IBM DB2 Alphablox

Developer's Reference

Version 8.2

SC18-9435-00

IBM DB2 Alphablox

Developer's Reference

Version 8.2

SC18-9435-00

Note:
Before using this information and the product it supports, read the information in "Notices" on page 919.

First Edition (November 2004)

This edition applies to version 8, release 2, of IBM DB2 Alphablox for Linux, UNIX and Windows (product number 5724-L14) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Preface

This Preface contains introductory material for the *Developer's Reference*. 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.

About This Book

The *Developer's Reference* contains general Blox reference information, as well as specific information for each Blox supported by DB2 Alphablox for developing applications using the default DHTML client.

This book is written for application developers and database administrators with the following skills and knowledge:

- application design experience
- knowledge of HTML and JavaScript™
- knowledge of JSP and Java™ programming
- working knowledge of the data sources used in Alphablox applications
- working knowledge of the languages used for querying data sources

The *Developer's Reference* contains the following chapters:

- Chapter 1, "Using This Reference," on page 1 provides a brief overview of DB2 Alphablox, describes how the Blox reference information is organized, and includes general information about working with Blox.
- Chapter 2, "Overview of Blox, Object Model, and UI Model," on page 3 describes the object model you can use to access different Blox through their Java objects.
- Chapter 3, "General Blox Reference Information," on page 17, provides general reference information about Blox and JSP pages.
- Chapter 4, "Common Blox Reference," on page 29 describes those properties and methods that are common to multiple Blox.
- Chapter 5, "Client-Side API Reference," on page 63 describes the client-side events available in the DHTML client.
- Chapters 6 through 23, arranged alphabetically by Blox name, describe properties, methods, and usage unique to specific Blox.
- Chapter 24, "Blox Form Tag Reference," on page 699, lists and describes the properties of FormBlox and their tags and attributes in the Blox Form Tag Library.
- Chapter 25, "Business Logic Blox and TimeSchema DTD Reference," on page 735, lists and describes the business logic Blox tags, their tags and attributes in the Blox Logic Tag Library, and the TimeSchema DTD.
- Chapter 26, "Blox UI Tags Reference," on page 791, lists and describes the tags and their attributes in the Blox UI Tag Library.

- Chapter 27, "XML Resource Files Reference," on page 839, lists and describes the XML formats for creating resource files based on the Blox UI model.
- Chapter 28, "Using the Alphablox XML Cube," on page 855, describes the Alphablox XML cube document type definition (DTD) extended Document Object Model (DOM).
- Chapter 29, "Extended DOM API Reference," on page 863, includes the syntax and descriptions for the extended DOM APIs.
- Appendix A, "JSP Custom Tag Copy and Paste," on page 881, provides a reference of all of the Alphablox tag libraries so you can cut and paste them into your JSP files.
- Appendix B, "DB2 Alphablox XML Cube DTD," on page 903, describes the DTD.
- Appendix C, "Examples Cross References," on page 907 contains examples using a variety of Blox and methods.
- Appendix D, "Deprecated APIs," on page 915 lists deprecated properties, methods, classes, tags, and/or URL attributes.

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 DB2 Alphablox Home Page as well as within Alphablox applications. The DB2 Alphablox documentation set includes the following books:

Title	Description
Administrator's Guide	Contains information about setting up and managing DB2 Alphablox and about DB2 Alphablox in a J2EE environment.
Developer's Guide	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.
Developer's Reference	A complete API reference for developing applications using the DHTML client; contains information on each Blox, including its JSP syntax, properties, methods, and objects.
Relational Reporting Developer's Guide	Contains information about setting up ReportBlox to build a report from relational data.
DB2 Alphablox Cube Server Administrator's Guide	Contains information about setting up DB2® Alphablox cubes. DB2 Alphablox cubes allow you to present a multidimensional view of data stored in a relational data warehouse or data mart database.
Installation Guide	Contains information on system requirements, installing and configuring DB2 Alphablox, installing sample data, and migrating applications from previous versions.

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

`<alphablox_dir>/system/documentation/javadoc`

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

Access to Online Documentation

The DB2 Alphablox documentation is 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 DB2 Alphablox Home Page.

The entry page to Online Documentation opens in a separate browser window and provides links to both the HTML and PDF versions of the books. It also provides links to Javadoc for server-side API, Relational Reporting API, and Fast Forward API.

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.ibmlink.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).

Chapter 1. Using This Reference

This book is designed for use as an API reference for developing applications to run on the DHTML client. This chapter describes how to find information in the *Developer's Reference*.

Locating Blox Reference Information

The Blox API chapters of the *Developer's Reference* are organized into the following sections whenever appropriate:

Section	Description
Overview	Provides a brief description of the Blox and a description of its user interface (if any).
JSP Custom Tag Syntax	Provides the syntax for the custom tag libraries used to create a Blox.
Properties and Methods by Category	Lists all properties and methods on a Blox; maps properties to their related methods.
Blox Properties and Associated Methods	Lists the name of each property available on the Blox, its use, its valid and default values, and the name, syntax (Java methods), and usage of any methods associated with the property. The properties are organized alphabetically by property name.
Methods	Lists the methods which do not have a property associated with them and describes their purpose, syntax, returned values, and usage examples. The methods are organized alphabetically by method name.

The table of contents lists each of these sections, and the index lists individual APIs by their names.

Using the API Descriptions

The APIs for each Blox are documented in their corresponding reference chapters; that is, the APIs for DataBlox are in the Chapter 11, "DataBlox Reference," on page 319 chapter, the ChartBlox APIs are in the Chapter 8, "ChartBlox Reference," on page 193 chapter, and so on. Within each chapter, the APIs are divided into one section for all of the Blox properties and their associated methods, and one section for methods not associated with properties.

This section describes how each API description is organized.

Property or Method Name

A brief description of what the API does.

Data Sources

Lists the data source types to which this API applies (for example, Multidimensional).

Syntax

JSP Tag Attribute

```
Lists the syntax of the custom tag library (for properties only)
```

Java Methods

```
Lists the Java signature (syntax) for any Java methods available. The Java
methods are server-side methods, typically called in Java scriptlets or Java
classes, and they are executed on the server. If there is no listing, then
no Java methods are available.
```

where:

Argument	Default	Description

This table describes any arguments the APIs take.

Usage

This section provides any usage notes relevant to the API.

Examples

This section shows example of using the APIs or links to other code examples.

See Also

This section lists cross references to related APIs.

Using the Javadoc

Javadoc is generated documentation that contains the signature of all of the Java APIs available as well as any comments added to the source code about those APIs. Javadoc is an easy way for Java developers to have a quick reference to the APIs available to them.

DB2 Alphablox includes Javadoc for the server-side Blox API, the ReportBlox API, and FastForward API. All the Javadoc can be accessed from the **Help** menu on the DB2 Alphablox Home Page. Or, to access the server-side Blox API, you can enter the following location in a browser:

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

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

To access the Javadoc for the ReportBlox API, open the following file in a browser:

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

To access the Javadoc for FastForward API, open the following file in a browser:

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

Chapter 2. Overview of Blox, Object Model, and UI Model

This chapter describes the Blox categories, the Blox object model, and the Blox UI model. For conceptual information about DB2 Alphablox, see the *Developer's Guide*.

Blox Categories

DB2 Alphablox includes a set of Blox components for building powerful analytic applications. This section provides an overview of these Blox components by category as follows:

- User Interface Blox
- Data Blox
- Analytic Infrastructure Blox
- Blox UI Components
- Business Logic Blox
- FormBlox
- Relational Reporting Blox

User Interface Blox

These Blox provide visual components to support data navigation in grids and charts formats, supported by page filters, toolbars, a menubar, and a data layout panel. Blox in this category includes:

- GridBlox: presents data in a table format.
- ChartBlox: presents data in charts.
- DataLayoutBlox: provides a data layout panel that allows users to move dimensions to desired row, column, or page filter axis.
- PageBlox: provides a page filter to the data presented in the grid and chart.
- ToolbarBlox: provides easy access to data navigation and analysis functionality with a click of the icons.
- PresentBlox: combines all the above user interface Blox in one container for better layout and communications among the Blox.
- ContainerBlox: the foundation Blox for all the user interface Blox. If you want to build custom user interface, you can start with a ContainerBlox.

The custom JSP tags for these Blox are available in the Blox Tag Library (`blox.tld`).

Data Blox

These Blox provide access to data sources. DataBlox, in particular, is needed for the user interface Blox to connect to and obtain the result set from the data source of interest. Blox components in this category includes:

- DataBlox: accesses the data sources and retrieves the result set for the user interface Blox.

- StoredProceduresBlox: allows you to create a connection to a relational database and prepare a stored procedure statement for use.
- ResultSetBlox: lets you to arbitrarily push a ResultSet into the associated DataBlox. You can also extend the normal functions associated with a JDBC data source by attaching a method to intercept queries in the DataBlox and to return an arbitrary ResultSet to the DataBlox.
- JDBCConnection bean: allows you to get information about an Alphablox relational data source, perform JDBC calls, or override properties of a JDBC data source defined in DB2 Alphablox.

The custom JSP tags for DataBlox and StoredProceduresBlox are available in the Blox Tag Library (`blox.tld`).

Analytic Infrastructure Blox

These Blox provide means to building the analytic infrastructure. Blox in this category includes:

- AdminBlox: provides programmatic access to information on the server, users, groups, roles, data sources, and applications set through the DB2 Alphablox Admin Pages.
- BookmarksBlox: allows you to programmatically create and manage bookmarks and dynamically set the bookmark properties.
- CommentsBlox: provides cell commenting (also known as cell annotations) as well as general page/application commenting functionality to your application.
- RepositoryBlox: provides a means for you to save and retrieve user and application properties stored in the DB2 Alphablox Repository.

The custom JSP tags for these Blox are also available in the Blox Tag Library (`blox.tld`).

Blox UI Components

The DHTML client is built on the Blox UI model with three distinct parts: the visual elements on a page (the Components), the Controllers that process events from the components, and Events that communicate state changes from the user interfaces and underlying application logic. These UI components are extensible, allowing you to extend beyond the out-of-the-box functionality the user interface Blox provide.

The tags in the Blox UI Tag Library (`bloxui.tld`) allow you to:

- customize components in the user interface such as adding items to the menubar or toolbar or creating your own menubar or toolbar
- customize layout such as adding empty columns/rows or moving row headers to specified position (the butterfly layout)
- add custom data analysis functionality that can be fully integrated into the user interface

The Blox UI Tag Library (`bloxui.tld`) is described in Chapter 26, "Blox UI Tags Reference," on page 791. For a list of all the components and their methods, see the com.alphablox.blox.uimodel.* packages in the Javadoc.

Business Logic Blox

These Blox components let you add business logic to your application:

- MDBQueryBlox: enables OLAP queries to be built with one language regardless of the underlying server's query language
- MemberSecurityBlox: gives you the ability to hide members from unauthorized users
- TimeSchemaBlox: supports dynamic time series, such as showing data from the "last 3 months"

The custom JSP tags for business logic Blox are available in the Blox Logic Tag Library (`bloxlogic.tld`), described in Chapter 25, "Business Logic Blox and TimeSchema DTD Reference," on page 735.

FormBlox

These Blox are built on top of the Blox UI components to create form-based interface. A series of FormBlox are available to provide a familiar HTML form interface that allows users to select the data source, dimensions, members, or other options you provide to create personalized query.

The custom JSP tags for FormBlox are available in the Blox Form Tag Library (`bloxform.tld`), described in Chapter 24, "Blox Form Tag Reference," on page 699.

Relational Reporting Blox

A set of Blox designed to build interactive reports from relational data sources. For details, see the *Relational Reporting Developer's Guide*.

Blox Object Model

The Blox API consists of many Java objects, and you can programmatically access many of the objects through other objects. For example, PresentBlox has the methods `getDataBlox()`, `getPageBlox()`, `getGridBlox()`, `getChartBlox()`, `getToolbarBlox()`, and `getDataLayoutBlox()`, which are used to access the DataBlox, PageBlox, GridBlox, ChartBlox, ToolbarBlox, and DataLayoutBlox nested within. This section describes the overall object model in the Blox API. As with most object models, there are many paths you can use to navigate through the Blox API; this section describes the most basic and common access points to help familiarize you with the API.

Tip: For developers experienced in using Javadoc, Javadoc is a useful tool in learning about the object model. The Javadoc for the server-side Blox APIs is installed with the rest of the documentation and is available in the following location:

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

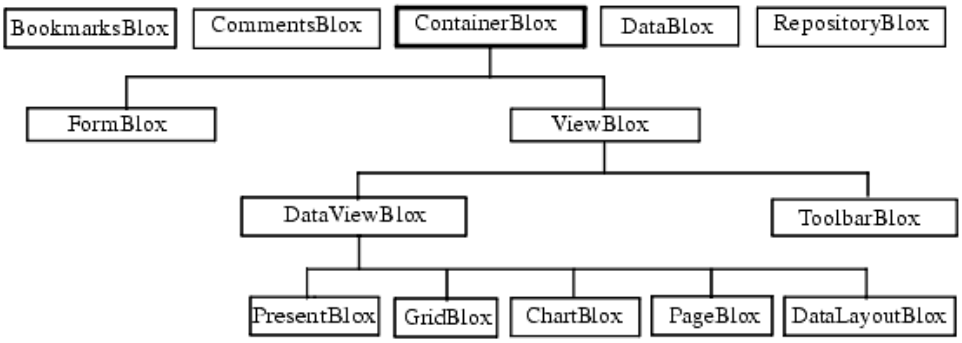
This section covers the following topics:

ContainerBlox— The Container for User Interface Blox

The ContainerBlox is base class for all user interface Blox. These Blox inherit the `bloxModel` property from ContainerBlox. Via the Blox's `getBloxModel()` method, you can access the UI model in effect for this Blox. Each Blox model consists of a header container and a body container, each contains a number of named standard

components. You can use these names to find a component and customize it. This is discussed further in "Blox UI Model" on page 10.

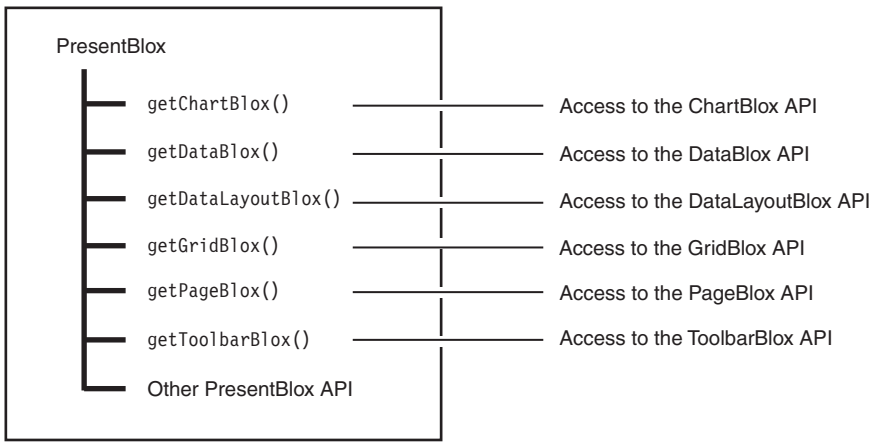
The following diagram shows the hierarchy of inheritance of various Blox components:



PresentBlox—A Single Blox with Nested User Interface Blox

PresentBlox is a convenient way to display a chart, a grid, a data layout panel, and a toolbar all in a single Blox. Using a *nested Blox*, you can control all of the individual elements of each part that displays in a PresentBlox. Since you access each of the individual elements through the PresentBlox, you can think of the PresentBlox as a container for the other Blox, and those other Blox inside the PresentBlox container are known as nested Blox. Each Blox has properties that represent the state of the Blox, and you can access the nested Blox properties either by specifying values for the properties in the tag library used to create the Blox or by using the API to programmatically get and set the properties.

The following figure shows how you can access other Blox through PresentBlox:



Nested Blox

Some Blox can contain other, nested Blox. For example, ChartBlox and GridBlox (each of which can be a standalone Blox) are nested Blox within a nesting PresentBlox. DataBlox can be nested within PresentBlox, ChartBlox, or any of the Blox that take a data source. To apply a Blox-specific property to a nested Blox, add the nested tag. Nested Blox are accessed using the object model. Start with the outer Blox and then access the inner Blox calling the get method for the Blox you want (for example, `getDataBlox()`) to obtain access to the inner Blox object.

You can use the Blox Tag Library to create and access nested Blox. The following example, for an IBM DB2 OLAP Server or Hyperion Essbase data source, shows a DataBlox nested within a ChartBlox:

```
<blox:chart id="myChart"
...ChartProperty="ChartPropertyValue" >
   <blox:data
      dataSourceName="FinancialCube">
      query="!">
   </blox:data>
</blox:chart>
```

A PresentBlox typically contains multiple Blox that share one DataBlox:

```
<blox:present id = "profitPresent"
   height = "80%"
   width = "96%"
   dividerLocation = "0.60" >

   <blox:data
      dataSourceName = "QCC-Essbase"
      useAliases = "true"
      query = "!">
   </blox:data>

   <blox:chart
      chartType = "Vertical Bar, Side-by-Side"
      legend = "All Products"
      XAxis = "All Time Periods" >
   </blox:chart>

   <blox:grid
      paginate = "false">
   </blox:grid>

</blox:present>
```

If you have an explicit DataBlox to be used by multiple presentation Blox, you can use the DataBlox as a nested Blox via the `bloxRef` tag attribute:

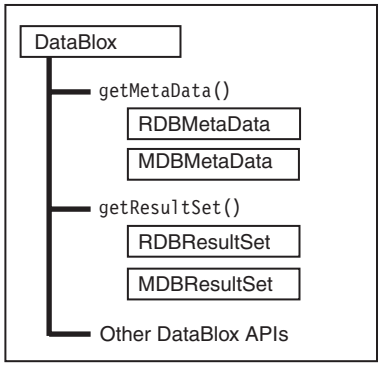
```
<blox:data id="FinancialCube"
   dataSourceName="FinancialCube">
   query="!" />

<blox:chart id="myChart"
...ChartProperty="ChartPropertyValue" >
   <blox:data bloxRef="FinancialCube" />
</blox:chart>

<blox:grid id="myGrid"
...GridProperty="GridPropertyValue" >
   <blox:data bloxRef="FinancialCube" />
</blox:chart>
```

For the syntax of each of the Blox custom tags, see the "JSP Custom Tag Syntax" section for a given Blox.

DataBlox—Access to Metadata and Result Sets

DataBlox provides access to data sources not only in terms of retrieving queries, but also in terms of searching through the metadata in the database (that is, the information about the data such as what members belong to a given dimension or what columns belong to a given table, what are the names of the dimensions or what are the names of the tables, etc.) and in searching through the data that is retrieved in a result set.

The following figure shows how you can access the metadata and result set objects through DataBlox. The metadata and result set objects each contain several objects.



To use the APIs associated with RDBMetaData and RDBResultSet, you need to import the com.alphablox.blox.data.rdb package in your JSP page:

```
<%@ page import="com.alphablox.blox.data.rdb.*" %>
```

To use the APIs associated with MDBMetaData and MDBResultSet, you need to import the com.alphablox.blox.data.mdb package in your JSP page:

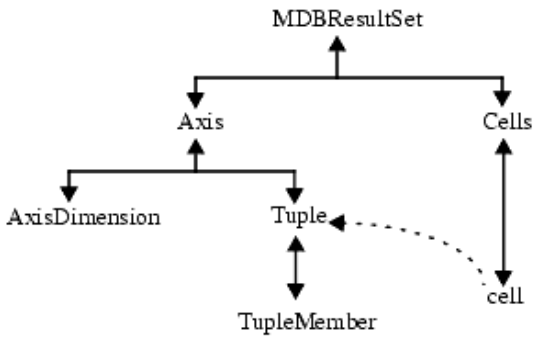
```
<%@ page import="com.alphablox.blox.data.mdb.*" %>
```

Metadata and Result Sets

Result sets and metadata provide a way of walking the data in a hierarchical fashion. They are represented as objects with a rich set of APIs that give you full control of data presentation and interaction. You can access these objects through the `getMetaData()` and `getResultSet()` DataBlox methods.

The result sets objects involve actual data values from a query. Therefore, they have a restricted set of axes, tuples, dimensions, and members. Metadata objects do not need a result set from a query, and only involve the cubes, dimensions, and members (outline) of the data source. Generally speaking, you use MDB or RDB result sets if you are performing data source specific tasks such as calculations, writeback, and custom view. When what you do involves browsing outline or forming queries, you should use the metadata objects.

MDBResultSet: The following figure shows the object hierarchy of MDBResultSet. The direction of the arrows indicates whether you can reference the parent or child of an object. The dotted arrow means that once you get to the individual cell, you can find out its tuple, which allows you to access the axis it is on or a specific tuple member.



Note that a MDBResultSet object typically contains multiple axes and multiple cells, and each axis typically contains multiple tuples or dimensions. Therefore, you have one method that returns an array containing all the axes, dimensions, tuples, or cells, and another method that returns one particular axis, dimension, tuple, or cell if you specify a 0-based index. For example, `getAxes()` returns an array containing all the axes in the result set, and `getAxis(0)` returns the first axis in the result set.

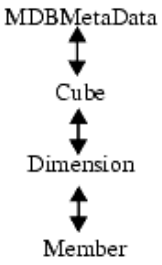
Some of the objects have types that give you easier access to the child object you want. For example, the Axis object has fields called `ROW_AXIS`, `COLUMN_AXIS`, `PAGE_AXIS`, and `SLICER_AXIS`. This allows you to easily access an axis of a specific type. Likewise, AxisDimension has types such as `ATTR_DIMENSION`, `MEASURES_DIMENSION`, and `TIME_DIMENSION` so you can easily access a dimension of a specific type.

RDBResultSet: The following figure shows the object hierarchy of RDBResultSet. The direction of the arrows indicates whether you can reference the parent or child of an object.

The RDBResultSet object contains a ResultColumn object that gives you information on the column type, column name, or position (0-based index) in the result set. The row iterator is an array of objects (`Object[]`) that lets you iterate through the rows to get the data.

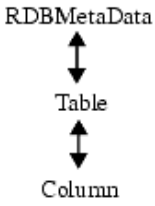
Similar to the MDBResultSet object, an RDBResultSet object typically contains multiple columns. Therefore, you have one method that returns an array containing all the ResultColumn objects, and another method that returns one particular column. For example, `getColumns()` returns an array of result columns in this result set, and `getColumn(0)` returns the first result column within this result set.

MDBMetaData: The following figure shows the object hierarchy of MDBMetaData. The direction of the arrows indicates whether you can reference the parent or child of an object.



Again, a MDBMetaData object may contain multiple cubes (in IBM DB2 OLAP Server or Hyperion Essbase, there is only one cube). Each cube typically contains multiple dimensions, and each dimension typically contains multiple members. As a result, you often have one method that returns an array containing all the cubes, dimensions, or members, and another method or methods that returns one particular cube, dimension, or member if you specify a 0-based index. For example, `getCubes()` returns an array of cubes, and `getCube(0)` returns the first cube within the database this MDBMetaData object is describing.

RDBMetaData: The following figure shows the object hierarchy of RDBMetaData.



An RDBMetaData object may contain multiple tables, and each table typically contains multiple columns. As a result, you often have one method that returns an array containing all the tables and columns, and another method that returns one particular table or column if you specify a 0-based index. For example, `getTables()` returns an array of tables, and `getTable(0)` returns the first table within the database this RDBMetaData object is describing.

For a listing of the methods on each object discussed in this section, see "DataBlox Properties and Methods by Category" on page 322

Blox UI Model

The DHTML client is based on the Blox UI Model with three distinct concepts in the framework: components, controllers, and events. Components make up the visual elements on a page, such as buttons, edit fields, images, texts, toolbars, and menus. Controllers process events from the components, translating generic component behaviors such as `ClickEvent`, `RightClickEvent`, `DoubleClickEvent`, or `SelectedEvent` into application-defined action. Events communicates state changes from the user interface, the underlying application logic, and from the model itself to the components and controllers.

The following section provides a high-level overview of Components, Controllers and Events. These are common concepts and terms you will come across when you work with the Blox UI Tag Library or when you want to customize the behavior of a component. This general information will help you understand the UI model in the DHTML client and provide a foundation for customizing and extending the

Blox UI model. Extending the Blox UI model is an advanced topic and is discussed in the *Developer's Guide*. Details on the objects and their associated methods are available in the DB2 Alphablox server-side API Javadoc under the com.alphablox.blox.uimodel.* packages.

Component

Every visual user interface object in the Blox UI model descends from the Component base class. This model provides a number of core, basic user interface components such as Button, CheckBox, RadioButton, Edit (edit fields), ListBox, DropDownList, Menu, Menubar, Toolbar, ToolbarButton, DropDownToolbarButton, and ComponentContainer. All these components share a number of common properties and behaviors and are arranged in a hierarchy that provides both formatting control as well as central management of primitive components. The grouping of these components are made possible by the ComponentContainer, which allows grouping of components for the purposes of layout, behavior, and style. For details on the components, their hierarchy, and their methods, see the com.alphablox.blox.uimodel.core package in the Javadoc.

These UI components are further combined into compound components in the server's Blox Models. `BloxModel` is the base class for `GridBloxModel`, `ChartBloxModel`, `DataLayoutBloxModel`, `PageBloxModel`, and `PresentBloxModel`. It is used to represent the current visual state of ViewBlox-derived Blox (see the Blox object hierarchy in "ContainerBlox— The Container for User Interface Blox" on page 5).

Each Blox model consists of two named Containers:

- `ModelConstants.BLOXUI_HEADER` — Container which contains the toolbars and menubar
- `ModelConstants.BLOXUI_BODY`— Container which contains the model(s) that provide the unique Blox functionality

Inside each of the main containers is a number of specifically named standard components that can be customized and/or removed. You can use these names to easily find a component during customization. All component names are available as constants in the interface class `ModelConstants`.

The following table lists the Blox UI model components. These are the components that made up the Blox user interface:

Blox UI Model Component	Description
Button	A push button component.
CheckBox	A checkbox component.
Component	The abstract base class for all UI model visual components. This class provides default behaviors and properties which are common across all visual components.
ComponentContainer	Generic container for UI model objects. The component container is used to group components for the purposes of layout, behavior, and style. For example, if you want three Buttons to line up horizontally from left to right, put them in a ComponentContainer and set the layout to

	horizontal. The order of component in the container is significant and affects the display order. A component can only exist exactly once in exactly one container. If a component is added to a different container, it is automatically removed from the previous container.
Controlbar	The base class for controlbars (menus and toolbars) that can be contained in a ControlbarContainer.
ControlbarContainer	The container for Controllbar.
DateChooser	This component extends the Edit component by adding a calendar icon next to the edit field. Clicking the icon launches a calendar widget for selecting a date to populate the edit field.
Dialog	Dialogs are floating containers that are used to collect input from and/or show status to users. Create a dialog and then populate the dialog with components such as Buttons, CheckBoxes and RadioButtons (among other) to present users with option lists or to make a decision.
DropDownList	A DropDownList consists of a single displayed option with a list of other choices. Only one choice may be selected at a time. Use a DropDownList when space is limited and the constant display of possible choices is not required.
DropDownToolbarButton	The DropDownToolbarButton has both a drop-down list of selections as well as an action button to invoke the currently displayed drop down list. The control generates a ClickEvent when the selection is changed in addition to when the action button is clicked.
Edit	Edit field component. An Edit component allows the user to enter and modify one or more lines of text. Text can be copied, moved, and inserted into the edit field using the standard user UI mechanisms.
GroupBox	GroupBox component provides a titled container for dialogs and other models. The GroupBox is primarily used to group components in dialog boxes. For example, if there are a number of components dedicated to setting options for a chart, then these can be grouped together in a GroupBox with the title “Chart Options.” When RadioButton components are contained inside of a `GroupBox`, all unnamed RadioButtons in a named group will become automatically grouped. Pressing one radio button will unselect others in the group.
Image	An Image component used to display any GIF,

	JPEG, or other compatible image. The image will generate a ClickEvent when clicked.
ListBox	A ListBox component that creates a selection list.
Menu	A `Menu` component consisting of MenuItems and other Menus. Menus inside of Menus will be treated as submenus with the appropriate submenu behavior. MenuItems will display as selections and will generate a ClickEvent when selected. By default, a MenuItem's name is used to construct a handler method in controllers. For example, a MenuItem with the name ″drillDown″ will map to a method called ″actionDrillDown″ in controllers. All new menus are assigned a vertical layout by default.
Menubar	A Menubar component.
MenuItem	A MenuItem component. This is a selectable item in menu.
MessageBox	MessageBox dialog. The MessageBox dialog provides a convenient way to present simple information to the user. It also provides a simple mechanism to collect YES/NO and OK/Cancel responses from users.
RadioButton	RadioButton component. Typically RadioButtons are grouped together so that only a single RadioButton in a named group can be selected at any one time. Selecting a RadioButton in a group of RadioButtons will automatically unselect the currently selected button in that group. GroupBoxes can be used to automatically group together sets of RadioButtons.
Spacer	Spacer component, used to add fixed height and/or width spacing between components.
SpinnerButton	Spinner component, used to accept integer input from the user and provide buttons to increase/decrease the value. You can set the initial value, increment, and low and high values.
SplitterContainer	SplitterContainer component, used to display two components with a splitter bar between them that the user can adjust. Use either the HorizontalLayout or the VerticalLayout to control the orientation of the splitter.
Static	Static component, used to display simple static text such as instructions, labels, or values, where interaction is not needed. You may attach a ClientLink to a static component in order to make it respond to a user selection. The component's title field is used to store the Static text.
StaticImage	Component to render a static image which does

not respond to user input. For images that will generate a ClickEvent, use the Image component.

TabbedContainer

Container window with tabs for all child containers. This container is used to display a list of tabs corresponding to all child containers. A typical use would be to implement a tabbed dialog box. This container can only contain other component containers. The style attached to this container is applied to the tabs.

The title of the child container is used for that container's tab label. The selection state of the child containers is used to determine the selected tab. If no child containers are selected, then the first container is automatically selected. If multiple child containers are marked as selected, then the first one encountered is considered selected.

The order in which the child containers are added to the tabbed container determines the tab order. For top and bottom (horizontal), the first container is on the right. For left and right (vertical), the first container is on the top.

Toolbar

Toolbar component used in conjunction with ControlbarContainer to display toolbars.

ToolbarButton

ToolbarButton component. Toolbar buttons can be used anywhere in the component model, but they are primary designed to work in ControlbarContainers. The name of the component is used to construct the image name with a `.gif` extension.

Events

The DHTML client creates events as JavaScript objects that can be created, sent, and intercepted by client-side JavaScript code. The DHTML client does not have any domain knowledge such as whether an event is data drill up or drill down, nor does it understand dimensions or members. All of the domain-specific logic and information is stored on the server. The DHTML client only knows when an object such as a menu item or a help button is clicked. It is only aware of a simple set of events such as single click, double click, right click, scroll, drag and drop, selected, unselected, selection changed, contents changed, and closed.

Controller

When users press a `Button` or select a `MenuItem` from a `Menu`, the controller is responsible for calling and executing the action associated with this user event. The `Controller` class is the base class for all model component controllers. Controllers can be attached to any model object that are derived from Component.

Server-Side API and Client-Side API

The APIs for developing applications using the DHTML client in DB2 Alphablox are available on the server-side, where a developer accesses them through Java calls (for example, in a Java scriptlet on a JSP page). The reason the Java APIs are called *server-side* APIs is because the code executes on the server before it is sent to the browser.

Executing code on the server is often more efficient, and also makes it easier to create web pages that work correctly on multiple browsers. The DHTML client is designed to keep the client and the server in sync without page refreshing. When you execute code on the server, only affected areas in the Blox UI is refreshed, not the whole page.

There are also times when you want to use the DHTML client's Client API for tasks that are best handled on the client. These are called client-side APIs because they are interpreted by the browsers. Often times you want to call some server-side code to change Blox properties on the server via some JavaScript code on the client when a user clicks a button or link on the page.

The DHTML client has a relatively straightforward API on the client-side. See Chapter 5, "Client-Side API Reference," on page 63 for more information.

Chapter 3. General Blox Reference Information

This section provides general reference information that applies to all Blox. For information on APIs common to all Blox, see Chapter 4, "Common Blox Reference," on page 29.

Tips for Working with Blox

Before working with Blox, note the following key points:

- Appendix D, "Deprecated APIs," on page 915 includes a listing of deprecated Blox methods and properties, the release in which they were deprecated, and their replacements. When working with existing applications, refer to this list to determine necessary changes to method and property names.
- If an invalid property is used on a Blox, the JSP file will not compile successfully.
- All Blox properties (and their corresponding tag attributes) are case-sensitive. With a few exceptions, the property names follow the Java bean naming convention: begin the name with a lowercase letter, each new word or phrase in the name begins with an uppercase letter (for example, `dataSourceName`).
- All URLs that run through DB2 Alphablox are case-sensitive.
- To override a default or inherited value for a specific Blox, include the property keyword and local value in the custom tag used to create the Blox. For example, the following attribute within the `<blox:chart>` custom tag causes ChartBlox to display a pie chart:

  ```
  chartType="Pie"
  ```

Working with Different Data Sources

DB2 Alphablox includes a Data Manager and associated data adapters that provide support for:

- browsing a collection of pre-configured, named connections to application data sources
- exposing the available databases within each data source
- publishing the compatible query types for a specific data source
- allowing the traversal of the data source metadata
- managing data source connections for user sessions
- translating query objects into the underlying native query language
- executing queries against a data source
- interrogating a query result set's data and schema

- modifying a result set by pivoting, expanding, and drilling

DB2 Alphablox data adapters can access and retrieve data from relational databases, multidimensional databases, and DB2 Alphablox cubes. (DB2 Alphablox cubes transform data from relational databases into multidimensional format).

Most Blox properties and methods apply to all types of data sources. For those that do not, there is a section in the API description stating which data sources a particular API works with (for example, all, multidimensional, relational, IBM DB2 OLAP Server™ and Hyperion Essbase only, etc.).

Blox in a JSP File

This section describes the components of a JSP page that contains Blox. It shows a sample JSP file and then describes the different sections of it.

Sample JSP File Containing Blox

The following code listing shows a JSP file containing all the elements necessary to render a Blox on a page.

```
<%-- Import any packages used in scriptlets --%>
<%@ page import="com.alphablox.blox.*" %>
<%@ page import="com.alphablox.blox.data.*" %>
<%@ page import="com.alphablox.blox.data.mdb.*" %>

<%-- Import the Blox custom tag libraries --%>
<%@ taglib uri="bloxtld" prefix="blox"%>

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

<%-- Create the Blox --%>
<blox:present
   id="regionsBlox"
   visible="false"
   width="650"
   height="350"
   splitPane="false"
   visible="false">

   <blox:data
      dataSourceName="TBC"
      query="<SYM <ROW(Product) <ICHILD Product <COLUMN(Year, Scenario)
         Qtr1 Qtr2 <CHILD Scenario Sales !"
      useAliases="true"
      selectableSlicerDimensions="Market" >
   </blox:data>

   <blox:grid
      bandingEnabled="true" >
   </blox:grid>

   <blox:chart
    chartType="Vertical Bar, Stacked" >
   </blox:chart>

</blox:present>
<!-- HTML and JavaScript Elements -->
<html>
<head>
<title>Sample Blox JSP File</title>

<%-- Insert the Blox header --%>
```

```
<blox:header />

<%--Insert some JavaScript, if needed (with or without any
Blox APIs)--%>
<script language="JavaScript">
</script>

</head>
<body>

<%-- You can include scriptlets or JavaScript containing
     Blox APIs as needed --%>
<p>Put the Blox here <br />

<%-- Display the Blox --%>
<blox:display bloxRef="regionsBlox" />
</p>

</body>
</html>
```

Package and Tag Libraries Imports

This section is usually at the top of the JSP file. The package import statements are only necessary if you are using any APIs in those packages. The tag libraries import section is required to use any of the Blox tag libraries.

```
<%-- Import the Blox Tag Library --%>
<%@ taglib uri="bloxtld" prefix="blox"%>

<%-- Import the Blox UI Tag Library --%>
<%@ taglib uri="bloxuitld" prefix="bloxui"%>

<%-- Import the Blox Form Tag Library --%>
<%@ taglib uri="bloxformtld" prefix="bloxform"%>

<%-- Import the Blox Logic Tag Library --%>
<%@ taglib uri="bloxlogictld" prefix="bloxform"%>

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

Also use this section to import any Java packages used in Java API calls. The Java APIs might be called from a scriptlet on the JSP page. For example:

- If you are using the MDBMetaData object or the MDBResultSet object, you will need the following import statement:

  ```
  <%@ page import="com.alphablox.blox.data.mdb.*" %>
  ```

- If you are using the RDBMetData object or the RDBResultSet object, you will need the following import statement:

  ```
  <%@ page import="com.alphablox.blox.data.rdb.*" %>
  ```

- If you are using the BookmarksBlox and its associate objects, you will need the following import statement:

  ```
  <%@ page import="com.alphablox.blox.repository.*" %>
  ```

- If you are using the Comment object, you will need the following import statement:

  ```
  <%@ page import="com.alphablox.blox.comments.*" %>
  ```

- If you are using the server-side Event Filter object, you will need the following import statement:

  ```
  <%@ page import="com.alphablox.blox.filter.*" %>
  ```

- If you are using the StoredProcedure object, you will need the following import statement:

  ```
  <%@ page import="com.alphablox.blox.data.rdb.storedprocedure" %>
  ```

- If you are using the <jsp:useBean> tag to create Blox rather than using the Blox tag libraries, you will need the following import statement:

  ```
  <%@ page import="com.alphablox.blox.*" %>
  ```

- If you are using the BloxModel API, you will need the following import statement:

  ```
  <%@ page import="com.alphablox.blox.uimodel.*" %>
  ```

If an import statement is needed in order to use an API, this information is listed in the beginning of each of the API section.

Adding Content Type Character Set Declaration

To ensure proper character set encoding, add the following line to your JSP file to set the character set:

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

This is particularly important if you are running on a Japanese or other foreign language system.

Blox Creation Tags

You can place your custom tag libraries to create the Blox anywhere on the JSP page, but it is a good idea, and can make your code cleaner looking and more readable, to place them before creating the HTML sections of the page. This helps separate the application logic from the display elements of your page. If you do place the Blox tag libraries before any HTML elements, you also have to set the `visible` property to `false`, then use the <blox:display> tag to actually display the Blox on your page. For details on the <blox:display> tag, see "The <blox:display> Tag" on page 24.

<blox:header> Tag in the HTML <head>

The <blox:header> tag should go in the HTML <head> section of your page. It is required for pages rendered in the DHTML mode to display properly as DB2 Alphablox substitutes the proper theme and style information in the header for the Header tag. It also adds a few lines of code that manage file caching for pages. More importantly, it provides the foundation for communications services between the client and the server, making it possible to execute server-side code from JavaScript objects on the client.

For more information on additional tag attributes for specifying page URL and context path for portlet integration or client bean registration, see the "The <blox:header> Tag" on page 25.

Tip: The <blox:header> tag must come before the Blox is set to display in your JSP file; that is, it either must come before a Blox creation tag with a `visible` property of `true` (the default) or it must come before the <blox:display> tag makes the Blox visible on the page.

Scriptlets Containing Blox APIs

You can put any Java code you want anywhere within a JSP page, and the code is executed on the server before the page is sent to the user. The Java code can use Blox APIs in a scriptlet or it can use anything available in Java or in the environment in which your web application server is running. If the Java code

uses Blox APIs, the Blox definition to which the code scripts must come before the scripting code. To place Java code on your JSP page, place any valid Java code between the following sets of characters:

```
<%
%>
```

The JSP engine recognizes this as Java code and compiles and executes it (it will only be compiled the first time the page is loaded or if the page has changed since the last compile). For example, the following scriptlet uses the server-side ChartBlox APIs and standard `out.write()` Java method to print the values of the `rowSelections` and `columnSelections` properties to the Java console:

```
<%
String RowSelections = mypresent.getChartBlox().getRowSelections();
String ColumnSelections =
       mypresent.getChartBlox().getColumnSelections();
       out.write("The value of columnSelections is:" +
                           ColumnSelections);
       out.write("The value of rowSelections is:" +
                           RowSelections);
%>
```

Note: If you are running an applicaton server and the DB2 Alphablox console is not available, you can use other techniques such as writing the output to a log file, or using the UI Model's MessageBox to display the output during development.

Note: JSP technology includes many techniques for scripting. For details on different ways to script in a JSP file, see a JSP reference book.

How Scriptlets are Evaluated—Inside the Tag versus Outside the Tag

The Blox tags are only evaluated the first time a page is loaded for a user session, while everything outside of the tag is evaluated each time the page is loaded. When the Blox tag is evaluated, the state of all the properties at that time is rendered to the page. If you have scriptlets inside a Blox tag, the code in the scriptlet is executed before the Blox is rendered, therefore any changes that the code might make to a property will be reflected in the Blox rendered to the page.

Because code outside the Blox tags is evaluated after the Blox is evaluated and rendered to the page, any changes to properties in a scriptlet outside the Blox tag will not show up in the Blox on the page until the page is reloaded. Therefore, if you put a scriptlet which changes the value of a Blox property inside the Blox tag, it is evaluated before the Blox is rendered to the page, so the changes appear on the initially rendered page; if you put a scriptlet which changes the value of a Blox property outside the tag, it is evaluated after the Blox is rendered to the page, so the property change is not reflected in the Blox until the page is reloaded.

Sometimes you might need to perform some logic in a scriptlet to determine how to set a property, but you also want that logic to execute each time the page is loaded (not just the first time the page is loaded). Putting the code inside the Blox tag would execute the code for the first load of the page in a user's session, but subsequent page refreshes in that session will not execute the code in the Blox tag. In this situation, you can set the `visible` property in the Blox tag to `false` and put the code to set the property in a scriptlet outside the Blox tag. Then, later on in the page, use the `<blox:display>` tag to display the Blox on the page. This technique

results in properties you set outside the Blox tag to be reflected in the Blox that is displayed on the users page. The following pseudo-code demonstrates this technique:

```
<!--The Blox tag creates the Blox, but since the visible
    property is set to false, the Blox is not yet sent to
    the browser -->
<blox:grid
    id="myBlox"
    visible="false"
    ....
    ...the rest of the tag definition >
</blox:grid>

<%
   // this scriptlet executes some code to set a property
   // (for example, based on who the user is)
   // Because it is outside the tag, it will execute when each
   // time the page is loaded
%>
<!--Use the display tag after the code has executed.
<blox: display
      bloxRef="myBlox"
      visible="true" />
```

JavaScript Code Containing Blox APIs

Anywhere within the HTML section of your JSP page, you can place JavaScript elements using the HTML `<script>` tag.

Tip: The best practice for putting `<script>` tags in HTML pages is to locate them between the `<head>` tags, the `<body>` tags, or between the `<html>` tags if there are no `<head>` or `<body>` tags. The exception is if the `<script>` tag is writing out the `<head>` or `<body>` tags.

HTML and JavaScript Elements

Of course, you can put any HTML or JavaScript elements on your JSP page and it will just be passed on through to the browser. The JSP engine ignores all HTML and JavaScript elements.

URL Attributes

DB2 Alphablox provides several URL attributes as a convenient way to change the render mode, saved state, and theme for an application. URL attributes can be added to the application's URL to define run-time processing. URL attributes take the following form:

```
attribute=value
```

For example, the `render` attribute specifies the format into which DB2 Alphablox renders a page before it is delivered to a client browser. The following attribute specifies that the page is to be delivered in DHTML:

```
render=dhtml
```

To add a single attribute to a URL, append the attribute at the end of the URL preceded by the "?" symbol, as in the following example:

```
http://<serverName>/<App_Context>/MyApp.jsp?render=dhtml
```

To add other URL attributes, append them with the & character, as follows:

```
http://<serverName>/<App_Context>/MyApp.jsp?theme=financial&render=dhtml
```

This section describes the following valid URL attributes:

- "render" on page 23
- "theme" on page 23

Tip: URL attributes are case-sensitive; they are all lowercase.

For an example of using a URL attribute and RepositoryBlox to load a saved application state, see "restoreApplicationState()" on page 653.

render

`render=string`

Specifies the delivery format for all Blox on this application page. Note that the render property on a Blox takes precedence over this attribute.

Possible values include the following:

`dhtml`	Render in fully interactive DHTML format (the default). Requires the `<blox:header />` tag in your JSP page.
`printer`	Render in a format suitable for printing (many browsers do not support printing the output of interactive Java applets). Requires the `<blox:header />` tag in your JSP page.
`xls`	Render a format suitable for export to Microsoft® Excel, and set the MIME type to XLS. In order for the MIME type to be set so the page opens in Excel, you must put the `<blox:header />` tag in your JSP page. For information on the `<blox:header />` tag, see "<blox:header> Tag in the HTML <head>" on page 20.
`xml`	Render in XML format.

For more information on delivery formats, see "Render Formats Available to the DHTML Client" of the *Developer's Guide*.

Important: Using the render attribute without the `theme` attribute causes DB2 Alphablox to use the default theme and automatic browser detection.

theme

`theme=themeName`

Used in `dhtml` render mode. Specifies the theme to use when rendering this page. If the theme name is default, or if the attribute is not used, DB2 Alphablox automatically selects the most appropriate theme based on browser type, browser version, client operating system, and rendering format.

Note: For the `theme` attribute to be recognized, the following line must be added between the `<head>` and `</head>` tags in the HTML part of the JSP page:

```
<blox:header />
```

Valid values for the `theme` attribute are `coleman` (default) and `financial`. The coleman theme has a grey and blue tone, whereas the financial theme has a pale green tone.

Data Type Mappings

The following table shows how Alphablox data types map to JDBC and Java data types. The data ranges of Java types may differ from the ranges supported by a specific database.

JDBC Type	Alphablox Type	Java Type	Java Range
TINYINT	IntegerOperand	int	32-bit signed two's-complement integers
SMALLINT	IntegerOperand	int	32-bit signed two's-complement integers
INTEGER	IntegerOperand	int	32-bit signed two's-complement integers
BIT	boolean	boolean	true or false
BIGINT	LongOperand	long	64-bit signed two's-complement integers
REAL	FloatOperand	Float	32-bit IEEE 754
FLOAT	DoubleOperand	double	64-bit IEEE 754
DOUBLE	DoubleOperand	double	64-bit IEEE 754
NUMERIC	CurrencyOperand	double	64-bit IEEE 754
DECIMAL	CurrencyOperand	double	64-bit IEEE 754
CHAR	String	String	
VARCHAR	String	String	
LONGVARCHAR	String	String	

Note: Note the following about data type mappings:

- The content of a specific Java data type may also differ from that of a database. For example, Java date values begin with the year 1900; most databases accommodate earlier date values.
- JDBC data types may not map directly to your RDBMS. For help in this area, contact the vendor of your JDBC drivers.

The <blox:display> Tag

The `<blox:display>` tag references a Blox that has already been created and displays it wherever you place the tag. The most common use of this tag is to, after some processing logic is done, display a presentation Blox that already exists but has its `visible` attribute set to `false`. Place the `<blox:display>` tag where you want your Blox to display on your HTML portion of your JSP page.

