>px</code> , or <code>pts</code> . The default is <code>1.25in</code> .
top	No	1in	Specifies the top margin. Units can be <code>in</code> , <code>cm</code> , <code>px</code> , or <code>pts</code> . The default is <code>1in</code> .
width	No	8.5in	Specifies the width of the page. Units can be <code>in</code> , <code>cm</code> , <code>px</code> , or <code>pts</code> . The default is <code>8.5in</code> .

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.

The `<bloxreport:persistence>` Tag

Tag Attribute	Required	Default	Description
<code>id</code>	No		The unique identifier for this instance of PersistenceBlox.
<code>bloxName</code>	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 138.
<code>location</code>	Yes		The folder name under the repository's <code>reportingpersistence/</code> folder. You can specify sub-folders such as “ <code>sales/east</code> ”, which creates an <code>east/</code> folder under <code>sales/</code> under the <code>reportingpersistence/</code> folder in the repository if they do not already exist.
<code>operation</code>	Yes		The operation to perform on the bookmark specified (via the <code>persistedName</code> attribute). Valid values are <code>save</code> and <code>load</code> .
<code>persistedName</code>	Yes		The name of the bookmark.

Tag Attribute	Required	Default	Description
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 drillthrough, since there may be multiple reports defined for a cell, use drillThroughReportName to specify the report of interest.

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 138.
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 drillthrough report.
rowCoordinate	No		The row coordinate of the specified cell.

Example 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 63.

See Also The DrillThrough example in Blox Sampler’s Retrieving Data section (MSAS version) and the Retrieving Data chapter in the *Developer’s Guide for the DHTML Client*.

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 34 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` and one `DataSourceConnectionBlox`. Or it can take an `RDBResultSetDataBlox`, 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`.

The `<bloxreport:report>` Tag

Tag Attribute	Required	Default	Description
<code><bloxreport:report></code>			
<code>id</code>	No		The unique identifier for this instance of <code>ReportBlox</code> if <code>bloxName</code> is not specified. If <code>bloxName</code> is specified, <code>id</code> is the local Java scripting variable name. For details, see the <code>bloxName</code> entry in <i>Developer's Reference for the DHTML Client</i> .
<code>bloxName</code>	No		The unique identifier for this instance of <code>ReportBlox</code> on the server that allows you to dynamically set its name. See “Managing Session Scope” on page 138 and the <code>bloxName</code> entry in <i>Developer's Reference for the DHTML Client</i> .

Tag Attribute	Required	Default	Description
errors	No	false	<p>Intercepts exceptions thrown.</p> <p>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 <code>noDataMessage</code> and <code>noDataDueToErrorMessage</code> attributes.</p> <p>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 <code>true</code>. See “Error Handling Using ErrorBlox” on page 156 for more information on tracking the root cause of an exception.</p>
interactive	No	false	<p>The attribute to set whether the report should be rendered in DHTML table for interactive report editing.</p> <p>The default value is <code>false</code>, that is, there will not be the interactive context menus available for users.</p> <p>When <code>interactive</code> is set to <code>true</code>, a stylesheet 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 34 for details on styles.</p>
noDataMessage	No	No data	<p>The message displayed in the rendered report when the ReportBlox returns no data. Note that <code>errors</code> attribute needs to be set to <code>false</code> (the default) or exceptions will be displayed.</p>

Tag Attribute	Required	Default	Description
<code>noDataDueToErrorMessage</code>	No	An error occurred while generating report data	<p>The message displayed in the rendered report when errors occur during data transformation in the pipeline.</p> <p>When the ReportBlox returns no data due to such an exception, users will see both <code>noDataMessage</code> and <code>noDataDueToErrorMessage</code> in one string: “No data: An error occurred while generating report data.”</p>

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.

The `<bloxreport:sort>` Tag

Tag Attribute	Required	Default	Description
<code><bloxreport:sort></code>			
id	No		The unique identifier for this instance of SortBlox.
bloxName	No		The unique identifier for this instance of SortBlox on the server that allows you to dynamically set its name. See “Managing Session Scope” on page 138.
<code><bloxreport:sort></code> , <code><bloxreport:rule></code>			

Tag Attribute	Required	Default	Description
ascending	No	true	Whether to sort ascendingly.
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 49 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.

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 138.
query	Yes		The SQL command to extract data from the relational data source. The data source should be specified using the <code><bloxreport:dataSourceConnection></code> tag, a tag that nests within <code><bloxreport:sqlData></code> .

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>
```

```
</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 stylesheet, and use the StyleBlox to set all data in text columns to be black, then the data will be in black.