```
<blox:display
    bloxRef="myPresentBlox" />
```

Note the following:

- Since this is an empty tag with no elements, it should be closed using the shorthand notation as shown above. Depending on the application server, you may get an error if you close it using a `</blox:display>` closing tag.
- Before using the `<blox:display>` tag, the Blox to which it refers (the value of the `bloxRef` attribute) must already be instantiated. The instantiation occurs either if the Blox was previously created in another page or if the Blox comes before the `<blox:display>` tag in the JSP file.

Tag attributes available to the <blox:display> tag are as follows:

```
<blox:display
    bloxRef="myPresentBlox"
    render="dhtml"
    width="600"
    height="800" />
```

Any attributes that you set on the <blox:display> tag override any of those attributes that might be set in the tag it references. The above example shows a <blox:display> tag that will display the Blox named myPresentBlox that was defined earlier, overriding the original render, width, and height settings with the ones specified in the display tags.

Note: The <blox:display> tag cannot reference a relational reporting Blox. Relational reporting Blox are Blox that support ReportBlox for producing interactive reports from relational data sources. The usage and reference material for those Blox are documented in the *Relational Reporting Developer's Guide*.

The <blox:header> Tag

The <blox:header> tag is required for pages rendered in dhtml mode to display properly. It:

- Tells the DB2 Alphablox to substitute the proper theme and style information in the header for the Header tag
- Adds code that manage file caching for pages rendered in either dhtml or html mode
- Provides the foundation for communications services between the client and the server, making it possible to execute server-side code from JavaScript objects on the client.

In cases such as portals or other proxy front-ends where the servelet request URI does not reference back to DB2 Alphablox or the application's context path, the <blox:header> tag has two attributes—pageURL and contextPath—that allow you to explicitly specify the page URL and application context path.

The <blox:header> tag also allows you to register your custom bean using its <blox:clientBean> inner tag. This registers the server-side bean with the DB2 Alphablox programming framework and makes the bean's methods available on the client. You can then invoke the bean's server-side method from client-side JavaScript. The following example shows the registration of a custom bean called myBean and its method called changeColor.

```
<blox:header>
   <blox:clientBean name="myBean" protect="false">
      <blox:method name="changeColor" />
   </blox:clientBean>
</blox:header>
```

You then can call the changeColor method on the server using the client API's callBean() method:

```
<a href="bloxAPI.callBean('myBean','changeColor');">Change Color</a>
```

For more information, see the *Developer's Guide*.

Tip: The <blox:header> tag must come before the Blox is set to display in your JSP file; that is, it either must come before a Blox creation tag with a visible property of true (the default) or it must come before the <blox:display> tag makes the Blox visible on the page.

The <blox:session> Tag

This tag lets you synchronize the creation of the DB2 Alphablox session. This is useful when you are using framesets, iframes, or portlets, where you do not want multiple session cookies sent to the browser, or you do not want to use the <blox:header/> tag on each JSP in the frames to put unnecessary JavaScripts and styles on pages that do not have a Blox. This tag has an optional key attribute that should be unique for the application/browser session (that is, the frameset session ID or the portal session ID). If the key attribute is not specified, an DB2 Alphablox session will be created and the session ID cookie will be returned. If a unique identifier is passed, it will prompt DB2 Alphablox to check if an DB2 Alphablox session has already been created with the key. For example:

```
<blox:session key="<%=session.getID()%>" />
```

or

```
<blox:session key="<%=request.getParameter( "syncKey" ) %>" />
```

where the syncKey is passed in when this page is called.

If a non-unique or invalid key is specified, you will have unexpected or undesired results such as data showing up in wrong sessions or session expired messages.

Tags for Rendering to PDF

ToolbarBlox contains an Export to PDF icon, allowing users to convert their current view of Blox on the page to PDF format for printing or archive. As a developer, you can use the pdfReport tag to specify the header, footer, their heights, page margin, and page size. In addition, you can customize the popup dialog to prompt users to specify these various settings, or use the provided tags to render multiple Blox on the page to one PDF. Below are the tags and attributes:

```
<blox:pdfReport
    header=""
    headerHeight=""
    footer=""
    footerHeight=""
    margin=""
    size=""
    theme=""
    themeListEnabled="" >
    <blox:pdfDialogInput
        index=""
        displayName=""
        defaultValue=""
    />
</blox:pdfReport>
```

The pdfDialogInput tag lets you specify settings that you want users to provide values for. For example, you may want users to specify the header or footer for this PDF report. With:

```
<blox:pdfReport>
   <blox:pdfDialogInput
      displayName="Report header"
      defaultValue="Enter your header here"
```

```
      index="1" />
   <blox:pdfDialogInput
      displayName="Report footer"
      defaultValue="Enter your footer here"
      index="2" />
</blox:pdfReport>
```

Your users will be prompted with a dialog that allows them to specify the report header and footer.

For a detailed discussion of this rendering to PDF feature and the use of the tags, see the Converting to PDF section in the *Developer's Guide*.

The <blox:logo> Tag

This tag adds a "DB2 Alphablox" logo with a hyperlink link to DB2 Alphablox product Web site. All that is required is the following:

```
<blox:logo />
```

Exceptions

The Blox Java APIs throw exceptions if they reach error conditions, and, if you want to, you can catch those exceptions and do something with them. For example, you might want to catch an exception and send a specific error message to the user with instructions how to continue. The exceptions thrown by each API are documented in the API signature in the syntax descriptions.

If you want to use some of the exceptions, they are documented in the Javadoc shipped with DB2 Alphablox. The Javadoc is located in the following directory:

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

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

The general practice for catching exceptions is to use the `try...catch` syntax similar to the following pseudocode:

```
<%
try {
%>

<% original JSP Code %>

<% catch {Exception e}
{
   out.println(e.getMessage());
}
%>
```

Whether or not you catch exceptions, it is a good practice to add a custom error page. For details, see a JSP/Java reference book or the "Error Handling" section of the *Developer's Guide*.

Chapter 4. Common Blox Reference

This chapter contains reference material for properties and methods that are common to multiple Blox. For general reference information about Blox, see Chapter 3, "General Blox Reference Information," on page 17. For information on how to use this reference, see Chapter 1, "Using This Reference," on page 1.

Common Blox Properties and Methods by Category

The following tables list HTML properties and their corresponding methods that are common to multiple Blox. The properties and methods in this table are organized as follows:

Application and Session Properties

These properties affect application instantiation and the user session.

Properties	Methods
`applicationName`	`getApplicationName()`
`helpTargetFrame`	`getHelpTargetFrame()` `setHelpTargetFrame()`
`localeCode`	`getLocaleCode()` `setLocaleCode()`

Blox Properties

The properties in this section affect Blox behavior and appearance.

Properties	Methods
	`getDataBlox()`
`bloxEnabled`	`isBloxEnabled()` `setBloxEnabled()`

bloxName	
	getBloxName()
applyPropertiesAfterBookmark	
	isApplyPropertiesAfterBookmark() setApplyPropertiesAfterBookmark()
	loadBookmark() saveBookmark() saveBookmarkHidden()
bookmarkFilter	
	getBookmarkFilter() setBookmarkFilter()
maximumUndoSteps	
	getMaximumUndoSteps() setMaximumUndoSteps()
propertyNames	
	getPropertyNames()
	getProperty()

Blox Qualifiers—Used for Nested Blox

The following table lists the methods used to access a nested Blox from the top-level Blox. For example, you can use the `getDataBlox()` method on GridBlox to access the client-side DataBlox methods through GridBlox.

Methods
getChartBlox()
getDataBlox()
getDataLayoutBlox()
getGridBlox()
getPageBlox()

Bookmark and Application State Properties and Methods

The following table lists properties and methods associated with bookmarks.

Properties	Methods
bookmarkFilter	
	getBookmarkFilter() setBookmarkFilter()
applyPropertiesAfterBookmark	
	isApplyPropertiesAfterBookmark() setApplyPropertiesAfterBookmark()
readEnabled	
	isReadEnabled()
writeEnabled	
	isWriteEnabled()
	loadBookmark()

	`saveBookmark()` `saveBookmarkHidden()`

Server-side Event Filters and Listeners Methods

The following table lists the server-side Java methods used to add and remove event filter objects for capturing user events on the server *before* they are processed, and methods used to add and remove event listener objects for capturing events *after* they have been processed on the server.

Methods
addEventFilter()
addEventListener()
removeEventFilter()
removeEventListener()

Client-Side APIs

The following table lists the client-side JavaScript methods used to invoke server-side code from any user interface Blox on your page:

Methods
call()
getBloxAPI()
flushProperties()
getName()
isBusy()
setBusy()
setDataBusy()
updateProperties()

Rendering Properties

These properties affects the delivery of Blox in any of several formats.

Properties	Methods
`removeAction`	
	`getRemoveAction()` `setRemoveAction()`
`render`	
	`getRender()` `setRender()`
`rightClickMenuEnabled`	
	`isRightClickMenuEnabled()` `setRightClickMenuEnabled()`

Menubar Properties

The property below determines if the menubar is visible in a PresentBlox, GridBlox, or ChartBlox:

Property	Methods

menubarVisible	
	isMenubarVisible() setMenubarVisible()

Popped Out Properties

The following table lists the properties regarding displaying a PresentBlox, a standalone GridBlox, or a standalone ChartBlox in a separate, popped out browser window.

Chart Labels

Properties	Methods
enablePoppedOut	
	isEnablePoppedOut() setPoppedOut()
poppedOut	
	isPoppedOut() setPoppedOut()
poppedOutHeight	
	getPoppedOutHeight() setPoppedOutHeight()
poppedOutTitle	
	getPoppedOutTitle() setPoppedOutTitle()
poppedOutWidth	
	getPoppedOutWidth() setPoppedOutWidth()

Properties and Associated Methods Common to Multiple Blox

This section describes the properties supported by multiple Blox and the methods associated with those properties. The properties are listed alphabetically by property name. For a list of common methods with which no properties are associated, see "Methods Common to Multiple Blox" on page 48.

For each Blox to which a property is valid, there is an entry in the property section for that Blox with a cross reference to the description in this section. Also, the custom tag section for each Blox lists all the properties supported on that Blox.

applicationName

The application context name.

Data Sources

All

Syntax

Java Method

```
String getApplicationName();
```

applyPropertiesAfterBookmark

Specifies whether, after retrieving a bookmark, the Blox properties should override those in the bookmark.

Data Sources
All

Syntax
JSP Tag Attribute

```
applyPropertiesAfterBookmark="applyAfterBookmark"
```

Java Methods

```
boolean isApplyPropertiesAfterBookmark();   // returns boolean
void setApplyPropertiesAfterBookmark(boolean
                                     applyAfterBookmark);
```

where:

Argument	Default	Description
applyAfterBookmark	false	Specify true to apply any properties specified in the Blox custom tags after a bookmark is loaded.

Usage
A value of true overwrites the bookmark property values with those on the application page.

Note: The DataBlox dataSourceName property ignores the applyPropertiesAfterBookmark() setting. If data source A is currently used by a PresentBlox on a page and the user loads a bookmark that was saved on data source B, data source B will be used and loaded even if applyPropertiesAfterBookmark is set to true.

Examples

```
isApplyPropertiesAfterBookmark();
setApplyPropertiesAfterBookmark(true);
```

bookmarkFilter

Specifies a default location from which to store and load bookmarks. You can use this location to provide grouping and visibility for bookmarks, providing application developers control over the set of bookmarks available to end users. The filter you specify causes the bookmarks to be stored in a subdirectory called filterName under the usual bookmark directory (public, private, or group) in the DB2 Alphablox repository.

Additionally, the bookmarkFilter property can allow bookmarks to be shared across multiple Blox and/or multiple applications.

Data Sources
All

Syntax
JSP Tag Attribute

```
bookmarkFilter="filterName"
```

Java Methods

```
String getBookmarkFilter();
void setBookmarkFilter(String filterName);
```

where:

Argument	Default	Description
filterName	none	There are two forms of this argument. The first is a String specifying the name of subdirectory. The second form is a String with a comma separated list of the form: "*subDirectory*, name=*BloxID*, application=*AppContext*, user=*UserName*" The name, application, and user clauses are all optional; the subDirectory clause is required.

Usage

The bookmarkFilter property allows you to categorize the bookmarks of each Blox, providing greater flexibility of bookmarks.

For example, consider an application with a single PresentBlox that has two entry points (for example, two links that run different queries from different data sources, but display the results in the same PresentBlox), one for marketing and one for sales. You might want to set the bookmarkFilter property for the first link so that all users in marketing see bookmarks created from the marketing part of the application, and set the bookmarkFilter property for the second link so all users in sales see the bookmarks created in the sales part of the application. The net result is that users in sales see one set of bookmarks while users in marketing see another, even if they are both using the same PresentBlox to display the data. This scheme allows you to minimize the number of Blox on your page. Minimizing the number of Blox is a useful resource optimization tool, especially useful in applications running in Netscape browsers, where multiple page refreshes with multiple applets on the page sometimes causes the browser to behave unexpectedly.

The second form of the filterName argument described in the syntax section is useful in applications where the Blox is created dynamically based on programmatic details (for example, user profiles, fiscal quarter, etc.). Such dynamic Blox creation often results in the Blox having a different name each time the Blox is created, making it difficult to have a consistent set of bookmarks across different Blox instances. Setting the bookmarkFilter based on some criteria (BloxID, application, and/or user) ensures the desired set of bookmarks are available to the appropriate users.

Examples

```
getBookmarkFilter();
   // returns the current setting of the bookmarkFilter property

setBookmarkFilter("sales");   // Sets the bookmarkFilter
   // Property to store bookmarks in the "sales" subdirectory
   // of the bookmarks folder in the DB2 Alphablox repository.
   // Other Blox or applications can then access these bookmarks by
   // appropriately setting the bookmarkFilter property.
```

The following example sets the bookmarks so they are stored and retrieved in the `marketingBookmarks` subdirectory of the `marketing` application of the DB2 Alphablox repository.

```
setBookmarkFilter("marketingBookmarks, name="myPresentBlox",
                  application=marketing");
```

See Also

"saveBookmarkHidden()" on page 58

bloxEnabled

Specifies whether or not the Blox interface is interactive and greyed out.

Data Sources

All

Syntax

JSP Tag Attribute

```
bloxEnabled="enable"
```

Java Methods

```
boolean isBloxEnabled();
void setBloxEnabled(boolean enable);
```

where:

Argument	Default	Description
`enable`	`true`	Specify `true` to enable interactivity and display the Blox, `false` to disable interactivity and grey out the Blox.

Usage

A value of `false` presents a greyed-out Blox that is not interactive. A value of `true` presents an interactive interface, allowing the user to drill up and down, change chart types, and so forth. If you want the Blox to display (not greyed out), but do not want users to interact with the data, use the `bloxui:component` tag's `clickable` attribute. See "Example 3: Setting a PresentBlox Unclickable" on page 796.

Examples

```
isBloxEnabled();
setBloxEnabled(false);
```

bloxName

Specifies the name of the Blox. This is an optional attribute and is an advanced feature that allows you to dynamically set the name of Blox and its corresponding JavaScript name.

Data Sources

All

Syntax

JSP Tag Attribute

```
bloxName="bloxName"
```

Java Method

```
String getBloxName();
```

Usage

When you define a Blox using Blox tags, you need to uniquely identify the outmost Blox using the `id` attribute (nested Blox cannot have a separate `id` and is referenced using the qualifier, as described in "Blox Qualifiers—Used for Nested Blox" on page 30). This `id` is required, cannot be dynamically set, and is used for two purposes:

- as the scripting variable name within your JSP page, and
- as the name of the Blox and its corresponding JavaScript object created under DB2 Alphablox (used in JavaScripting your Blox within the browser)

If the optional `bloxName` attribute is not specified, the `id` is used as both a Java scripting variable and the name of the Blox object on the server.

The use of `id` alone as both the Blox name and the Java scripting variable name is probably sufficient for all your development needs. Only in a few cases you may want to separate the two so you can dynamically create Blox names with tags. It is also useful for reusing server-side code (such as using the client-side call() method to execute server-side code that acts on the Blox or as demonstrated in the examples below). If you specify the value of `bloxName` for a Blox, then:

- this `bloxName` will be the name of the Blox DB2 Alphablox knows this object as
- this `bloxName` will be the name of the rendered JavaScript object (to be used in your JavaScript code when referencing the Blox)
- `id` will now only serve as the Java scripting variable you use in your JSP page.

Separating the scripting variable name and the Blox name.: This following example demonstrates the differences between the scripting variable name and the Blox name when `bloxName` is specified. The code creates a GridBlox called `salesDataGrid`, which will not be displayed initially (`visible="false"`):

```
...
<blox:grid id="myGrid" bloxName="salesDataGrid"
    visible="false"
    width="400
    height="360"
    <blox:data dataSourceName="qcc"
        query="!" />
    <%
    //you can set properties using id within the grid tag as
    //it is now a scripting variable
    myGrid.setBandingEnabled(true);
    ....
    %>

</blox:grid>
...
//In your scriptlet within the page, you script to the grid using
//its id
<%
    myGrid.getDataBlox().setQuery(newQuery);
    myGrid.getDataBlox().connect();
%>
```

Notice that you script to this grid using its scripting variable name, which is the value of `id`. After some processing logic is done, this GridBlox is displayed using the `<blox:display>` tag:

```
//In other Blox tags that reference this Blox, use the Blox name
<blox:display bloxRef="salesDataGrid" />
```

Dynamically setting the value of the bloxName attribute: You may have a JSP page named `setAlerts.jsp` that sets cell alert format for the specified threshold on any GridBlox that is passed in:

```
<!--This page is called by another JSP, with two parameters-->
<!--"blox" and "low" passed in along with the request.-->
<%@ taglib uri="bloxtld" prefix="blox"%>
<%
    //Blox name is passed in as a request parameter
    String gridName = request.getParameter("blox");
    String lowValue = request.getParameter("low");
%>

<blox:grid id="someGrid" bloxName="<%= gridName %>"/>

<%
    someGrid.setCellAlert(1,"condition=LT,value=" + lowValue +
",foreground=white,background=red");
    return;
%>
```

Or you may want to reuse some generic JSP code:

```
<blox:grid id="someGrid" bloxName="<%=currBloxName %>"
    .... />
<%@ include file="gridDefaults.jsp" %>
```

where `gridDefaults.jsp` does the following:

```
someGrid.setBandingEnabled(true);
someGrid.setCellFormat(1, "format=#,##0.00,
      scope={Accounts:COGS}");
```

In summary:

- `id` is required; `bloxName` is optional.
- The name of a Blox is the value of the `bloxName` attribute if it is specified in the tag. If it is not supplied, the value of the `id` attribute is used both as the Blox name and the Java scripting variable name.
- Throughout this documentation set, the phrase “Blox name” refers to the value of the `id` attribute unless the value of the `bloxName` attribute is supplied.
- In most cases, you only need to specify the `id` and do not need to worry about `bloxName`. Only when you need to separate the scripting variable name and the Blox name, do you need to specify `bloxName`.
- If you do specify the value of the `bloxName` attribute:
  - `id` is the Java scripting variable name to script to in your JSP page
  - `bloxName` is the Blox name you should use when referencing it

Note: `bloxName` cannot be a number, start with a number, or contain any special characters such as `~`, `!`, `@`, `#`, `$`, `%`, `^`, `&`, `*`, `-`, `+`, `=`, `(`, `)`, `?`, `<`, `>`, `/`, `:`, `;`, `'`, or `"`.

Note: If you call the `getBloxName()` method on a nested Blox, it returns the name generated for the Blox.

Examples

The following code creates a local scripting variable named `myGrid` and a Grid peer named `salesGrid`.

```
<% String bloxName="salesGrid"; %>
<blox:grid id="myGrid" bloxName="<%= bloxName %>" .../>
```

To script to this grid, use the grid's scripting variable name (`id`). The following code show the results of the `getBloxName()` method. The comments indicate the returned values.

```
<%
    myGrid.getBloxName(); // returns the string "salesGrid"
    myGrid.getDataBlox().getBloxName();
    //returns the generated name for the nested DataBlox (for
    //example, "salesGrid_data")
```

See Also

"id" on page 39

bloxModel

This is a ContainerBlox property. See "bloxModel" on page 312.

bloxRef

Specifies the name of another Blox to use. The `bloxRef` attribute is available through the DataBlox (`blox:data`) and Display (`blox:display`) custom tag libraries.

Data Sources

All

Syntax

JSP Tag Attribute

`bloxRef="`*`bloxName`*`"`

Usage

Use the `bloxRef` tag attribute in a nested Blox to refer to a Blox that was created as a separate Blox.

Examples

If the following DataBlox was created in the `<head>` section of an HTML page:

```
<blox:data id="DataBlox1"
           dataSourceName="TBC"
           query="!"
</blox:data>
```

You could then reference that DataBlox within other Blox (for example, GridBlox) as a nested Blox, referring to it with the `bloxRef` attribute as follows:

```
<blox:grid id="myGrid" >
    <blox:data bloxRef="DataBlox1" />
</blox:grid>
```

enablePoppedOut

This is a property inherited from ContainerBlox. For a complete description, see "enablePoppedOut" on page 313.

height

Specifies the height of the Blox on the page.

Data Sources

All

Syntax

JSP Tag Attribute

```
height="height"
```

Java Methods

```
String getHeight();
void setHeight(String height);
```

Usage

Specifies the height of the Blox display area. The value can be expressed as pixels (`height="300"`) or as a percentage of the browser display area (`height="40%"`).

helpTargetFrame

Identifies the target browser window or frameset frame in which user help appears.

Data Sources

All

Syntax

JSP Tag Attribute

```
helpTargetFrame="helpTargetFrame"
```

Java Methods

```
String getHelpTargetFrame();
            throws ServerBloxException
void setHelpTargetFrame(String helpTargetFrame);
            throws InvalidBloxPropertyValueException,
                   ServerBloxException
```

where:

Argument	Default	Description
`helpTargetFrame`	`"AlphabloxHelp"`	String identifying a browser window or frameset.

Usage

The default is `AlphabloxHelp`, which is a separate browser window.

Examples

```
getHelpTargetFrame();
setHelpTargetFrame("Browser Window Name");
```

id

Specifies the name of the Blox. This name can then be referenced from other Blox or from Java or JavaScript code on the JSP page.

Data Sources

All

Syntax

JSP Tag Attribute

```
id="idString"
```

Usage

The `id` attribute is valid only on the outer Blox; nested Blox cannot have an `id` attribute. If you specify the optional `bloxName` attribute, then `id` will serve only as the Java scripting variable name in your JSP page. The value of `bloxName` will be the name of the Blox peer created on the server and the name of the rendered JavaScript object.

where:

Argument	Default	Description
`idString`	none	An identification string representing the name of the Blox. The `idString` can be referenced from other Blox tags using the `bloxRef` attribute.

Note: `id` cannot be a number, start with a number, or contain any special characters such as `~`, `!`, `@`, `#`, `$`, `%`, `^`, `&`, `*`, `-`, `+`, `=`, `(`, `)`, `?`, `<`, `>`, `/`, `:`, `;`, `'`, or `"`.

See Also

"bloxName" on page 35

lastAppliedApplicationStateName

The name of the last applied application state.

Data Sources

All

Syntax

Java Method

```
String getLastAppliedApplicationStateName();
```

localeCode

Sets the locale for formatting numeric values. You can use this property to display numeric formats in a different locale than the one in which DB2 Alphablox is running. Typically, you will need to add code to your application to set the `localeCode` property based on the user so users in France see numbers formatted for their locale, users in Germany see numbers formatted for their locale, and so on.

Data Sources

All

Syntax

JSP Tag Attribute

```
localeCode="locale"
```

Java Methods

```
String getLocaleCode();
void setLocaleCode(String locale);
        throws InvalidBloxPropertyValueException,
                ServerBloxException
```

where:

Argument	Default	Description

`locale`	none	This is a two part parameter. The first part is the 2 digit language code and the second part is the two digit country code. They are separated by an underscore (_). You can find the language codes from the following:
		`http://lcweb.loc.gov/standards/iso639-2/langcodes.html`
		You can find the country codes at the following:
		`ftp://ftp.ripe.net/iso3166-countrycodes.txt`
		In both cases it is the two digit code which is used.

Usage

You should set the `localeCode` property on the outer Blox (for example, on a PresentBlox); setting the property on an inner Blox does not affect the outer Blox.

If you use the `setLocaleCode()` method to set the `localeCode` on an outer Blox (for example, on a PresentBlox), the inner Blox (for example, a GridBlox) will use that value for the `localeCode` property; however calling `getLocaleCode()` on the inner Blox (for example, on the GridBlox) returns the original value, not the value that is being used.

To personalize the `localeCode` property based on the user profile, define a custom user property with valid values for users in different countries. Then, create a RepositoryBlox on your application page, get the value of your custom user property (with the RepositoryBlox's getUserProperty() method), and set the `localeCode` property accordingly (using the `setLocaleCode()` method).

If you do not set the `localeCode` property, the default value is the locale in which DB2 Alphablox is running. Do not set the `localeCode` property to different values for different Blox on the same page.

Examples

To set the locale code to English in the United States:

```
setLocaleCode("en_US");
```

To set the locale code to English in the United Kingdom:

```
setLocaleCode("en_GB");
```

maximumUndoSteps

Specifies the maximum number of steps to be tracked in the Menubar's Undo button.

Data Sources

All

Syntax

JSP Tag Attribute:

```
maximumUndoSteps="steps"
```

Java Methods

```
int getMaximumUndoSteps();  //throws ServerBloxException
```

```
void setMaximumUndoSteps(int steps);
  // throws InvalidBloxPropertyValueException,ServerBloxException
```

where:

Argument	Default	Description
steps	50	The maximum numbers of steps to be tracked.

Usage

This property applies to PresentBlox, GridBlox, ChartBlox, DataLayoutBlox, and PageBlox. When this property is set to 0, the Undo and Redo buttons and menu items will be removed from the toolbar and menubar.

menubarVisible

Specifies whether or not a text-based menubar will appear at the top of the Blox.

Data Sources

All

Syntax

JSP Tag Attribute

```
menubarVisible="visible"
```

Java Methods

```
boolean isMenubarVisible();
void setMenubarVisible(boolean visible);
```

where:

Argument	Default	Description
visible	true	Set to true to show the menubar, set to false to hide the menubar. The default is true (when the default application render mode is set to DHTML).

Usage

The contents of the menubar and its drop-down menus automatically match the contents of the Blox.

Examples

```
isMenubarVisible();
setMenubarVisible(true);
```

noDataMessage

Sets the string to be displayed on the Blox when it has no data.

Data Sources

Multidimensional

Syntax

JSP Tag Attribute

```
noDataMessage="message"
```

Java Methods

```
String getNoDataMessage();
void setNoDataMessage(String message);
```

where:

Argument	Default	Description
message	"Data not available"	Any string.

Usage

When a new message is set, the screen will not get updated until next time a result set is returned with no data (such as after a call to `updateResultSet()` or `connect()`). With DataLayoutBlox, this property is ignored and no message appears if no data is available.

Examples

```
getNoDataMessage();
setNoDataMessage("No data at this time");
```

poppedOut

This is a property inherited from ContainerBlox. For a complete description, see "poppedOut" on page 314.

poppedOutHeight

This is a property inherited from ContainerBlox. For a complete description, see "poppedOutHeight" on page 315.

poppedOutTitle

This is a property inherited from ContainerBlox. For a complete description, see "poppedOutTitle" on page 315.

poppedOutWidth

This is a property inherited from ContainerBlox. For a complete description, see "poppedOutWidth" on page 316.

propertyNames

A String array containing a list of all properties.

Data Sources

All

Syntax

Java Method

```
String[] getPropertyNames();
```

Usage

Use this list in conjunction with `getProperty(String` *propertyName*`)` to get the specific value of each property. Returns `null` if this Blox does not support any properties.

See Also

"getProperty()" on page 53

readEnabled

Specifies if the current user has permissions to read from the repository. For example, read permissions would allow the current user to load a bookmark.

Data Sources

All

Syntax

Java Method

```
boolean isReadEnabled();
```

Usage

Returns `true` if the current user has read permissions. This method is useful to check for read permissions on the DB2 Alphablox repository before attempting to read from it.

See Also

"writeEnabled" on page 48

removeAction

Specifies which (if any) data analysis actions to remove from the right-click menu and the Data menu in the menubar.

Data Sources

All

Syntax

JSP Tag Attribute

```
removeAction="dataActions"
```

Java Methods

```
String getRemoveAction();
void setRemoveAction(String dataActions);
```

where:

Argument	Default	Description
`dataActions`	empty string	Comma-delimited list of actions to remove.

Usage

Valid entries in the list are:

- `Find`
- `Drill Up`
- `Drill Down`
- `Pivot`
- `Data Sort`
- `Remove Only`
- `Keep Only`
- `Hide Only`
- `Show Only`
- `Show All`
- `Expand All`
- `Show Bottom Level`
- `Show Siblings`

- Swap
- Drill Through
- Member Filter
- Comments
- Traffic Lights

Examples

Using the tag:

```
removeAction="Keep Only, Remove Only, Pivot"
```

Using the Java methods:

```
setRemoveAction("Keep Only", "Remove Only", "Pivot");
```

render

Specifies the delivery format for a specific Blox on an application page.

Data Sources

All

Syntax

JSP Tag Attribute

```
render="renderMode"
```

Java Methods

```
String getRender();
              throws ServerBloxException
void setRender(String renderMode);
              throws InvalidBloxPropertyValueException,
              ServerBloxException
```

where:

Argument	Default	Description
renderMode	dhtml	The mode in which the page is rendered. See the table below for the possible values.

Usage

Using this property enables different delivery formats for Blox on the same page. Setting this property on an individual Blox overrides the render attribute on the application's URL. The render attribute applies to all Blox on the page; this property applies to a specific Blox. Therefore, to ensure that a Blox is delivered in a specific format only, use the render property on the Blox.

Possible values are:

Value	
dhtml	Render in fully interactive DHTML format (the default). Requires the <blox:header> tag in your JSP page
html	Render in static HTML format. Requires the <blox:header> tag in your JSP page.
printer	Render in a format suitable for printing (many browsers do not support printing the output of

	interactive Java applets). Requires the `<blox:header>` tag in your JSP page.
`xls`	Render a format suitable for export to Microsoft Excel, and set the MIME type to XLS. **Note:** In order for the MIME type to be set so the page opens in Excel, you must put the `<blox:header>` tag in your JSP page. For information on the `<blox:header>` tag, see "<blox:header> Tag in the HTML <head>" on page 20.
`xml`	Render in XML format. This format applies only to DataBlox. For more information, see Chapter 28, "Using the Alphablox XML Cube," on page 855. **Note:** To render to Microsoft Excel format, you must use the URL `render=xls` attribute.

rightClickMenuEnabled

Specifies whether the right-click menu in the Blox user interface should be turned on or off.

Data Sources

All

Syntax

JSP Tag Attribute

```
rightClickMenuEnabled="enabled"
```

Java Methods

```
boolean isRightClickMenuEnabled();
void setRightClickMenuEnabled(boolean enabled)
```

where:

Argument	Default	Description
`enabled`	`true`	When set to `true`, the right-click menu is enabled, allowing users to perform data analysis or manipulation tasks. When set to `false`, the right-click menu is disabled. The default is `true`.

Usage

Only GridBlox and ChartBlox have a right-click menu with various data navigation options.

visible

Specifies whether a Blox is visible on the page.

Data Sources

All

Syntax

JSP Tag Attribute

```
visible="boolean"
```

Java Methods

```
boolean isVisible();
     throws ServerBloxException
void setVisible(boolean)
     throws InvalidBloxPropertyValueException,
            ServerBloxException
```

where:

Argument	Default	Description
boolean	true	Set to true to cause the Blox to render to the page, set to false if you want the Blox created but not to display on the page.

Usage

Set the visible property to false to create a Blox but not display it. You can later display the Blox using the <blox:display> tag. The default value is true.

If using the visible property on ToolbarBlox, carefully consider the user implications of turning the toolbar off. Most applications need to provide some of the functionality provided through either the Blox toolbars or menubars. If the menubar is turned off in a Blox, options such as Undo/Redo buttons, export to PDF/Excel, and turning on/off the grid, the chart, the page filter, and the data layout panel are only available through the toolbars.

width

Specifies the width of the Blox on the page.

Data Sources

All

Syntax

JSP Tag Attribute

```
width="width"
```

Java Methods

```
String getWidth();
void setWidth(String width);
```

where:

Argument	Default	Description
width	none	A string representing the valid HTML width values.

Usage

Specifies the width of the Blox display area. The value can be expressed as pixels (width="500") or as a percentage of the browser display area (width="50%").

writeEnabled

Specifies if the current user has permissions to write to the repository. For example, write permissions would allow the current user to save a bookmark.

Data Sources

All

Syntax

Java Method

```
boolean isWriteEnabled();
```

Usage

Returns `true` if the current user has write permissions. This method is useful to check for write permissions on the DB2 Alphablox repository before attempting to write to it. Typically, a user must be assigned the `AlphabloxAdministrator` role to write to the repository.

See Also

"readEnabled" on page 44

Methods Common to Multiple Blox

This section describes the methods common to multiple Blox that are not associated with a specific property. To see if a method is valid for a particular Blox, see the methods section for that Blox. For the syntax and descriptions of the common methods that have a property associated with them, see "Properties and Associated Methods Common to Multiple Blox" on page 32.

addEventFilter()

Adds a server-side event filter so the specified filter is called *before* the event is processed on the server.

Data Sources

All

Syntax

Java Method

```
void addEventFilter(EventFilter filter)
        throws ServerBloxException
```

where:

Argument	Description
`filter`	An event filter object. See available event filters in Chapter 13, "Event Filter Objects," on page 463.

Usage

Event filters let you capture some user event such as drilling down, pivoting, and adding a bookmark and perform some custom actions before the event is actually processed. For example, to capture a user drilldown event:

1. Add a server-side drilldown event filter to the DataBlox inside the Blox tag:

```
<blox:present id="myPresentBlox">
  ...
  <%
```

```
    myPresentBlox.getDataBlox().addEventFilter( new DDFilter() );
  %>
  ...
</blox:present>
```

2. Have your event filter object implement the DrillDownFilter interface:

```
<%!
  public class DDFilter implements DrillDownFilter
  {
    ...
  }
%>
```

3. Add actions to take when the `drillDown` method is called. The method takes a `DrillDownEvent` object as input:

```
<%!
public class DDFilter implements DrillDownFilter
{
   BloxModel model;

// drillDown is the method to implement to capture a drilldown
// events. It takes a DrillDownEvent object as input.
   public void drillDown( DrillDownEvent dde ) throws Exception
   {
      DataBlox blox = dde.getDataBlox();
      StringBuffer msg = new StringBuffer("-------------------------");
      msg.append("DRILL DOWN event on " + blox.getBloxName() + "\n");
      msg.append("Axis ID: " + dde.getAxisIndex() + ", " );
      msg.append("Nest level: " + dde.getNestLevel() + ", " );
      msg.append("Member index: " + dde.getMemberIndex() + ", " );
      msg.append("Member: " + dde.getMember().getDisplayName());

      //Write the output using a MessageBox. Note that this requires
      //importing the com.alphablox.blox.uimodel.core.MessageBox and
      //com.alphablox.blox.uimodel.BloxModel packages.
      MessageBox msgBox = new MessageBox(msg.toString(), "DrillDown Filter
Message", MessageBox.MESSAGE_OK, null);
      model.getDispatcher().showDialog(msgBox);
   }
}
%>
```

To remove an event filter, use removeEventFilter(). For post-operation notification, use addEventListener().

See Also

"removeEventFilter()" on page 55. For details on event filters and associated methods, see Chapter 13, "Event Filter Objects," on page 463 and the `com.alphablox.blox.filter` package in the Javadoc.

addEventListener()

Adds a server-side event listener so the specified listener is called *after* the event has been processed on the server.

Data Sources

All

Syntax

Java Method

```
void addEventListener(java.util.EventListener listener)
        throws ServerBloxException
```

where:

Argument	Description
`listener`	An event listener.

Usage

Event listeners let you capture some user action such as drilling down, pivoting, and adding a bookmark and perform some custom actions *after* the user action has been processed. For example, you may want to update another Blox, handle an exception that is a side-effect of the event, or sending messages back to the client based on the results of the event. To add a server-side event listener for the drilldown operation:

1. Add a server-side drilldown event listener to the DataBlox:

```
<blox:present id="myPresentBlox">
   <blox:data bloxRef="myData"/>
   ...

<%
  myPresentBlox.getDataBlox().addEventListener( new DDHandler);
%>
...
</blox:present>
```

   Note that the listener is added inside the Blox tag so it is added only once even if the page is reloaded.

2. Have your event listener object implement the appropriate event listener interface:

```
<%!
  public class DDHandler implements DrillDownListener
  {
    ...
  }
%>
```

3. Add actions to take after the `drillDown` method is called. The `drillDown` method must be implemented, and the method takes a `DrillDownEvent` object as input:

```
  <%!
  public class DDFilter implements DrillDownListener
  {
   BloxModel model;

  // drillDown is the method to implement to capture a drilldown
  // events. It takes a DrillDownEvent object as input.
    public void drillDown( DrillDownEvent dde )
    {
      DataBlox blox = dde.getDataBlox();
      StringBuffer msg = new StringBuffer("-----------");
      msg.append("DRILL DOWN event on " + blox.getBloxName() + "\n");
      msg.append("Axis ID: " + dde.getAxisIndex() + ", ");
      msg.append("Nest level: " + dde.getNestLevel() + ", ");
      msg.append("Member index: " + dde.getMemberIndex() + ", ");
      msg.append("Member: " + dde.getMemberName());

      //Write the output using a MessageBox. Note that this requires
      //importing the com.alphablox.blox.uimodel.core.MessageBox and
      //com.alphablox.blox.uimodel.BloxModel packages.
      MessageBox msgBox = new MessageBox(msg.toString(), "DrillDown Event
Listener Message", MessageBox.MESSAGE_OK, null);
      model.getDispatcher().showDialog(msgBox);
    }
  }
  %>
```

To remove an event listener, use removeEventListener(). For pre-operation notification, use addEventFilter().

See Also

"removeEventListener()" on page 56. For details on event listener and associated methods, see Chapter 14, "Event Listener Objects," on page 501 and the `com.alphablox.blox.event` package.

call()

Calls a URL to execute on the server and returns the results of the HTTP request as a String. This method is useful to execute server-side code from the client. The `call()` method executes the server-side code without refreshing the page.

Data Sources

All

Syntax

JavaScript Method

```
call(callURL); // returns String
```

where:

Argument	Description
`callURL`	A String containing a URL of a file (typically a JSP file) to be run on the server.

Usage

Use the `call()` method to execute, from a client-side method, server-side code. You might use this method to set properties on the server or execute other server-side logic. This code is executed without refreshing the page (if your application is rendered in `html` mode, the `call()` method does cause a page refresh).

The `call()` method automatically flushes any pending transactions on the Blox, ensuring that any properties that have been set by users have propagated down to the server.

The `callURL` string can reference a JSP file that does not actually send anything to the client, but just performs various server actions. The URL can be absolute or relative:

- 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. Note that 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.

Note that for absolute URLs, if the render mode is Java, you must call the same server that delivered the applet. This is due to the Java applet security policy.

Note: The default encoding of the response text from the `call()` method is UTF-8 if not otherwise specified. If you need a different encoding, specify your encoding in your JSP page directive, for example:

```
<%@ page contentTyp"text/html; charset=SHIFT_JIS" %>
```

Examples

```
myPresent.call("http://myserver/myapp/RunSomeCode.jsp"); //absolute URL
myPresent.call("/myapp/RunSomeCode.jsp");   //relative to server root
```

See Also

"BloxAPI Methods" on page 65, "setDataBusy()" on page 60; "Example 2: Set Chart Properties on the Server Using the bloxAPI.call() Method" on page 911.

flushProperties()

Ensures that all properties set on the client (for example, through user actions in the user interface) are propagated ("flushed") to the server.

Data Sources

All

Syntax

JavaScript Method

```
flushProperties(); // no return value
```

Usage

This method flushes to DB2 Alphablox all pending property changes for all of the Blox, so you only need to call it from one Blox to flush all properties on all Blox.

See Also

"call()" on page 51, "updateProperties()" on page 61

getApplicationName()

Returns the context name of the J2EE application to which the page belongs.

Data Sources

All

Syntax

Java Method

```
String getApplicationName();
         throws ServerBloxException
```

getBloxAPI()

Returns the global framework object.

Data Sources

All

Syntax

JavaScript Method

```
BloxAPI getBloxAPI();
```

Usage

This BloxAPI object can also be obtained using the `bloxAPI` variable available in each frame.

See Also

Chapter 5, "Client-Side API Reference," on page 63

getDataBlox()

Returns an interface to the server-side (for the Java method) DataBlox from GridBlox, ChartBlox, PresentBlox, and DataLayoutBlox.

Data Sources

All

Syntax

Java Method

```
DataBlox getDataBlox();
        throws ServerBloxMissingResourceException,
                ServerBloxException
```

See Also

"setDataBlox()" on page 60

getName()

Returns the name of the Blox.

Data Sources

All

Syntax

JavaScript Method

```
String getName();
```

getProperty()

Returns the value for a specified property.

Data Sources

All

Syntax

Java Method

```
String getProperty(String name);
```

where:

Argument	Default	Description
`name`	none	Name of a property.

Usage

The `getProperty()` method returns `null` if a property has not been set.

See Also

"setProperty()" on page 61

getServerContextPath()

Returns the properly formatted AlphabloxServer Context URL including any prefix.

Data Sources

All

Syntax
Java Method

```
String getServerContextPath();
```

isBusy()

Returns the current busy state for the Blox.

Data Sources
All

Syntax
JavaScript Method

```
boolean isBusy();
```

Usage
Returns `true` if the Blox is busy; `false` if not.

See Also
"setBusy()" on page 59, Chapter 5, "Client-Side API Reference," on page 63

init()

Sets the peer to which a server-side Blox binds. If the peer does not exist, it is created automatically by the `init()` method.

Data Sources
All

Syntax
Java Method

```
boolean init(BloxContext bloxContext,
             java.lang.String bloxName);
       // throws ServerBloxException
```

where:

Argument	Default	Description
`bloxContext`	none	The BloxContext in which this Blox will be contained.
`bloxName`	none	The name of the Blox to which this Blox binds.

Usage
This method is used when creating a bean using the JSP `useBean` syntax; if you create a Blox using the custom tag libraries, the `init()` method is not needed. To get the BloxContext:

```
<%@ page import="com.alphablox.blox.*"%>
<%
BloxContext bc = (BloxContext) session.getAttribute(BloxContext.BLOX_CONTEXT_ATTR);
%>
```

loadBookmark()

Restores the state of the Blox to the state stored in the bookmark.

Data Sources
All

Syntax

Java Method

```
void loadBookmark(int visibility, String owner,
                  String bookmarkName);
void loadBookmark(Bookmark bookmark);
          // throws ServerBloxException
```

where:

Argument	Default	Description
bookmarkName	null	String name of bookmark.
visibility	VISIBILITY_ APPLICATION	Visibility bookmark is saved under. Cannot be changed once specified. The valid integer values are RepositoryBlox.VISIBILITY_GROUP, RepositoryBlox.VISIBILITY_APPLICATION, and RepositoryBlox.VISIBILITY_PRIVATE.
owner	null	The name of the bookmark group. It is ignored if the visibility is not RepositoryBlox.VISIBILITY_GROUP.
bookmark	null	A Bookmark object. See “Bookmark Object Properties and Methods Cross References” on page 139.

Usage

Bookmarks are a good way to keep track of different data views in an application. You can set a bookmark’s visibility so that only users in certain access groups can see it. Bookmarks can be public, private, or belong to another defined user group.

Examples

```
loadBookmark(RepositoryBlox.VISIBILITY_GROUP, "Team 5",
               "profit analysis");
```

See Also

“saveBookmark()” on page 57, “bookmarkFilter” on page 33.

removeEventFilter()

Removes the specified server-side event filter that was added with the addEventFilter() method.

Data Sources

All

Syntax

Java Method

```
void removeEventFilter(EventFilter filter)
         throws ServerBloxException
```

where:

Argument	Description
filter	An event filter object added using the addEventFilter() method.

See Also

See "addEventFilter()" on page 48 for details on what event filter objects are and how to use them. See Chapter 13, "Event Filter Objects," on page 463 and the `com.alphablox.blox.filter` package for details on event filters and associated methods.

removeEventListener()

Removes the specified server-side event listener that was added with the addEventListener() method.

Data Sources

All

Syntax

Java Method

```
void removeEventListener(java.util.EventListener listener)
          throws ServerBloxException
```

where:

Argument	Description
`listener`	An event listener object added using the addEventListener() method.

See Also

See "addEventListener()" on page 49 for details on what event listener objects are and how to use them. See Chapter 14, "Event Listener Objects," on page 501 and the `com.alphablox.blox.event` package for details on event listeners and associated methods.

render()

Renders the Blox.

Data Sources

All

Syntax

Java Methods

```
void render(javax.servlet.http.HttpServletRequest request,
            java.io.Writer out, String renderMethod, String width,
            String height);
void render(javax.servlet.http.HttpServletRequest request,
            java.io.Writer out, String renderMethod);
void render(javax.servlet.http.HttpServletRequest request,
            java.io.Writer out);
       throws ServerBloxMissingResourceException,
              ServerBloxRenderException,
              InvalidParameterException,
              ServerBloxException,
              DataBloxCannotConnectException
```

where:

Argument	Default	Description
`request`	none	The HTTP request object.

Argument	Default	Description
out	none	The Writer Object.
renderMethod	none	The mode in which to render the Blox .
width	none	The width of the Blox to render.
height	none	The height of the Blox to render.

Usage

There are versions of this method that do not require you to specify the renderMethod, width, and height arguments.

The render() method will render the Blox regardless of the value of the visible property for a Blox.

renderHtmlHeader()

Generate the HTML code for the document's render mode and theme.

Data Sources

All

Syntax

Java Method

```
void renderHtmlHeader(javax.servlet.http.HttpServletRequest request,
                      javax.servlet.http.HttpServletResponse response);
```

where:

Argument	Default	Description
request	none	The HTTP request object.
response	none	The HTTP response object.

Usage

This is the method behind the <blox:header> tag, which has two tag attributes—contextPath and pageURL and has a nested <blox:clientBean> tag. The <blox:clientBean> tag registers the server-side bean with the Alphablox programming framework and makes the bean's methods available on the client.

saveBookmark()

Save the current state to the specified bookmark.

Data Sources

All

Syntax

Java Method

```
void saveBookmark(int visibility, String owner,
                  String bookmarkName, String description)
```

where:

Argument	Default	Description
bookmarkName	null	String name of bookmark.
description	null	New bookmark description.

Argument	Default	Description
owner	null	The name of the bookmark group. It is ignored if the visibility is not RepositoryBlox.VISIBILITY_GROUP.
visibility	none	Visibility bookmark is saved under. Cannot be changed once specified. The valid integer values are RepositoryBlox.VISIBILITY_GROUP, RepositoryBlox.VISIBILITY_APPLICATION, and RepositoryBlox.VISIBILITY_PRIVATE.

Usage

Bookmarks are a good way to keep track of different data views in an application. You can set a bookmark's visibility so that only users in certain access groups can see it. Bookmarks can be public, private, or belong to another defined user group.

Bookmarks can also be organized into folders and subfolders to make finding information easier. To place a bookmark in a folder, add the name of the folder to the beginning of the bookmark name, followed by the percent sign:

```
saveBookmark(RepositoryBlox.VISIBILITY_PRIVATE, "", "My Folder%My
Subfolder%My Bookmark", "Some new info");
```

This will save a private bookmark "My Bookmark" in the subfolder "My Subfolder" of the folder "My Folder". The second argument, which is blank in this example, is ignored since it is not group visible.

Folders, unlike bookmarks, do not have visibilities. When users open the bookmark dialog, all of the folders are visible. However, within each folder users can only access bookmarks that have visibilities appropriate for their user group.

Examples

The following Java code saves a group-visible bookmark called "profit analysis" for group "Team 1."

```
saveBookmark(RepositoryBlox.VISIBILITY_GROUP, "Team 1",
        "profit analysis", "Profit analysis report for Q1FY02");
```

See Also

"bookmarkFilter" on page 33, "saveBookmarkHidden()" on page 58.

saveBookmarkHidden()

Saves the current state of the Blox as a bookmark with the specified visibility (public, private, or specific group name).

Data Sources

All

Syntax

Java Method

```
saveBookmarkHidden(int visibility,
                   String owner,
                   String bookmarkName,
                   String description);
            // throws ServerBloxException
```

where:

Argument	Default	Description
bookmarkName	none	Any String representing the name of the bookmark to be restored.
description	none	Any String representing the description to be saved for the bookmark.
owner	null	The name of the bookmark group. This argument is for the Java method only. It is ignored if the visibility is not RepositoryBlox.VISIBILITY_GROUP.
visibility	none	The valid integer values are expressed as constants: RepositoryBlox.VISIBILITY_GROUP, RepositoryBlox.VISIBILITY_APPLICATION, and RepositoryBlox.VISIBILITY_PRIVATE

Usage

"Hidden" bookmarks are not visible in the user interface. They do not show up in the bookmark drop list accessible from the right-click menu or the Toolbar's bookmark button. They are accessible only through the API. For server-side Java methods to set and access hidden bookmarks, use the BookmarkBlox API. See "hidden" on page 152 in the Chapter 7, "BookmarksBlox Reference," on page 123 section.

Examples

```
saveBookmarkHidden("Hidden Bookmark", "My hidden view", "private");
```

See Also

"bookmarkFilter" on page 33

setBookmarkFilter()

For a description of this method, see "bookmarkFilter" on page 33.

setBusy()

Temporarily controls a Blox's busy state on the client.

Data Sources

All

Syntax

JavaScript Method

```
void setBusy(boolean busy);
```

where:

Argument	Default	Description
busy	none	When true, the Blox will indicate its busy state by disabling interactivity in its user interface and animating the logo in the menubar.

See Also

"isBusy()" on page 54, Chapter 5, "Client-Side API Reference," on page 63

setDataBlox()

Sets the DataBlox to use with the outer Blox (for example, ChartBlox, GridBlox, DataLayoutBlox, PageBlox, or PresentBlox).

Data Sources

All

Syntax

Java Method

```
void setDataBlox(DataBlox bloxName);
          throws ServerBloxMissingResourceException,
                 ServerBloxException
```

where:

Argument	Default	Description
`bloxName`	none	The name of the DataBlox you want to use.

See Also

"getDataBlox()" on page 53

setDataBusy()

Set the busy state on the client. This method is useful when you execute some server-side code that performs data operations.

Availability

Render Modes All

Data Sources All

Syntax

JavaScript Method

```
setDataBusy(state); // no return value
```

where:

Argument	Default	Description
`state`	`false`	A boolean argument. A value of `true` sets the state of the client-side blox to busy, thus disallowing any client-side code to execute. A value of `false` indicates that the state is not busy so client-side code can freely execute.

Usage

Set the `setDataBusy()` method on a client-side DataBlox that will be updated by the any server-side code, for example, code that is executed via the `call()` method.

See Also

"call()" on page 51

setInitialProperty()

Sets the initial property.

Data Sources
All

Syntax
Java Method

```
void setInitialProperty(String propertyName,
                        String propertyValue);
             throws InvalidBloxPropertyNameException,
                    InvalidBloxPropertyValueException,
                     ServerBloxException
```

where:

Argument	Description
`propertyName`	Name of the property to set on a Blox
`propertyValue`	The initial value of the property

setProperty()
Sets a specified property to a specified value.

Data Sources
All

Syntax
Java Method

```
void setProperty(String name, String value);
```

where:

Argument	Default	Description
`name`	none	Name of property to set.
`value`	none	Value to assign to specified property.

See Also
"getProperty()" on page 53

updateProperties()
Sends a refresh message to DB2 Alphablox to update the current Blox properties. This causes the Blox to transmit its working state (client-side) properties to the server-side Blox peer, thus insuring consistency between the server-side peer and the client-side Blox.

Data Sources
All

Syntax
JavaScript Method

```
updateProperties();    // no return value
```

See Also
"call()" on page 51, "flushProperties()" on page 52

Chapter 5. Client-Side API Reference

This chapter contains reference material for client-side APIs available in the DHTML client. For information on how to use this reference, see Chapter 1, "Using This Reference," on page 1.

Client-Side API Overview

The Client API for the DHTML client has a relatively simple set of JavaScript objects, methods and events as compared to the other clients in DB2 Alphablox. The primary intent of the DHTML client is to enable easy access to server-side application logic and APIs. A summary of the four main components in the Client API are described next. For in-depth discussions of the client-side API and examples, see the *Developer's Guide* and accompanying examples in Blox Sampler.

Blox Methods

Each Blox has an associated JavaScript Blox object in the frame. When you create a PresentBlox with an `id` of "salesPresent", a JavaScript object of the same name is available on the page. A handful of JavaScript methods—such as `call()`, `getBloxAPI()`, `getName()`, `isBusy()`, `setBusy()`, `setDataBusy()`— allow you to call a JSP page on the server, update Blox properties on the server, set the busy state of a Blox and more.

These methods are common to user interface Blox (PresentBlox, GridBlox, ChartBlox, PageBlox, and ToolbarBlox) and are described in "Client-Side APIs" on page 31 in the Chapter 4, "Common Blox Reference" chapter.

BloxAPI

Each frame has one BloxAPI object that controls all incoming and outgoing traffic between the server and the client. This object is available to your JavaScript code through the global bloxAPI object. It provides methods to poll the server, send an event to the server, call a JSP page or invoke a method on a server-side bean, or add an event listener. The methods available to the bloxAPI object are described in "BloxAPI Methods" on page 65. For a complete example, see "Example 2: Set Chart Properties on the Server Using the bloxAPI.call() Method" on page 911.

Events

To intercept a user action such as loading a bookmark or swapping axes, you should use the associated server-side event filters (see "Event Filter Objects Overview" on page 463). Alternatively, you can intercept the click event on the DHTML client before the event is sent to the server. For example, if you need to send a click event to the server, you can use the client API's (the `bloxAPI` object) `sendEvent` method:

```
bloxAPI.sendEvent( new ClientEvent( bloxname, uid, name) );
```

Sometimes it may be desirable or even necessary to intercept the events on the client-side before any server intervention. For example, if you need to cancel out a drillthrough event, using the server-side event filter's `cancelEvent()` method will only cancel the data operation, but not the pop-up window. Intercepting the event on the client-side allows you to cancel out the entire operation before it gets to the server.

Reference information on these client-side events are described in "Client-Side Events Reference" on page 72.

Custom Events

You can also create custom events that can be sent from the DHTML client to your components based on the Blox UI model on the server. This involves creating a class that extends the `ModelEvent` class in the `com.alphablox.blox.uimodel.core.event` package. See "Creating Custom Events" on page 76

Client Bean Registration using the Blox Header Tag

You can register a bean on the server for the server to remote the bean interface to the client in the form of a JavaScript object. This allows you to call any of the bean's methods from JavaScript just like any other JavaScript object. For example, if you have a bean called "myBean" on the server and it has the method:

```
String myMethod( String argument )
```

You can register the bean in the Blox header tag as follows:

```
<blox:header>
   <blox:clientBean name="myBean">
      <blox:method name="myMethod"/>
   </blox:clientBean>
</blox:header>
```

You can then call this registered method as follows:

```
function myFunction() {
   var result = myBean.myMethod( "somevalue" );
   alert(result);
}
```

See the *Developer's Guide* for in-depth discussions on the use of client beans.

DHTML Client API Cross References

This section provides cross reference tables for the following:

- "BloxAPI Methods" on page 65
- "Client-Side Events and Event Methods" on page 66
- "Blox JavaScript Object Methods" on page 65

BloxAPI Methods

The following table lists the client-side JavaScript exposed via the `bloxAPI` Object:

Methods
"addBusyHandler()" on page 66
"addErrorHandler()" on page 67
"addEventListener()" on page 67
"addResponseListener()" on page 68
"call()" on page 69
"callBean()" on page 69
"getEnablePolling()" on page 70
"getPollingInterval()" on page 71
"poll()" on page 71
"sendEvent()" on page 71
"setEnablePolling()" on page 71
"setPollingInterval()" on page 72

Blox JavaScript Object Methods

The following table lists the client-side JavaScript exposed via the Blox JavaScript object. These methods are described in the Chapter 4, "Common Blox Reference," on page 29.

JavaScript Methods
"call()" on page 51
"flushProperties()" on page 52
"getBloxAPI()" on page 52
"getName()" on page 53
"isBusy()" on page 54
"setBusy()" on page 59
"setDataBusy()" on page 60
"updateProperties()" on page 61

Client-Side Events and Event Methods

The following table lists all the client-side events available in the DHTML client:

Event
"ClickEvent" on page 73
"CaretPositionChangedEvent" on page 73
"ContentsChangedEvent" on page 73
"ClosedEvent" on page 73
"DoubleClickEvent" on page 73
"DragDropEvent" on page 73
"ExpandCollapseEvent" on page 73
"HScrollEvent" on page 73
"ResizeEvent" on page 73
"RightClickEvent" on page 73
"SelectedEvent" on page 73
"SelectionChangedEvent" on page 73
"UnselectedEvent" on page 73
"VscrollEvent" on page 73

The following table lists the methods common to all the client-side events:

Method
"getBloxName()" on page 74
"getDestinationName()" on page 74
"getDestinationUID()" on page 74
"getEventClass()" on page 75
"isReplaceDuplicate()" on page 75
"isUrgent()" on page 75
"setAttribute()" on page 75
"setReplaceDuplicate()" on page 75
"setUrgent()" on page 75

BloxAPI Reference

The following methods are global methods available on the BloxAPI object in the frame.

addBusyHandler()

Adds a busy handler for all Blox on the page.

Syntax

JavaScript Method

```
addBusyHandler(busyHandler);
```

where:

Argument	Description
busyHandler	The name of a JavaScript function.

Usage

Busy handlers are invoked whenever a Blox's busy state changes. The busy handler provided will be called with the Blox object. This allows you to supply your own custom handler. The busy handler is a JavaScript function of the form:

```
boolean busyHander( Blox blox );
```

Note: The default action will gray out the Blox when busy. Return `true` to prevent further processing of the state changes.

addErrorHandler()

Adds an error handler to the framework.

Syntax

JavaScript Method

```
addErrorHandler(eventListener);
```

where:

Argument	Description
eventListener	The name of a JavaScript function.

Usage

Error handlers are invoked when the framework receives a communication or Simple Object Access Protocol (SOAP) error response from the server. The error handler is a JavaScript function of the form:

```
boolean errorHandler( SoapResponse response )
```

where `SoapResponse` has the following method and attributes:

- `boolean SoapResponse.hasFault();`
- `String SoapResponse.faultReason;`
- `String SoapResponse.faultCode;`
- `String SoapResponse.faultSubcode;`

The error handler should return `true` if it has handled the error and should stop any further processing of the error. If the error handler returns `false`, the error is sent to any remaining error handlers in the list. In any case, after an error, the framework returns without updating Blox UI contents or busy states.

addEventListener()

Adds an event handler to the framework.

Syntax

JavaScript Method

```
addEventListener(eventListener);
```

where:

Argument	Description
eventListener	The name of a JavaScript function.

Usage

Event listeners are invoked for each event sent by either the DHTML client code or by a developer using the sendEvent() method. The event listener is a JavaScript function of the form:

```
boolean eventListener( [ClientEvent] event )
```

where `ClientEvent` is any client event as listed in "Client-Side Events Reference" on page 72. The listener should return `true` to stop all further processing of the event. If the event handler returns `false`, the client API continues to process the event by sending the event to any remaining listeners and then on to the server.

addResponseListener()

Adds a response listener for all Blox on the page.

Syntax

JavaScript Method

```
addResponseListener(responseListener);
```

where:

Argument	Description
`responseListener`	The name of a JavaScript function.

Usage

This method lets you add a JavaScript function which will be notified whenever a DHTML client-RPC returns a success response. This can be used to coordinate actions across frames or other post-event processing. The JavaScript function gets both the request and the response

Examples

The following example demonstrates how the request and response are captured:

```
<%@ taglib uri='bloxtld' prefix='blox'%>

<html>
<head>
<blox:header />
<script>
  function responseListener( request, response ) {
    var text = "REQUEST:\r\n\r\n" + request.getRequest() + "\r\n\r\n-----
\r\n\r\n";
    text += "RESPONSE: \r\n\r\n" + response.getResponse();
    responseOutput.value = text;
  }
  bloxAPI.addResponseListener( responseListener );
</script>
</head>
...
<body>
<blox:present id="present"
   ...>
</blox:present>
...
<textarea id=responseOutput rows=100 cols=200>
...
</body>
</html>
```

call()

Makes an HTTP request to the supplied URL and returns the results of the request as a String.

Syntax

JavaScript Method

```
call(url);
```

where:

Argument	Description
`url`	A String containing a URL of a file (typically a JSP file) to be run on the server.

Usage

This method also polls the server for changes immediately after the HTTP request and before the method returns the results.

callBean()

Invokes the specified method on the indicated server-side bean.

Syntax

JavaScript Method

```
callBean(beanName, methodName);
callBean(beanName, methodName, argumentArray, argumentTypeArray);
```

where:

Argument	Description
`beanName`	The name of bean. The bean must have been previously registered with the HttpSession on the server.
`methodName`	The name of a bean method.
`argumentArray`	An array of all arguments to be passed to the bean method
`argumentTypeArray`	Optional. An array of the data type of the argument(s). This helps match the arguments to the proper method on the client-side bean. See the table below for the data types supported.

Usage

The return value will be converted to an appropriate JavaScript data type. If the bean method throws a Java Exception, the return value will be a JavaScript Exception object. See the *Developer's Guide* for more on the JavaScript Exception object.

Supported method argument types are case-sensitive and are limited to the following primitive data type:

Java Data Type	Valid `argumentTypes` Value
String	`string` or `unspecified`
Integer	`integer` or `int`

Java Data Type	Valid argumentTypes Value
Boolean	boolean
Long	long
Double	double
Float	float
Byte	byte
Array	JavaScript array

If an argument is not typed and the server-side bean does not have a method signature matching the argument listed, the server will look for methods that take Java objects other than primitive types. In this case, the server will try to create the required object using a String-based constructor for the object.

Examples

The following example calls the myMethod method on a bean called myBean on the server with two arguments: a string and an integer.

```
var results = bloxAPI.callBean('myBean', 'myMethod', new Array('arg1',
'2'), newArray('string', 'int');
```

If you have a server bean myBean with a single method as follows:

```
String beanMethod( MyObject object ) // myBean's method signature
```

If you call the above bean method from the client using the following statement without specifying the argument's data type:

```
bloxAPI.callBean( "myBean", "beanMethod", "foobar" );
```

The server will invoke the method as follows:

```
myBean.beanMethod( new MyObject("foobar"));
```

Notice the creation of a MyObject object uses a String-based constructor. If the MyObject object's String-based constructor can make sense of the supplied value from the client, the method will be invoked and the results returned to the client.

Important: It is recommended that you limit client-side code to calling only methods that take and return primitive data types rather than Java objects.

getEnablePolling()

Returns the polling enabled setting.

Syntax

JavaScript Method

```
getEnablePolling();
```

Usage

If true, the automatic polling mechanism is enabled and the framework underlying the DHTML client does a very slow poll of the server designed to pick up, in very rare occasions, any asynchronous changes to the Blox in the frame.

See Also

"setEnablePolling()" on page 71

getPollingInterval()

Returns the polling interval for non-busy polling in milliseconds.

Syntax

JavaScript Method

```
getPollingInterval();
```

Usage

This is the normal polling interval that checks the server for asynchronous updates. The polling mechanism uses a different interval when the server informs the client that it is busy.

See Also

"setPollingInterval()" on page 72

poll()

Immediately polls the server for changes to any of the Blox in the frame.

Syntax

JavaScript Method

```
poll();
```

Usage

The server will respond with any pending changes as well as the busy state for each of the Blox. Normally you do not need to explicitly poll the server since the framework handles this automatically. This is only needed if you modify the state of the server. In this case, polling may be needed in order to pick up those changes in a timely manner. Typically this means circumventing the DHTML client in some way such as using multiple frames.

sendEvent()

Immediately sends the named event to all registered event handlers and then ultimately to the server.

Syntax

JavaScript Method

```
sendEvent(ClientEvent event);
```

where:

Argument	Description
`ClientEvent`	One of the JavaScript event objects: `ClickEvent`, `ContentsChangedEvent`, `ClosedEvent`, `DoubleClickEvent`, `DragDropEvent`, `RightClickEvent`, `ScrollEvent`, `SelectedEvent`, `SelectionChangedEvent`, `UnselectedEvent`.
`event`	The name of the event.

Usage

If the client successfully sends the event to the server, the function returns `true`. In all other cases, the function returns `false`.

setEnablePolling()

Controls automatic server polling.

Syntax

JavaScript Method

```
setEnablePolling(enable);
```

where:

Argument	Description
`enable`	Specify `true` to enable the automatic polling mechanism; `false` to disable it.

Usage

When set to `false`, the framework underlying the client will not automatically poll the server. The automatic polling mechanism is a very slow poll of the server designed to pick up, in very rare occasions, any asynchronous changes to the Blox in the frame. By default, polling is enabled.

See Also

"getEnablePolling()" on page 70

setPollingInterval()

Sets the polling interval for non-busy polling in milliseconds.

Syntax

JavaScript Method

```
setPollingInterval(intervalMS);
```

where:

Argument	Description
`intervalMS`	The non-busy polling interval in milliseconds.

Usage

This is the normal polling interval that checks the server for asynchronous updates. The polling mechanism uses a different interval when the server informs the client that it is busy.

See Also

"getPollingInterval()" on page 71

Client-Side Events Reference

The Blox UI model exposes a set of events as JavaScript objects. As a result, you can use JavaScript to create event objects, send the events to the server, or intercept these events. Each event has a class name that matches the name of the class on the server used to dispatch the event. The class name is always available in the `EVENT_CLASS` static attribute on each event. For example, for ClickEvent, the class name is `ClickEvent.EVENT_CLASS`. For SelectionChangedEvent, the class name is `SelectionChangedEvent.EVENT_CLASS`. All the events have a common set of methods that let you identify the destination Blox name or component UID, specify whether an event should be dispatched immediately, and more.

This section provides information on the following:

• "Creating Custom Events" on page 76

Client-Side Events and Syntax

The following table lists the client-side event objects available in the DHTML client. Note that the argument types are provided to help describe the arguments. In JavaScript, all arguments are variables.

Event	Syntax
ClickEvent	ClickEvent(String *bloxName*, int *uid*, String *name* [, Boolean *checked*]);
CaretPositionChangedEvent	CaretPositionChangedEvent(String *bloxName*, int *uid*, String *name*, int *caretPosition*, String *textSelection*);
ContentsChangedEvent	ContentsChangedEvent(String *bloxName*, int *uid*, String *name*, *newContents*);
ClosedEvent	ClosedEvent(String *bloxName*, int *uid*, String *name*);
DoubleClickEvent	DoubleClickEvent(String *bloxName*, int *uid*, String *name*);
DragDropEvent	DragDropEvent(String *bloxName*, int *uid*, *operation*, *droppedComponent* [, int *positionAfterComponentUID*]);
ExpandCollapseEvent	ExpandCollapseEvent(String *bloxName*, int *uid*, String *name*, boolean *expanded*);
HScrollEvent	HScrollEvent(String *bloxName*, int *uid*, String *name*, int *newHorizontalPosition*);
ResizeEvent	ResizeEvent(String *bloxName*, int *uid*, String *name*, int *newWidth*);
RightClickEvent	RightClickEvent(String *bloxName*, int *uid*, String *name*);
SelectedEvent	SelectedEvent(String *bloxName*, int *uid*, String *name*);
SelectionChangedEvent	SelectionChangedEvent(String *bloxName*, int *uid*, String *name*, Array *integerSelections*);
UnselectedEvent	UnselectedEvent(String *bloxName*, int *uid*, String *name*);
VScrollEvent	VScrollEvent(String *bloxName*, int *uid*, String *name*, int *newVerticalPosition*);

The arguments are described below:

Arguments

`bloxName`	The name of the Blox the event should be sent to.
`uid`	The unique id associated with the component on which the event will be called.
`name`	Optional. The component name which is the actual text-based name of the component generating the event. In the cases where `name` is not needed and yet it is not the last argument, it should be set to null.
`caretPosition`	Location of the text cursor.
`checked`	Optional. The current state of the checkbox on the client (for CheckBox components).
`droppedComponent`	The `uid` of the component that is being dropped.
`expanded`	`true` if expanded.

integerSelections	Either an array of integers containing a list of uid of the selections or an array of integers containing the indices for multiple selection lists.
newContents	The changed content. This is usually a string such as that entered in an Edit component (an text edit box).
newHeight	The new height in pixels.
newHorizontalPosition	A 0-based horizontal scroll index. The scroll unit is defined by the component using the event. For example, for the grid component, this would be the column number.
newVerticalPosition	A 0-based vertical scroll index. The scroll unit is defined by the component using the event. For example, for the grid component, this would be the row number.
newWidth	The new width in pixels.
operation	The operation performed. Currently, move is only valid operation.
positionAfterComponentUID	Optional. The UID of the component the target component should be dropped after.
textSelection	The current selected text that is marked with the mouse.

Common Event Methods

Each event exposes the following methods:

getBloxName()

Returns the destination Blox name.

Syntax:

```
String getBloxName();
```

getDestinationName()

Returns the destination component's name.

Syntax:

```
String getDestinationName();
```

Usage: You should use getDestinationUID() whenever possible as this method may return null for some components. Use this only in cases when getDestinationUID() does not work for you (for example, when you need to identify whether the destination component is your custom component).

getDestinationUID()

Returns the destination component's UID.

Syntax:

```
int getDestinationUID();
```

getEventClass()
Returns the server-side Java class name associated with the event.

Syntax:

```
String getEventClass();
```

isReplaceDuplicate()

Returns `true` if this event will replace duplicate events in the client-side event queue.

Syntax:

```
boolean isReplaceDuplicate();
```

isUrgent()

Returns `true` if this is an urgent event to be sent immediately to the server

Syntax:

```
boolean isUrgent();
```

setAttribute()
Sets the value of the attribute name in the event.

Syntax:

```
void setAttribute( String name, String value [, String type] );
```

Usage: The attribute value is passed along with the event to the server. The type of the attribute can be specified using the optional type argument. For supported data type, see "Support Data Type" on page 69.

setReplaceDuplicate()
Specifies whether an event of the same event class, destination UID and destination Blox name in the client-side event queue should be replaced.

Syntax:

```
void setReplaceDuplicate( boolean replaceDuplicate);
```

Usage: When `true`, an event of the same event class, destination UID and destination Blox name residing in the client-side event queue (that is, non-urgent events) will get replaced. The process removes the existing duplicate event from the queue and adds the replacement to the end of the queue.

setUrgent()
Specifies to sent urgent events to the server immediately.

Syntax:

```
void setUrgent( boolean isUrgent );
```

Usage: Urgent events are sent to the server immediately with no waiting. Non-urgent events are sent when convenient, such as during a poll or other server communications including the sending of an urgent event.

Generating an Event

To generate an event:

```
var myClickEvent = new ClickEvent (bloxName, uid [, componentName] };
```

Then to send an event:

```
bloxAPI.sendEvent( myClickEvent );
```

To intercept an event:

```
bloxAPI.addEventListener(eventHandler);
```

and the eventHandler JavaScript function may look like the following:

```
<script>
function eventHandler(event) {
  alert( "At handler for event " + event.getEventClass() +
  " on component UID " + event. getDestinationUID() );
  return false;
}
</script>
```

Returning `false` from the handler will allow the event to be processed and sent to the server. Returning `true` will stop all further processing of the event.

The following JavaScript example demonstrates how to make sure that events occurring on a custom UI component (in this example, the component's name is ″Show″) get dispatched immediately:

```
function eventListener( event )
{
  if ( event.getDestinationName() == "Show" )
  {
    // Make this event not dispatch immediately
    event.setUrgent( false );

    // Set busy on the blox
    myShowContainer.setBusy( true );
    myGrid.setBusy( true );

    // After a bit of time, make sure the event is sent out
    setTimeout( "bloxAPI.flushEvents();", 0 );
  }
  return false;
}

bloxAPI.addEventListener( eventListener );
```

For more details and examples see the DHTML Client API chapter in the *Developer's Guide*.

Creating Custom Events

You can create you own custom events that work like the supplied events. Custom events should use the existing client-side event infrastructure if these events are to be sent from the client. The recipe for generating client-side custom events is as follows:

1. Define a JavaScript function with the name of the event.
2. Add additional event parameters to the constructor.
3. Set the class name of the event on the function by setting the `EVENT_CLASS` attribute.
4. Call the `_modelEventConstructor()` in the function.
5. Set any additional parameters on the event using the `setAttribute()` method.

```
function MyEvent( bloxName, uid, name, myvalue )
{
   // Begin constructor
   _modelEventConstructor( this, MyEvent.EVENT_CLASS, bloxName, uid, name );
```

```
   this.setAttribute( "MyEvent.myValue", myvalue, "int" );
   // End constructor
}
MyEvent.EVENT_CLASS = "my.package.MyEvent";
```

Assuming that the event my.package.MyEvent has been defined on the server:

```
package my.package;
import com.alphablox.blox.uimodel.core.event.ModelEvent;
Public class MyEvent extends ModelEvent
{
  ...
}
```

Chapter 6. AdminBlox Reference

This chapter contains reference material for AdminBlox. For general reference information about Blox, see Chapter 3, "General Blox Reference Information," on page 17. For information on how to use this reference, see Chapter 1, "Using This Reference," on page 1.

AdminBlox Overview

AdminBlox provides programmatic access to information on the server, users, groups, roles, data sources, the Alphablox log system, and applications set through the Administration tab in the DB2 Alphablox Home Page.

The Administration tab under the DB2 Alphablox Home Page provides a means for server administrators to specify properties such as server log file name, message level, telnet console port, clustering options, and telnet username and password. The Data Sources, Users, Groups, Roles, and Applications links under the Administration tab allow administrators to define data sources, users, groups, roles, and applications to be served by DB2 Alphablox. Once specified, this information is stored in the repository. Application developers can access this information via AdminBlox and its related objects and methods.

	Method		Object Returned
AdminBlox —>	`getApplication(...)` `getApplications()`	—>	Application
	`getDataSource(...)` `getDataSources()`	—>	DataSource
	`getGroup(...)` `getGroups()`	—>	Group
	`getLog(...)`	—>	Log
	`getRole(...)` `getRoles()`	—>	Role
	`getServer(...)`	—>	Server

Method		Object Returned
`getUser(...)` `getUsers()`	—>	User

While AdminBlox and RepositoryBlox both let you access information stored in the repository, AdminBlox is specific to the general, administrative details on the server, applications, users, and data sources. For example, it allows you to get information regarding all data sources available to the server, find out the session timeout setting of an application, or identify the specified SMTP server. RepositoryBlox, on the other hand, gives you access to information on the current user, application, and custom properties.

With AdminBlox, you can build your own administration application to serve your specific server monitoring and management needs. Because of this power, care should be taken at the application level to ensure the appropriate access control is in place. There is no built-in security for AdminBlox.

The Application Object

The Application object represents an Alphablox application in the repository. It provides methods for you to get information specified through the Application link under the Administration tab. With the getter methods provided, you can find out information such as an application's default saved state, display name, header links defined, and the amount of inactive time for the session to time out.

The DataSource Object

The DataSource object represents an Alphablox data source in the repository. It provides methods for you to get information specified through the Data Source link under the Administration tab. With the getter methods provided, you can find out information such as the data source's adapter type, application/catalog/database/schema name, default username and password to log in to the data source, and maximum columns and rows to return.

The User Object

The User object represents an Alphablox user in the repository. It provides methods for you to access information specified through the Users link under the Administration tab. With the getter and setter methods provided, you can find out and modify information such as the user's name, email address, and primary group association.

The Group Object

The Group object represents an Alphablox group in the repository. It provides methods for you to access information specified through the Groups link under the Administration tab. With the getter and setter methods provided, you can find out and modify information such as which users or subgroups belong to a specified group.

The Role Object

The Role object represents an Alphablox role in the repository. It provides methods for you to access information specified through the Roles link under the Administration tab. With the methods provided, you can find out and modify information such as which users or groups belong to a specified role.

The Log Object

The Log object is used to place messages into the Alphablox log system. The message levels are, in order of severity from minor to the most severe: DEBUG, VERBOSE, INFO, SYSTEM, WARNING, and ERROR. These messages will be logged to the log file and registered consoles depending on their message level setting. The log file is located inside the DB2 Alphablox repository under the `<alphablox_dir>/repository/logs/<instance_name>/logs`.

The Server Object

The Server object represents server-related information that DB2 Alphablox stored in the repository. It provides methods for you to get information specified through the Server link under the Administration tab.

AdminBlox Example

This example demonstrates how to log messages to the Alphablox log system through AdminBlox. This is particularly useful in monitoring, logging, and debugging problems. It gives you the ability to log both messages and Exceptions.

Note the logging mechanism is multi-threaded so that you may get messages slightly out of the order from what you expect.