<bloxreport:style> Tag and Its Sub Tags

Tag Attribute	Required	Default	Description
<code><bloxreport:style></code>			
<code>id</code>	No		The unique identifier for this instance of StyleBlox.
<code>bloxName</code>	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 138.
Nested <code>text</code> , <code>negative</code> , <code>banding</code> Tags			
<code>style</code>	Yes		The style to apply. Specify the style using CSS attributes in the format of attribute: value pair within the quotes, separated with “;”.

Tag Attribute	Required	Default	Description
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.



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:style>
```

```

<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

Below is the sample output:

Type	Country	State	City	Revenue
Deluxe Supermarket	Canada	BC	Vancouver	23,112
Deluxe Supermarket	Mexico	DF	San Andres	#Missing
Deluxe Supermarket	Mexico	Yucatan	Merida	30,797
Deluxe Supermarket	Mexico	Zacatecas	Hidalgo	30,584
Deluxe Supermarket	USA	OR	Salem	27,694
Deluxe Supermarket	USA	WA	Tacoma	33,858
Gourmet Supermarket	Mexico	Zacatecas	Camacho	23,759
Gourmet Supermarket	USA	CA	Beverly Hills	23,688
Mid-Size Grocery	Canada	BC	Victoria	34,452
Mid-Size Grocery	Mexico	DF	Mexico City	36,509

Note: The text displayed for missing value is set using FormatBlox.

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.

Nested Tags Inside `<bloxreport:text>`

The following table describes TextBlox's sub tags and their attributes.

Attribute	Required	Default	Description
Nested <code>columnHeader</code> and <code>data</code> Tags			
<code>columnName</code>	Yes for <code>columnHeader</code> ; No for <code>data</code>		The name of the column this column header text or data text should be applied to.
<code>text</code>	Yes	<code><member/></code> for <code>columnHeader</code> ; <code><value/></code> for <code>data</code>	<p>The text to display for the named column, or for all columns if <code>columnName</code> is not specified. If <code>text</code> is not specified, the entire <code>columnHeader</code> and <code>data</code> tags will be ignored.</p> <p>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 Alphablox Relational Reporting engine are the two substitution variables—<code><member/></code> and <code><value/></code>. See “Special Substitution Variables for Displaying Member Names and Values” on page 97.</p>
Nested <code>groupFooter</code> , <code>groupHeader</code> , and <code>groupTotal</code> Tags			
<code>columnName</code>	No		<p>Applies to the <code>groupTotal</code> tag only.</p> <p>The name of the column this column header text should be applied to.</p>

Attribute	Required	Default	Description
level	No		<p>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).</p> <p>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.</p> <p>If level is not specified, the specified footer, header, or total text and style will be applied to all grouping levels.</p>
text	Yes	<code><member/></code> for <code>groupHeader</code> and <code>groupFooter</code> ; <code><value/></code> for <code>groupTotal</code>	<p>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.</p> <p>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 Alphablox Relational Reporting engine are the two substitution variables—<code><member/></code> and <code><value/></code>. See “Special Substitution Variables for Displaying Member Names and Values” on page 97.</p>



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 Alphaslo 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 <code>groupHeader</code> , <code>groupFooter</code> , or <code>groupTotal</code> 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 Alphaslo 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 `TextBox`'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 curent 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 97.

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>

```

Below is a sample output:

Profitability by Product					
Product: Caramel Suckers					
Week_Ending	Location	Cost	Units Sold	Sales	
4/1/00	Napa		203.4	72	648
4/1/00	Sonoma		191.76	68	612
4/8/00	Beverly Hills		19.74	7	63
4/8/00	Napa		231.24	82	738
4/8/00	Sonoma		217.14	77	693
			total: 863.28	total: 306	total: 2,754
Product: Coffee Suckers					
Week_Ending	Location	Cost	Units Sold	Sales	
4/1/00	Napa		155.1	55	495
4/1/00	Sonoma		141	50	450
4/8/00	Beverly Hills		4.2	7	14
4/8/00	Napa		169.2	60	540
4/8/00	Sonoma		157.92	56	54
			total: 623.22	total: 221	total: 1,539
⋮					
Product: White Chocolate Blocks					
Week_Ending	Location	Cost	Units Sold	Sales	
4/1/00	Napa		43.8	73	146
4/1/00	Sonoma		41.4	69	138
4/8/00	Beverly Hills		4.2	7	14
4/8/00	Napa		48	80	160
4/8/00	Sonoma		46.2	77	154
			total: 183.6	total: 306	total: 612
			Grand Total Cost: 2,819.1	Overall Total Units: 2,580	Grand Total Sales: 8,913.5

See the Grouping and Adding Group-based Summary Columns section in BloX Sampler - Relational Reporting for an example. This section also has an example that demonstrates a use case when `<value/>` is used in group headers to produce a report as follows:

Product Sales by Location				
Location: Napa				
				Rank: 1 — (1)
Napa: 2000-04-01				
				Product count: 7 — (2)
Product	Code	Sales	Rank	
Milk Chocolate Blocks	210	\$240	17	
Dark Chocolate Blocks	220	\$98	26	
White Chocolate Blocks	230	\$146	22	
Milk Chocolate Blocks w Almonds	240	\$378	9	
Milk Chocolate Bunny	252	\$242	16	
Caramel Suckers	431	\$648	3	
Coffee Suckers	432	\$495	6	
		\$2,247	2	
Napa: 2000-04-08				
				Product count: 7 — (2)
Product	Code	Sales	Rank	
Milk Chocolate Blocks	210	\$282	12	
Dark Chocolate Blocks	220	\$108	24	

- 1 This report adds ranking information to each level 2 group header to identify the location that made the most sales. The calculation of ranking is set to level 1 so the group total value for the calculated member "Rank" will be available in the group header.
- 2 A product count is added to indicate the number of products carried in each location for each week. This is done by calculating a temporary column using the `runningCount` calculation function and later use the TextBlox's `data` tag to set the text in the column to be blank.

See the Grouping and Adding Group-based Summary Columns section in Blox Sampler - Relational Reporting for the complete code.

A

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 205
- “All Tag Attributes for PdfBlox” on page 207

All Tags Nested Within `<bloxreport:report>`