```
<%@ taglib uri="bloxtld" prefix="blox" %>
<%@ page import="com.alphablox.blox.repository.Log" %>
<html>
<head>
   <blox:header />
</head>

<body>
<blox:admin id="myAdminBlox" />
<%
   Log log = myAdminBlox.getLog();
   log.sendMessage( Log.MESSAGE_LEVEL_INFO, "My Info Message Title",
"My Info Message" );
   Exception e = new Exception( "My dummy Exception" );
   log.sendException( Log.MESSAGE_LEVEL_INFO, "My Info Exception
Title", e);

%>
The Log test is done.
</body>
</html>
```

1. Import the com.alphablox.blox.repository.Log class.
2. Add an AdminBlox using the `<blox:admin>` tag.
3. Access the Log object via AdminBlox's `getLog()` method.
4. Send a message to the log using `sendMessage(...)`.
5. Send an Exception to the log using `sendException(...)`.

This would generate the following entries in the log file:

7/28/04 1:29:52 PM [INFO] My Info Message Title: My Info Message 7/28/04 1:29:52 PM [INFO] My Info Exception Title: My dummy Exception

7/28/04 1:29:52 PM [INFO] My Info Message Title: My Info Message 7/28/04 1:29:52 PM [INFO] My Info Exception Title: My dummy Exception

```
java.lang.Exception: My dummy Exception
at org.apache.jsp._log4._jspService(_log4.java:126)
at com.ibm.ws.webcontainer.jsp.runtime.HttpJspBase.service(HttpJspBase.java:89)
at javax.servlet.http.HttpServlet.service(HttpServlet.java:853)
at
com.ibm.ws.webcontainer.jsp.servlet.JspServlet$JspServletWrapper.service(JspServlet.ja
va:344)
at com.ibm.ws.webcontainer.jsp.servlet.JspServlet.serviceJspFile(JspServlet.java:662)
at com.ibm.ws.webcontainer.jsp.servlet.JspServlet.service(JspServlet.java:760)
at javax.servlet.http.HttpServlet.service(HttpServlet.java:853)
at
com.ibm.ws.webcontainer.servlet.StrictServletInstance.doService(StrictServletInstance.ja
va:110)
at
com.ibm.ws.webcontainer.servlet.StrictLifecycleServlet._service(StrictLifecycleServlet.ja
va:174)
[ more stack traces below omitted... ]
```

AdminBlox JSP Custom Tag Syntax

The Alphablox Tag Libraries provides custom tags to use in a JSP page for creating each blox. This section describes how to create the custom tag to create an AdminBlox. For a copy and paste version of the tag with all the attributes, see "AdminBlox JSP Custom Tag" on page 881.

Syntax

```
<blox:admin
    [attribute="value"] >
</blox:admin>
```

where:

attribute is one of the attributes listed in the attribute table.

value is a valid value for the attribute.

and where the attributes are one of the following:

Attribute
id
bloxName

Usage

Each custom tag can have one or more attributes, separated by one or more space or new line characters. Any extra space or new line characters are ignored. For readability, attributes can each go on a separate line with the same indenting.

You can substitute the closing `</blox:admin>` tag using the shorthand notation, closing the tag at the end of the attribute list that looks as follows:

```
id="myAdminBlox" />
```

Examples

```
<blox:admin
     id="namedAdminBlox" />
```

Methods Cross References

This section lists all unique properties and methods for BookmarksBlox and its associated objects:

AdminBlox Methods Cross References

The following table lists all methods for AdminBlox:

Methods	Methods
createUser()	getGroups()
getApplication()	getLog()
getApplicationNames()	getRole()
getApplications()	getRoleNames()
getDataSource()	getRoles()
getDataSourceNames()	getServer()
getDataSources()	getUser()
getGroup()	getUserNames()
getGroupNames()	getUsers()

Application Object Methods Cross References

Methods	Methods
getContextName()	getSessionTimeout()
getDefaultSavedState()	getType()
getDescription()	getURI()
getDisplayName()	getURL()
getDocBase()	getWriteRole()
getEntApp()	isAutosave()
getHeaderLinks()	isRestoreSavedState()
getImageURL()	refresh()
getPrimaryName()	

DataSource Object Methods Cross References

Methods	Methods
getAdapterName()	getMaxRows()
getAdapterType()	getName()

Methods	Methods
getAliasTable()	getProvider()
getApplication()	getSchema()
getCatalog()	getServer()
getDatabase()	getUserName()
getDescription()	isAASUserAuthorizationEnabled()
getAdapterType()	getName()
getAliasTable()	getProvider()
getApplication()	getSchema()
getCatalog()	getServer()
getDatabase()	getUserName()
getDescription()	isAASUserAuthorizationEnabled()
getMaxColumns()	isMDB()
getMaxColumns()	isMDB()

Group Object Methods Cross References

Methods	Methods
addGroup()	isUserInGroup()
addUser()	removeGroup()
getName()	save()
getDescription()	removeUser()
isGroupInGroup()	

Log Object Methods Cross References

Methods	Methods
`getMinimumServerMessageLevel()`	`sendMessage()`
`sendException()`	

Role Object Methods Cross References

Methods	Methods
`addGroup()`	`isUserInRole()`
`addUser()`	`removeGroup()`
`getDescription()`	`removeUser()`
`getName()`	`save()`
`isGroupInRole()`	

Server Object Methods Cross References

Methods	Methods
getApplicationServerType()	getRepositoryDatabasePort()

Methods	Methods
getAuthorizedClientList()	getRepositoryDatabaseUser()
getClusteringLeadIpAddress()	getRepositoryFileDirectory()
getClusteringLeadPort()	getRepositoryServiceProvider()
getClusteringMaxHosts()	getServerBuildVersion()
getClusteringStartupWait()	getServerIdleDuration()
getCommandFileName()	getServerIncrementVersion()
getDefaultMessageLevel()	getServerLogFileName()
getHtmlClientTheme()	getServerMajorVersion()
getInstanceName()	getServerMinorVersion()
getMaxCubes()	getServerVersion()
getMessageHistorySize()	getSmtpServer()
getNewLogEndMessageLevel()	getTelnetConsoleName()
getNewLogStartMessageLevel()	getTelnetConsolePort()
getPoweredBy()	getTelnetTimeout()
getRepositoryDatabaseAdapter()	isAuthenticationEnabled()
getRepositoryDatabaeDriver()	isAutoCreateUsers()
getRepositoryDatabaseHostName()	isClusteringEnabled()
getRepositoryDatabaseIsolationLevel()	isMaxCubesEnabled()
getRepositoryDatabaseAdapter()	isSaveOnExit()
getRepositoryDatabaeDriver()	isServerLogEnabled()
getRepositoryDatabaseHostName()	levelIntToString()
getRepositoryDatabaseIsolationLevel()	levelStringToInt()
getRepositoryDatabaseName()	

User Object Methods Cross References

Methods	Methods
delete()	save()
getDescription()	setCanEdit()
getEmail()	setDescription()
getGroupNames()	setEmail()
getName()	setFullName()
getPrimaryGroupName()	setPassword()
isCanEdit()	setPrimaryGroupName()

AdminBlox Methods

This section describes all methods for AdminBlox.

createUser()

Creates a User object for use in creating a user.

Data Sources
All

Syntax
Java Method

```
User createUser(String userName, String password);
        // throws ServerBloxException
```

where:

Argument	Description
`userName`	The user name to associate with the new user.
`password`	The password to associate with the new user.

Usage
Use this method to get a User object when creating a user. Then call `User.save()` to save the user to the repository. If the user already exists, the method will throw a ServerBloxException.

See Also
"User Object Methods" on page 118, "save()" on page 119

getApplication()
Returns an Application object for the given application name.

Data Sources
All

Syntax
Java Method

```
Application getApplication(String applicationName);
                // throws ServerBloxException
```

where:

Argument	Description
`applicationName`	The name of the application to get.

See Also
"Application Object Methods" on page 91

getApplicationNames()
Gets the names of all application as a String array.

Data Sources
All

Syntax
Java Method

```
String[] getApplicationNames();
     // throws ServerBloxNotFoundException
```

Usage
This method works faster than `getApplications()` or `Application.getContextName()`.

See Also
"getApplications()" on page 87, "getContextName()" on page 91

getApplications()
Gets a list of all applications as Application objects.

Data Sources
All

Syntax
Java Method

```
Application[] getApplications();
     // throws ServerBloxNotFoundException
```

Usage
The returned array contains one Application object for each application defined to DB2 Alphablox.

See Also
"getApplicationNames()" on page 86, "Application Object Methods" on page 91.

getDataSource()
Returns a DataSource object for the given data source.

Data Sources
All

Syntax
Java Method

```
DataSource getDataSource(String dataSourceName);
    // throws ServerBloxNotFoundException
```

where:

Argument	Description
dataSourceName	The name of the data source to get.

See Also
"DataSource Object Methods" on page 96

getDataSourceNames()
Gets a list of data source names.

Data Sources
All

Syntax
Java Method

```
String[] getDataSourceNames();
    // throws ServerBloxNotFoundException
```

Usage
This method works faster than `getDataSources()` or `DataSource.getName()`.

getDataSources()

Gets a list of all data sources as DataSource objects.

Data Sources

All

Syntax

Java Method

```
DataSources[] getDataSources();
    // throws ServerBloxNotFoundException
```

Usage

The returned array contains a DataSource object for each data source defined to DB2 Alphablox.

See Also

"DataSource Object Methods" on page 96

getGroup()

Returns a Group object for the given group name.

Data Sources

All

Syntax

Java Method

```
Group getGroup(String groupName); // throws ServerBloxNotFoundException
```

where:

Argument	Description
`groupName`	The name of the group to get. Note that group names are converted to all lowercase letters in the repository.

See Also

"Group Object Methods" on page 99

getGroupNames()

Gets a list of group names.

Data Sources

All

Syntax

Java Method

```
String[] getGroupNames(); // throws ServerBloxNotFoundException
```

Usage

This method works faster than `getGroup()` or `Group.getName()`.

See Also

"Group Object Methods" on page 99, "getGroup()" on page 88

getGroups()

Gets a list of all groups as Group objects.

Data Sources

All

Syntax

Java Method

```
Group[] getGroups(); // throws ServerBloxNotFoundException
```

Usage

The returned array contains a Group object for each group defined to DB2 Alphablox.

See Also

"Group Object Methods" on page 99

getLog()

Gets the Log object to write messages and exceptions to the Alphablox log.

Data Sources

All

Syntax

Java Method

```
Log getLog();
```

See Also

"Log Object Methods" on page 102

getRole()

Gets a Role object.

Data Sources

All

Syntax

Java Method

```
Role getRole(String roleName); // throws ServerBloxNotFoundException,
                                  //        ServerBloxNotSupportedException
```

where:

Argument	Description
roleName	The name of the role to get.

See Also

"Role Object Methods" on page 103, "getRoleNames()" on page 89

getRoleNames()

Gets a list of role names.

Data Sources

All

Syntax
Java Method

```
String[] getRoleNames(); // throws ServerBloxNotFoundException
```

Usage
This method works faster than `getRole()` or `Role.getName()`. This is for application servers other than Tomcat.

See Also
"getRole()" on page 89

getRoles()
Gets all the roles as a list of Role objects.

Data Sources
All

Syntax
Java Method

```
Role[] getRoles();// throws ServerBloxNotFoundException,
                  //         ServerBloxNotSupportedException
```

Usage
The returned array contains a Role object for each role defined to DB2 Alphablox.

See Also
"Role Object Methods" on page 103

getServer()
Gets the Server object.

Data Sources
All

Syntax
Java Method

```
Server getServer();
```

Usage
The returned Server object can be used to access various server information.

See Also
"Server Object Methods" on page 106

getUser()
Gets a User object for the given username.

Data Sources
All

Syntax
Java Method

```
User getUser(String); // throws ServerBloxNotFoundException
```

See Also
"User Object Methods" on page 118

getUserNames()

Gets a list of user names.

Data Sources
All

Syntax
Java Method

```
String[] getUserNames(); // throws ServerBloxNotFoundException
```

Usage
This method works faster than `getUser()` or `User.getName()`.

See Also
"getUser()" on page 90

getUsers()

Gets all users as an array of User objects.

Data Sources
All

Syntax
Java Method

```
Users[] getUsers(); // throws ServerBloxNotFoundException
```

Usage
The returned array contains a User object for each user known to DB2 Alphablox.

See Also
"User Object Methods" on page 118

Application Object Methods

This section describes methods associated with the Application object. To access this object from AdminBlox, use the `AdminBlox.getApplication(...)` or `AdminBlox.getApplications()` methods. To use any method for this object, import the `com.alphablox.blox.repository` package in your JSP.

getContextName()

Gets the context name for this application.

Data Sources
All

Syntax
Java Method

```
String getContextName();
```

getDefaultSavedState()

Gets the saved state to use as the default saved state for this application.

Data Sources
All

Syntax
Java Method

```
String getDefaultSavedState();
```

Usage
The default saved state is specified and created by the application developers if they desire their users to access a saved state version of the application. If a default saved state is not specified under the Application link in the Administration tab, the restored state is the state that the application was in upon browser shutdown or timeout, and the method returns null.

getDescription()
Gets the description for this application.

Data Sources
All

Syntax
Java Method

```
String getDescription();
```

getDisplayName()
Gets the full, display name of the application.

Data Sources
All

Syntax
Java Method

```
String getDisplayName();
```

getDocBase()
Gets the document base for this application.

Data Sources
All

Syntax
Java Method

```
String getDocBase();
```

Usage
Returns the document base path on the disk (for example, on a Windows® system, `D:\Alphablox\webapps\SalesApp`).

getEntApp()
Gets the enterprise application setting for this application.

Data Sources
All

Syntax
Java Method

```
String getEntApp();
```

Usage
This method applies only to enterprise applications.

getHeaderLinks()
Gets the header links specified for this application.

Data Sources
All

Syntax
Java Method

```
String getHeaderLinks();
```

Usage
Header links are the tags displayed on the member names to pop up a link to a URL. The tags use an information icon as a link to the URLs specified. Each link specified in the Header Links text box in the Applications definition page under the Administration tab has the form `<member name> = URL` with a line break between each link. The string returned from this method also includes the line breaks.

getImageURL()
Gets the image URL for this application

Data Sources
All

Syntax
Java Method

```
String getImageURL();
```

Usage
Each application can have an image displayed next to the application name on the Applications page. This is an optional entry. This method returns the exact text string specified in the Image URL text box in the Applications definition page under the Administration tab. If no URL is specified, the method returns null.

getPrimaryName()
Gets the primary name (the application context name) for the application.

Data Sources
All

Syntax
Java Method

```
String getPrimaryName();
```

See Also
"getDisplayName()" on page 92

getSessionTimeout()

Ges the amount of inactive time, in minutes, for any session to timeout.

Data Sources
All

Syntax
Java Method

```
int getSessionTimeout();
```

getType()

Gets the type of application.

Data Sources
All

Syntax
Java Method

```
String getType();
```

Usage
Returns "internal" for applications, examples, or tools that come with DB2 Alphablox in most cases. Returns "external" for custom applications.

getURI()

Gets the URI of the application.

Data Sources
All

Syntax
Java Method

```
String getURI();
```

Usage
Applies to enterprise applications only. For applications running under WebSphere® or WebLogic, this is the location of the application relative to the location of the enterprise application. Returns null for Tomcat.

See Also
"getEntApp()" on page 92

getURL()

Gets the URL of this application.

Data Sources
All

Syntax
Java Method

```
String getURL();
```

Usage
Returns the exact string specified in the Home URL text box in the Applications definition page under the Administration tab.

See Also
"getURI()" on page 94

getWriteRole()
The Role the application has write privilege in.

Data Sources
All

Syntax
Java Method

```
String getWriteRole();
```

isAutosave()
Identifies if the application instance should save its state when its session times out

Data Sources
All

Syntax
Java Method

```
boolean isAutosave();
```

Usage
Returns `true` if autosave is enabled.

isRestoreSavedState()
Identifies if the saved state should be restored when the application is loaded.

Data Sources
All

Syntax
Java Method

```
boolean isRestoreSavedState();
```

Usage
Returns `true` if the Restore Saved Application State option is set to yes in the Applications definition page under the Administration tab.

refresh()
Refreshes a Tomcat application.

Data Sources
All

Syntax
Java Method

```
void refresh();    // throws ServerBloxException
```

Usage
This method can be used to have the server reread the Blox initialization parameters, for example.

DataSource Object Methods

This section describes methods associated with the DataSource object. To access this object from AdminBlox, use the `AdminBlox.getDataSource(...)` or `AdminBlox.getDataSources()` methods. To use any method for this object, import the `com.alphablox.blox.repository` package in your JSP.

getAdapterName()

Gets the adapter name for this data source.

Data Sources
All

Syntax
Java Method

```
String getAdapterName();
```

Usage
Returned string is identical to the adapter name as it appears in the Adapter drop list in the Data Sources definition page under the Administration tab.

getAdapterType()

Gets the adapter type for this data source.

Data Sources
All

Syntax
Java Method

```
int getAdapterType();
```

Usage
Returns 0 if the data source is multidimensional; 1 if relational. You should evaluate the returned integer with the constants: `TYPE_MDB` and `TYPE_RDB`. This helps avoid problems if the integer values should change.

getAliasTable()

Gets the alias table for this data source.

Data Sources
IBM DB2 OLAP Server, Hyperion Essbase

Syntax
Java Method

```
String getAliasTable();
```

Usage
Returns the name of the alias table.

getApplication()

Gets the application (catalog) for this data source.

Data Sources

Multidimensional

Syntax

Java Method

```
String getApplication();
```

getCatalog()

Gets the catalog (application) for this data source.

Data Sources

All

Syntax

Java Method

```
String getCatalog();
```

getDatabase()

Gets the database (schema) for this data source.

Data Sources

All

Syntax

Java Method

```
String getDatabase();
```

getDescription()

Gets the description (if any) for this data source.

Data Sources

All

Syntax

Java Method

```
String getDescription();
```

getMaxColumns()

Gets the maximum columns setting for this data source.

Data Sources

All

Syntax

Java Method

```
int getMaxColumns();
```

getMaxRows()

Gets the maximum rows setting for this data source.

Data Sources
All

Syntax
Java Method

```
int getMaxRows();
```

getName()
Gets the name of the data source.

Data Sources
All

Syntax
Java Method

```
String getName();
```

getProvider()
Gets the provider for this data source

Data Sources
All

Syntax
Java Method

```
String getProvider();
```

Usage
Applies to Microsoft Analysis Services data sources only.

getSchema()
Gets the schema (database) for this data source.

Data Sources
All

Syntax
Java Method

```
String getSchema();
```

getServer()
Gets the server name for this data source.

Data Sources
All

Syntax
Java Method

```
String getServer();
```

getUserName()
Gets the default user name used to log in for this data source.

Data Sources
All

Syntax
Java Method

```
String getUserName();
```

isAASUserAuthorizationEnabled()
Gets the DB2 Alphablox user authentication setting for this data source.

Data Sources
All

Syntax
Java Method

```
boolean isAASUserAuthorizationEnabled();
```

Usage
If this method returns `true`, DB2 Alphablox will use the user's Alphablox username and password to log into the database.

isMDB()
Identifies if the data source is multidimensional.

Data Sources
All

Syntax
Java Method

```
boolean isMDB();
```

Usage
Returns `true` if the data source is multidimensional.

Group Object Methods
This section describes methods associated with the Group object. To access this object from AdminBlox, use the `AdminBlox.getGroup(...)` or `AdminBlox.getGroups()` method. To use any method for this object, import the `com.alphablox.blox.repository` package in your JSP.

addGroup()
Adds a subgroup to this group.

Data Sources
All

Syntax
Java Method

```
void addGroup(Group group);
void addGroup(String groupName);
```

where:

Argument	Description

`group`	A Group object.
`groupName`	The name of a group. This name is converted to all lowercase letters in the repository.

addUser()

Adds a user to this group.

Data Sources

All

Syntax

Java Method

```
void addUser(User user);
void addGroup(String userName);
```

where:

Argument	Description
`user`	A User object.
`userName`	A username.

Usage

The user will not be added until save() is called.

See Also

"save()" on page 102.

getDescription()

Gets the description of this group.

Data Sources

All

Syntax

Java Method

```
String getDescription();
```

getName()

Gets the name of this group.

Data Sources

All

Syntax

Java Method

```
String getName();
```

isGroupInGroup()

Identifies if the specified group is a subgroup in this group.

Data Sources

All

Syntax

Java Method

```
boolean isGroupInGroup(Group group);
boolean isGroupInGroup(String groupName);
                // throws ServerBloxException
```

where:

Argument	Description
group	A Group object.
groupName	The name of a group. Group names are stored in the repository in all lowercase letters.

isUserInGroup()

Identifies if the specified user is in this group.

Data Sources

All

Syntax

Java Method

```
boolean isUserInGroup(User user);
boolean isUserInGroup(String userName);
                // throws ServerBloxException
```

where:

Argument	Description
user	A User object.
userName	A username.

removeGroup()

Removes a subgroup from this group.

Data Sources

All

Syntax

Java Method

```
void removeGroup(Group group);
void removeGroup(String groupName);
                // throws ServerBloxException
```

where:

Argument	Description
group	A Group object.
groupName	The name of a group. Note that group names are stored in all lowercase letters in the repository.

removeUser()

Removes a user from this group.

Data Sources
All

Syntax
Java Method

```
void removeUser(User user);
void removeUser(String userName);
                // throws ServerBloxException
```

where:

Argument	Description
user	A User object.
userName	A username

save()

Saves all changes to this group to the repository.

Data Sources
All

Syntax
Java Method

```
void save();   // throws ServerBloxException
```

Log Object Methods

This section describes methods associated with the Log object. To access this object from AdminBlox, use the AdminBlox.getLog() method. To use any method for this object, import the com.alphablox.blox.repository.Log package in your JSP since the message level constants are in this package.

getMinimumServerMessageLevel()

Identifies the minimum message level that will be sent to the Alphablox log system.

Data Sources
All

Syntax
Java Method

```
int getMinimumServerMessageLevel();
```

See Also
See "Server Message Level" on page 122.

sendException()

Sends an exception message to the Alphablox log system.

Data Sources
All

Syntax

Java Method

```
void sendException(int messageLevel, String messageTitle, Exception exception);
```

where:

Argument	Description
messageLevel	The message level. Valid values are MESSAGE_LEVEL_DEBUG, MESSAGE_LEVEL_VERBOSE, MESSAGE_LEVEL_INFO, MESSAGE_LEVEL_SYSTEM, MESSAGE_LEVEL_WARNING, and MESSAGE_LEVEL_ERROR. See "Server Message Level" on page 122 for descriptions on these message levels.
messageTitle	The title of this message.
exception	The Exception to be logged. A full stack trace will be placed in the log.

sendMessage()

Sends a message to the Alphablox log system.

Data Sources

All

Syntax

Java Method

```
void sendMessage(int messageLevel, String messageTitle, String message);
```

where:

Argument	Description
messageLevel	The message level. Valid values are MESSAGE_LEVEL_DEBUG, MESSAGE_LEVEL_VERBOSE, MESSAGE_LEVEL_INFO, MESSAGE_LEVEL_SYSTEM, MESSAGE_LEVEL_WARNING, and MESSAGE_LEVEL_ERROR. See "Server Message Level" on page 122 for descriptions on these message levels.
messageTitle	The title of this message.
message	The message text.

Role Object Methods

This section describes methods associated with the Role object. To access this object from AdminBlox, use the `AdminBlox.getRole(...)` or `AdminBlox.getRoles()` method. To use any method for this object, import the `com.alphablox.blox.repository` package in your JSP.

addGroup()

Adds a subgroup to this Role.

Data Sources

All

Syntax

Java Method

```
void addGroup(Group group, int rights);
void addGroup(String groupName, int rights);
```

where:

Argument	Description
group	A Group object.
groupName	The name of a group. Note that this name is converted to all lowercase letters in the repository.
rights	Valid values are: NO_ACCESS, READ_ONLY_ACCESS, READ_WRITE_ACCESS.

addUser()

Adds a user to this Role.

Data Sources

All

Syntax

Java Method

```
void addUser(User user, int rights);
void addUser(String userName, int rights);
```

where:

Argument	Description
user	A User object.
userName	A username.
rights	Valid values are: NO_ACCESS, READ_ONLY_ACCESS, READ_WRITE_ACCESS.

Usage

The user will not be added until save() is called.

See Also

"save()" on page 106.

getDescription()

Gets the description of this Role.

Data Sources

All

Syntax

Java Method

```
String getDescription();
```

getName()

Gets the name of this Role.

Data Sources
All

Syntax
Java Method

```
String getName();
```

isGroupInRole()
Identifies if the specified group is in this Role.

Data Sources
All

Syntax
Java Method

```
boolean isGroupInRole(Group group);
boolean isGroupInRole(String groupName);
                // throws ServerBloxException
```

where:

Argument	Description
group	A Group object.
groupName	The name of a group. Group names are stored in all lowercase letters in the repository.

isUserInRole()
Identifies if the specified user is in this Role.

Data Sources
All

Syntax
Java Method

```
boolean isUserInRole(User user);
boolean isUserInRole(String userName);
                // throws ServerBloxException
```

where:

Argument	Description
user	A User object.
userName	A username.

removeGroup()
Removes a subgroup from this Role.

Data Sources
All

Syntax
Java Method

```
void removeGroup(Group group);
void removeGroup(String groupName);
                // throws ServerBloxException
```

where:

Argument	Description
group	A Group object.
groupName	The name of a group. Group names are stored in all lowercase letters in the repository.

removeUser()

Removes a user from this Role.

Data Sources

All

Syntax

Java Method

```
void removeUser(User user);
void removeUser(String userName);
                // throws ServerBloxException
```

where:

Argument	Description
user	A User object.
userName	A username

save()

Saves all changes to this Role to the repository.

Data Sources

All

Syntax

Java Method

```
void save();   // throws ServerBloxException
```

Server Object Methods

This section describes methods associated with the Server object. To access this object from AdminBlox, use the `AdminBlox.getServer()` method. To use any method for this object, import the `com.alphablox.blox.repository` package in your JSP.

getApplicationServerType()

Gets the application server type.

Data Sources

All

Syntax
Java Method

```
short getApplicationServerType();
```

Usage
Returned result should be compared with the following constants: APPLICATION_SERVER_WEBSPHERE, APPLICATION_SERVER_WEBLOGIC, APPLICATION_SERVER_TOMCAT, or APPLICATION_SERVER_UNKNOWN.

getAuthorizedClientList()
Gets the list of users authorized to access the server.

Data Sources
All

Syntax
Java Method

```
String getAuthorizedClientList();
```

Usage
Returns the exact string specified in the Authorized Client List text box in the Server configuration page under the Administration tab.

getClusteringLeadIpAddress()
Gets the hostname or the IP address of the computer in which the lead host node of the DB2 Alphablox cluster is running.

Data Sources
All

Syntax
Java Method

```
String getClusteringLeadIpAddress();
```

getClusteringLeadPort()
Gets the port number of the computer in which the lead host node of the DB2 Alphablox cluster is running.

Data Sources
All

Syntax
Java Method

```
int getClusteringLeadPort();
```

getClusteringMaxHosts()
Gets the maximum amount of hosts that can be in the DB2 Alphablox cluster.

Data Sources
All

Syntax
Java Method

```
int getClusteringMaxHost();
```

getClusteringStartupWait()

Gets the amount of time, in seconds, this server cluster in which DB2 Alphablox is running will wait for a successful connection to the cluster lead node.

Data Sources

All

Syntax

Java Method

```
int getClusteringStartupWait();
```

getCommandFileName()

Gets the command file name.

Data Sources

All

Syntax

Java Method

```
String getCommandFileName();
```

Usage

This is the name of an optional file that DB2 Alphablox reads during startup. The file contains commands using the syntax described in Console Commands in the *Administrator's Guide*.

getDefaultMessageLevel()

Gets the minimum (least severe) level of messages to display and write to the log file.

Data Sources

All

Syntax

Java Method

```
int getDefaultMessageLevel();
```

Usage

Returns an integer to be evaluated against the message level constants: `MESSAGE_LEVEL_DEBUG`, `MESSAGE_LEVEL_ERROR`, `MESSAGE_LEVEL_FATAL`, `MESSAGE_LEVEL_INFO`, `MESSAGE_LEVEL_SYSTEM`, `MESSAGE_LEVEL_VERBOSE`, `MESSAGE_LEVEL_WARNING`.

See Also

"Server Message Level" on page 122

getHtmlClientTheme()

Gets the default theme used.

Data Sources

All

Syntax
Java Method

```
String getHtmlClientTheme();
```

Usage
Returns the name of the theme.

getInstanceName()
Gets the DB2 Alphablox instance name.

Availability

Render Modes	All
Data Sources	All

Syntax
Java Method

```
String getInstanceName();
```

Usage
The instance name is specified during installation. The default name is `Alphablox`.

getMaxCubes()
Gets the maximum number of cubes that can be active at one time.

Data Sources
Multidimensional

Syntax
Java Method

```
int getMaxCubes();
```

Usage
This method only applies when isMaxCubesEnabled() is set to `true`.

See Also
"isMaxCubesEnabled()" on page 116.

getMessageHistorySize()
Gets the number of messages saved to the message history area.

Data Sources
All

Syntax
Java Method

```
int getMessageHistorySize();
```

Usage
Returns the number of messages saved to the message history area. When the area fills, the server wraps to the beginning, overwriting the oldest messages. The default value is 100.

getNewLogEndMessageLevel()

Gets the most severe message level to write to a new log file.

Data Sources

All

Syntax

Java Method

```
int getNewLogEndMessageLevel();
```

Usage

The most severe message level is 7 (`MESSAGE_LEVEL_FATAL`).

See Also

"getNewLogStartMessageLevel()" on page 110, "Server Message Level" on page 122

getNewLogStartMessageLevel()

Gets the least severe message level to write to a new log file.

Data Sources

All

Syntax

Java Method

```
int getNewLogStartMessageLevel();
```

Usage

The least severe message level is 1 (`MESSAGE_LEVEL_DEBUG`).

See Also

"getNewLogEndMessageLevel()" on page 110, "Server Message Level" on page 122

getPoweredBy()

Returns the descriptive text about the server.

Data Sources

All

Syntax

Java Method

```
String getPoweredBy();
```

Usage

Returns a string describing the details of the server. The returned string looks as follows:

```
DB2 Alphablox Release 8.2.0 Build 74 [General Availability] / IBM
WebSphere Application Server/5.1
```

getRepositoryDatabaseAdapter()

Gets the database adapter name if the server is using a database repository.

Data Sources

All

Syntax
Java Method

```
String getRepositoryDatabaseAdapter();
```

Usage
Throws a ServerBloxException if the server is not using a database repository.

getRepositoryDatabaeDriver()
Gets the database driver if the server is using a database repository.

Data Sources
All

Syntax
Java Method

```
String getRepositoryDatabaseDriver();
```

Usage
Throws a ServerBloxException if the server is not using a database repository.

getRepositoryDatabaseHostName()
Gets the database host name if the server is using a database repository.

Data Sources
All

Syntax
Java Method

```
String getRepositoryDatabaseHostName();
```

Usage
Throws a ServerBloxException if the server is not using a database repository.

getRepositoryDatabaseIsolationLevel()
Gets the database isolation level used on most transactions if the server is using a database repository.

Data Sources
All

Syntax
Java Method

```
String getRepositoryDatabaseIsolationLevel();
```

Usage
Throws a ServerBloxException if the server is not using a database repository.

getRepositoryDatabaseName()
Gets the database name if the server is using a database repository.

Data Sources
All

Syntax
Java Method

```
String getRepositoryDatabaseName();
```

Usage
Throws a ServerBloxException if the server is not using a database repository.

getRepositoryDatabasePort()
Gets the database port if the server is using a database repository.

Data Sources
All

Syntax
Java Method

```
int getRepositoryDatabasePort();
```

Usage
Throws a ServerBloxException if the server is not using a database repository.

getRepositoryDatabaseUser()
Gets the user name used to log into the database if the server is using a database-based repository.

Data Sources
All

Syntax
Java Method

```
String getRepositoryDatabaseUser();
```

Usage
Throws a ServerBloxException if the server is not using a database repository.

getRepositoryFileDirectory()
Gets the repository directory if the server is using a file-based repository.

Data Sources
All

Syntax
Java Method

```
String getRepositoryFileDirectory();
```

Usage
Throws a ServerBloxException if the server is not using a file-based repository.

getRepositoryServiceProvider()
Gets the repository service provider type that is used by the Repository Manager.

Data Sources
All

Syntax
Java Method

```
int getRepositoryServiceProvider();
```

Usage
Returns a value that should be evaluated against one of the following constants: `PROVIDER_TYPE_DB`, `PROVIDER_TYPE_FILE`, and `PROVIDER_TYPE_UNKNOWN`.

getServerBuildVersion()
Gets the server build version.

Data Sources
All

Syntax
Java Method

```
String getRepositoryFileDirectory();
```

getServerIdleDuration()
Gets the number of minutes of idle time before the server automatically goes into suspend mode.

Data Sources
All

Syntax
Java Method

```
int getServerIdleDuration();
```

getServerIncrementVersion()
Gets the server increment version.

Data Sources
All

Syntax
Java Method

```
int getServerIncrementVersion();
```

Usage
When the server version is 5.0.1.2 (as returned by getServerVersion()), the server increment version would be 1.

See Also
"getServerVersion()" on page 114

getServerLogFileName()
Gets the Alphablox log file name used to store Alphablox messages.

Data Sources
All

Syntax
Java Method
```
String getServerLogFileName();
```

getServerMajorVersion()
Gets the server major version number.

Data Sources
All

Syntax
Java Method
```
int getServerMajorVersion();
```

Usage
When the server version is 5.0.1.2 (as returned by getServerVersion()), the server major version would be 5.

See Also
"getServerVersion()" on page 114

getServerMinorVersion()
Gets the server minor version number.

Data Sources
All

Syntax
Java Method
```
int getServerMinorVersion();
```

Usage
When the server version is 5.0.1.2 (as returned by getServerVersion()), the server minor version would be 0.

See Also
"getServerVersion()" on page 114

getServerVersion()
Gets the server version string.

Data Sources
All

Syntax
Java Method
```
String getServerIncrementVersion();
```

Usage
Returns a string such as 5.0.1.2 that shows the major version number, minor version number, patch/increment number, and build number.

See Also
"getServerBuildVersion()" on page 113, "getServerIncrementVersion()" on page 113, "getServerMajorVersion()" on page 114, "getServerMinorVersion()" on page 114

getSmtpServer()

Gets the name of the SMTP server used in some applications (such as Quick View and Fast Forward application template builder) to send email.

Data Sources

All

Syntax

Java Method

```
String getSmtpServer();
```

getTelnetConsoleName()

Gets the username used when accessing the telnet console.

Data Sources

All

Syntax

Java Method

```
String getTelnetConsoleName();
```

getTelnetConsolePort()

Gets the port on which the telnet version of the server console operates.

Data Sources

All

Syntax

Java Method

```
int getTelnetConsolePort();
```

getTelnetTimeout()

Gets the time, in minutes, when the telnet console times out due to no activity in the telnet session.

Data Sources

All

Syntax

Java Method

```
int getTelnetTimeout();
```

isAuthenticationEnabled()

Identifies if DB2 Alphablox authenticates users against the User manager.

Data Sources

All

Syntax

Java Method

```
boolean isAuthenticationEnabled();
```

Usage

Returns `true` if authentication is turned on. The default is `true`.

isAutoCreateUsers()

Identifies if a user account is automatically created when the user logs into DB2 Alphablox.

Data Sources
All

Syntax
Java Method

```
boolean isAutoCreateUsers();
```

Usage
Returns `true` if a user account will be automatically created when a new user logs in.

This is usually used in cases where some other external system (such as Windows NT®) is used to authenticate users. This enables you to use that system for authentication instead of having to maintain user accounts in multiple places. However, when this option is enabled, Authorized Client List should also be specified to avoid unwanted access. See the *Administrator's Guide* for more details.

isClusteringEnabled()

Identifies if server cluster mode is enabled.

Data Sources
All

Syntax
Java Method

```
boolean isClusteringEnabled();
```

Usage
Returns `true` when server clustering is enabled.

isMaxCubesEnabled()

Identifies if maximum active cube restriction is enabled.

Data Sources
Multidimensional

Syntax
Java Method

```
boolean isMaxCubesEnabled();
```

Usage
Returns `true` when maximum active cube restriction is enabled. Use getMaxCubes() to find out the maximum number of active cube allowed.

See Also
"getMaxCubes()" on page 109

isSaveOnExit()

Identifies if the server and user settings will be saved when DB2 Alphablox is shut down or a session ends.

Data Sources
All

Syntax
Java Method

```
boolean isSaveOnExit();
```

Usage
Returns `true` if the server and user settings will be saved when DB2 Alphablox is shut down or a session ends.

isServerLogEnabled()
Identifies if the server will log messages to the log file.

Data Sources
All

Syntax
Java Method

```
boolean isServerLogEnabled();
```

Usage
Returns `true` if messages are logged.

levelIntToString()
Converts the system message level from constants (integers) to strings.

Data Sources
All

Syntax
Java Method

```
String levelIntToString();
```

Examples
The following code:

```
The warning message level string is: <%=
myAdmin.getServer().levelIntToString(Server.MESSAGE_LEVEL_WARNING) %>
```

produces the output as shown below:

```
The warning message level string is: WARNING
```

See Also
"Server Message Level" on page 122

levelStringToInt()
Converts the system message level from a string to an integer.

Data Sources
All

Syntax
Java Method

```
int levelStringToInt();
```

Examples

The following code:

```
The value for string "DEBUG" =
<%= myAdmin.getServer().levelStringToInt("DEBUG")%>
```

produces the output as shown below:

```
The value for string "DEBUG" = 1
```

See Also

"Server Message Level" on page 122

User Object Methods

This section describes methods associated with the User object. To access this object from AdminBlox, use the `AdminBlox.getUser(...)` or `AdminBlox.getUsers()` methods. To use any method for this object, import the `com.alphablox.blox.repository` package in your JSP.

delete()

Deletes this user from the repository.

Data Sources

All

Syntax

Java Method

```
void delete();
```

Usage

Throws a ServerBloxException if the user cannot be deleted.

getDescription()

Gets the description for this User.

Data Sources

All

Syntax

Java Method

```
String getDescription();
```

getEmail()

Gets the user's email address.

Data Sources

All

Syntax

Java Method

```
String getEmail();
```

getGroupNames()

Returns a String array containing the names of the groups that the current user is associated with.

Data Sources
All

Syntax
Java Method

```
String[] getGroupNames();   //throws ServerBloxException
```

getName()

Gets the username of this User.

Data Sources
All

Syntax
Java Method

```
String getName();
```

Usage
Returns the username.

getPrimaryGroupName()

Gets the user's primary group name.

Data Sources
All

Syntax
Java Method

```
String getPrimaryGroupName();
```

Usage
This group has top priority when the user is authenticated

isCanEdit()

Identifies if the user can edit his or her own user profile.

Data Sources
All

Syntax
Java Method

```
boolean isCanEdit();
```

Usage
Returns `true` if the user is allowed to edit his or own user profile.

save()

Saves this user to the DB2 Alphablox repository.

Data Sources
All

Syntax
Java Method

```
void save();  //throws ServerBloxException
```

Usage
This method must be called for any changes set through all the User object's set methods to be saved to the repository.

setCanEdit()
Specifies if the user can edit his or her own user profile.

Data Sources
All

Syntax
Java Method

```
void setCanEdit(boolean canEdit);
```

where:

Argument	Description
`canEdit`	`true` if the user is allowed to edit his or her own user profile.

Usage
Must call save() for the changes to be saved to the repository.

setDescription()
Sets the description for this user.

Data Sources
All

Syntax
Java Method

```
void setDescription(String description);
```

where:

Argument	Description
`description`	A description about the user.

Usage
Must call save() for the changes to be saved to the repository.

setEmail()
Sets the user's email address.

Data Sources
All

Syntax
Java Method

```
void setEmail(String email);
```

where:

Argument	Description
`email`	The user's email address.

Usage
Must call save() for the changes to be saved to the repository.

setFullName()
Sets the description for this user.

Data Sources
All

Syntax
Java Method

```
void setFullName(String fullName);
```

where:

Argument	Description
`fullName`	The user's full name.

Usage
Must call save() for the changes to be saved to the repository.

setPassword()
Sets the user's password.

Data Sources
All

Syntax
Java Method

```
void setPassword(String newPassword, String oldPassword);
```

where:

Argument	Description
`newPassword`	The new password.
`oldPassword`	The old password.

Usage
The correct old password must be supplied. Throws a ServerBloxException if the old password is not specified correctly. Must call save() for the changes to be saved to the repository.

setPrimaryGroupName()
Sets the user's primary group name.

Data Sources
All

Syntax

Java Method