```
<bloxreport:report id="reportId"
  bloxName="bloxName"
  errors="true|false"
  interactive="true|false"
  noDataMessage="messageText"
  noDataDueToErrorMessage="messageText">

  <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"  
footerVisible = "true"  
headerVisible = "true" >  
report="ReportBlox_id"  
</bloxreport:pdf>
```



The attribute values shown are the default unless they are in italic.

B

Deprecated Tags for Relational Reporting

This section lists deprecated tags and the replacements for the deprecated functionality.

Deprecated tags and APIs receive support for a limited time but are no longer a part of strategic product direction. IBM recommends eliminating their use as soon as possible. Unless explicitly stated otherwise, a deprecated tag or API receives support for three major releases, including the one in which the release notes announced its deprecation. Major releases are, for example, 5.0.0 or 5.5.0. Minor releases are, for example, 5.0.1.

Contents

- “Deprecated Tags and Attributes in Release 5.5” on page 210

Deprecated Tags and Attributes in Release 5.5

Deprecated Tags/Attributes	New Tags/Attributes
<p>The following nested tags inside the <code><bloxreport:report></code> tag are deprecated:</p> <p>columnText footerText headerText</p>	<p>Use <code>TextBlox</code> (<code><bloxreport:text></code>) and its nested tags:</p> <p>columnHeader groupFooter groupHeader</p> <p>See “TextBlox” on page 197 for details.</p>
<p>The following attribute for nested tags for <code>StyleBlox</code> are deprecated:</p> <p>member</p>	<p>Use <code>columnName</code> instead.</p> <p>Example of new attribute:</p> <pre data-bbox="627 638 1000 793"><bloxreport:style> <bloxreport:column style="" columnName="" /> <bloxreport:data style="" columnName="" /> </bloxreport:style></pre>

Index

B

- Blox tags
 - Relational Reporting Blox, listing 205
- Blox tags, *see* tags
- Blox tags, using 160
- bookmark
 - relational reports 121
- break group
 - totals, specifying 107
- break groups
 - footers, specifying 107
 - headers, specifying 107
 - style classes, listing 35
- browsers
 - Relational Reporting 30

C

- CalculateBlox
 - definition 25
 - percentOfTotal() function 113
 - rank() function 113
 - runningCount() function 113
 - runningTotal() function 113
 - tag reference 161
- calculated members
 - adding, in relational reports 71
- calculating data
 - in relational reports 71
- cell banding
 - setting colors 100
 - turning off 100
- cell banding, Relational Reporting 90

- cell values, accessing 147
- column header
 - Context Menu 28
- column headers
 - Context Menu 28
- column headers, renaming and formatting 93
- components
 - Relational Reporting 23
- concepts
 - Relational Reporting 32
- custom tags
 - see* Blox Reporting Tag Library

D

- data
 - accessing, Relational Reporting 54
 - display style 90
 - filtering 70
 - formatting 87
 - hiding and showing members, in relational reports 75
 - removing members, in relational reports 74
 - sorting, in relational reports 68
- data columns
 - formatting 96
 - style classes in Relational Reporting, listing 35
 - width, color, and style, setting 96
- data columns, renaming and formatting column headers 93
- data row, accessing 147

- data sources
 - Relational Reporting, defining access to 54
- DataSourceConnectionBlox
 - definition 24
 - tag reference 163
- design considerations
 - Relational Reporting 150
- display areas 80
- display names 49

E

- error reporting
 - ErrorBlox, using 56
- ErrorBlox
 - definition 26
 - getNextException() method 156
 - style classes 41
 - tag reference 165
 - using 56
- ErrorBlox()
 - getRootCause() method 156
- Excel, exporting to 127
- exporting, to Excel 127
- exporting, to static HTML 119

F

- FilterBlox
 - definition 25
 - tag reference 166
- filtering
 - data, in relational reports 70
- footer, for break groups 107
- FormatBlox
 - definition 25
 - HTML formatting, adding 89
 - tag reference 168
- functions
 - percentOfTotal() 113
 - rank() 113
 - runningCount() 113
 - runningTotal() 113

G

- getData() method 139

- group header
 - Context Menu 28
- group headers
 - Context Menu 28
- group total
 - Context Menu 29
- group totals
 - Context Menu 29
- GroupBlox
 - definition 25
 - tag reference 172

H

- header
 - break group headers, setting 107
 - column headers, renaming and formatting 93
- HTML code, adding to data returned 89

I

- interactive
 - context menus, style classes 36
 - relational reports, essential steps to creating 58

L

- localization
 - Relational Reporting 23

M

- member names, variable 97
- member values, variable 97
- members
 - definition of, in relational reports 34
 - member identifier vs. display names 49
- MembersBlox
 - definition 25
 - tag reference 176
- message logging
 - Relational Reporting 155

- missing data
 - calculation, involving 73
 - display text, setting 87
 - hiding or showing, in relational reports 76

O

- OrderBlox 25
 - definition 25
 - tag reference 178

P

- PDF, saving reports to 125
- PdfBlox
 - definition 25
 - tag reference 180
- percentOfTotal() function 113
- performance
 - relational report rendering 152
- persistence
 - bookmarking, relational reports 121
- PersistenceBlox
 - definition 25
 - tag reference 182
 - usage 121

R

- rank() function 113
- RDBResultSetDataBlox
 - definition 25
 - tag reference 184
- refreshReport() JavaScript method 144

Relational Reporting

- Column Header Context Menu 28
- components 23
- concepts 32
- custom tags, general syntax 43
- custom tags, nested 44
- default stylesheets 34
- design considerations 150
- development steps, general 54
- development tips 52
- error reporting 56
- expression syntax, evaluating 46
- Group Header Context Menu 28
- Group Total Context Menu 29
- localization 23
- overview 22
- rendering 26
- rendering options 32
- Report Editor user interface 26
- Report Style dialog box, listing of style
 - classes 36
- simplest interactive report, creating 58
- simplest report, creating 57
- style classes, listing 35
- relational reports 70
 - background image, adding 102
 - bookmarking 121
 - creating, essential steps to 57
 - data columns, setting styles 96
 - exporting to Excel 127
 - exporting, to static HTML 118
 - images, adding 102
 - performance considerations, using
 - StyleBlox 152
 - rendering 26
 - rendering to PDF 29
 - report display area, setting 101
 - saving in browsers, options 118
 - saving, as HTML tables 119
 - saving, in PDF 125
 - sorting data 68
 - state, saving and retrieving 121
 - styles, defining 38
 - text alignment, default 91
- rendering to PDF 29

Report Editor

- Column Header Context Menu 28
- Group Footer Context Menu 29
- Group Header Context Menu 28
- specifying styles 132
- style classes, listing 133
- user interface, Relational Reporting 26

Report Style dialog box

- style classes 41
- style classes, listing 36

ReportBlox 24

- definition 24
- tag reference 186

runningCount() function 113**runningTotal() function 113****S****session scope**

- dynamic reports 138
- Relational Reporting tags 48
- terminating 138

setInput() method 139**SortBlox**

- definition 25
- tag reference 189

sorting, in relational reports 68**SQLDataBlox**

- definition 24
- setting queries, dynamically 142
- tag reference 191

style classes

- break groups, listing 35
- column and data, listing 35
- ErrorBlox 41
- interactive context menus, listing 36
- relational reports 35
- Report Style dialog box, listing 36

StyleBlox

- definition 25
- tag reference 193

stylesheets

- referencing, in Relational Reporting 55

styling

- relational reports 34

T**tags**

- general rules
- nested tags, Relational Reporting 44
- Relational Reporting vs. other Blox 160
- syntax, Relational Reporting 43

text alignment

- relational reports, default 91

TextBlox

- definition 25
- tag reference 197

troubleshooting

- Relational Reporting 155

U**user help**

- Relational Reporting 151