```
void setPrimaryGroupName(String primaryGroupName);
```

where:

Argument	Description
primaryGroupName	The primary group name. Note that group names are converted to all lowercase letters in the repository.

Usage

Must call save() for the changes to be saved to the repository.

Server Message Level

The DB2 Alphablox provides seven message levels to log messages for server monitoring and debugging purposes. Administrators can specify the New Log Start Message Level and New Log End Message Level in the System page under the Administration tab. Setting the values on these two properties creates a log containing messages within the range of the specified levels.

The following table lists the message level constants and a description for the kind of message each level logs. It also shows the string and integer values used by the levelIntToString() and levelStringToInt() methods.

Message Level Constant	String	Value	Description
MESSAGE_LEVEL_DEBUG	DEBUG	1	Messages that aid debugging the system.
MESSAGE_LEVEL_VERBOSE	VERBOSE	2	All system messages.
MESSAGE_LEVEL_INFO	INFO	3	Minor system events for which no administrator action is necessary.
MESSAGE_LEVEL_SYSTEM	SYSTEM	4	Messages for major, normal system events (such as new sessions).
MESSAGE_LEVEL_WARNING	WARNING	5	Messages that indicate a minor, recoverable, error has occurred, suggesting issues the administrator may want to investigate.
MESSAGE_LEVEL_ERROR	ERROR	6	Messages that indicate an operation cannot be completed and a non-recoverable error has occurred.
MESSAGE_LEVEL_FATAL	FATAL	7	Messages that indicate that a fatal error has occurred that can cause the server to terminate.

Chapter 7. BookmarksBlox Reference

This chapter contains a general overview of bookmarks and reference material for BookmarksBlox properties, methods and objects. For general reference information about Blox, see Chapter 3, "General Blox Reference Information," on page 17. For information on how to use this reference, see Chapter 1, "Using This Reference," on page 1.

BookmarksBlox Overview

Through the Blox user interface, end-users can bookmark data views with either private, public, or group accessibility for later retrieval. Bookmarking a view is done via the Bookmark button in the Toolbar or the Bookmark option from the right-click menu. Users can also load, delete, or rename existing bookmarks that are visible to them.

A bookmark is essentially a collection of property sets. Each bookmark contains the following information:

- the name of the Blox whose state is stored
- the change in properties from the initial application state of the Blox to the current state when the bookmark is added
- the name of the user who owns the bookmark
- the bookmark's visibility
- some description about the bookmark

When a bookmark is saved, only the difference between a Blox's current state (after the user interacts with the data) and its initial state (the default property values or the values specified when the Blox is created) are stored in the

repository. When a bookmark is loaded, live data is retrieved from the data source based on the Blox properties information stored in the repository.

BookmarksBlox, with its extensive API, allows you to programmatically create and manage bookmarks and dynamically set the bookmark properties. For example, you can create time-series reports or reports that always fetch the data for the current quarter by dynamically modifying the data query stored with a bookmark. You can use custom bookmark properties to store each user's choice of report layout or implement your own security. You can modify the query stored with a bookmark in the case of change of member names or outline in the data source. You can even create your own bookmark management user interface.

To use the BookmarksBlox API, add a BookmarksBlox to your page. This gives you access to each bookmark as a Bookmark object.

Bookmark Concepts and Features

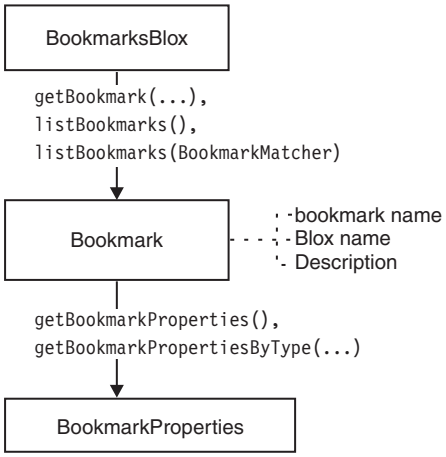
Bookmarking is a powerful feature with an extensive API that allows you to perform various custom actions. This section discusses the following key concepts and features of bookmarks and related Bookmark objects:

What is a Bookmark?

A Bookmark is a collection of property sets. A Bookmark, by itself, has its own properties like application, description, name, and visibility. It also stores information with regard to an individual Blox. This Blox can be a standalone Blox that has no nested Blox (such as a DataBlox), or a Blox with nested Blox (such as a PresentBlox). For example, if a Bookmark is added on a PresentBlox, information on the individual nested Blox is also stored.

You can use the BookmarksBlox API to access specific bookmarks by specifying the search criteria such as the bookmark's name, the Blox name, the owner's name, and its visibility. In addition, the BookmarksBlox API allows you to modify the properties or even apply the bookmark to a different Blox.

The following diagram shows the object hierarchy of BookmarksBlox.



Note: To access the Bookmark and BookmarkProperties objects, you should add the following import directive in your JSP:

```
<%@ page import="com.alphablox.blox.repository.*" %>
```

Note: Bookmark names can only contain the following characters: A-Z, a-z, 0-9, and underscore (_).

Blox Default States vs. Initial Application State vs. Current Blox State

The `BookmarkProperties` object only contains properties that are not the same as the initial Blox state. For example, if ChartBlox's `chartType` is not set in the tags, the default chart type is "Vertical Bar, Side-by-Side, 3D Effect." If a bookmark is saved and the current displayed chart type is "Vertical Bar, Side-by-Side, 3D Effect," then the chart type property will not be in the list of properties stored for the Blox.

Besides Blox default states, you may set the chart type to "Pie" through the ChartBlox tag. This specified property, together with the default values of the other unspecified Blox properties, dictates how the Blox are instantiated and rendered. This is the initial application state. The state is changed when a user interacts with the data, such as changing the chart type, drilling down, hiding members, swapping axes, or setting some other cell banding style. When he saves a bookmark on the current view, what is stored with the bookmark is the difference between the initial application state and the current Blox state.

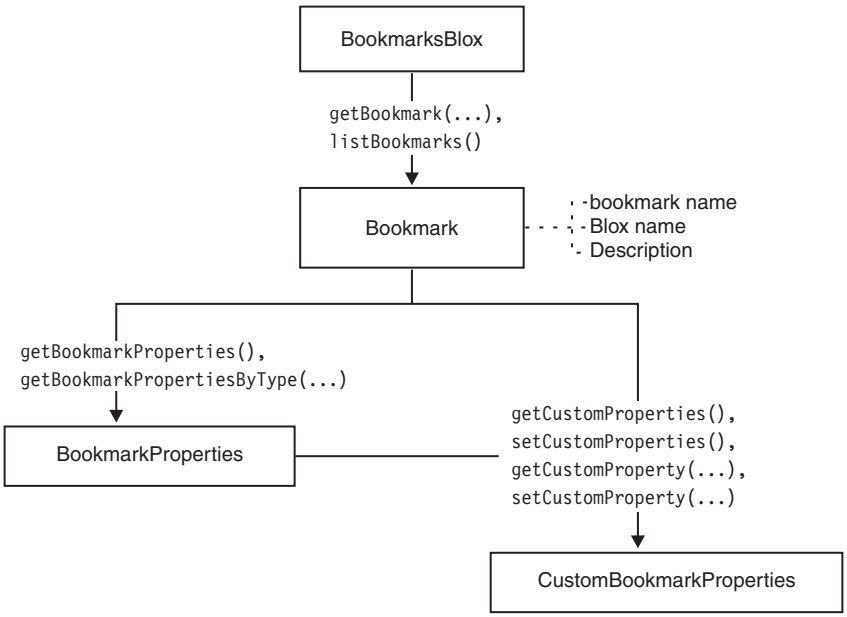
Custom Bookmark Properties

Besides the default bookmark properties, you can also add custom properties to bookmarks. Similar to the custom user properties and custom application properties available in RepositoryBlox, custom bookmark properties allow you to store any information in a name-value pair that you may need in your application. For instance, you may want to build a navigation tree menu for bookmarks. Using the custom bookmark properties, you can store the folder names for dynamically building the tree menu. Or you can implement access control so only certain users or groups can see certain folders in your navigation tree. These properties do not affect the behavior of bookmarks in any way but allow you to save and get custom properties.

Custom bookmark properties are different from custom user/application properties in that they are not defined through the DB2 Alphablox Admin Pages and are not

accessed via the RepositoryBlox. To create and access custom bookmark properties, first add a BookmarksBlox to your JSP file and then you can:

- Use the `BookmarksBlox.getBookmark(...).setCustomProperties()` method to set custom properties
- Use the `BookmarksBlox.getBookmark(...).getCustomProperties()` method to get all custom properties associated with a bookmark
- Access individual custom properties by their key using the `getCustomProperty(`*key*`)` method.



Bookmark Visibility

A bookmark can be private, public, or group visible only. By default, bookmarks are added as private bookmarks unless the users (through the Blox user interface) or developers (through BookmarksBlox API) specify otherwise. Bookmark visibility is marked using the following static fields:

- `PRIVATE_VISIBILITY`
- `PUBLIC_VISIBILITY`

For group visibility, use the name of the group to get group bookmarks.

Blox Types and Binding

When a bookmark is saved on a Blox through the Blox user interface, properties of all nested Blox are also saved if they are not in their initial state. If a bookmark on a PresentBlox is saved, in the folder for the bookmark, there may be a separate folder for each of the nested Blox like the following if the Blox is in a state different from its initial state:

- `<blox name>_data` (if using an implicit DataBlox that is not explicitly defined with an `id` outside a presentation Blox)
- `<blox name>` (for the PresentBlox)
- `<blox name>_chart`
- `<blox name>_datalayout`
- `<blox name>_grid`
- `<blox name>_page`

- <blox name>_toolbar

Through the BookmarksBlox API, you can access the property set of a nested Blox by specifying its Blox type. Blox types are marked using static fields:

- CHART_BLOX_TYPE
- DATA_BLOX_TYPE
- DATALAYOUT_BLOX_TYPE
- GRID_BLOX_TYPE
- PAGE_BLOX_TYPE
- PRESENT_BLOX_TYPE
- TOOLBAR_BLOX_TYPE

This allows you to directly access and modify the property set of a specific Blox type.

The physical location of the bookmark is called the binding. A binding is the association of an object with a logical name and a context. It is based on the Java Naming and Directory Interface (JNDI), which provides Java technology-enabled applications with a unified interface to seamlessly navigate across databases, files, directories, objects, and networks. J2EE containers use this information to locate needed resources. Using the `getContainer()` and `getBinding()` methods on the Bookmark object, you can get the physical location of a bookmark.

Bookmark Matchers and Bookmark Filters

You can find a list of bookmarks that match a certain criterion, or get a list of bookmarks for an application, for a specific group of users, or for a specific user. Since application, user, and group specific information is stored in the repository, objects supporting bookmark filtering are in the `com.alphablox.blox.repository` package. These objects include BookmarkMatcherApplications, BookmarkMatcherGroups, BookmarkMatcherUsers, and BookmarkMatcherAll.

The BookmarkMatcherApplications object is used to find bookmarks based upon which application owns a bookmark. An application bookmark is equivalent to a public bookmark. The BookmarkMatcherGroups object is used to find bookmarks based upon which group owns a bookmark. The BookmarkMatcherUsers object is used to find bookmarks based upon which user owns a bookmark. A user bookmark is equivalent to a private bookmark. The BookmarkMatcherAll object lets you find all bookmarks for a specified application, user, visibility or Blox name.

All these BookmarkMatcher objects work in much the same way as an extended Java SDK File Filter class except that BookmarkMatcherUsers has a `setUser()` method that can be optionally called to find specific bookmarks for a user; BookmarkMatcherApplications has a `setApplication()` method that can be optionally called to find specific bookmarks for an application; and BookmarkMatcherGroups has a `setVisibility()` method that can be optionally called to find specific bookmarks for a group. Each of these objects has an `accept()` method. This method is called for every Bookmark object to see if it should be included in the list of bookmarks returned.

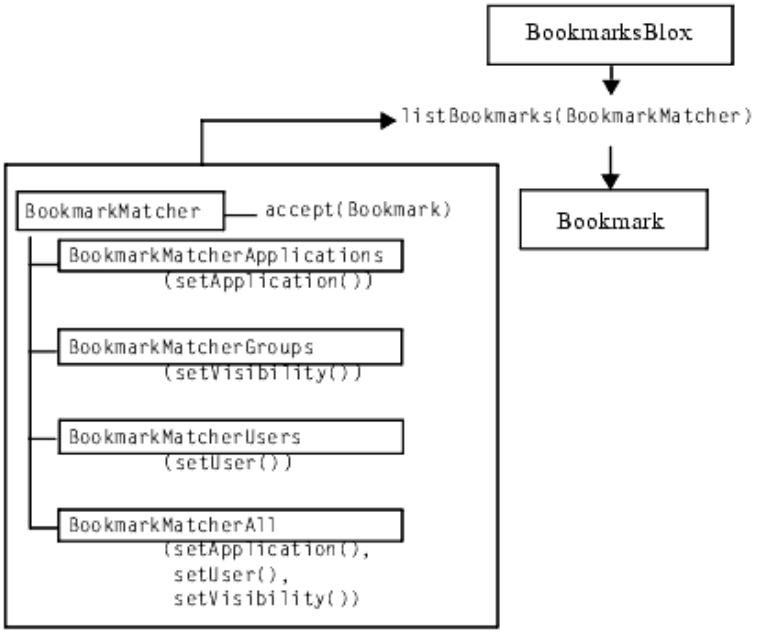
You can use these objects or extend them depending on what type of bookmark matching you want to perform. It is recommended that if any type of custom application/group/user bookmark matching is to be performed that it uses or extends BookmarkMatcherApplications, BookmarkMatcherGroups,

BookmarkMatcherUsers, or BookmarkMatcherAll. DB2 Alphablox has optimized these classes to perform quick searches for applications, groups, and users.

Note: To access all these BookmarkMatcher objects, you should add the following import directive in your JSP:

```
<%@ page import = "com.alphablox.blox.repository.*" %>
```

The following diagram shows the how these objects are related to the Bookmark object.



Bookmark Events and Event Filters

You can intercept events when a user clicks to delete, edit, add, or save a bookmark. Using the server-side event filters, you can intercept the events and perform some actions *before* the server processes them. To use server-side event filters, generally involves two steps.

1. First you add the specific event filter object using the common Blox method addEventFilter(). For example,

   ```
   <blox:present id="myPresent">
   ...
   <%
     myPresent.addEventFilter(new LoadFilter() );
   %>
   </blox:present>
   ```

2. Then write your own class that implements the corresponding event filter object (`BookmarkDeleteFilter`, `BookmarkLoadFilter`, `BookmarkRenameFilter` and `BookmarkSaveFilter`) and the corresponding method (bookmarkDelete(BookmarkDeleteEvent), bookmarkLoad(BookmarkLoadEvent), bookmarkRename(BookmarkRenameEvent), and bookmarkSave(BookmarkSaveEvent)) that will be called with the event is triggered. For example:

   ```
   public class LoadFilter implements BookmarkLoadFilter
   {
      public void bookmarkLoad( BookmarkLoadEvent bre )
   ```

```
    {
       //actions to take when the event is triggered
    }
}
```

For more information on bookmark events and event filters, see "Example 5: Using server-side bookmarkLoad event filter" on page 136, "Bookmark and Application State Properties and Methods" on page 30, and "Event Filter Objects Overview" on page 463.

Serialized Query and Textual Query

When a bookmark is first created, the *delta* between the original query set in the underlying DataBlox and the associated query that generates the current data view is saved as well. In the case of a file repository, two files are saved in the bookmark's `<blox name>_data` folder in the repository— *bookmarkName*`.data` and *bookmarkName*`.query`. The `.data` file is a text file that contains the basic properties for reconnecting to the data source, such as application name, data source name, last executed query, and the page axis members. It looks like the following:

```
Associated.query = q2report
ResultSet.Market = East,West,South,Central,Market
applicationName = SalesApp
connectOnStartup = true
dataSourceName = TBC
dimensionsOnPageAxis = {[null]}
parentFirst = {[null]}
query = <Row(Market) <ICHILD Market <Column(Year) Year !
```

Textual Queries

The `.data` text file is created when a bookmark is first added. Depending on how the bookmark is created, you may or may not see the `query` entry. If the bookmark is created through the API by setting the bookmark object's `query` property, you will see the query string in the text file. This file, however, is not updated when the bookmark is resaved. To keep the textual query in sync with the serialized query when users try to resave a bookmark with a different data view:

1. First use the DataBlox `generateQuery()` method to get the textual query for the current data view.
2. Use the `addEventFilter()` common Blox method to add a method that implements the `BookmarkSaveFilter` interface to update the query stored in the bookmark every time a bookmark is resaved.

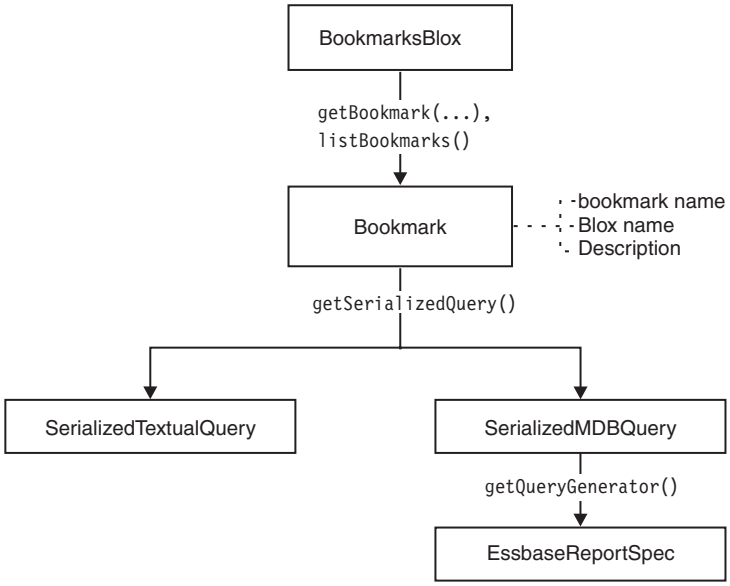
Keeping the textual query in sync allows you to modify the textual query later in cases such as data outline change. Textual queries may be more efficient since DB2 Alphablox does not need to manipulate the result set to match the serialized object.

When a bookmark is loaded, by default, the serialized query is used. To load a bookmark using the textual query, set the DataBlox `textualQueryEnabled` property to `true`. For an example of how to change the query before a bookmark is loaded, see "Example 6: Getting a bookmark's query when it is loaded" on page 137.

Serialized Queries

The `.query` file contains the serialized object that, in many ways, are similar to the GridBlox result set except that it has no data. It stores information on the axes, tuples, dimensions, and members. You can programmatically access the axes, tuples, dimensions, and members and modify the query before a bookmark is loaded. Or you can modify all bookmarks stored in the repository in cases where member names or the data outline have changed.

The following diagram shows how you can access the SerializedMDBQuery object (for multidimensional data sources) and the SerializedTextualQuery object (for relational data sources) through BookmarksBlox. The SerializedMDBQuery object lets you get information on the axis, dimension, tuple and member involved and replace a old member with a new member. You can also access the EssbaseReportSpec object in order to obtain specific Essbase Report Scripts. The SerializedTextualQuery object lets you get the saved query and set a new query.



Static Fields for the Bookmark Object

The Bookmark object contains the following static fields to indicate Blox type, bookmark visibility, and null dimension:

Static Field by Category

Blox Type

```
CHART_BLOX_TYPE

DATA_BLOX_TYPE

DATALAYOUT_BLOX_TYPE

GRID_BLOX_TYPE

PAGE_BLOX_TYPE

PRESENT_BLOX_TYPE

TOOLBAR_BLOX_TYPE

UNKNOWN_BLOX_TYPE
```

Bookmark Visibility

```
PRIVATE_VISIBILITY
```

PUBLIC_VISIBILITY

Null Dimension

NULL_DIMENSION

These static fields give you a way to specify and identify Blox type and bookmark visibility using constants.

BookmarksBlox JSP Custom Tag Syntax

The Alphablox Tag Libraries provides custom tags to use in a JSP page for creating each blox. This section describes how to create the custom tag to create a BookmarksBlox. For a copy and paste version of the tag with all the attributes, see "BookmarksBlox JSP Custom Tag" on page 881.

Parameters

```
<blox:bookmarks
    [attribute="value"] >
</blox:>
```

where:

attribute is one of the attributes listed in the attribute table.

value is a valid value for the attribute.

Valid attributes are listed in the following table:

Attribute
id
bloxName

Usage

Each custom tag can have one or more attributes, separated by one or more space or new line characters. Any extra space or new line characters are ignored. For readability, attributes can each go on a separate line with the same indenting.

You can substitute the closing `</blox:bookmarks>` tag using the shorthand notation, closing the tag at the end of the attribute list that looks as follows:

```
id="myBookmarksBlox" />
```

Examples

```
<blox:bookmarks
   id = "myBookmarksBlox" />
```

BookmarksBlox Examples

This sections provide examples that demonstrate how to use BookmarksBlox, its associated objects and related methods:

- "Example 1: Getting a count of all bookmarks" on page 132
- "Example 2: Getting the properties set for a Bookmark" on page 132
- "Example 3: Getting a list of bookmarks that match the specified criteria" on page 134

Example 1: Getting a count of all bookmarks

This example demonstrates the following:

- the use of BookmarksBlox and its `listBookmarks()` method to gain access to all bookmarks stored in the repository. The `listBookmarks()` method returns an array of bookmark objects
- how to get a count of the total number of bookmarks by getting the length of the array

```
<%@ taglib uri="bloxtld" prefix="blox" %>
<!--import the following package in order to access the
    com.alphablox.blox.repository.Bookmark class-->
<%@ page import="com.alphablox.blox.repository.*" %>

<blox:bookmarks id="myBookmarksBlox"/>

<%
   Bookmark bks[] = null;
   bks = myBookmarksBlox.listBookmarks();
%>
There are <%= bks.length %> bookmark(s).
```

Example 2: Getting the properties set for a Bookmark

This example demonstrates how to access a bookmark based on the bookmark name, application name, user name, Blox name, and bookmark visibility and get information on its properties set. In particularly, it demonstrates:

- the use of the BookmarksBlox to access individual bookmarks (the Bookmark object)
- the use of the Bookmark object's `getName()`, `getVisibility()`, `getDescription()`, `getBloxType()`, and `getBinding()` methods
- the use of the Bookmark object's `getBookmarkProperties()` method to access the individual properties (one for each nested Blox)

The generated output looks like the following:

The bookmark you are looking for exists.

1. The Repository JNDI binding for this bookmark is:
   `users/admin/salesapp/mygrid/bookmark/q2fy02WestSales/properties`
2. The bookmark name is: q2fy02WestSales
3. The type of Blox this bookmark was saved for is: grid
4. The bookmark description is: The Q2 West Sales
5. The bookmark visibility is: private
6. The bookmark contains Blox properties in the repository
7. Types of Blox properties saved in the bookmark:
   - grid
   - data

The code is as follows:

```
<%@ page import="com.alphablox.blox.repository.*,
                 com.alphablox.blox.ServerBloxMissingResourceException,
                 com.alphablox.blox.ServerBloxException,
```

```
                com.alphablox.blox.BookmarksBlox" %>
<%@ page import="java.util.*" %>
<%@ page import="java.io.*" %>
<%@ taglib uri="bloxtld" prefix="blox"%>

<html>
<head>
<!-- Blox header tag -->
<blox:header/>
</head>

<body>
<!-- Get an SSPM BookmarkBlox -->
<blox:bookmarks id="bookmarks" />
<ol>

<!--getting the bookmark you want-->
<%
   String bookmarkName = "q2fy02WestSales";
   String applicationName = "SalesApp";
   String userName = "admin";
   String bloxName = "myGrid";
   String visibility = Bookmark.PRIVATE_VISIBILITY;
   Bookmark bk = null;
   try {
      bk = bookmarks.getBookmark(bookmarkName, applicationName, userName,
bloxName, visibility);
   } catch (ServerBloxException e){
      if (e instanceof ServerBloxMissingResourceException){
         %><li><b>The bookmark DOES NOT EXIST!</b></li><%
      }
      else
         throw e;
   }
   if (bk != null)  {
      %><li><b>The bookmark you are looking for exists.</b></li>
   <li><b>The Repository JNDI binding for this bookmark is:<br>
         <%= bk.getBinding() %></b></li>
   <li><b>The bookmark name is: <%= bk.getName() %></b></li>
   <li><b>The type of Blox this bookmark was saved for is: <%=
      bk.getBloxType() %></b></li>
   <li><b>The bookmark description is: <%= bk.getDescription() %>
      </b></li>
   <li><b>The bookmark visibility is: <%= bk.getVisibility() %>
      </b></li>

   <!---Getting the individual BookmarkProperties ---->
   <%
      BookmarkProperties props[] = bk.getBookmarkProperties();
      if (props != null) {
         %><li><b>The bookmark contains Blox properties in the
         repository</b><br>
         Types of Blox properties saved in the bookmark:
         <ul>
         <%
         for (int i = 0; i < props.length; i++) {
            %><li><%= props[i].getType() %></li><%
         }
         %></ul><br></li><%
      }
      else {
         %><li><b>The bookmark DOES NOT CONTAIN Blox properties in the
         repository</b></li><%
      }
   }%>
 </body>
 </html>
```

Example 3: Getting a list of bookmarks that match the specified criteria

This example demonstrates the following:

- getting bookmarks for a specified user, and in this example, the user "admin" with the use of the BookmarkMatcher object
- the use of the Bookmark object's `getBinding()` and `getBloxType()` methods and their output

The generated output is as follows:

Got 5 Bookmark Object(s) for user admin.

The Bookmarks are:

users/admin/salesapp/salesgrid/bookmark/salesq1fy03/properties (grid)

users/admin/salesapp/salespresent/bookmark/eastq2fy03/properties (present)

users/admin/budgetapp/mypresent/bookmark/eastq3budget/properties (present)

users/admin/budgetapp/mypresent/bookmark/westq3budget/properties (present)

users/admin/budgetapp/present2/bookmark/mybudget/properties (present)

The code is as follows:

```
<%@ taglib uri="bloxtld" prefix="blox" %>
<!--import the following package in order to access the
    com.alphablox.blox.repository.BookmarkMatcherUsers class-->
<%@ page import="com.alphablox.blox.repository.*" %>
<html>
<head>
   <blox:header/>
</head>
<body>
<blox:bookmarks id="myBookmarksBlox" />
<%
   Bookmark bks[] = null;
   BookmarkMatcherUsers matcher = new BookmarkMatcherUsers();
   bks = null;
   matcher.setUser("admin");
   bks = myBookmarksBlox.listBookmarks(matcher);
%>
   <div>Got <%= bks.length %> Bookmark Object(s) for
        user <%= matcher.getUser() %></div>
   <div>The Bookmarks are:</div><br>
<%
   for (int i = 0; i < bks.length; i++) {
%><%= bks[i].getBinding() %> (<%= bks[i].getBloxType() %>)<br>
<%
      }
   %></div>
</body>
</html>
```

Example 4: Creating a bookmark using BookmarksBlox API

This example shows how to use a BookmarksBlox, Bookmark and BookmarkProperties classes to create a new bookmark. There are two ways to create a bookmark programmatically:

- Supply all the bookmark options to `BookmarksBlox.createBookmark(...)`
- Supply a Blox along with other information needed to `BookmarksBlox.createBookmark(...)`

This example demonstrates the later approach.

1. We specify the bookmark name, application name, user name, Blox name, visibility, and description associated with the bookmark.
2. Then we create a Bookmark object called "`bk`" using the `createBookmark()` method, and specify the Blox type to be `GRID_BLOX_TYPE`.
3. For the "`bk`" object, we create an instance of the BookmarkProperties object called "`gridBloxProp`" to store GridBlox specific properties and another called "`dataBloxProp`" to store DataBlox specific properties. For `gridBloxProp`, we set `cellBandingEnabled` to `true`; for `dataBloxProp`, we set the query to "`!`" and specify to reconnect to the data source.
4. Call the `saveAll()` method to save the bookmark we just created into the repository.

The generated output looks like the following:

(a GridBlox here)

```
We've got a Bookmark object from BookmarksBlox.createBookmark()!
Created a bookmark: q2fy02WestSales
At binding: users/jdoe/salesapp/mygrid/bookmark/q2fy02westsales/properties
```

Here is the code:

```
<%@ taglib uri="bloxtld" prefix="blox" %>
<!--import the following package in order to access the
    com.alphablox.blox.repository.BookmarkMatcherUsers class-->
<%@ page import="com.alphablox.blox.repository.*" %>
<blox:header />
<blox:bookmarks id="myBookmarksBlox" />

<blox:grid id="myGrid" width="500" height="320">
   <blox:data dataSourceName="qcc-essbase" query="!"/>
</blox:grid>
<%
// (1) Specify the bookmark properties
   String bookmarkName = "q2fy02WestSales";
   String applicationName = "SalesApp";
   String userName = "jdoe";
   String bloxName = "myGrid";
   String visibility = myBookmarksBlox.PRIVATE_VISIBILITY;
   String description = "Bookmark for Q2FY02 West Region Sales";
   Bookmark bk = null;

// (2) Create a Bookmark object called "bk"
   bk = myBookmarksBlox.createBookmark(bookmarkName,
      applicationName, userName, bloxName, visibility,
      myBookmarksBlox.GRID_BLOX_TYPE);

%>
<p>We've got a Bookmark object from BookmarksBlox.createBookmark()!<p/>

<%
// (3) Set the bookmark's description and its GridBlox and DataBlox
```

```
//     properties
   bk.setDescription(description);
   bk.setCustomProperty("Report", "West Region Sales Report");

   BookmarkProperties gridBloxProp =
      bk.createBookmarkProperties(myBookmarksBlox.GRID_BLOX_TYPE);
   gridBloxProp.setProperty("bandingEnabled", true);
   BookmarkProperties dataBloxProp =
      bk.createBookmarkProperties(myBookmarksBlox.DATA_BLOX_TYPE);
   dataBloxProp.setProperty("connectOnStartup", true);
   dataBloxProp.setProperty("query", "!");

// (4) Save the bookmarks to the repository. Must call save() or
//     saveAll() to save the bookmark to the repository.
   bk.saveAll();

%>
Created a bookmark: <%= bookmarkName %><br>
      At binding: <%= bk.getBinding() %>
<%
   bk = null;
%>
```

Example 5: Using server-side bookmarkLoad event filter

This example demonstrates how to use the server-side event filters to perform custom tasks (in this example, we pop up a MessageBox notifying the name of the loaded bookmark) when the bookmarkLoad event is triggered.

1. To use server-side event filters, first add the specific event filter object using the common Blox method addEventFilter().

   ```
   <blox:present id="myPresent">
   ...
   <%
     myPresent.addEventFilter(new LoadFilter() );
   %>
   </blox:present>
   ```

2. Then write your own class that implements the corresponding event filter object (BookmarkLoadFilter) and the corresponding method (bookmarkLoad(BookmarkLoadEvent)) that will be called with the event is triggered. This requires adding the com.alphablox.blox.filter.* package import statement in your JSP.

   ```
   public class LoadFilter implements BookmarkLoadFilter
   {
      public void bookmarkLoad( BookmarkLoadEvent bre )
      {
         //actions to take when the event is triggered
      }
   }
   ```

Here is the code:

```
<%@ page import="com.alphablox.blox.filter.*" %>
<%@ page import="com.alphablox.blox.*" %>
<%@ page import="com.alphablox.blox.repository.Bookmark,
                 com.alphablox.blox.uimodel.core.MessageBox,
                 com.alphablox.blox.uimodel.BloxModel" %>
<%@ taglib uri="bloxtld" prefix="blox"%>

<html>
<head>
  <title>Bookmarks Filter Events</title>
  <!-- Blox header tag -->
  <blox:header/>
</head>
```

```
<body>
<blox:present id="myPresent" >
   <blox:data dataSourceName="QCC-Essbase" query="!"/>
<%
   myPresent.addEventFilter(new LoadFilter(myPresent.getBloxModel()));
%>
</blox:present>
</body>
</html>

<%!
public class LoadFilter implements BookmarkLoadFilter {
   BloxModel model;
   public LoadFilter (BloxModel model) {
      this.model = model;
   }
   public void bookmarkLoad( BookmarkLoadEvent ble ) throws Exception {
      Bookmark bookmark = ble.getBookmark();
      String name = bookmark.getName();
      StringBuffer msg = new StringBuffer("A bookmark called " + name + " is
loaded.");

      MessageBox msgBox = new MessageBox(msg.toString(), "Bookmark Loaded",
MessageBox.MESSAGE_OK, null);
      model.getDispatcher().showDialog(msgBox);
      }
   }
%>
```

Example 6: Getting a bookmark's query when it is loaded

This example demonstrates how to get the textual query stored with a bookmark when a bookmarkLoad event is triggered.

1. Use server-side event filters BookmarkLoadFilter to trigger our custom action when a bookmark is loaded. See "Example 5: Using server-side bookmarkLoad event filter" on page 136 for an example of the server-side event filter. Note that the event filter should be added inside the PresentBlox tag so the filter is only added once rather than each time the page is reloaded:

   ```
   <blox:present id="myPresent" ...>

     <% myPresent.addBookmarkLoadFilter(new LoadFilter()); %>

   </blox:present>
   ```

2. Set the DataBlox's `textualQueryEnabled` property to `true` to apply the textual query when the bookmark is loaded:

   ```
   <blox:present id="myPresent" ...>

     <blox:data
       ...
       textualQueryEnabled="true" />

     <% myPresent.addBookmarkLoadFilter(new LoadFilter()); %>

   </blox:present>
   ```

3. When a bookmark is loaded, get the textual query from the bookmark's SerializedMDBQuery object (for multidimensional data sources) or the SerializedTextualQuery object (for relational data sources). SerializedMDBQuery has a `generateQuery()` method and SerializedTextualQuery has a `getQuery()` method that return the textual query. Note that the `generateQuery()` method only works for IBM DB2 OLAP Server or Hyperion Essbase.

Here is the complete code:

```
<%@ page import="com.alphablox.blox.filter.*,
                  com.alphablox.blox.repository.BookmarkProperties,
                  com.alphablox.blox.repository.SerializedQuery,
                  com.alphablox.blox.repository.SerializedTextualQuery,
                  com.alphablox.blox.repository.SerializedMDBQuery,
                  com.alphablox.blox.repository.Bookmark,
                  com.alphablox.blox.uimodel.core.MessageBox,
                  com.alphablox.blox.uimodel.BloxModel" %>

<%@ taglib uri="bloxtld" prefix="blox"%>
<html>
<head> <title>Bookmarks Filter Events</title>
<blox:header/>

</head>
<body>
<blox:present id="myPresent" width="800" height="600">
  <blox:data dataSourceName="QCC-Essbase"
    query="<ROW (\"All Locations\") Central East West <COLUMN (\"All Time
Periods\") 2001 !"
    useAliases="true"
    textualQueryEnabled="true" />

  <% myPresent.addEventFilter(new LoadFilter(myPresent.getBloxModel())); %>

</blox:present>

</body>
</html>


<%! public class LoadFilter implements BookmarkLoadFilter
{
  BloxModel model;
  public LoadFilter (BloxModel model) {
    this.model = model;
  }

  public void bookmarkLoad( BookmarkLoadEvent ble ) throws Exception
  {
    Bookmark bookmark = ble.getBookmark();
    SerializedQuery sq = bookmark.getSerializedQuery();
    SerializedTextualQuery stq = null;
    SerializedMDBQuery smq = null;
    String query = null;
    if( sq instanceof SerializedTextualQuery )
    {
       stq = (SerializedTextualQuery)sq;
       query = stq.getQuery();
    }
    else if( sq instanceof SerializedMDBQuery )
      {
        smq = (SerializedMDBQuery)sq;
        query  = smq.generateQuery();
      }
     StringBuffer msg = new StringBuffer("query=" + query);

     MessageBox msgBox = new MessageBox(msg.toString(), "Bookmark Event
Filter Message", MessageBox.MESSAGE_OK, null);
     model.getDispatcher().showDialog(msgBox);

  }
}
%>
```

Properties and Methods Cross References

This section lists all unique properties and methods for BookmarksBlox and its associated objects:

BookmarksBlox Properties and Methods Cross References

BookmarksBlox has no unique Blox properties. The following table lists all BookmarksBlox methods for which there are no corresponding properties. For lists of properties and methods common to several Blox, see "Common Blox Properties and Methods by Category" on page 29.

Methods
`bookmarkExists()`
`createBookmark()`
`getBookmark()`
`listBookmarks()`

Bookmark Object Properties and Methods Cross References

The following properties and methods are available on the Bookmark object. To access this object from BookmarksBlox, use the `BookmarksBlox.getBookmark(...)` or `BookmarksBlox.listBookmarks()` methods.

Properties	Methods
`binding`	`getBinding()`
`bloxType`	`getBloxType()`
`bookmarkProperties`	`getBookmarkProperties()`
`container`	`getContainer()`
`customProperties`	`getCustomProperties()` `setCustomProperties()`

description	getDescription() setDescription()
hidden	isHidden() setHidden()
name	getName() setName()
serializedQuery	getSerializedQuery()
userName	getUserName() setUserName()
visibility	getVisibility() setVisibility()
	bookmarkExists()
	clearCustomProperties()
	createBookmarkProperties()
	delete()
	deleteCustomProperty()
	getBookmarkPropertiesByType()
	getCustomProperty() setCustomProperty()
	getCustomPropertyAsBoolean()
	getCustomPropertyAsDouble()
	getCustomPropertyAsInt()
	getCustomPropertyAsLong()
	save()
	saveAll()
	saveSerializedQuery()

BookmarkDescriptor Object Methods Cross References

The following properties and methods are available on the BookmarkDescriptor object. To access this object from BookmarksBlox, use the

BookmarksBlox.modifyBookmark() method.

Methods
getApplicationName() setApplicationName()
getBloxName() setBloxName()
getDescription() setDescription()
getModifyMode() setModifyMode()
getName() setName()
getUserName() setUserName()
getVisibility() setVisibility()
isHidden() setHidden()
isOverwriteable() setOverwriteable()

BookmarkProperties Object Properties and Methods Cross References

The following properties and methods are available on the BookmarkProperties object. To access this object from BookmarksBlox, use the BookmarksBlox.getBookmark(...).getProperties() or BookmarksBlox.getBookmark(...).getPropertiesByType(...) methods.

Properties	Methods
binding	getBinding()
customProperties	getCustomProperties() setCustomProperties()
defaultProperties	getDefaultProperties()
properties	getProperties()
propertiesWithDefaults	getPropertiesWithDefaults()
type	getType()
	clearCustomProperties()
	clearProperties()
	delete()

deleteCustomProperty()

deleteProperty()

getCustomProperty()
getCustomProperty()

getCustomPropertyAsBoolean()
getCustomPropertyAsDouble()
getCustomPropertyAsInt()
getCustomPropertyAsLong()

getProperty()
getPropertyAsBoolean()
getPropertyAsDouble()
getPropertyAsInt()
getPropertyAsLong()

save()

setProperties()

setProperty()

BookmarkMatcher Objects Methods Cross References

The following methods are available on BookmarkMatcherAll, BookmarkMatcherApplications, BookmarkMatcherGroups, and BookmarkMatcherUsers. These objects are part of the com.alphablox.blox.repository package. BookmarkMatcher objects are used to find bookmarks that match the specified criteria via the BookmarksBlox's listBookmarks(BookmarkMatcher *matcher*) method.

BookmarkMatcherAll Methods

Methods
accept()
getApplication() setApplication()
getBloxName() setBloxName()
getUser() setUser()
getVisibility() setVisibility()

BookmarkMatcherApplications Methods

Methods
accept()
getApplication() setApplication()
getVisibility()

BookmarkMatcherGroups Methods

Methods
accept()
getVisibility() setVisibility()

BookmarkMatcherUsers Methods

Methods
accept()
getVisibility()
getUser() setUser()

SerializedMDBQuery Methods Cross References

This section provides cross reference tables for methods related to SerializedMDBQuery and associated objects. To access SerializedMDBQuery and its inner classes from BookmarksBlox, use the BookmarksBlox.getBookmark(...).getSerializedQuery() method. These objects are part of the com.alphablox.blox.repository package.

- "SerializedMDBQuery Methods Cross Reference Table" on page 143
- "SerializedMDBQuery.Axis Inner Class Methods Cross Reference Table" on page 144
- "SerializedMDBQuery.Dimension Inner Class Methods Cross Reference Table" on page 144
- "SerializedMDBQuery.Member Inner Class Methods Cross Reference Table" on page 144
- "SerializedMDBQuery.Tuple Inner Class Methods Cross Reference Table" on page 144

SerializedMDBQuery Methods Cross Reference Table

Methods
generateQuery()
getAxes()
getAxisCount()
getColumnAxis()
getQueryGenerator()
getRowAxis()
getSlicerAxis()
replaceMembers()
save()
update()

SerializedMDBQuery.Axis Inner Class Methods Cross Reference Table

Methods
getDimensionCount()
getDimensions()
getNestedDimensionCount()
getTuple()
getTupleCount()
getTuples()
getType()

SerializedMDBQuery.Dimension Inner Class Methods Cross Reference Table

Methods
getCubeName()
getName()
getType()
getUniqueName()

SerializedMDBQuery.Member Inner Class Methods Cross Reference Table

Methods
getGenerationLevel()
isLeaf()
getName()
getType()
getUniqueName()

SerializedMDBQuery.Tuple Inner Class Methods Cross Reference Table

Methods
getMember()
getMemberCount()
getMembers()

SerializedTextualQuery Methods Cross References

The following methods are available on `SerializedTextualQuery`. To access this object from BookmarksBlox, use the `BookmarksBlox.getBookmark(...).getSerializedQuery()` method. These objects are part of the `com.alphablox.blox.repository` package.

Methods
getQuery()

Methods
save()
setQuery()
update()

BookmarksBlox Properties and Associated Methods

This section describes the properties supported by BookmarksBlox and the methods associated with those properties. The properties are listed alphabetically by property name. For a list of BookmarksBlox methods with which no properties are associated, see "BookmarksBlox Methods" on page 145. For complete descriptions of common Blox properties, see "Properties and Associated Methods Common to Multiple Blox" on page 32.

id

This is a common Blox property. For a complete description, see "id" on page 39.

applicationName

This is a common Blox property. For a complete description, see "applicationName" on page 32.

bloxName

This is a common Blox property. For a complete description, see "bloxName" on page 35.

propertyNames

This is a common Blox property. For a complete description, see "propertyNames" on page 43.

BookmarksBlox Methods

This section describes BookmarksBlox methods that are not associated with a specific property. For the syntax and descriptions of BookmarksBlox methods that have a property associated with them, see "BookmarksBlox Properties and Associated Methods" on page 145.

bookmarkExists()

Checks to see if this Bookmark exists in the repository.

Data Sources

All

Syntax

Java Methods

```
boolean bookmarkExists(Blox blox,
                       String bookmarkName,
                       String visibility);
       //throws ServerBloxException
```

```
boolean bookmarkExists(String bookmarkName,
                       String applicationName,
                       String userName,
                       String bloxName,
                       String visibility);
```

where:

Argument	Description
blox	the Blox used to see if the bookmark exists.
bookmarkName	the name of the bookmark
visibility	the bookmark visibility: i.e. PRIVATE_VISIBILITY, PUBLIC_VISIBILITY, or <*group_name*>
applicationName	The application name for this bookmark
userName	The user name for this bookmark
bloxName	The Blox name for this bookmark

Usage

Returns true if the bookmark exists; false if it does not.

call()

This is a common Blox method. For a complete description, see "call()" on page 51.

createBookmark()

Creates a bookmark for the Blox that is passed in. The bookmark will be created using the information from the Blox. Can also create a bookmark based on the information provided that is not bound to any Blox.

Data Sources

All

Syntax

Java Methods

```
Bookmark createBookmark(Blox blox,
                        String bookmarkName,
                        String visibility);
                    // throws ServerBloxException
Bookmark createBookmark(String bookmarkName,
                        String applicationName,
                        String userName,
                        String bloxName,
                        String visibility,
                        String bloxType);
                   // throws ServerBloxException
```

where:

Argument	Description
blox	the Blox to create the bookmark for
bookmarkName	the name of the bookmark
visibility	the bookmark visibility: i.e. PRIVATE_VISIBILITY, PUBLIC_VISIBILITY, or <*group_name*>

`applicationName`	the application name this bookmark will be used on
`userName`	the user name this bookmark will be used on
`bloxName`	the Blox name this bookmark will be used on
`bloxType`	the Blox type this bookmark will be used on; see "Blox Types and Binding" on page 126 for valid values and examples.

Usage
Returns a Bookmark object representing the bookmark to be created. If the bookmark is to be saved to the repository, `Bookmark.save()` or `Bookmark.saveAll()` must also be called.

Examples
See "Example 4: Creating a bookmark using BookmarksBlox API" on page 135

See Also
"save()" on page 158, "saveAll()" on page 158

flushProperties()
This is a common Blox method. For a complete description, see "flushProperties()" on page 52.

getBookmark()
Finds a single bookmark from the repository using:

- a Blox to supply most of the bookmark information, or
- Strings to supply the bookmark information

Data Sources
All

Syntax
Java Method

```
Bookmark getBookmark(Blox blox,
                     String bookmarkName,
                     String visibility);
            // throws ServerBloxException
Bookmark getBookmark(String bookmarkName,
                     String applicationName,
                     String userName
                     String bloxName,
                     String visibility);
            // throws ServerBloxException
```

where:

Argument	Description
`blox`	the Blox to use for most of the bookmark information
`bookmarkName`	the name of the bookmark to find
`visibility`	the visibility of the bookmark to find
`userName`	the user name for the bookmark to find

`bloxName`	the Blox name for the bookmark to find

Usage

Returns a Bookmark object. Throws `ServerBloxMissingResourceException` if the bookmark was not found. Throws `ServerBloxException` if an unknown problem occurred.

Examples

```
<% Bookmark bookmark = myBookmarksBlox.getBookmark("myBookmark1", "SalesApp",
"Admin", "myPresentBlox", BookmarksBlox.PRIVATE_VISIBILITY); %>
```

The above example gets a private bookmark named “myBookmarksBlox” for a PresentBlox called “myPresentBlox” in the “SalesApp” application.

getProperty()

This is a common Blox method. For a complete description, see “getProperty()” on page 53.

This is a common Blox method. For a complete description, see “getServerContextPath()” on page 53.

listBookmarks()

Retrieves all the bookmarks from the repository. If matching criteria are specified, retrieves all the bookmarks from the repository that match the specified criteria.

Data Sources

All

Syntax

Java Method

```
Bookmark[] listBookmarks();
                  // throws ServerBloxException
Bookmark[] listBookmarks(BookmarkMatcher matcher);
                  // throws ServerBloxException
```

where:

Argument	Description
`matcher`	the BookmarkMatcher objects

Examples

The following code snippet shows how to get all bookmarks from the repository:

```
<% Bookmark bks[] = myBookmarksBlox.listBookmarks(); %>
```

The following code snippet shows how to get the bookmarks owned by the user “admin”:

```
<%
   Bookmark bks[] = null;
   BookmarkMatcherUsers matcher = new BookmarkMatcherUsers();
   matcher.setUser("admin");
   bks = myBookmarksBlox.listBookmarks(matcher);
%>
```

For complete examples, see “Example 2: Getting the properties set for a Bookmark” on page 132 and “Example 3: Getting a list of bookmarks that match the specified criteria” on page 134.

See Also

"BookmarkMatcherUsers Methods" on page 179, "BookmarkMatcherApplications Methods" on page 176, "BookmarkMatcherGroups Methods" on page 178

modifyBookmark()

Modify a bookmark object and save it to the repository.

Data Sources

All

Syntax

Java Method

```
Bookmark modifyBookmark(Bookmark bookmark,
           BookmarksBlox.BookmarkDescriptor newDescriptor);
```

Argument	Description
`bookmark`	The Bookmark object to modify.
`ndwDescriptor`	The BookmarkDescriptor object containing the Bookmark's new property values.

Usage

This is a convenient method that allows you to modify a bookmark's name, visibility, owner, description, and associated Blox, or to specify if the bookmark should be hidden from the bookmark UI. This cannot be used to modify bookmarks that involve an explicit DataBlox.

This method returns the modified bookmark object while the changes to the bookmark are saved to the repository automatically.

setInitialProperty()

This is a common Blox method. For a complete description, see "setInitialProperty()" on page 60.

setProperty()

This is a common Blox method. For a complete description, see "setProperty()" on page 61.

Bookmark Object Properties and Associated Methods

This section describes the properties supported by the Bookmark object and the methods associated with those properties. The properties are listed alphabetically by property name. For a list of Bookmark object methods with which no properties are associated, see "Bookmark Object Methods" on page 153.

To access this object from BookmarksBlox, use the `BookmarksBlox.getBookmark(...)` or `BookmarksBlox.listBookmarks()` methods. To use any method for this object, import the `com.alphablox.blox.repository` package in your JSP.

applicationName

This is a common Blox property. For a complete description, see "applicationName" on page 32.

binding

Gets the JNDI binding String used to get the Blox's bookmark properties from the repository.

Data Sources

All

Syntax

Java Method

```
String getBinding(); //throws ServerBloxException
```

Usage

If the user "jdoe" saved a private bookmark on a PresentBlox called "salesPresent" in the application called "SalesApp" and named the bookmark "q1fy03data", using the `getBinding()` method on the Bookmark object, you will get a String like the following:

```
users/jdoe/SalesApp/salesPresent/bookmark/q1fy03data/properties
```

Examples

See "Example 3: Getting a list of bookmarks that match the specified criteria" on page 134.

bloxName

This is a common Blox property. For a complete description, see "bloxName" on page 35.

bloxType

The type of Blox the bookmark is saved on.

Data Sources

All

Syntax

Java Methods

```
String getBloxType();
```

Usage

The result from this method should be evaluated against the Bookmark object's static fields for Blox type.

Examples

The following code snippet shows how to find out the visibility of an instance of the Bookmark object called `bookmark`:

```
String visibility = bookmark.getVisibility();
if (visibility.equals(bookmark.PRIVATE_VISIBILITY)) {
   //This is a private bookmark
} else if (visibility.equals(bookmark.PUBLIC_VISIBILITY)) {
   //This is a public bookmark
} else {
   //This is a group bookmark
}
```

bookmarkProperties

The BookmarkProperties object representing the property set of a Blox within a Bookmark.

Data Sources
All

Syntax
Java Methods

```
BookmarkProperties[] getBookmarkProperties();
```

See Also
"customProperties" on page 151, "createBookmarkProperties()" on page 154, "getBookmarkPropertiesByType()" on page 155

container
The container in which the bookmarks are stored.

Data Sources
All

Syntax
Java Method

```
String getContainer();   // throws ServerBloxException
```

customProperties
The CustomBookmarkProperties object representing the custom property set of a Blox within a Bookmark.

Data Sources
All

Syntax
Java Methods

```
HashMap getCustomProperties();
void setCustomProperties(HashMap map,
                         boolean clearFirst);
```

where:

Argument	Description
`map`	a Hashtable containing all the custom properties to set
`clearFirst`	If `true`, the Blox's existing custom bookmark properties will be removed and the new properties added. If `false`, the existing Blox's custom bookmark properties will be untouched unless the new properties have duplicate keys.

See Also
"bookmarkProperties" on page 150

description
The description associated with the Bookmark.

Data Sources
All

Syntax

Java Methods

```
String getDescription();
void setDescription(String description);
```

Usage

This description can be provided by users when they save a bookmark using the Blox user interface or via the `setDescription()` method.

hidden

Whether the bookmark is hidden.

Data Sources

All

Syntax

Java Methods

```
boolean isHidden();
void setHidden(boolean hidden);
```

where:

Argument	Default	Description
`hidden`	`false`	Whether the bookmark should be hidden from the Bookmark user interface

Usage

"Hidden" bookmarks are not visible in the user interface. They do not show up in the bookmark drop list accessible from the right-click menu or the Toolbar's bookmark button. Other than that, you can access and manipulate hidden bookmarks the same way as regular bookmarks using the BookmarksBlox API.

name

Name of the bookmark.

Data Sources

All

Syntax

Java Methods

```
String getName();
void setName();
```

Usage

Bookmark names can only contain the following characters: A-Z, a-z, 0-9, and underscore (_).

serializedQuery

The query for the bookmark as a SerializedQuery object.

Data Sources

Multidimensional

Syntax

Java Method

```
SerializedQuery getSerializedQuery();
                        // throws ServerBloxException
```

See Also

"SerializedMDBQuery Methods Cross References" on page 143,
"SerializedTextualQuery Methods Cross References" on page 144

userName

Owner of the bookmark.

Data Sources

All

Syntax

Java Methods

```
String getUserName();
void setUserName(String userName);
```

where:

Argument	Default	Description
`userName`	none	The user who owns the bookmark.

visibility

The visibility of the bookmark.

Data Sources

All

Syntax

Java Methods

```
String getVisibility();
void setVisibility(String visibility);
```

where:

Argument	Default	Description
`visibility`		Valid values are: • `PRIVATE_VISIBILITY` • `PUBLIC_VISIBILITY`

Examples

See "Example 2: Getting the properties set for a Bookmark" on page 132

Bookmark Object Methods

This section describes Bookmark object methods that are not associated with a specific property. For the syntax and descriptions of methods that have a property associated with them, see "Bookmark Object Properties and Associated Methods" on page 149.

To access this object from BookmarksBlox, use the `BookmarksBlox.getBookmark(...)` or `BookmarksBlox.listBookmarks()` methods. To use any method for this object, import the `com.alphablox.blox.repository` package in your JSP.

bookmarkExists()

Checks to see if this Bookmark exists in the repository.

Data Sources

All

Syntax

Java Method

```
boolean bookmarkExists();
```

Usage

Returns `true` if the bookmark exists; `false` if not.

See Also

"bookmarkExists()" on page 145

clearCustomProperties()

Clears the custom properties.

Data Sources

All

Syntax

Java Method

```
HashMap clearCustomProperties();
```

Usage

Returns all the bookmark properties for the bookmark as a Hash table

createBookmarkProperties()

Creates a BookmarkProperties object for the specified Blox type.

Data Sources

All

Syntax

Java Method

```
BookmarkProperties createBookmarkProperties(String bloxType);
          // throws ServerBloxException
```

where:

Argument	Description
`bloxType`	Valid Blox types are: • `CHART_BLOX_TYPE` • `DATA_BLOX_TYPE` • `DATALAYOUT_BLOX_TYPE` • `GRID_BLOX_TYPE` • `PAGE_BLOX_TYPE`

- PRESENT_BLOX_TYPE
- TOOLBAR_BLOX_TYPE

delete()

Deletes a Bookmark.

Data Sources
All

Syntax
Java Method

```
void delete();  //throws ServerBloxException
```

deleteCustomProperty()

Deletes the custom bookmark property based upon the key specified.

Data Sources
All

Syntax
Java Method

```
String deleteCustomProperty(String key);
```

where:

Argument	Description
key	The key used to look up the property in the set of properties

getBookmarkPropertiesByType()

Gets a Bookmark's property set by Blox type.

Data Sources
All

Syntax
Java Method

```
BookmarkProperties getBookmarkPropertiesByType(String bloxType);
```

where:

Argument	Description
bloxType	Valid Blox types are: • CHART_BLOX_TYPE • DATA_BLOX_TYPE • DATALAYOUT_BLOX_TYPE • GRID_BLOX_TYPE • PAGE_BLOX_TYPE • PRESENT_BLOX_TYPE • TOOLBAR_BLOX_TYPE

Examples

The following example gets the DataBlox related bookmark property from a `Bookmark` object into a `BookmarkProperties` object, and then resets the dimension on the page axis to null. When the bookmark is loaded, there will be no dimension, overwriting what was originally saved with the bookmark.

```
<%
Bookmark bookmark = myBookmarksBlox.getBookmark();
BookmarkProperties props =
bookmark.getBookmarkPropertiesByType(bookmark.DATA_BLOX_TYPE);
props.setProperty("dimensionsOnPageAxis", bookmark.NULL_DIMENSION);
%>
```

getCustomProperties()

Gets the custom properties.

Data Sources

All

Syntax

Java Method

```
java.util.HashMap getCustomProperties();
```

getCustomProperty()

Gets a particular custom bookmark property as a String.

Data Sources

All

Syntax

Java Methods

```
String getCustomProperty(String key);
```

where:

Argument	Description
`key`	The key used to look up the property in the set of properties.

getCustomPropertyAsBoolean()

Gets a particular custom bookmark property as a Boolean.

Data Sources

All

Syntax

Java Method

```
boolean getCustomPropertyAsInt(String key, boolean defaultValue);
```

where:

Argument	Description
`key`	The key used to look up the property in the set of properties
`defaultValue`	The bookmark property to get

See Also

"getCustomPropertyAsDouble()" on page 157, "getCustomPropertyAsInt()" on page 157, "getCustomPropertyAsLong()" on page 157, "getCustomProperty()" on page 156

getCustomPropertyAsDouble()

Gets a particular custom bookmark property as a Double.

Data Sources

All

Syntax

Java Method

```
double getCustomPropertyAsInt(String key, double defaultValue);
```

where:

Argument	Description
`key`	The key used to look up the property in the set of properties
`defaultValue`	The bookmark property to get

See Also

"getCustomPropertyAsBoolean()" on page 156, "getCustomPropertyAsInt()" on page 157, "getCustomPropertyAsLong()" on page 157, "getCustomProperty()" on page 156

getCustomPropertyAsInt()

Gets a particular custom bookmark property as an Integer.

Data Sources

All

Syntax

Java Method

```
int getCustomPropertyAsInt(String key, int defaultValue);
```

where:

Argument	Description
`key`	The key used to look up the property in the set of properties
`defaultValue`	The bookmark property to get

See Also

"getCustomPropertyAsBoolean()" on page 156, "getCustomPropertyAsDouble()" on page 157, "getCustomPropertyAsLong()" on page 157, "getCustomProperty()" on page 156

getCustomPropertyAsLong()

Gets a particular custom bookmark property as a Long.

Data Sources

All

Syntax

Java Method

```
long getCustomPropertyAsLong(String key, long defaultValue);
```

where:

Argument	Description
`key`	The key used to look up the property in the set of properties
`defaultValue`	The bookmark property to get

See Also

"getCustomPropertyAsBoolean()" on page 156, "getCustomPropertyAsDouble()" on page 157, "getCustomPropertyAsInt()" on page 157, "getCustomProperty()" on page 156

save()

Saves the bookmark to the repository.

Data Sources

All

Syntax

Java Method

```
void save();  // throws ServerBloxException
```

Usage

The bookmark's individual properties or serialized query is not saved with this method.

See Also

"saveAll()" on page 158

saveAll()

Saves all bookmarks to the repository.

Data Sources

All

Syntax

Java Method

```
void saveAll();    // throws ServerBloxException
```

Usage

The bookmark's properties and serialized query are also saved to the repository.

Examples

See "Example 4: Creating a bookmark using BookmarksBlox API" on page 135.

saveSerializedQuery()

Saves the query associated with this bookmark as a serialized query.

Data Sources

All

Syntax

Java Method

```
void saveSerializedQuery();
               // throws RepositoryIOException
```

See Also

"Serialized Query and Textual Query" on page 129, "textualQueryEnabled" on page 381

setCustomProperty()

Sets a particular custom bookmark property as a key-value pair.

Data Sources

All

Syntax

Java Method

```
String setCustomProperty(String key, String property);
String setCustomProperty(String key, int value);
String setCustomProperty(String key, double value);
String setCustomProperty(String key, long value);
String setCustomProperty(String key, boolean value);
```

where:

Argument	Description
`key`	The key used to look up the property in the set of properties
`property`	The bookmark property string
`value`	The value for the bookmark property

Examples

In the following code example, a custom bookmark property called "StandardStyle" with a value of "Finance" is created for the public bookmark called "finance_bookmark."

```
<%
Bookmark bookmark = myBookmarkBlox.getBookmark("finance_bookmark",
"FinanceApp", "admin", "myPresent", Bookmark.PUBLIC_VISIBILITY);
BookmarkProperties gridProps =
bookmark.getBookmarkPropertiesByType(Bookmark.GRID_BLOX_TYPE);

dataProps.setCustomProperty("StandardStyle", "Finance");
%>
```

BookmarkDescriptor Object Methods

This section describes methods for the BookmarkDescriptor class. The BookmarkDescriptor object describes the changes to a bookmark. It is used in the BookmarksBlox's `modifyBookmark()` method.

To use any method for this object, import the `com.alphablox.blox.repository` package in your JSP.

getApplicationName()

Gets the name of the application associated with the bookmark.

Data Sources
All

Syntax
Java Method

```
String getApplicationName();
```

See Also
"setApplicationName()" on page 162

getBloxName()
Gets the name of the Blox associated with the bookmark.

Data Sources
All

Syntax
Java Method

```
String getBloxName();
```

See Also
"setBloxName()" on page 162

getDescription()
Gets the description stored with the bookmark.

Data Sources
All

Syntax
Java Method

```
String getDescription();
```

See Also
"setDescription()" on page 162

getModifyMode()
Gets the modify mode of the bookmark.

Data Sources
All

Syntax
Java Method

```
short getModifyMode();
```

Usage
Returned result should be evaluated against the constants: `BookmarkDescriptor.MODE_COPY` and `BookmarkDescriptor.MODE_MOVE`.

See Also
"setModifyMode()" on page 163

getName()

Gets the name of the bookmark.

Data Sources
All

Syntax
Java Method

```
String getName();
```

See Also
"setName()" on page 163

getUserName()

Gets the owner associated with a private bookmark.

Data Sources
All

Syntax
Java Method

```
String getUserName();
```

getVisibility()

Gets the visibility of the bookmark.

Data Sources
All

Syntax
Java Method

```
String getVisibility();
```

Usage
The returned string should be evaluated against the following constants to find out the visibility: `PUBLIC_VISIBILITY`, `PRIVATE_VISIBILITY`, or the name of the group.

isHidden()

Identifies if a bookmark is hidden from the bookmark user interface.

Data Sources
All

Syntax
Java Method

```
Boolean isHidden();
```

See Also
"setHidden()" on page 163

isOverwriteable()

Identifies if saving this modified bookmark can overwrite an existing bookmark in a different location.

Data Sources
All

Syntax
Java Method

```
boolean isOverwriteable();
```

See Also
"setOverwriteable()" on page 164

setApplicationName()
Specifies the name of the application associated with the bookmark.

Data Sources
All

Syntax
Java Method

```
void setApplicationName(String applicationName);
```

where:

Argument	Description
applicationName	Name of the application.

See Also
"getApplicationName()" on page 159

setBloxName()
Sets the Blox associated with the bookmark.

Data Sources
All

Syntax
Java Method

```
void setBloxName(String bloxName);
```

where:

Argument	Description
bloxName	Name of the Blox.

See Also
"getBloxName()" on page 160

setDescription()
Sets the description associated with the bookmark.

Data Sources
All

Syntax
Java Method

```
voide setDescription(String description);
```

where:

Argument	Description
`description`	Description of the bookmark.

See Also
"getDescription()" on page 160

setHidden()
Sets the bookmark to be hidden from the user interface.

Data Sources
All

Syntax
Java Method

```
voide setHidden(Boolean hidden);
```

where:

Argument	Description
`hidden`	`true` — the bookmark is hidden from the user interface.

See Also
"isHidden()" on page 161

setModifyMode()
Specifies the mode of bookmark modification.

Data Sources
All

Syntax
Java Method

```
String setModifyMode(short modifyMode);
```

where:

Argument	Description
`modifyMode`	One of the two modes: `BookmarkDescriptor.MODE_COPY` and `BookmarkDescriptor.MODE_MOVE`.

See Also
"getModifyMode()" on page 160

setName()
Sets the name of the bookmark.

Data Sources
All

Syntax
Java Method

```
voide setName(String name);
```

where:

Argument	Description
name	The name of the bookmark.

See Also
"getName()" on page 161

setOverwriteable()
Specifies whether saving this modified bookmark can overwrite an existing bookmark in a different location.

Data Sources
All

Syntax
Java Method

```
void setOverwriteable(boolean overwriteable);
```

where:

Argument	Description
overwriteable	true —sets the bookmark to be overwriteable.

See Also
"isOverwriteable()" on page 161

setUserName()
Sets the bookmark owner's name.

Data Sources
All

Syntax
Java Method

```
void setUserName(String userName);
```

See Also
"getUserName()" on page 161

setVisibility()
Sets the bookmark's visibility.

Data Sources
All

Syntax

Java Method

```
void setVisibility(String visibility);
```

where:

Argument	Description
`visibility`	The visibility of the bookmark: `PRIVATE_VISIBILITY`, `PUBLIC_VISIBILITY`, or group name.

See Also

"getVisibility()" on page 161

BookmarkProperties Object Properties and Associated Methods

This section describes the properties supported by the BookmarkProperties object and the methods associated with those properties. The properties are listed alphabetically by property name. For a list of BookmarkProperties object methods with which no properties are associated, see "BookmarkProperties Methods" on page 167.

To access this object from BookmarksBlox, use the `BookmarksBlox.getBookmark(...).getProperties()` or `BookmarksBlox.getBookmark(...).getPropertiesByType(...)` methods. To use any method for this object, import the `com.alphablox.blox.repository` package in your JSP.

binding

Gets the JNDI binding String used to get the Blox's bookmark properties from the repository.

Data Sources

All

Syntax

Java Method

```
String getBinding();
```

customProperties

Gets a list of the custom properties defined by application developers.

Data Sources

All

Syntax

Java Method

```
HashMap getCustomProperties();

void setCustomProperties(HashMap map,
                         boolean clearFirst);
```

where:

Argument	Description

`map`	a Hashtable containing all the custom properties to set
`clearFirst`	If `true`, the Blox's existing custom bookmark properties will be removed then the new properties added. If `false`, the existing Blox's custom bookmark properties will be untouched unless the new properties have duplicate keys.

Examples

The following code snippet shows how to get the DataBlox properties from a Bookmark object and then get the custom DataBlox properties. The returned HashMap contains all the custom DataBlox properties and their values.

```
<%
Bookmark bookmark =
   myBookmarksBlox.getBookmark("mySalesBookmark", "SalesApp",
   "Admin", "myPresent", Bookmark.PRIVATE_VISIBILITY);
BookmarkProperties dataProps =
   bookmark.getBookmarkPropertiesByType(Bookmark.DATA_BLOX_TYPE);
HashMap defaults = dataProps.getCustomProperties();
...
%>
```

defaultProperties

Gets a list of the default properties for the Blox type specified. Custom properties are not included.

Data Sources

All

Syntax

Java Method

```
HashMap getDefaultProperties();
```

Examples

The following code snippet shows how to get the DataBlox properties from a Bookmark object and then get the default DataBlox properties. The returned HashMap contains all the DataBlox properties and their default values.

```
<%
Bookmark bookmark =
   myBookmarksBlox.getBookmark("mySalesBookmark", "SalesApp",
   "Admin", "myPresent", Bookmark.PRIVATE_VISIBILITY);
BookmarkProperties dataProps =
   bookmark.getBookmarkPropertiesByType(Bookmark.DATA_BLOX_TYPE);
HashMap defaults = dataProps.getDefaultProperties();
...
%>
```

properties

Gets the properties and their values for the specified Blox type as saved in the bookmark. The properties returned do not include the custom properties.

Data Sources

All

Syntax
Java Method

```
HashMap getProperties();
```

Usage
Returns an empty HashMap if no properties exist.

Examples
The following code snippet shows how to get the DataBlox properties from a Bookmark object and then get the DataBlox properties saved in the bookmark. The returned HashMap contains all the DataBlox properties and their default values. In this case, the properties and their values returned are extracted from the bookmark's *bookmarkName*.data file.

```
<%
Bookmark bookmark =
   myBookmarksBlox.getBookmark("mySalesBookmark", "SalesApp",
   "Admin", "myPresent", Bookmark.PRIVATE_VISIBILITY);
BookmarkProperties dataProps =
   bookmark.getBookmarkPropertiesByType(Bookmark.DATA_BLOX_TYPE);
HashMap defaults = dataProps.getProperties();
...
%>
```

propertiesWithDefaults
Gets just the Blox's bookmark properties (not the custom properties).

Data Sources
All

Syntax
Java Method

```
HashMap getPropertiesWithDefaults();
```

Usage
This returns an empty HashMap if no properties exist.

type

Get the type of Blox this bookmark property belongs to. This value can be compared against the com.alphablox.blox.Bookmark.CHART_BLOX_TYPE, DATA_BLOX_TYPE, DATALAYOUT_BLOX_TYPE, GRID_BLOX_TYPE,PAGE_BLOX_TYPE, PRESENT_BLOX_TYPE, and UNKNOWN_BLOX_TYPE Strings.

Data Sources
All

Syntax
Java Method

```
String getType();
```

BookmarkProperties Methods

This section describes BookmarkProperties object methods that are not associated with a specific property. For the syntax and descriptions of methods that have a property associated with them, see "BookmarkProperties Object Properties and Associated Methods" on page 165.

To access this object from BookmarksBlox, use the `BookmarksBlox.getBookmark(...).getProperties()` or `BookmarksBlox.getBookmark(...).getPropertiesByType(...)` methods. To use any method for this object, import the `com.alphablox.blox.repository` package in your JSP.

clearCustomProperties()

Clears all the Blox's custom bookmark properties.

Data Sources

All

Syntax

Java Method

```
HashMap clearCustomProperties();
```

Usage

Returns all the bookmark properties for the bookmark as a Hash table

clearProperties()

Clears all the Blox's bookmark properties (not custom properties).

Data Sources

All

Syntax

Java Method

```
HashMap clearProperties();
```

Examples

Returns all the bookmark properties for the bookmark as a Hash table.

delete()

Deletes a BookmarkProperties object.

Data Sources

All

Syntax

Java Method

```
void delete();     //throws ServerBloxException
```

deleteCustomProperty()

Deletes the custom property that matches the property key.

Data Sources

All

Syntax

Java Method

```
String deleteCustomProperty(String key);
```

where:

Argument	Description

`key`	The key used to look up the property in the set of properties

deleteProperty()

Deletes a property from a Blox's bookmark properties (not in custom properties) list.

Data Sources

All

Syntax

Java Method

```
String deleteProperty(String key);
```

where:

Argument	Description
`key`	The property to remove

Usage

Returns the value of the property removed or null if it did not exist.

getCustomProperty()

Gets a particular custom bookmark property as a String.

Data Sources

All

Syntax

Java Method

```
String getCustomProperty(String key);
```

where:

Argument	Description
`key`	The key used to look up the custom bookmark property in the set of properties

getCustomPropertyAsBoolean()

Gets a particular custom bookmark property as a Boolean.

Data Sources

All

Syntax

Java Method

```
boolean getCustomPropertyAsBoolean(String key, boolean defaultValue);
```

where:

Argument	Description
`key`	The key used to look up the property in the set of properties

`defaultValue`	The default value to return if the specified property is not found

getCustomPropertyAsDouble()

Gets a particular custom bookmark property as a Double.

Data Sources

All

Syntax

Java Method

```
double getCustomPropertyAsDouble(String key,
                                 double defaultValue);
```

where:

Argument	Description
`key`	The key used to look up the property in the set of properties
`defaultValue`	The default value to return if the specified property is not found

getCustomPropertyAsInt()

Gets a particular custom bookmark property as an Integer.

Data Sources

All

Syntax

Java Method

```
int getCustomPropertyAsInt(String key, int defaultValue);
```

where:

Argument	Description
`key`	The key used to look up the property in the set of properties
`defaultValue`	The default value to return if the specified property is not found

getCustomPropertyAsLong()

Gets a particular Blox's custom bookmark property as a Long.

Data Sources

All

Syntax

Java Method

```
long getCustomPropertyAsLong(String key, long defaultValue);
```

where:

Argument	Description

`key`	The key used to look up the property in the set of properties
`defaultValue`	The default value to return if the specified property is not found

getProperty()

Gets a particular Blox's bookmark property as a String.

Data Sources

All

Syntax

Java Method

```
String getProperty(String key);
```

where:

Argument	Description
`key`	The key used to look up the property in the set of properties

Usage

Returns the value of the updated property; null if it did not exist.

getPropertyAsBoolean()

Gets a particular Blox's bookmark property (not in custom properties) as a Boolean.

Data Sources

All

Syntax

Java Method

```
boolean getPropertyAsBoolean(String key, boolean defaultValue);
```

where:

Argument	Description
`key`	The key used to look up the property in the set of properties
`defaultValue`	The default value to return if the specified property is not found

Usage

Returns the default value if the property does not exist.

getPropertyAsDouble()

Gets a particular Blox's bookmark property (not in custom properties) as a Double.

Data Sources

All

Syntax

Java Method

```
double getPropertyAsDouble(String key, double defaultValue);
                     //throws InvalidBloxPropertyValueException
```

where:

Argument	Description
key	The key used to look up the property in the set of properties
defaultValue	The default value to return if the specified property is not found

Usage

Returns the default value if the property does not exist.

getPropertyAsInt()

Gets a particular Blox's bookmark property (not in custom properties) as an integer.

Data Sources

All

Syntax

Java Method

```
int getPropertyAsInt(String key,
                     int defaultValue);
              throws InvalidBloxPropertyValueException
```

where:

Argument	Description
key	The key used to look up the property in the set of properties
defaultValue	The default value to return if the specified property is not found

Usage

This returns the default value if the property does not exist

getPropertyAsLong()

Gets a particular Blox's bookmark property (not in custom properties) as a Long.

Data Sources

All

Syntax

Java Method

```
long getPropertyAsLong(String key, long defaultValue);
                      // throws InvalidBloxPropertyValueException
```

where:

Argument	Description
key	The key used to look up the property in the set of properties

`defaultValue`	The default value to return if the specified property is not found

Usage
Returns the default value if the property does not exist.

save()
Saves the Blox bookmark properties (including custom properties) to the repository.

Data Sources
All

Syntax
Java Method

```
void save();   // throws ServerBloxException
```

Usage
To save an entire bookmark's properties, use `Bookmark.save()`.

See Also
"save()" on page 158

setProperties()
Sets a particular Blox's bookmark properties (not in custom properties) from a Hashtable

Data Sources
All

Syntax
Java Method

```
HashMap setProperties(HashMap map, boolean clearFirst);
```

where:

Argument	Description
`map`	a Hashtable containing all the properties to set
`clearFirst`	If `true`, the Blox's existing bookmark properties will be removed then the new properties added. If `false`, the existing Blox's bookmark properties will be untouched unless the new properties has duplicate keys.

setProperty()
Sets a particular Blox's bookmark property (not in custom properties).

Data Sources
All

Syntax
Java Method

```
String setProperty(String key, String property);
String setProperty(String key, int value);
String setProperty(String key, double value)
String setProperty(String key, long value);
String setProperty(String key, boolean value);
```

where:

Argument	Description
key	The key used to look up the property in the set of properties
property	The bookmark property to save
value	The value to save for the *key*

Usage

Returns the value of the updated property; null if it did not exist.

BookmarkMatcherAll Methods

This section describes all methods associated with the BookmarkMatcherAll object. To use any method for this object, import the com.alphablox.blox.repository package in your JSP.

accept()

Identifies if this bookmark should be included in the list of bookmarks returned.

Data Sources

All

Syntax

Java Method

```
boolean accept(Bookmark bookmark);
                   // throws ServerBloxException
```

where:

Argument	Description
bookmark	A bookmark object.

Usage

Returns true if the bookmark should be added to the list of bookmarks returned.

getApplication()

Gets the application name that was set using setApplication().

Data Sources

All

Syntax

Java Method

```
String getApplication();
```

getBloxName()

Gets the name of the Blox that was set using setBloxName().

Data Sources
All

Syntax
Java Method

```
String getBloxName();
```

getUser()

Gets the user name that was set using setUser().

Data Sources
All

Syntax
Java Method

```
String getUser();
```

getVisibility()

Gets the visibility for a user bookmark.

Data Sources
All

Syntax
Java Method

```
String getVisibility();
          //throws ServerBloxException
```

Usage
This should always return `PRIVATE_VISIBILITY` for a user bookmark. See "Bookmark Visibility" on page 126 for the value of the static fields.

setApplication()

Sets the name of the application to find all the bookmarks for.

Data Sources
All

Syntax
Java Method

```
void setApplication(String application);
```

where:

Argument	Description
`application`	The name of the application

Usage
If the application name is not set, when searching for bookmarks (usinglistBookmarks(), for example), bookmarks for all applications will be returned.

setBloxName()
Sets the name of the Blox to find all the bookmarks for.

Data Sources
All

Syntax
Java Method

```
void setBloxName(String bloxName);
```

where:

Argument	Description
bloxName	The name of the Blox.

setUser()
Sets the name of the user to find all the bookmarks for.

Data Sources
All

Syntax
Java Method

```
void setUser();
```

Usage
If this is not specified, bookmarks for all users will be returned.

setVisibility()
Sets the name of the group to find all the bookmarks for.

Data Sources
All

Syntax
Java Method

```
void setVisibility(String visibility);
          //throws ServerBloxException
```

where:

Argument	Description
visibility	PUBLIC_VISIBILITY or PRIVATE_VISIBILITY.

BookmarkMatcherApplications Methods
This section describes all methods associated with the BookmarkMatcherApplications object. To use any method for this object, import the com.alphablox.blox.repository package in your JSP.

accept()

Identifies if this bookmark matches the criteria for this application.

Data Sources

All

Syntax

Java Method

```
boolean accept(Bookmark bookmark);
                   // throws ServerBloxException
```

where:

Argument	Description
`bookmark`	A bookmark object.

Usage

Returns `true` if the bookmark is added to the list of bookmarks returned by `BookmarksBlox.listUsers(BookmarkMatcher)`. If this returns `false`, it does not.

getApplication()

Gets the application name that was set using setApplication().

Data Sources

All

Syntax

Java Method

```
String getApplication();
```

setApplication()

Sets the name of the application to find all the bookmarks for.

Data Sources

All

Syntax

Java Method

```
void setApplication(String application);
```

where:

Argument	Description
`application`	The name of the application

Usage

If the application name is not set, when searching for bookmarks (usinglistBookmarks(), for example), bookmarks for all applications will be returned.

getVisibility()

Gets the visibility for an application's bookmark.

Data Sources
All

Syntax
Java Method

```
String getVisibility();
          //throws ServerBloxException
```

BookmarkMatcherGroups Methods

This section describes all methods associated with the BookmarkMatcherGroups object. To use any method for this object, import the `com.alphablox.blox.repository` package in your JSP.

accept()

Identifies if this bookmark matches the criteria for this application.

Data Sources
All

Syntax
Java Method

```
boolean accept(Bookmark bookmark);
                   // throws ServerBloxException
```

where:

Argument	Description
`bookmark`	A bookmark object.

Usage
Returns `true` if the bookmark is added to the list of bookmarks returned by `BookmarksBlox.listUsers(BookmarkMatcher)`. If this returns `false`, it does not.

getVisibility()

Gets the visibility for a group bookmark.

Data Sources
All

Syntax
Java Method

```
String getVisibility();
          //throws ServerBloxException
```

setVisibility()

Sets the visibility of the bookmark.

Data Sources
All

Syntax
Java Method

```
void setVisibility(String visibility);
                        // throws ServerBloxException
```

where:

Argument	Description
`visibility`	`PUBLIC_VISIBILITY` or `PRIVATE_VISIBILITY`.

BookmarkMatcherUsers Methods

This section describes all methods associated with the BookmarkMatcherUsers object. To use any method for this object, import the `com.alphablox.blox.repository` package in your JSP.

accept()

Identifies if this bookmark matches the criteria for this application.

Data Sources

All

Syntax

Java Method

```
boolean accept(Bookmark bookmark);
                    // throws ServerBloxException
```

where:

Argument	Description
`bookmark`	A bookmark object.

Usage

Returns `true` if the bookmark is added to the list of bookmarks returned by `BookmarksBlox.listUsers(BookmarkMatcher)`. If this returns `false`, it does not.

getVisibility()

Gets the visibility for a user's bookmark.

Data Sources

All

Syntax

Java Method

```
String getVisibility();
           //throws ServerBloxException
```

getUser()

Gets the user name that was set using setUser().

Data Sources

All

Syntax

Java Method

```
String getUser();
```

setUser()

Sets the name of the user to find all the bookmarks for.

Data Sources
All

Syntax
Java Method

```
void setUser(String user);
```

where:

Argument	Description
user	The username to find all the bookmarks for.

Usage
If the user is not set, bookmarks for all users will be returned.

EssbaseReportSpec Methods

This section describes all methods associated with the EssbaseReportSpec object. To access this object from BookmarksBlox, use the BookmarksBlox.getBookmark(...).getSerializedQuery().getEssbaseReportSpec() method. To use any method for this object, import the com.alphablox.blox.repository package in your JSP.

generateColumnSpec()

Generates the IBM DB2 OLAP Server or Hyperion Essbase column report specification for the current query.

Data Sources
IBM DB2 OLAP Server, Hyperion Essbase

Syntax
Java Method

```
String generateColumnSpec();   // throws ServerBloxException;
```

generatePageSpec()

Generates the IBM DB2 OLAP Server or Hyperion Essbase page report specification for the current query.

Data Sources
IBM DB2 OLAP Server, Hyperion Essbase

Syntax
Java Method

```
String generatePapgeSpec();   // throws ServerBloxException;
```

generateQuery()

Generates the IBM DB2 OLAP Server or Hyperion Essbase report specification for the current query.

Data Sources
IBM DB2 OLAP Server, Hyperion Essbase

Syntax
Java Method

```
String generateQuery();   // throws ServerBloxException;
```

generateRowSpec()

Generates the IBM DB2 OLAP Server or Hyperion Essbase row report specification for the current query.

Data Sources
IBM DB2 OLAP Server, Hyperion Essbase

Syntax
Java Method

```
String generateRowSpec();   // throws ServerBloxException;
```

isAllowAsymCols()

Identifies if the columns portion of the report spec should be allowed to be asymmetric.

Data Sources
IBM DB2 OLAP Server, Hyperion Essbase

Syntax
Java Method

```
boolean isAllowAsymCols();
```

Usage
The default is `true`.

See Also
"setAllowAsymCols()" on page 181

isAllowAsymRows()

Identifies if the rows portion of the report spec should be allowed to be asymmetric.

Data Sources
IBM DB2 OLAP Server, Hyperion Essbase

Syntax
Java Method

```
boolean setAllowAsymRows();
```

Usage
The default is `true`.

See Also
"setAllowAsymRows()" on page 182

setAllowAsymCols()

Allows the column portion of the report spec to be the result of an asymmetric query.

Data Sources
IBM DB2 OLAP Server, Hyperion Essbase

Syntax
Java Method

```
void setAllowAsymCols(boolean allowAsymCols);
```

where:

Argument	Description
`allowAsymCols`	`true` — columns will be the result of an asymmetric query. The default is `true`.

See Also
"isAllowAsymCols()" on page 181

setAllowAsymRows()
Allows the rows portion of the report spec to be the result of an asymmetric query.

Data Sources
IBM DB2 OLAP Server, Hyperion Essbase

Syntax
Java Method

```
void setAllowAsymRows(boolean allowAsymRows);
```

where:

Argument	Description
`allowAsymRows`	`true` — rows will be the result of an asymmetric query. The default is `true`.

See Also
"isAllowAsymRows()" on page 181

SerializedMDBQuery Methods
This section describes all methods associated with the `SerializedMDBQuery` object. This object gives you access to the serialized query stored in the `<bookmark_name>.query` file for multidimensional data sources. To access this object from BookmarksBlox, use the `BookmarksBlox.getBookmark(...).getSerializedQuery()` method. To use any method for this object, import the `com.alphablox.blox.repository` package in your JSP.

generateQuery()
Returns an IBM DB2 OLAP Server or Hyperion Essbase report spec query string from the SerializedMDBQuery object that this SerializedMDBQuery object represents.

Data Sources
IBM DB2 OLAP Server, Hyperion Essbase

Syntax
Java Method

```
String generateQuery();
String generateQuery(boolean returnHtmlSafeString);
```

where:

Argument	Description
`returnHtmlSafeString`	`true` to return an HTML-safe query string that can be used in display in an HTML page.

Usage

Returns the Essbase Report Spec query. Without argument, this method returns a query string that contains the "<" sign and therefore care should be taken when using the returned string in HTML code. To get an HTML-safe query string, use the second syntax with an argument of `true`.

getAxes()

Gets all axes in the stored serialized query.

Data Sources

Multidimensional

Syntax

Java Method

```
SerializedMDBQuery.Axis[] getAxes()
```

Usage

Returns an Axis inner class. See "SerializedMDBQuery.Axis Inner Class Methods" on page 185.

getAxisCount()

Gets a count of the axes.

Data Sources

Multidimensional

Syntax

Java Method

```
int getAxisCount();
```

getColumnAxis()

Gets the column axis.

Data Sources

Multidimensional

Syntax

Java Method

```
SerializledMDBQuery.Axis getColumnAxis();
```

getQueryGenerator()

Gets the EssbaseReportSpec object in order to create Essbase Report Spec query strings.

Data Sources
IBM DB2 OLAP Server, Hyperion Essbase

Syntax
Java Method

```
EssbaseReportSpec getQueryGenerator();
```

Usage
Returns the EssbaseReportSpec interface class. See "EssbaseReportSpec Methods" on page 180.

getRowAxis()
Gets the row axis.

Data Sources
Multidimensional

Syntax
Java Method

```
SerializledMDBQuery.Axis getRowAxis();
```

Usage
Returns an Axis inner class. See "SerializedMDBQuery.Axis Inner Class Methods" on page 185.

getSlicerAxis()
Gets the slicer axis.

Data Sources
Multidimensional

Syntax
Java Method

```
SerializledMDBQuery.Axis getSlicerAxis();
```

Usage
Returns an Axis inner class. See "SerializedMDBQuery.Axis Inner Class Methods" on page 185.

replaceMembers()
Replaces the members in the query and returns the number of members changed.

Data Sources
Multidimensional

Syntax
Java Method

```
int replaceMembers(String oldUniqueMemberName,
                   String newUniqueMemberName,
                   String newMemberName);

int replaceMembers(String oldUniqueMemberName,
                   TupleMember newMember);
```

where:

Argument	Description
oldUniqueMemberName	The unique name of the old member to be replaced.
newUniqueMemberName	The unique name of the new member.
newMemberName	The name of the new member.
newMember	The new member of type TupleMember.

Usage
The unique name should be a unique name in IBM DB2 OLAP Server and Hyperion Essbase or a fully qualified name in data sources that use MDX (Microsoft Analysis Services and DB2 Alphablox cubes). See "TupleMember" on page 331.

save()
Saves the SerializedMDBQuery object.

Data Sources
Multidimensional

Syntax
Java Method

```
void save();    //throws RepositoryIOException
```

update()
Updates the query.

Data Sources
All

Syntax
Java Method

```
void update();
```

SerializedMDBQuery.Axis Inner Class Methods
This section describes all methods associated with SerializedMDBQuery.Axis. To use any method for this object, import the com.alphablox.blox.repository package in your JSP.

getDimensionCount()
Gets a count of the dimensions on the axis.

Data Sources
Multidimensional

Syntax
Java Method

```
int getDimensionCount();
```

getDimensions()
Gets the dimensions on the axis.

Data Sources
Multidimensional

Syntax
Java Method

```
SerializedMDBQuery.Dimension[] getDimensions();
```

getNestedDimensionCount()
Gets a count of nested dimensions on the axis.

Data Sources
Multidimensional

Syntax
Java Method

```
int getNestedDimensionCount();
```

getTuple()
Get the tuple at the specified index.

Data Sources
Multidimensional

Syntax
Java Method

```
SerializedMDBQuery.Tuple getTuple(int tupleIndex);
```

where:

Argument	Description
tupleIndex	The zero-based index of the tuple.

Usage
Returns a Tuple inner class. See "SerializedMDBQuery.Tuple Inner Class Methods" on page 188.

getTupleCount()
Gets a count of tuples on the axis.

Data Sources
Multidimensional

Syntax
Java Method

```
int getTupleCount();
```

getTuples()
Gets all tuples on the axis.

Data Sources
Multidimensional

Syntax
Java Method

```
SerializedMDBQuery.Tuple[] getTuples();
```

Usage
Returns an array of the Tuple inner class. See "SerializedMDBQuery.Tuple Inner Class Methods" on page 188.

getType()
Gets the Axis type.

Data Sources
Multidimensional

Syntax
Java Method

```
int getType();
```

Usage
Returns an integer corresponding to one of the following static fields: `MEMBER_SETS`, `NOT_SET`, or `TUPLES`.

SerializedMDBQuery.Dimension Inner Class Methods
This section describes all methods associated with `SerializedMDBQuery.Dimension`. To use any method for this object, import the `com.alphablox.blox.repository` package in your JSP.

getCubeName()
Gets the name of the cube.

Data Sources
Multidimensional

Syntax
Java Method

```
String getCubeName();
```

getName()
Gets the name of the dimension.

Data Sources
Multidimensional

Syntax
Java Method

```
String getName();
```

getType()
Gets the Dimension type.

Data Sources
Multidimensional

Syntax
Java Method

```
int getType();
```

Usage
Returns an integer corresponding to one of the following static fields: `MEASURES`, `NORMAL`, `TIME`, or `UNKNOWN`.

getUniqueName()
Gets the unique dimension name.

Data Sources
Multidimensional

Syntax
Java Method

```
String getUniqueName();
```

SerializedMDBQuery.Tuple Inner Class Methods
This section describes all methods associated with `SerializedMDBQuery.Tuple`. To use any method for this object, import the `com.alphablox.blox.repository` package in your JSP.

getMember()
Gets the member at the specified index in the tuple or with the specified unique member name.

Data Sources
Multidimensional

Syntax
Java Methods

```
SerializedMDBQuery.Member getMember(int memberIndex);
SerializedMDBQuery.Member getMember(String uniqueMemberName);
```

where:

Argument	Description
`memberIndex`	The zero-based index for the member.
`uniqueMemberName`	A unique member name

Usage
Returns a Member inner class. See "SerializedMDBQuery.Member Inner Class Methods" on page 189.

getMemberCount()
Gets a count of the members in the tuple.

Data Sources
Multidimensional

Syntax
Java Method

```
int getMemberCount();
```

getMembers()

Gets all members in the tuple.

Data Sources
Multidimensional

Syntax
Java Method

```
SerializedMDBQuery.Member getMembers();
```

Usage
Returns an array of the Member inner class. See "SerializedMDBQuery.Member Inner Class Methods" on page 189.

SerializedMDBQuery.Member Inner Class Methods

This section describes all methods associated with `SerializedMDBQuery.Member`. To use any method for this object, import the `com.alphablox.blox.repository` package in your JSP.

getGenerationLevel()

Gets the generation level of the member.

Data Sources
Multidimensional

Syntax
Java Method

```
int getGenerationLevel();
```

isLeaf()

Identifies if this is a leaf member.

Data Sources
Multidimensional

Syntax
Java Method

```
boolean isLeaf();
```

getName()

Gets the name of the member.

Data Sources
Multidimensional

Syntax
Java Method

```
String getName();
```

getType()

Get the member type.

Data Sources

Multidimensional

Syntax

Java Method

```
int getType();
```

Usage

Returns an integer corresponding to one of the following static fields: `MEMBER_SET_EXPRESSION`, `ODBO_CALCULATED`, or `REGULAR`.

getUniqueName()

Gets the unique name of the member.

Data Sources

Multidimensional

Syntax

Java Method

```
String getUniqueName();
```

SerializedTextualQuery Methods

This section describes all methods associated with the `SerializedTextualQuery` object. This object gives you access to the serialized query stored in the `<bookmark_name>.query` file for relational data sources. To use any method for this object, import the `com.alphablox.blox.repository` package in your JSP.

getQuery()

Gets the textual query.

Data Sources

Relational

Syntax

Java Method

```
String getQuery();
```

save()

Saves the textual query.

Data Sources

Relational

Syntax

Java Method

```
void save();   // throws RepositoryIOException
```

setQuery()

Sets the textual query.

Data Sources
Relational

Syntax
Java Method

```
String setQuery(String query);
```

update()
Updates the query.

Data Sources
All

Syntax
Java Method

```
void update();
```

Chapter 8. ChartBlox Reference

This chapter contains reference material for ChartBlox. For general reference information about Blox, see Chapter 3, "General Blox Reference Information," on page 17. For information on how to use this reference, see Chapter 1, "Using This Reference," on page 1.

ChartBlox Overview

ChartBlox displays data in a wide variety of pie, bar, and line formats. Users can change chart attributes, such as chart type and orientation, through the ChartBlox graphical user interface.

Graphical User Interface

The ChartBlox graphical user interface (GUI) consists of a chart display area and optional chart controls. Users can right-click on a member (can be on the legend, labels, or in the chart itself) to bring up the right-click menu, which gives them data navigation options such as drill up, drill down, pivot, and hide/show members. To change chart types, axes placement, or configure data, users can access the Chart Options dialog via the menubar's Chart > Options... menu.

Available Chart Types

The following table lists the valid names of all available chart types when the ChartBlox is rendered in the DHTML client. When using one of these names as a value for the `chartType` property, omit any parenthetical comments.

Note: Note the following about chart types:

- The `chartType` property takes only the text string as a value. Be sure to spell it exactly as it appears in this table.
- The `get/setChartTypeAsInt()` methods take the integer shown to the left of each chart name.

Integer Value	Chart
200	Vertical Bar, Side-by-Side, 3D Effect (the default)
201	Vertical Line, Absolute, 3D Effect
202	Vertical Area, Absolute, 3D Effect
0	3D Bar
17	Vertical Bar, Side-by-Side (or simply "Bar")
18	Vertical Bar, Stacked

19	Vertical Bar, Side-by-Side, Dual Axis
20	Vertical Bar, Stacked, Dual Axis
21	Vertical Bar, Side-by-Side, Bipolar
22	Vertical Bar, Stacked, Bipolar
24	Horizontal Bar, Side-by-Side
25	Horizontal Bar, Stacked
26	Horizontal Bar, Side-by-Side, Dual Axis
27	Horizontal Bar, Stacked, Dual Axis
28	Horizontal Bar, Side-by-Side, Bipolar
29	Horizontal Bar, Stacked, Bipolar
31	Vertical Area, Absolute
32	Vertical Area, Stacked
33	Vertical Area, Absolute, Bipolar
34	Vertical Area, Stacked, Bipolar
35	Vertical Area, Percentage
41	Vertical Line, Absolute (or simply "Line")
42	Vertical Line, Stacked
43	Vertical Line, Absolute, Dual Axis
44	Vertical Line, Stacked, Dual Axis
45	Vertical Line, Absolute, Bipolar
46	Vertical Line, Stacked, Bipolar
47	Vertical Line, Percentage
55	Pie
61	Scatter
67	Radar, Line
68	Radar, Area
85	Histogram, Vertical
86	Histogram, Horizontal
89	Bubble Chart
502	Dial (See "Dial Charts" on page 195 for more information on how to use dial charts).
510	Waterfall
511	Pareto

The charting engine for the DHTML client does not support true 3D chart types. Therefore there is only one ordinal axis (O1). Chart types that have 3D effects include:

- 3D Bar (this is basically a Vertical Bar, Side-by-Side chart with the depth optimized).
- Vertical Bar, Side-by-Side, 3D Effect (the default)

- Vertical Line, Absolute, 3D Effect
- Vertical Area, Absolute, 3D Effect

Some chart types require you to specify more than one data value per each charted element. The following table lists these charts along with how many data values per element they require and the ordering requirements.

Integer	Type	Data Values and Order
61	Scatter	Two values per marker: X and Y, in that order.
89	Bubble Chart	Three values per marker: X, Y, and Z, in that order.

Dial Charts

Dial charts require specifications of several parameters before they can be drawn. These include the starting and ending numbers of the dial and the step size. Several nested tags are available for specification of a dial chart. For details, see "ChartBlox Overview" on page 193 and "Dial Chart Tag Reference" on page 274.

Chart Axes

Depending on the chart type, a chart may include one ordinal axes (O1) and up to three numeric axes (X1, Y1, and Y2). The ordinal axis contain groups or categories, and an O1 axis is included in all chart types except bubble and scatter charts. An X1 axis is only included in bubble and scatter charts. A Y1 axis is included in all chart types except pies. A Y2 axis is only included in dual-axes charts.

Specifying Style

You can set the style for the chart's title, axis title, footnote, and label by specifying the font and foreground color. For example, the following tag attribute sets the title style to use bold, 24-point Arial font with the color of #990099 and footnote style to use italic, 14-point Monospace font in red.

```
<blox:chart id="myChart"
   titleStyle="font=Arial:Bold:24, foreground=#990099"
   footnoteStyle="font=Monospace:Italic:14, foreground=red"
/>
```

You can also specify the style using the related nested tags as follows:

```
<blox:chart id="myChart">
   <blox:titleStyle
      font="Arial:Bold:24"
      foreground="#990099"
   />
   <blox:footnoteStyle
      font="Monospace:Italic:14"
      foreground="red"
   />
</blox:chart>
```

Setting the title, footnote, label, or axis title style overrides the defaults in the underlying theme.

Font

The font attribute takes the font name, style, and point size, separated by a colon:

font name: style: *point*

where:

- *font name*: These are defined according to the operating system. The following font names are generally accepted:
  - Arial
  - Courier
  - Helvetica
  - TimesRoman
  - SansSerif
  - Serif
  - Monospace

  Acceptable font values vary widely by browser and client machine. Therefore, generic names are provided (Monospace, SansSerif, and Serif). Each browser defines the actual font it will substitute for a generic name.

  If no font is specified, the default is SansSerif. If the server is running on a non-western language system, some characters may not display correctly if they cannot be found in the font's character set. To avoid this problem, always specify a font that will display correctly in your locale.
- *Style*: The font style can be one of the following:
  - `plain`
  - `italic`
  - `bold`
  - `bolditalic`
- *Point*: An integer for point size (usually 8-36).

Tip: If no font is specified, the default is SansSerif. If the server is running on a non-Western language system, some characters may not display correctly if they cannot be found in the font's character set. To avoid this problem, always specify a font that will display correctly in your locale. Properties that involve font specification include axisTitleStyle, labelStyle, footnoteStyle, and titleStyle.

Tip: If any of the three attributes is not specified, the default or the currently inherited font value is applied. However, the colons separating the attributes should be included, as shown in the following table:

Attribute Value	Result
`font=:Bold:12`	Uses the current font, but changes the style to bold, the size to 12 points.
`font=Helvetica`	Changes the font to Helvetica, but does not change the style or size.
`font=::14`	Uses the current font and style, but changes the size to 14 points.
`font=:Plain:`	Uses the current font and size, but changes the style to Plain.

Foreground

Color names are case-insensitive. The recognized color names are:

```
Black Blue Cyan DarkGray Gray Green         Magenta Orange Pink Red White Yellow
LightGray
```

A color can also be expressed as an RGB value that specifies the intensity of red, green, and blue, respectively, in a color. To specify a color by RGB value, convert

each 3-number value to its hexadecimal or decimal equivalent. Then enter the resulting string, beginning a hexadecimal string with a number sign (#). For example, #00FF00 is the hexadecimal string for 100% green. The RGB, hexadecimal, and decimal values for the recognized color names are listed in the following table.

Tip: Check the web for palettes of browser-safe colors, such as:

- http://www.visibone.com/colorlab/

Color Name	RGB Value	Hex Value	Decimal Value
Black	000 000 000	000000	0
Blue	000 000 255	0000FF	255
Cyan	000 255 255	00FFFF	65535
DarkGray	064 064 064	404040	4210752
Gray	128 128 128	808080	8421504
Green	000 255 000	00FF00	62580
LightGray	192 192 192	C0C0C0	12632256
Magenta	255 000 255	FF00FF	16711935
Orange	255 200 000	FFC800	16762880
Pink	255 175 175	FFAFAF	16756655
Red	255 000 000	FF0000	16711680
White	255 255 255	FFFFFF	16777215
Yellow	255 255 000	FFFF00	16776960

ChartBlox JSP Custom Tag Syntax

The Alphablox Tag Libraries provides custom tags to use in a JSP page for creating each blox. This section describes how to create the custom tag to create a ChartBlox. For a copy and paste version of the tag with all the attributes, see "ChartBlox JSP Custom Tag" on page 882.

Parameters

```
<blox:chart
    [attribute="value"] >
       [<blox:axisTitleStyle
              [attribute="value"] />]
       [<blox:dial
              [attribute="value"] />
       [<blox:footnoteStyle
              [attribute="value"] />]
       [<blox:labelStyle
              [attribute="value"] />]
       [<blox:seriesFill
              [attribute="value"] />]
       [<blox:titleStyle
              [attribute="value"] />]
</blox:chart>
```

where:

attribute is one of the attributes listed in the attribute table.

value is a valid value for the attribute.

and where the attributes are one of the following:

Attribute
id
absoluteWarning
applyPropertiesAfterBookmark
areaSeries
autoAxesPlacement
axisTitleStyle
backgroundFill
barSeries
bloxEnabled
bloxName
bookmarkFilter
chartAbsolute
chartCurrentDimensions
chartFill
chartType
columnLevel
columnSelections
comboLineDepth
dataTextDisplay
dataValueLocation
depthRadius
dwellLabelsEnabled
enablePoppedOut
filter
footnote
footnoteStyle
formatProperties
gridLineColor
gridLinesVisible
groupSmallValues
height
helpTargetFrame
histogramOptions
labelStyle
legend
legendPosition
lineSeries
lineWidth
localeCode

Attribute
logScaleBubbles
markerShape
markerSizeDefault
maxChartItems
maximumUndoSteps
menubarVisible
mustIncludeZero
noDataMessage
o1AxisTitle
pieFeelerTextDisplay
poppedOut
poppedOutHeight
poppedOutTitle
poppedOutWidth
quadrantLineCountX
quadrantLineCountY
quadrantLineDisplay
removeAction
render
rightClickMenuEnabled
riserWidth
rowHeaderColumn
rowLevel
rowSelections
rowsOnXAxis
seriesColorList
showSeriesBorder
smallValuePercentage
title
titleStyle
toolbarVisible
totalsFilter
useSeriesShapes
visible
width
x1AxisTitle
x1LogScale
x1ScaleMax
x1ScaleMaxAuto
x1ScaleMin
x1ScaleMinAuto

Attribute
XAxis
XAxisTextRotation
y1Axis
y1AxisTitle
y1FormatMask
y1LogScale
y1ScaleMax
y1ScaleMaxAuto
y1ScaleMin
y1ScaleMinAuto
y2Axis
y2AxisTitle
y2FormatMask
y2LogScale
y2ScaleMax
y2ScaleMaxAuto
y2ScaleMin
y2ScaleMinAuto

<blox:axisTitleStyle> nested tag **See "axisTitleStyle" on page 209.**
Attribute
font
foreground

<blox:footnoteStyle> nested tag **See "footnoteStyle" on page 221.**
Attribute
font
foreground

<blox:labelStyle> nested tag **See "labelStyle" on page 226.**
Attribute
font
foreground

<blox:seriesFill> nested tag **See "seriesFill" on page 241.**
Attribute
index
value

<blox:titleStyle> nested tag **See "titleStyle" on page 244.**
Attribute
font
foreground

Usage

Each custom tag can have one or more attributes, separated by one or more space or new line characters. Any extra space or new line characters are ignored. For readability, attributes can each go on a separate line with the same indenting.

When there are no nested tags (such as the <blox:titleStyle> or <blox:footnoteStyle> tag), you can substitute the closing </blox:chart> tag using the shorthand notation, closing the tag at the end of the attribute list that looks as follows:

```
width="650" />
```

When there are nested tags, the shorthand notation is not valid and a closing tag is required.

Examples

```
<blox:chart
     height="400"
     width="400"
     chartType="bar" />
```

ChartBlox Properties and Methods by Category

The following tables list the unique ChartBlox properties and their corresponding methods, if any. The tables also list ChartBlox methods for which there are no corresponding properties. For lists of properties and methods common to several Blox, see "Common Blox Properties and Methods by Category" on page 29. The properties and methods supported by ChartBlox are organized in the cross reference as follows:

- "Chart Appearance Properties" on page 202
- "Chart Data Properties" on page 204
- "Chart Label Properties" on page 205
- "Chart Popped Out Properties" on page 206
- "Chart Output Method" on page 206
- "Server-side Event Listener and Event Filter Methods" on page 206
- For dial charts, see "Dial Charts Overview" on page 269

Chart Appearance Properties

The following table lists properties and methods related to the appearance of a chart.

Chart Appearance

Properties	Methods
areaSeries	getAreaSeries() setAreaSeries()
autoAxesPlacement	getAutoAxesPlacement() setAutoAxesPlacement()
backgroundFill	getBackgroundFill() setBackgroundFill()
barSeries	getBarSeries() setBarSeries()
chartFill	getChartFill() setChartFill()
chartType	getChartType() setChartType()
comboLineDepth	getComboLineDepth() setComboLineDepth()
dataTextDisplay	getDataTextDisplay() setDataTextDisplay()
depthRadius	getDepthRadius() setDepthRadius()
footnoteStyle	getFootnoteStyle() setFootnoteStyle()
formatProperties	getFormatProperties() setFormatProperties()
gridLineColor	getGridLineColor() setGridLineColor()
gridLinesVisible	getGridLineVisible() setGridLineVisible()
histogramOptions	getHistogramOptions() setHistogramOptions()
labelStyle	getLabelStyle() setLabelStyle()

lineSeries	
	getLineSeries() setLineSeries()
lineWidth	
	getLineWidth() setLineWidth()
markerShape	
	getMarkerShape() setMarkerShape()
markerSizeDefault	
	getMarkerSizeDefault() setMarkerSizeDefault()
quadrantLineCountX	
	getQuadrantLineCountX() setQuadrantLineCountX()
quadrantLineCountY	
	getQuadrantLineCountY() setQuadrantLineCountY()
quadrantLineDisplay	
	getQuadrantLineDisplay() setQuadrantLineDisplay()
removeAction	
	getRemoveAction() setRemoveAction()
rightClickMenuEnabled	
	isRightClickMenuEnabled() setRightClickMenuEnabled()
riserWidth	
	getRiserWidth() setRiserWidth()
seriesColorList	
	getSeriesColorList() setSeriesColorList()
seriesFill	
	getSeriesFill() setSeriesFill()
showSeriesBorder	
	isShowSeriesBorder() setShowSeriesBorder()
titleStyle	
	getTitleStyle() setTitleStyle()
trendLines	
	getTrendLine() getTrendLines() setTrendLines()
useSeriesShapes	
	getUseSeriesShapes() setUseSeriesShapes()

Chart Data Properties

The following table lists properties and methods related to the data in a chart.

Chart Data	
Properties	**Methods**
absoluteWarning	getAbsoluteWarning() setAbsoluteWarning()
chartAbsolute	isChartAbsolute() setChartAbsolute()
chartCurrentDimensions	isChartCurrentDimensions() setChartCurrentDimensions()
columnLevel rowLevel	getColumnLevel() setColumnLevel() getRowLevel() setRowLevel()
columnSelections rowSelections	getColumnSelections() setColumnSelections() getRowSelections() setRowSelections()
dataValueLocation	getDataValueLocation() setDataValueLocation()
filter	getFilter() setFilter()
groupSmallValues	getGroupSmallValues() setGroupSmallValues()
legend	getLegend() setLegend()
localeCode	getLocaleCode() setLocaleCode()
logScaleBubbles	isLogScaleBubbles() setLogScaleBubbles()
maxChartItems	getMaxChartItems() setMaxChartItems()
mustIncludeZero	getMustIncludeZero() setMustIncludeZero()
rowsOnXAxis	getRowsOnXAxis setRowsOnXAxis
smallValuePercentage	getSmallValuePercentage() setSmallValuePercentage()
totalsFilter	getTotalsFilter() setTotalsFilter()
x1LogScale	isX1LogScale() setX1LogScale()
x1ScaleMax x1ScaleMaxAuto	getX1ScaleMax() setX1ScaleMax() isX1ScaleMaxAuto() setX1ScaleMaxAuto()
x1ScaleMin x1ScaleMinAuto	getX1ScaleMin() setX1ScaleMin() isX1ScaleMinAuto() setX1ScaleMinAuto()
XAxis	getXAxis() setXAxis()

Chart Data	
Properties	**Methods**
y1Axis y2Axis	getY1Axis() setY1Axis() getY2Axis() setY2Axis()
y1FormatMask y2FormatMask	getY1FormatMask() setY1FormatMask() getY2FormatMask() setY2FormatMask()
y1LogScale y2LogScale	isY1LogScale() setY1LogScale() isY2LogScale() setY2LogScale()
y1ScaleMax y1ScaleMaxAuto y2ScaleMax y2ScaleMaxAuto	getY1ScaleMax() setY1ScaleMax() isY1ScaleMaxAuto() setY1ScaleMaxAuto() getY2ScaleMax() setY2ScaleMax() isY2ScaleMaxAuto() setY2ScaleMaxAuto()
y1ScaleMin y1ScaleMinAuto y2ScaleMin y2ScaleMinAuto	getY1ScaleMin() setY1ScaleMin() isY1ScaleMinAuto() setY1ScaleMinAuto() getY2ScaleMin() setY2ScaleMin() isY2ScaleMinAuto() setY2ScaleMinAuto()

Chart Label Properties

The following table lists properties and methods related to the labels that appear on a chart.

Chart Labels	
Properties	**Methods**
axisTitleStyle	getAxisTitleStyle() setAxisTitleStyle()
dwellLabelsEnabled	isDwellLabelsEnabled() setDwellLabelsEnabled()
footnote footnoteStyle	getFootnote() setFootnote() getFootnoteStyle() setFootnoteStyle()
labelStyle	getLabelStyle() setLabelStyle()
legendPosition	getLegendPosition() setLegendPosition()
o1AxisTitle	getO1AxisTitle() setO1AxisTitle()
pieFeelerTextDisplay	getPieFeelerTextDisplay() setPieFeelerTextDisplay()
rowHeaderColumn	getRowHeaderColumn() setRowHeaderColumn()

Chart Labels	
Properties	**Methods**
titletitleStyle	getTitle() setTitle() getTitleStyle() setTitleStyle()
XAxisTextRotation	getXAxisTextRotation() setXAxisTextRotation()
x1AxisTitle	getX1AxisTitle() setX1AxisTitle()
y1AxisTitle y2AxisTitle	getY1AxisTitle() setY1AxisTitle() getY2AxisTitle() setY2AxisTitle()

Chart Popped Out Properties

The following table lists the properties regarding displaying ChartBlox in a separate, popped out browser window.

Chart Labels

Properties	**Methods**
enablePoppedOut	isEnablePoppedOut() setPoppedOut()
poppedOut	isPoppedOut() setPoppedOut()
poppedOutHeight	getPoppedOutHeight() setPoppedOutHeight()
poppedOutTitle	getPoppedOutTitle() setPoppedOutTitle()
poppedOutWidth	getPoppedOutWidth() setPoppedOutWidth()

Chart Output Method

The following table lists the property for writing a chart to a GIF image file.

Chart Labels

Property	**Method**
	writeChartToFile()

Server-side Event Listener and Event Filter Methods

The following table lists the methods for capturing events for pre- and post-event processing.

Methods

addEventFilter()
addEventListener()
removeEventFilter()
removeEventListener()

ChartBlox Properties and Associated Methods

This section describes the properties supported by ChartBlox and the methods associated with those properties. The properties are listed alphabetically by property name. For a list of ChartBlox methods with which no properties are associated, see "ChartBlox Methods" on page 267.

id

This is a common Blox property. For a detailed description, see "id" on page 39.

absoluteWarning

When the user sets chart values to display absolute values and at least one data value is negative, this warning is appended to the chart's footnote.

Data Sources: All

Syntax: JSP Tag Attribute

```
absoluteWarning="warning"
```

Java Methods

```
String getAbsoluteWarning();
          throws ServerBloxException
void setAbsoluteWarning(String warning);
          throws InvalidBloxPropertyValueException,
                  ServerBloxException
```

where:

Argument	Default	Description
warning	"Warning: Values are absolute"	Warning message string.

applyPropertiesAfterBookmark

This is a common Blox property. For a detailed description, see "applyPropertiesAfterBookmark" on page 32.

areaSeries

Specifies which data series in a combination chart should be the area series.

Data Sources

All

Syntax

JSP Tag Attribute

```
areaSeries="series"
```

Java Methods

```
String getAreaSeries();    // throws ServerBloxException

void setAreaSeries(String series);
void setAreaSeries(String[] seriesArray);
  // throws InvalidBloxPropertyValueException, ServerBloxException
```

where:

Argument	Default	Description
series	empty string	Comma-delimited string defining the displayed member names in the area series
seriesArray	empty string	An array of String defining the displayed member names in the area series.

When displaying an area, bar, or line chart, you can display the chart as a combination of these three chart types. It is possible to make one data series a line and another a bar and a third an area.

This property identifies the members represented on the area chart type as part of a combination chart. The displayed member names are defined as a comma-delimited string. If there are multiple dimensions making up the legend item (as defined in Chart Axes Placement), you must use tabs ("\t") to separate the dimensions.

Examples

```
myPresent.getChartBlox().setAreaSeries("Qtr1\tAudio, Qtr2\tAudio,
Qtr3\tAudio");
```

See Also

"barSeries" on page 211, "lineSeries" on page 228

autoAxesPlacement

Determines how information should be placed on chart axes.

Data Sources

All

Syntax

JSP Tag Attribute

```
autoAxesPlacement="auto"
```

Java Methods

```
boolean isAutoAxesPlacement();
          throws ServerBloxException
void setAutoAxesPlacement(boolean auto);
          throws InvalidBloxPropertyValueException,
                  ServerBloxException
```

where:

Argument	Default	Description
auto	true	Valid values are true or false.

Usage
The default indicates that ChartBlox should use the normal defaults for placing data on the x axis, legend, filter, y1 axis, and y2 axis. If you want to explicitly set any of these axes, legend or filter, this property should be set to `false`.

Examples
```
isAutoAxesPlacement();
setAutoAxesPlacement(false);
```

See Also
"filter" on page 220, "legend" on page 227, "XAxis" on page 253, "y1Axis" on page 255, "y2Axis" on page 261

axisTitleStyle
Specifies the style (foreground colors and text format) for the chart axis title.

Data Sources
All

Syntax
JSP Tag Attribute
```
axisTitleStyle="style"
```

or
```
<blox:axisTitleStyle
    font=""
    foreground="">
</blox:axisTitleStyle>
```

Java Methods
```
String getAxisTitleStyle();
          throws ServerBloxException
void setAxisTitleStyle(String style);
          throws InvalidBloxPropertyValueException,
                 ServerBloxException
```

where:

Argument	Default	Description
`style`	empty string	String defining style attributes.

Usage
For details on how to specify the style string, see "Specifying Style" on page 195.

Examples
```
getAxisTitleStyle();
setAxisTitleStyle("foreground=white, font=Courier:Bold:10");
```

See Also
"footnoteStyle" on page 221, "labelStyle" on page 226, "titleStyle" on page 244, "backgroundFill" on page 209

backgroundFill
Allows you to specify a solid color, color gradients, or images as the fill for the area outside of the chart frame.

Data Sources

All

Syntax

JSP Tag Attribute

```
backgroundFill="fill"
```

Java Methods

```
String getBackgroundFill();
          throws InvalidBloxPropertyValueException,
                  ServerBloxException
void setBackgroundFill(String fill);
          throws InvalidBloxPropertyValueException,
                  ServerBloxException
```

where:

Argument	Default	Description
fill	null	String defining the color, gradient, or image for the chart background area.

Usage

The string `fill` can be either a solid color, a list of two colors for color gradients, or a URL to an image you want to display in the background. Specify the two colors using either standard Java color names or RGB values. If you use RGB values, enter them in the form `0xffffff`. If you want to use gradient colors, the string should be a comma-separated list of two colors. You can specify a gradient direction as the last item in the string by adding the appropriate gradient qualifier from the table below.

Gradient Direction	Qualifier
Right	1 (the default if a direction is not specified with a list of two colors)
Left	2
Down	3
Up	4
Down/Left	5
Up/Left	6
Down/Right	7
Up/Right	8

If you want to specify an image to use, it must be of one of the following:

- A relative URL from the application context to the image. For example, if your JSP resides in an application called “salesApp” and you want to use the image file `logo.gif` in the `salesApp/images/` directory for the background:

  ```
  backgroundFill = "images/logo.gif"
  ```

- An absolute URL that starts with “`http:`”

  ```
  backgroundFill = "http://serverName/path/to/image.gif"
  ```

  Note that the server where the referenced image file is located should not require authentication. If authentication is required, the image will not load and

the default series color will be used. This is because the charting engine does not have a username and password to be authenticated.

- A URL that starts with "file:" using the file protocol:

  ```
  backgroundFill = "file:///C|/Alphablox5/webapps/salesApp/images/logo.gif"
  ```

  This is the file path to the image on the server where DB2 Alphablox is running.

The image will tile by default. If you want the image to stretch to fill the area, add:

```
, stretch"
```

to the end of the URL.

Examples

The following example fills the background with a solid color:

```
backgroundFill = "red"
```

The following example fills the background with a gradient from blue to green, with a direction that goes down to the right.

```
backgroundFill = "blue, green, 7"
```

The following example fills the background with a gradient from yellow to green. Since a direction is not specified, the default is from left to right.

```
backgroundFill = "yellow, green"
```

The following example stretches an image to fill the background:

```
backgroundFill = "images/logo.gif, stretch"
```

See Also

"chartFill" on page 213, "seriesFill" on page 241

barSeries

Specifies which data series in a combination chart should be the bar series.

Data Sources

All

Syntax

JSP Tag Attribute

```
barSeries="series"
```

Java Methods

```
String getBarSeries();
          // throws ServerBloxException

void setBarSeries(String series);
void setBarSeries(String[] seriesArray);
 // throws InvalidBloxPropertyValueException, ServerBloxException
```

where:

Argument	Default	Description
series	empty string	Comma-delimited string defining the displayed member names in the bar series.
seriesArray	empty string	An array of String defining the displayed member names in the bar series.

Usage

When displaying an area, bar, or line chart, you can display the chart as a combination of these three chart types. It is possible to make one data series a line and another a bar and a third an area.

This property identifies the members represented on the area chart type as part of a combination chart. The displayed member names are defined as a comma-delimited string. If there are multiple dimensions making up the legend item (as defined in Chart Axes Placement), you should use tabs ("\t") to separate the dimensions.

Examples

```
myPresentBlox.getChartBlox().setBarSeries("Qtr1\tVideo, Qtr2\t Video,
Qtr3\tVideo"); .
```

See Also

"areaSeries" on page 207, "lineSeries" on page 228

bloxEnabled

This is a common Blox property. For a complete description, see "bloxEnabled" on page 35.

bloxModel

This is a common Blox property. For a complete description, see "bloxModel" on page 38

bloxName

This is a common Blox property. For a complete description, see "bloxName" on page 35.

bookmarkFilter

This is a common Blox property. For a complete description, see "bookmarkFilter" on page 33.

chartAbsolute

Specifies whether negative values should be treated as positive.

Data Sources

All

Syntax

JSP Tag Attribute

```
chartAbsolute="chartAbsolute"
```

Java Methods

```
boolean isChartAbsolute();
          throws ServerBloxException
void setChartAbsolute(boolean chartAbsolute);
          throws InvalidBloxPropertyValueException,
                 ServerBloxException
```

where:

Argument	Default	Description
chartAbsolute	false	A boolean value. Specify true for negative values to be treated as positive values, otherwise specify false.

Usage
In pie charts, for example, a negative value will not appear. By setting this property to true, the value appears as a positive value in the chart.

Tip: When one or more chart values is negative, ChartBlox displays a warning message. To modify the text of the message, use the absoluteWarning property.

Examples
```
isChartAbsolute();
setChartAbsolute(true);
```

See Also
"absoluteWarning" on page 207

chartCurrentDimensions
Specifies the current members to be used for the chart filters.

Data Sources
All

Syntax
JSP Tag Attribute
```
chartCurrentDimensions="members"
```

Java Methods
```
String[] getChartCurrentDimensions();
                        // throws ServerBloxException
void setChartCurrentDimensions(String[] members);
                          // throws ServerBloxException
```

where:

Argument	Default	Description
members	null	An array of strings which are the currently selected chart filter items. When setting the members, the members have to be in the same order the dimensions are in the chart's page filter. For example, if Products and Locations are on the chart's page filter, you can specify "Coke, East" to be the selected members.

chartFill
Allows you to specify a solid color or an image as the fill for the area inside the chart frame that is not the data representation.

Data Sources
All

Syntax

JSP Tag Attribute

```
chartFill="fill"
```

Java Methods

```
String getChartFill();
          throws ServerBloxException
void setChartFill(String fill);
          throws InvalidBloxPropertyValueException,
                  ServerBloxException
```

where:

Argument	Default	Description
fill	null	String defining the color or image for the chart background area. The default is #F0F0F0 (very light grey).

Usage

The string fill can be either a solid color or a URL to an image you want to display in the background. Specify the color using either standard Java color names or RGB values. If you use a RGB value, enter it in the form 0xffffff.

If you want to specify an image to use, it must be of one of the following:

- A relative URL from the application context to the image. For example, if your JSP resides somewhere in an application context called "salesApp" and you want to use the image file logo.gif in the salesApp/images/ directory, you should specify the following relative URL:

  ```
  chartFill = "images/logo.gif"
  ```

- An absolute URL that starts with "http:"

  ```
  chartFill ="http://serverName/path/to/image.gif"
  ```

  Note that the server where the referenced image file is located should not require authentication. If authentication is required, the image will not load and the default color will be used. This is because the charting engine does not have a username and password to be authenticated.

- A URL that starts with "file:" using the file protocol:

  ```
  chartFill ="file:///C|/DB2Alphablox/webapps/salesApp/images/logo.gif"
  ```

  This is the file path to the image on the server where DB2 Alphablox is running.

The image will tile by default. If you want the image to stretch to fill the area, add:

```
, stretch"
```

to the end of the URL.

Examples

```
chartFill = "red"
chartFill = "http://someServer/images/mypicture.gif"
chartFill = "file:///C|/Alphablox5/webapps/salesApp/images/logo.gif, stretch"
```

See Also

"footnoteStyle" on page 221, "labelStyle" on page 226, "titleStyle" on page 244, , "seriesFill" on page 241

chartType

Identifies the type of chart to display.

Data Sources

All

Syntax

JSP Tag Attribute

```
chartType="type"
```

Java Methods

```
String getChartType();
          throws ServerBloxException
boolean setChartType(String type);
          throws InvalidBloxPropertyValueException,
                  ServerBloxException
```

where:

Argument	Default	Description
type	"Vertical Bar, Side-by-Side, 3D Effect"	See "Available Chart Types" on page 193.

Usage

Frequently used values include "3D Bar", "Bar", "Pie", and "Line". The value must exactly match one of the entries in the table of "Available Chart Types" on page 193.

Note: The best way to view various types is to create a simple application page with a ChartBlox on it. Then invoke the application and use the **Chart Type** dialog box to preview chart types.

Examples

```
getChartType();
setChartType("Vertical Bar, Stacked");
```

columnLevel

Specifies the data generation that the chart should use.

Data Sources

All

Syntax

JSP Tag Attribute

```
columnLevel="levels"
```

Java Methods

```
int getColumnLevel(int level);
          throws ServerBloxException
int[] getColumnLevel();
          throws ServerBloxException
void setColumnLevel(int index, int level);
          throws InvalidBloxPropertyValueException,
```

```
                ServerBloxException
void setColumnLevel(int[] levels);
          throws InvalidBloxPropertyValueException,
                ServerBloxException
```

where:

Argument	Default	Description
index	none	Dimension level.
level	none	An integer specifying a dimension level, where level 0 is the parent of every level.
levels	none	An integer array specifying a set of dimension levels, where level 0 is the parent of every level.

Usage

This method requires that the totalsFilter property be set to 2.

Examples

```
getColumnLevel(2);
setColumnLevel(2, 4);
```

See Also

"rowLevel" on page 238, "totalsFilter" on page 246

columnSelections

Specifies a subset of data to be charted.

Data Sources

All

Syntax

JSP Tag Attribute

```
columnSelections="selections"
```

Java Methods

```
String getColumnSelections();
          throws ServerBloxException
void setColumnSelections(String selections);
          throws InvalidBloxPropertyValueException,
                ServerBloxException
```

where:

Argument	Default	Description
selections	null	A String consisting of semicolon separated tuples, where the members of the tuples are separated by commas.

Usage

The value is a string consisting of a list of tuples separated by semicolons, with the members of each tuple separated by commas. Both columnSelections and rowSelections are set automatically when the user selects data in the grid and chooses to chart selected data, but they can be defined in the Blox so that the chart displays the specified data when loaded in the DHTML client. You must set both

the rowSelections and columnSelections properties in order for the chart to display data. If one is not set, the chart will be empty.

The default value of null indicates that all the data is charted.

Note: If your member names have commas or semicolons in them, you need to put double-quotes around each member name and escape the double-quotes, as follows:

```
columnSelections="\"East\", \"Qtr1\"; \"East\", \"Qtr2\""
```

Examples

```
columnSelections="East, Qtr1; East, Qtr2"
rowSelections="Actual, Audio; Actual, Visual"
```

See Also

"rowSelections" on page 239. For an additional example, see "Scriptlets Containing Blox APIs" on page 20.

comboLineDepth

Specifies the line depth in a combo chart.

Data Sources

All

Syntax

Java Method

```
int getComboLineDepth();
void setComboLineDepth(int depth);
```

where:

Argument	Default	Description
depth	0	The depth of lines in a combo chart in pixels.

dataTextDisplay

Controls whether data values will be shown above each bar in a bar chart.

Data Sources

All

Syntax

JSP Tag Attribute

```
dataTextDisplay="display"
```

Java Methods

```
boolean isDataTextDisplay();
       throws ServerBloxException
void setDataTextDisplay(boolean display);
       throws InvalidBloxPropertyValueException,
              ServerBloxException
```

where:

Argument	Default	Description
display	false	A boolean argument. A value of true sets the display so the data values appear above bar charts, a value of false indicates that data values do not appear.

Examples

```
getDataTextDisplay();
setDataTextDisplay(true);
```

dataValueLocation

Specifies the dimension name and list of member names used in the chart.

Data Sources

All

Syntax

JSP Tag Attribute

```
dataValueLocation="data"
```

Java Methods

```
String getDataValueLocation();
       throws ServerBloxException
void setDataValueLocation(String data);
       throws InvalidBloxPropertyValueException,
              ServerBloxException
```

where:

Argument	Default	Description

data	null	String of the form:
		`"Dimension:Member1, Member2..."`
		If there are multiple dimensions on an axis, use "\t" to separate the dimensions and members on the dimensions:
		`"Dim1\tDim2...:Member1ofDim1\tMember1ofDim2, Member2ofDim1\tMember2ofDim2..."`

Usage

When charting multidimensional data, this property defines which data to use on chart types that require more than one data value per element. When charting relational data, you must always use this property to define what data to chart. Use columns with numerical data only. If the columns contain other data, the chart will use null values and the chart will not be meaningful.

The syntax for data is the dimension name followed by a colon and a comma separated list of member names. When there is more than one dimension on an axis (for example, in a bubble chart), you can use "\t" to separate the dimensions and the members:

```
dataValueLocation="Scenario\tMeasures: Var% LY\tFS Sales,
   Act\tPromo %, Act\tFS Sales"
```

In the above example, two dimensions make up the axis: Scenario and Measures. The dimensions are separated with "\t" between them, and the members to be used are also specified with "\t" between them.

With relational data, the name of the column dimension is always "Columns".

In order for certain charts to work correctly, you must define the data values in a specific order. The order depends on the type of chart you are using. For a listing of chart types and data value requirements, see "Available Chart Types" on page 193.

Examples

```
getDataValueLocation();
setDataValueLocation("Columns: Product1, Product2");
```

depthRadius

Sets the depth of the 3D effect on 2D charts.

Data Sources

All

Syntax

JSP Tag Attribute

```
depthRadius="radius"
```

Java Methods

```
int getDepthRadius();
       throws ServerBloxException
void setDepthRadius(int radius);
       throws InvalidBloxPropertyValueException,
              ServerBloxException
```

where:

Argument	Default	Description
radius	0	An integer between 0 to 100, indicating the degree of 3D effect.

Usage

The default value, 0, eliminates the 3D effect. The higher the value, the more pronounced the 3D effect.

Examples

```
getDepthRadius();
setDepthRadius(45);
```

dwellLabelsEnabled

Specifies whether a dwell label (a text description of a data value) should appear when the user moves the mouse over a chart element.

Data Sources

All

Syntax

JSP Tag Attribute

```
dwellLabelsEnabled="enabled"
```

Java Methods

```
boolean isDwellLabelsEnabled();
       throws ServerBloxException
void setDwellLabelsEnabeld(boolean enabled);
       throws InvalidBloxPropertyValueException,
              ServerBloxException
```

where:

Argument	Default	Description
enabled	true	A boolean argument. A value of true indicates that the mouse-over labels appears on the chart, a value of false indicates that the labels do not appear.

Examples

```
isDwellLabelsEnabled();
setDwellLabelsEnabled(false);
```

enablePoppedOut

This is a property inherited from ContainerBlox. If the ChartBlox is nested within a PresentBlox:

- If the poppedOut property and its related properties have been specified in the PresentBlox, the settings in the PresentBlox are used.
- If the poppedOut property and its related properties have not been specified in the PresentBlox, the popped out settings in the nested ChartBlox are applied to the PresentBlox.

For a complete description, see “enablePoppedOut” on page 313.

filter

Specifies the dimensions that appear on the chart dimension filter.

Data Sources

All

Syntax

JSP Tag Attribute

```
filter="filter"
```

Java Methods

```
String getFilter();    // throws ServerBloxException

void setFilter(String filter);
void setFilter(String[] filterArray);
 // throws InvalidBloxPropertyValueException, ServerBloxException
```

where:

Argument	Default	Description
`filter`	empty string	Comma-delimited string defining filter dimensions.
`filterArray`	empty string	An array containing the names of dimensions on the filter.

Usage
ChartBlox determines the dimension placement if you use the default.
The `setFilter()` method automatically refreshes the chart.

Examples
```
myPresentBlox.getChartBlox().setFilter("Product");
```

footnote
Specifies the text to appear in the chart footnote (at the bottom right of the chart).

Data Sources
All

Syntax
JSP Tag Attribute
```
footnote="text"
```

Java Methods
```
String getFootnote();
       throws ServerBloxException
void setFootnote(String text);
       throws InvalidBloxPropertyValueException,
              ServerBloxException
```

where:

Argument	Default	Description
`text`	empty string	Any string. The text appears in the chart footnote.

Examples
```
getFootnote();
setFootnote("Company Confidential");
```

See Also
"footnoteStyle" on page 221

footnoteStyle
Specifies the style (foreground colors and text format) for the footnote.

Data Sources
All

Syntax
JSP Tag Attribute
```
footnoteStyle="style"
```

or

```
<blox:footnoteStyle
    font=""
    foreground="">
</blox:footnoteStyle>
```

Java Methods

```
String getFootnoteStyle();
       throws ServerBloxException
void setFootnoteStyle(String style);
       throws InvalidBloxPropertyValueException,
              ServerBloxException
```

where:

Argument	Default	Description
style	empty string	String defining style attributes.

Usage

For details on how to specify the style string, see “Specifying Style” on page 195.

Examples

```
getFootnoteStyle();
setFootnoteStyle("foreground=white, font=Courier:Bold:10");
```

See Also

“axisTitleStyle” on page 209, “footnote” on page 221, “labelStyle” on page 226, “titleStyle” on page 244

formatProperties

Specifies chart format properties string to override the defaults. These format properties are used by the DHML client user interface to set colors, styles, and other attributes of the chart, such as data series colors or x-axis text rotation custom angle.

Data Sources

All

Syntax

JSP Tag Attribute

```
formatProperties="formatProperties"
```

Java Methods

```
String getFormatProperties();
       //throws ServerBloxException
void setFormatProperties(String text);
       //throws InvalidBloxPropertyValueException,
                 InvalidBloxPropertyValueException, ServerBloxException
```

where:

Argument	Default	Description
formatProperties	empty string	Argument should be formatted as a comma-separated string of object property/value strings. Each object's property/value string should have each *property*:*value* pair separated by a semicolon, enclosed in curly braces. For example: *ObjectName={property1:value1; property2:value2;}, ObjectName2={property4:value4; property5:value5;}"* For properties that involve multiple values, separate the values with comma: *ObjectName1={property1:value1; property2:value2a,value2b;}, ObjectName2={property3:value3a, value3b; property4:value4a,value4b;}"* Alternatively, you can also use scope strings as keys. For example: *{Dim0:Mem0}{Dim1:Mem1}= {property0:value0,value1;property1:value2;}, {Dim2:Mem2,Mem3}={property2:value3;}* See "Scoping" on page 347 in the DataBlox's calculatedMembers section for the syntax on scope strings.

Usage
This property is currently only used to set individual data series colors, x-axis text rotation custom angle, and waterfall color array. Everything else should be set through normal named chart properties.

Examples
```
formatProperties="colorSeries_default_0 = {foreground:yellow;},
colorSeries_default_1 = {foreground:red;},
colorSeries_default_4 = {foreground:#FF9900;},
chart={XAxisTextRotation:45;}"
```

gridLineColor
Sets the color of the grid line.

Data Sources
All

Syntax
JSP Tag Attribute

```
gridLineColor="color"
```

Java Methods

```
Color getGridLineColor();
String getGridLineColorAsString();

void setGridLineColor(String color);
void setGridLineColor(Color javaColor);
```

where:

Argument	Default	Description
color	black	The name or hexadecimal value of a color.
javaColor	black	A java color.

Usage
The default grid line color when the application is rendered in DHTML client is #D0D0D0 (light grey). For more information on Java colors, see http://java.sun.com/j2se/1.4.2/docs/api/java/awt/Color.html.

Examples
```
setGridLineColor("red");
setGridLineColor("#00ffff");
```

gridLinesVisible
Specifies whether lines appear underlaid on a two-dimensional chart.

Data Sources
All

Syntax
JSP Tag Attribute

```
gridLinesVisible="enabled"
```

Java Methods

```
boolean isGridLinesVisible();
void setGridLinesVisible(boolean visible);
```

where:

Argument	Default	Description
visible	true	Specify true to display grid lines; false to hide them.

Usage
Some charts, such as a 3D bar chart, do not display grid lines, even if gridLinesVisible is set to true.

Examples
```
getGridLinesVisible();
setGridLinesVisible(false);
```

groupSmallValues
Groups smaller values into an “Other” item on a pie chart. This property affects only pie charts.

Data Sources
All

Syntax
JSP Tag Attribute

```
groupSmallValues="groupSmall"
```

Java Methods

```
boolean isGroupSmallValues();
       throws ServerBloxException
void setGroupSmallValues(boolean groupSmall);
       throws InvalidBloxPropertyValueException,
              ServerBloxException
```

where:

Argument	Default	Description
groupSmall	true	A boolean argument. A value of true indicates that values that are too small to chart are grouped into an "other" category on the chart, a value of false indicates that they will be charted.

Usage

Pie charts with many smaller values can be difficult to read, and grouping items together can improve chart readability.

The minimum percentage used for this grouping is set by the smallValuePercentage property.

Examples

```
isGroupSmallValues();
setGroupSmallValues(false);
```

See Also

"smallValuePercentage" on page 243

height

This is a common Blox property. For a complete description, see "height" on page 38.

helpTargetFrame

This is a common Blox property. For a complete description, see "helpTargetFrame" on page 39.

histogramOptions

Sets the options for histogram charts.

Data Sources

All

Syntax

JSP Tag Attribute

```
histogramOptions="options"
```

Java Methods

```
String getHistogramOptions();
boolean setHistogramOptions(String options);
```

where:

Argument	Default	Description
options	binMode=basic; binSize=false; binCount=6; addCumm=false; sort=false	A semicolon-delimited string of option and value pairs. Valid options are: • addCumm: true or false; true to add a cumulative percentage line to the chart (as in Pareto chart). The default is false. • binCount: the number of bins to include in the chart • binList: a comma-delimited list of numerics. For custom binning (binMode=custom) used to explicitly set bin ranges. Each number in the list is the inclusive, upper value for a bin. • binMode: basic (default) or custom. In basic mode, the bins are set either through binCount or binSize. The value set in binCount determines the bin ranges. The value set in binSize determines the number of bins in the chart. • binSize: the size of bin used to sort values. It should be a positive numeric value. • maxBin: the biggest value to store in the last (highest) bin. Used in conjunction with either binCount or binSize to determine bin ranges or number of bins. If this is not set, the largest value in the data is used. • minBin: the lowest value to store in the last (lowest) bin. Used in conjunction with either binCount or binSize. If this is not set, the lowest value in the data is used. • useSize: true or false. For basic binning (binMode=basic), when useSize is true, bins are created using binSize. Otherwise, bins are created based on binCount. • sort: true or false. A value of true results in a descending sort. When this option is combined with addCumm (sort=true;addCumm=true), it creates a Pareto chart.

Examples

The following example creates a sorted histogram chart with 10 bins and the chart includes a cumulative percentage line.

```
<blox:chart histogramOptions="addCumm=true;sort=true;binCount=10" .../>
```

labelStyle

Specifies the style (foreground colors and font) for the chart labels.

Data Sources

All

Syntax

JSP Tag Attribute

```
labelStyle="style"
```

or

```
<blox:labelStyle
    font=""
    foreground="">
</blox:labelStyle>
```

Java Methods

```
String getLabelStyle();
       throws ServerBloxException
boolean setLabelStyle(String style);
       throws InvalidBloxPropertyValueException,
              ServerBloxException
```

where:

Argument	Default	Description
style	empty string	String defining style attributes.

Usage

For details on how to specify the style string, see “Specifying Style” on page 195.

Examples

```
getLabelStyle();
setLabelStyle("foreground=white, font=Courier:Bold:10");
```

See Also

“axisTitleStyle” on page 209, “footnoteStyle” on page 221,, “titleStyle” on page 244

legend

Specifies the dimensions that appear on the legend.

Data Sources

All

Syntax

JSP Tag Attribute

```
legend="legend"
```

Java Methods

```
String getLegend();   // throws ServerBloxException

void setLegend(String legend);
void setLegend(String[] legendArray);
  //throws InvalidBloxPropertyValueException,ServerBloxException
```

where:

Argument	Default	Description
legend	empty string	A comma-delimited string of dimensions.
legendArray	empty string	An array containing the names of dimensions on the legend.

Usage

ChartBlox determines the dimension placement if you use the default. The `setLegend()` method automatically refreshes the chart.

Examples

```
setLegend("Measures, Market");
```

See Also

"legendPosition" on page 228, "setDataBlox()" on page 268

legendPosition

Specifies if and where chart legends are to appear.

Data Sources

All

Syntax

JSP Tag Attribute

```
legendPosition="position"
```

Java Methods

```
String getLegendPosition();
       throws ServerBloxException
boolean setLegendPosition(String position);
       throws InvalidBloxPropertyValueException,
              ServerBloxException
```

where:

Argument	Default	Description
position	bottom	A String with one of the following values:none, bottom, right. When the application's rendering mode is DHTML, the default is bottom. If the application's rendering mode is set to Java, the default is right. For dial charts, when you specify the dials using the nested <blox:dial> tag, legendPosition is automatically set to none as the legend will not be meaningful to users. See "Creating a Dial Chart" on page 269 for more information.

Usage

Valid values are:

- none – do not display legends.
- bottom – display legend beneath the chart.
- right – display legend to the right of the chart.

The default is bottom if the application's default render mode is set to DHTML.

Examples

```
getLegendPosition();
setLegendPosition("none");
```

See Also

"legend" on page 227

lineSeries

Specifies which data series in a combination chart should be the line series.

Data Sources
All

Syntax
JSP Tag Attribute

```
lineSeries="series"
```

Java Methods

```
String getLineSeries();  // throws ServerBloxException

void setLineSeries(String series);
void setLineSeries(String[] seriesArray);
// throws InvalidBloxPropertyValueException, ServerBloxException
```

where:

Argument	Default	Description
series	empty string	A comma-delimited string defining the displayed member names in the line series.
seriesArray	empty string	An array of String defining the displayed member names in the line series.

Usage
When displaying an area, bar, or line chart, you can display the chart as a combination of these three chart types. It is possible to make one data series a line and another a bar and a third an area.

This property identifies the members represented on the area chart type as part of a combination chart. The displayed member names are defined as a comma-delimited string. If there are multiple dimensions making up the axis (as defined in Chart Axes Placement), you must use tabs ("\t") to separate the dimensions.

Examples
```
setLineSeries("Qtr1\tAll Products, Qtr2\tAll Products, Qtr3\tAll Products");
```

See Also
"areaSeries" on page 207, "barSeries" on page 211

lineWidth
Controls the width of lines drawn on line charts.

Data Sources
All

Syntax
JSP Tag Attribute

```
lineWidth="width"
```

Java Methods

```
int getLineWidth();
       throws ServerBloxException
void setLineWidth(int width);
       throws InvalidBloxPropertyValueException,
              ServerBloxException
```

where:

Argument	Default	Description
width	3	A positive integer defining the line width.

Examples

```
getLineWidth();
setLineWidth(7);
```

localeCode

This is a common Blox property. For a complete description, see "localeCode" on page 40.

logScaleBubbles

Specifies whether to use a logarithmic scale to set bubble sizes in bubble charts.

Data Sources

All

Syntax

JSP Tag Attribute

```
logScaleBubbles="useLogScale"
```

Java Methods

```
boolean isLogScaleBubbles(); // throws ServerBloxException
void setLogScaleBubbles(boolean useLogScale);
   // throws InvalidBloxPropertyValueException, ServerBloxException
```

where:

Argument	Default	Description
useLogScale	false	true to set bubble sizes using a logarithmic scale.

markerShape

Sets the shape of the markers.

Data Sources

All

Syntax

JSP Tag Attribute

```
markerShape="shape"
```

Java Methods

```
int getMarkerShape(); // throws ServerBloxException
void setMarkerShape(int index, int shape);
   // throws InvalidBloxPropertyValueException,ServerBloxException
int[] getMarkerShape(); // throws ServerBloxException
void setMarkerShape(int[] markerShapes);
   // throws InvalidBloxPropertyValueException, ServerBloxException
```

where:

Argument	Default	Description
index		A 0-based index of the data series (or lines).
shape		Valid values are: • 0 = null • 1 = Square • 2 = Circle • 3 = Diamond • 4 = Plus • 5 = Triangle/Down • 6 = Triangle/Up Returns -1 if the index is out of bound.
markerShapes		A comma-separated list of numbers. Valid values are the same as shown above. Returns -1 if the index is out of bound.

Usage

The shapes will repeat. Setting the property to ″1,3,4″ will result in a square for the marker for the first series, the second a diamond, the third a plus, and then the fourth will be a square and so on.

Examples

```
getMarkerShape(0); // gets the shape of the marker for the
                   // 1st series as an integer

int[] markerShapes = { 1, 3, 4 };
setMarkerShape(markerShapes);
```

markerSizeDefault

Sets the size of the marker which appears on line charts and bubble charts.

Data Sources

All

Syntax

JSP Tag Attribute

```
markerSizeDefault="size"
```

Java Methods

```
int getMarkerSizeDefault(); //throws ServerBloxException
void setMarkerSizeDefault(int size);
   // throws InvalidBloxPropertyValueException, ServerBloxException
```

where:

Argument	Default	Description
size	30	Valid values are 0 to 100. A value of 30 is about 10 pixels. A value of 100 is about 30 pixels.

Examples

```
getMarkertSizeDefault();
setMarkerSizeDefault(10);
```

maxChartItems

Sets the maximum number of items allowed in the chart result set. If the result set exceeds this number, chart generation stops and you get an error message.

Data Sources

All

Syntax

JSP Tag Attribute

```
maxChartItems="items"
```

Java Methods

```
int getMaxChartItems();
       throws ServerBloxException
void setMaxChartItems(int items);
       throws InvalidBloxPropertyValueException,
              ServerBloxException
```

where:

Argument	Default	Description
items	256	A positive integer indicating the maximum items that can be charted.

Usage

Some charts might become difficult to read after a certain number of items are charted. This property is useful for limiting the number of items to be charted. The actual number of items you can chart and still be able to read the chart effectively will get larger as the size of your chart gets larger.

Examples

```
getMaxChartItems();
setMaxChartItems(10);
```

maximumUndoSteps

This is a common Blox property. For a complete description, see "maximumUndoSteps" on page 41.

menubarVisible

This is a common Blox property. For a complete description, see "menubarVisible" on page 42.

mustIncludeZero

Specifies whether to include zero on the chart axes.

Data Sources: All

Syntax: JSP Tag Attribute

```
mustIncludeZero="includeZero"
```

Java Methods

```
boolean isMustIncludeZero();
       throws ServerBloxException
void setMustIncludeZero(boolean includeZero);
       throws InvalidBloxPropertyValueException,
              ServerBloxException
```

where:

Argument	Default	Description
includeZero	true	A boolean argument. A value of true indicates that zero is charted, a value of false indicates that zero is not charted.

Usage: If set to true, there will always be a zero on the chart axes; the chart will begin counting at zero, regardless of the actual starting point on the measurement. If set to false, the measurement begins at a point close to the smallest value on the chart. In order to use log scale ([axis]LogScale) or [axis]ScaleMin, mustIncludeZero must be set to false.

Examples:

```
isMustIncludeZero();
setMustIncludeZero(false);
```

noDataMessage

This is a common Blox property. For a complete description, see "noDataMessage" on page 42.

o1AxisTitle

Explicitly defines the title for the O1 axis.

Data Sources: All

Syntax: JSP Tag Attribute

```
o1AxisTitle="title"
```

Java Methods

```
String getO1AxisTitle();
       throws ServerBloxException
void setO1AxisTitle(String title);
       throws InvalidBloxPropertyValueException,
              ServerBloxException
```

where:

Argument	Default	Description
title	null	Any string, indicating the text for the axis title.

Usage: O1 axis is the first ordinal axis in a chart that contains groups or categories. See "Chart Axes" on page 195 for details on chart axes. When charting relational data, the chart will not automatically display any axis titles. You must define all titles that you want displayed on the chart.

You can also specify titles with multidimensional data sources, but it is not required. The default value in this case, null, will automatically set axes titles, and

an empty string will display no title. A returned value of `null` for the getter methods signifies that the chart automatically determined the axis titles from a multidimensional data source.

Examples:

```
getO1AxisTitle();
setO1AxisTitle("This is the O1 Axis");
```

See Also: "x1AxisTitle" on page 249, "y1AxisTitle" on page 255, "y2AxisTitle" on page 261

pieFeelerTextDisplay

For pie charts, this property specifies whether and how pie slices should be labeled.

Data Sources: All

Syntax: JSP Tag Attribute

```
pieFeelerTextDisplay="type"
```

Java Methods

```
int getPieFeelerTextDisplay();
       throws ServerBloxException
void setPieFeelerTextDisplay(int type);
       throws InvalidBloxPropertyValueException,
              ServerBloxException
```

where:

Argument	Default	Description
type	1	An integer between 0 to 3, inclusive.

Usage: Valid values and their meanings are:

`0` = Do not label pie slices.

`1` = Show each text label at the end of a "feeler line" (a line that extends from the pie slice to the text).

`2` = Show labels only, with no feeler lines. Labels are positioned just outside the slice.

`3` = Place labels directly on the pie slices.

Examples:

```
getPieFeelerTextDisplay();
setPieFeelerTextDisplay(3);
```

poppedOut

This is a property inherited from ContainerBlox. If the ChartBlox is nested within a PresentBlox:

- If the `poppedOut` property and its related properties have been specified in the PresentBlox, the settings in the PresentBlox are used.
- If the `poppedOut` property and its related properties have not been specified in the PresentBlox, the popped out settings in the ChartBlox are applied to the PresentBlox.

For a complete description, see "poppedOut" on page 314.

poppedOutHeight

This is a property inherited from ContainerBlox. For a complete description, see "poppedOutHeight" on page 315.

poppedOutTitle

This is a property inherited from ContainerBlox. For a complete description, see "poppedOutTitle" on page 315.

poppedOutWidth

This is a property inherited from ContainerBlox. For a complete description, see "poppedOutWidth" on page 316.

quadrantLineCountX

Sets the number of vertical lines that will appear on a bubble chart. It is ignored for all other chart types.

Data Sources

All

Syntax

JSP Tag Attribute

```
quadrantLineCountX="count"
```

Java Methods

```
int getQuadrantLineCountX(); //throws ServerBloxException
void setQuadrantLineCountX(int count);
   // throws InvalidBloxPropertyValueException, ServerBloxException
```

where:

Argument	Default	Description
count	1	The value cannot be less than 1.

Usage

To remove the quadrant lines altogether, use the quadrantLineDisplay property.

Examples

```
getQuadrantLineCountX();
setQuadrantLineCountX(2);
```

See Also

"quadrantLineCountY" on page 235, "quadrantLineDisplay" on page 236

quadrantLineCountY

Sets the number of horizontal lines that will appear on a bubble chart. It is ignored for all other chart types.

Data Sources

All

Syntax

JSP Tag Attribute

```
quadrantLineCountY="count"
```

Java Methods

```
int getQuadrantLineCountY(); //throws ServerBloxException
void setQuadrantLineCountY(int count);
   // throws InvalidBloxPropertyValueException, ServerBloxException
```

where:

Argument	Default	Description
count	1	The value cannot be less than 1.

Usage

To remove the quadrant lines altogether, use the quadrantLineDisplay property.

Examples

```
getQuadrantLineCountY();
setQuadrantLineCountY(2);
```

See Also

"quadrantLineCountX" on page 235, "quadrantLineDisplay" on page 236

quadrantLineDisplay

Sets whether or not to display quadrant lines on a bubble chart.

Data Sources

All

Syntax

JSP Tag Attribute

```
quadrantLineDisplay="display"
```

Java Methods

```
boolean isQuadrantLineDisplay(); //throws ServerBloxException
void setQuadrantLineDisplay(boolean display);
   // throws InvalidBloxPropertyValueException, ServerBloxException
```

where:

Argument	Default	Description
display	true	To display no quadrant lines, set this property to false.

Examples

```
isQuadrantLineDisplay();
setQuadrantLineDisplay(false);
```

See Also

"quadrantLineCountX" on page 235, "quadrantLineCountY" on page 235

removeAction

This is a common Blox property. For a complete description, see “removeAction” on page 44.

render

This is a common Blox property. For a complete description, see “render” on page 45.

rightClickMenuEnabled

This is a common Blox property. For a complete description, see “rightClickMenuEnabled” on page 46.

riserWidth

Sets the width of the risers in bar charts. This value is a percentage of the available space.

Data Sources

All

Syntax

JSP Tag Attribute

```
riserWidth="width"
```

Java Methods

```
int getRiserWidth(); //throws ServerBloxException
void setRiserWidth(int width);
  //throws InvalidBloxPropertyValueException, ServerBloxException
```

where:

Argument	Default	Description
width	75	The percentage of the available space. A value of 100 would leave no space between the different groups.

Examples

```
getRiserWidth();
setRiserWidth(85);
```

rowHeaderColumn

Defines which column contains the names of the row labels and which column the chart uses to create the axis labels.

Data Sources

Relational

Syntax

JSP Tag Attribute

```
rowHeaderColumn="name"
```

Java Methods

```
String getRowHeaderColumn();
       throws ServerBloxException
void setRowHeaderColumn(String name);
       throws InvalidBloxPropertyValueException,
              ServerBloxException
```

where:

Argument	Default	Description
name	null	A String containing a column name.

Usage
Used only for relational data.

Examples
```
getRowHeaderColumn();
setRowHeaderColumn("Column header name");
```

rowLevel
Returns the data generation that the chart should use.

Data Sources
All

Syntax
JSP Tag Attribute

```
rowLevel="levels"
```

Java Methods

```
int getRowLevel(int level);
          throws ServerBloxException
int[] getRowLevel();
          throws ServerBloxException
void setRowLevel(int index, int level);
          throws InvalidBloxPropertyValueException,
                  ServerBloxException
void setRowLevel(int[] levels);
          throws InvalidBloxPropertyValueException,
                  ServerBloxException
```

where:

Argument	Default	Description
index	none	Dimension level.
level	none	An integer specifying a dimension level, where level 0 is the parent of every level.
levels	none	An integer array specifying a set of dimension levels, where level 0 is the parent of every level.

Usage
This method requires that the totalsFilter property be set to 2.

Examples
```
getRowLevel(2);
setRowLevel(3, 1);
```

See Also

"columnLevel" on page 215, "totalsFilter" on page 246

rowsOnXAxis

When enabled, swaps the chart axes such that data appear on opposite axes.

Data Sources

Relational

Syntax

JSP Tag Attribute

```
rowsOnXAxis="rowsOnXAxis"
```

Java Methods

```
boolean isRowsOnXAxis();
          throws ServerBloxException
void setRowsOnXAxis(boolean );
          throws InvalidBloxPropertyValueException,
                  ServerBloxException
```

where:

Argument	Default	Description
rowsOnXAxis	false	A boolean argument. A value of true indicates that the row axis is charted on the X axis, a value of false indicates that the row axis is charted on the Y axis.

Examples

```
getRowsOnXAxis();
setRowsOnXAxis(true);
```

rowSelections

Specifies a subset of data to be charted.

Data Sources

All

Syntax

JSP Tag Attribute

```
rowSelections="selections"
```

Java Methods

```
String getRowSelections();
          throws ServerBloxException
void setRowSelections(String selections);
          throws InvalidBloxPropertyValueException,
                  ServerBloxException
```

where:

Argument	Default	Description
selections	none	A String consisting of semicolon separated tuples, where the members of the tuples are separated by commas.

Usage

The value is a string consisting of a list of tuples separated by semicolons, with the members of each tuple separated by commas. These properties are set automatically when the user selects data in the grid and chooses to chart selected data, but they can be defined in the Blox so that the chart displays the specified data when loaded in the DHTML mode. You must set both the `rowSelections` and `columnSelections` properties in order for the chart to display data. If one is not set, the chart will be empty.

The default value of `null` indicates that all the data is charted.

Note: If your member names have commas or semicolons in them, you need to put double-quotes around each member name and escape the double-quotes, as follows:

```
rowSelections="\"Actual\", \"Audio\"; \"Actual\", \"Visual\""
```

Examples

```
columnSelections="East, Qtr1; East, Qtr2"
rowSelections="Actual, Audio; Actual, Visual"
```

See Also

"columnSelections" on page 216

seriesColorList

Sets the list of colors that will be used when charting the current series.

Data Sources

All

Syntax

JSP Tag Attribute

```
seriesColorList="list"
```

Java Methods

```
String[] getSeriesColorList();
          throws ServerBloxException
String getSeriesColorList(int index);
          throws ServerBloxException
void setSeriesColorList(String[] list);
          throws InvalidBloxPropertyValueException,
                  ServerBloxException
void setSeriesColorList(int index, String color);
          throws InvalidBloxPropertyValueException,
                  ServerBloxException
```

where:

Argument	Default	Description
`index`	null	The number in the series.
`color`	null	The name of a color.
`list`	null	Comma-delimited string of colors.

Usage

Sets the list of colors that will be used when charting the current series. The colors are specified in a comma separated string, and can be standard Java color names

or hexadecimal values. The default color when the application is rendered in DHTML is "#9691C7", "#00B09B", "#68AEE0", "#008B87", "#99CCCC", "#005699", "#C2C4C6", "#998300", "#CCAE99", "#A76100", "#E0CB68", "#B03400".

Be sure to list enough colors for the data you are charting. If you do not define enough colors, the colors you specify will be repeated in sequence for the remaining series.

Examples

```
String[] colors = {"red", "gree", "blue", "#FFCCFF"};
setSeriesColorList(colors);

getSeriesColorList(0); //gets the color of the first data series
```

See Also

"seriesFill" on page 241

seriesFill

Allows you to specify color gradients or images as the fill for the bars, lines, or areas that represent the data within the chart. This API takes an `index` argument which indicates the series of data in the chart to which you are applying the fill.

Data Sources

All

Syntax

JSP Tag

```
<blox:seriesFill
   index=""
   value="" >
</blox:seriesFill>
```

Java Methods

```
String getSeriesFill();
          throws ServerBloxException
void setSeriesFill(int index, String value);
          throws InvalidBloxPropertyValueException,
                 ServerBloxException
```

where:

Argument	Default	Description
`index`	null	An integer representing the number of any data series present.
`value`	null	String defining the color gradient, or image for the specified data series.

Usage

The `value` tag attribute/method argument is a string that can be either a list of two colors for color gradients or a URL to an image you want to display in an area. Specify the colors using either standard Java color names or RGB values. If you use RGB values, enter them in the form `0xffffff` or `#ffffff`. If you want to use gradient colors, the string should be a comma-separated list of two colors.

If you want the bars to contain gradients, you can specify a gradient direction as the last item in the string by adding the appropriate gradient qualifier from the table below. If you want the bars to contain solid colors, do not set the `seriesFill` property; instead, set the seriesColorList property.

Gradient Direction	Qualifier
Right	1 (the default if a direction is not specified)
Left	2
Down	3
Up	4
Down/Left	5
Up/Left	6
Down/Right	7
Up/Right	8

If you skip indices when specifying the `seriesFill`, the skipped series will use the same `seriesFill` as the last specified `seriesFill`. In the following example, series 2 and 3 will use the same gradient as series 1:

```
<blox:seriesFill index="1" value="green, yellow"/>
<blox:seriesFill index="4" value="blue, green"/>
```

If you want to specify an image to use, it must be of one of the following:

- A relative URL from the application context to the image. For example, if your JSP resides in an application called "salesApp" and it uses the image file `logo.gif` in the `salesApp/images/` directory:

  ```
  <blox:seriesFill index="1" value="images/logo.gif" />
  ```

- An absolute URL that starts with "`http:`"

  ```
  <blox:seriesFill index="2" value="http://serverName/path/to/image.gif" />
  ```

  Note that the server where the referenced image file is located should not require authentication. If authentication is required, the image will not load and the default series color will be used. This is because the charting engine does not have a username and password to be authenticated.

- A URL that starts with "`file:`" using the file protocol:

  ```
  <blox:seriesFill index="2"
  value="file:///C|/DB2Alphablox/webapps/salesApp/images/logo.gif" />
  ```

  This is the file path to the image on the server where DB2 Alphablox is running.

The image will always tile. If you use a transparent GIF image, it will overlay the series color (see "seriesColorList" on page 240).

Examples

```
<blox:chart ...>
   <blox:seriesFill index="1" value="green, yellow, 2"/>
</blox:chart>
```

The tag example above sets the first data series to a gradient of green to yellow, with a direction that goes to the left. The following are two Java method examples:

```
setSeriesFill(2, "blue, green, 5");
   // The second data series will be filled with a gradient from blue to
   // green, with a direction that goes down to the left
setSeriesFill(3, "red, yellow");
   // Since a direction is not specified, the default (going right) will be
   // applied
```

See Also

"footnoteStyle" on page 221, "labelStyle" on page 226, "titleStyle" on page 244, "chartFill" on page 213, "seriesColorList" on page 240

showSeriesBorder

Specifies whether or not a border is shown around the bars on a chart and the legend squares.

Data Sources

All

Syntax

JSP Tag Attribute

```
showSeriesBorder="show"
```

Java Methods

```
boolean isShowSeriesBorder();
          throws ServerBloxException
void setShowSeriesBorder(boolean show);
          throws InvalidBloxPropertyValueException,
                  ServerBloxException
```

where:

Argument	Default	Description
show	false	true — to show border around bars and legend squares; false— not to show border. The default is false.

Usage

Applies to bar charts only. The default is `false` when the application's rendering mode is set to DHTML.

smallValuePercentage

Sets the minimum percentage to group smaller values into an "Other" item on a pie chart. This property affects only pie charts.

Data Sources

All

Syntax

JSP Tag Attribute

```
smallValuePercentage="percentage"
```

Java Methods

```
double getSmallValuePercentage();
          throws ServerBloxException
void setSmallValuePercentage(double percentage);
          throws InvalidBloxPropertyValueException,
                  ServerBloxException
```

where:

Argument	Default	Description
`percentage`	`5.0`	An argument of type double. Valid values are between 0.01 and 10.0, inclusive.

Usage
Pie charts with many smaller values can be difficult to read, and grouping items together can improve chart readability.

Examples
```
getSmallValuePercentage();
setSmallValuePercentage(7.2);
```

See Also
"groupSmallValues" on page 224

title
Specifies the text to appear as a title above the chart.

Data Sources
Multidimensional

Syntax
JSP Tag Attribute
```
title="text"
```

Java Methods
```
String getTitle();
          throws ServerBloxException
void setTitle(String text);
          throws InvalidBloxPropertyValueException,
                  ServerBloxException
```

where:

Argument	Default	Description
`style`	empty string	String defining style attributes.

Examples
```
getTitle();
setTitle("My Title");
```

See Also
"titleStyle" on page 244

titleStyle
Specifies the style (foreground colors and text format) for the chart's title.

Data Sources
All

Syntax

JSP Tag Attribute

```
titleStyle="style"
```

or

```
<blox:titleStyle
    font=""
    foreground="">
</blox:titleStyle>
```

Java Methods

```
String getTitleStyle();
          throws ServerBloxException
void setTitleStyle(String style);
          throws InvalidBloxPropertyValueException,
                  ServerBloxException
```

where:

Argument	Default	Description
style	empty string	String defining style attributes.

Usage

For details on how to specify the style string, see “Specifying Style” on page 195.

Examples

```
getTitleStyle();
setTitleStyle("foreground=white, font=Courier:Bold:10");
```

See Also

“axisTitleStyle” on page 209, “footnoteStyle” on page 221, “labelStyle” on page 226, “title” on page 244

toolbarVisible

Specifies if the toolbar is visible.

Data Sources

All

Syntax

JSP Tag Attribute

```
toolbarVisible="visible"
```

where:

Argument	Default	Description
visible	true	true— the toolbar is visible; false— the toolbar is not visible. When the application’s rendering mode is set to DHTML, toolbar is visible by default.

Usage

By default, the toolbar is visible in a standalone ChartBlox. If a nested <blox:toolbar> tag is added, its setting overwrites the value of this attribute. For example, the following code will result in a visible toolbar.

```
<blox:chart id="myChart" toolbarVisible="false" ....>
   <blox:toolbar visible="true" />
   <blox:data bloxRef="myDataBlox"/>
</blox:chart>
```

Tip: `toolbarVisible` is only a tag attribute, not a property.

totalsFilter

Specifies if and how totals appear in a chart.

Data Sources

Multidimensional

Syntax

JSP Tag Attribute

```
totalsFilter="type"
```

Java Methods

```
int getTotalsFilter();
          throws ServerBloxException
void setTotalsFilter(int type);
          throws InvalidBloxPropertyValueException,
                  ServerBloxException
```

where:

Argument	Default	Description
`type`	1	An integer between 0 and 2, inclusive.

Usage

Valid values and their meanings are:

`0` = No filter; all totals appear.
`1` = Show the last (lowest-drilled-to) generation of each dimension on the chart.
`2` = Show a user-specified generation.

Tip: A value of "2" causes a set of radio buttons to appear at the bottom of the chart, with which the user selects the generation to appear on the chart. To limit the scope of the generation available, see the properties "rowLevel" on page 238 and "columnLevel" on page 215.

Examples

```
getTotalsFilter();
setTotalsFilter(0);
```

See Also

"rowLevel" on page 238, "columnLevel" on page 215

trendLines

Creates trendlines in the chart.

Data Sources

All

Syntax

JSP Tag

```
trendLines="trendLines"
```

Java Methods

```
String getTrendLine(int index);
String[] getTrendLines();
void setTrendLines(String[] trendLines)
   // throws ServerBloxException
```

where:

Argument	Default	Description

`trendLines`	null	A comma-separated string of trend lines:
		`trendLines="[trendLine1],[trendLine2],...,[trendLineN]"`
		
		Each trend line is a semicolon separated string of *parameter=value* pairs:
		`param1=value1;param2=value2;...;paramN=valueN`
		
		Valid parameters are:
		• `name`: Required. A string for the name of the trend line.
		• `member`: Required. One or more `member` parameter can be added. The member to use to plot the trend line, in the form of *uniqueDimensionName*: *uniqueMemberName* For example: `member=All Locations:East`
		• `type`: Required. Valid types are:
		– `exponential`
		– `linear`
		– `logarithmic`
		– `moving average(N)`: where N is at least 2
		– `polynomial(N)`: where N is greater or equal to 2 and less or equal to 100.
		– `power`
		• `drilldownscope`: Optional. Valid values are `descendents` and `none`.
		• `replace`: Optional. Set to `true` for the trend line to be drawn replacing the line or bar with which it is associated; `false`, not to replace.
		• `forecastforward`: Optional. A number of periods or units to forecast. Not supported for moving average trend line type.
		• `color`: Optional. Specify a Java color or a hexadecimal value (for example, #CCCCFF).
		• `style`: Optional. Set to `solid` or `dashed` for the line style.

`index`	null	A 0-based index of the trend line. For example: `myPresent.getChartBlox().getTrendLine(0)` gets the first trend line defined using the `trendLines` tag.

Usage

Trend lines can be added to line, bar, area or scatter charts. The trend lines added can be modified by end users via the **Chart** > **Trendlines...** option from the menubar.

The following table describes the types of trend lines supported:

Type	Description	Equation
Linear	The least squares fit for a line represented.	$y = c_0 + c_1 x$ where C1 is the slope and C0 is the intercept.
Logarithmic	The least squares fit through the data points.	$y = c_0 + c_1 \ln x$ where C0 and C1 are constants, and ln is the natural logarithm function.
Polynomial	The least squares fit through the data points.	$y = c_0 + c_1 x + c_2 x^2 + c_3 x^3 + \ldots + c_n x^n$ where C0 and C1...Cn are constants, where N is greater than or equal to 2, and N is less than or equal to 100.
Power	The least squares fit through the data points.	$y = cx^b$ where c and b are constants
Exponential	The least squares fit through the data points.	$y = ce^{bx}$ where c and b are constants, and e is the base of the natural logarithm.
Moving Average	The average over the a specified time period. The number of periods in a moving average trend line equals the total number of points in the series less the number you specify for the period.	$F_{(t+1)} = \frac{1}{N} \sum_{j=0}^{N-1} A_{t-j+1}$ where N is the number of prior periods to include in the moving average; Aj is the actual value at time j; Fj is the forecasted value at time j.

Examples

The following example creates two trend lines, one polynomial with an order of 3 and the other, linear.

```
trendLines="name=poly(3);member=All Locations:All Locations;
replace=true; type=polynomial(3),
name=line;member=All Locations:Central;type=linear"
```

This creates a polynomial trend line plotted for each member in All Locations, whereas the Central location has a linear trend line. The original data series lines/bars are replaced (`replace=true`).

useSeriesShapes

Sets whether the legend for a line chart displays the shapes used for the data points in the chart.

Data Sources

All

Syntax

JSP Tag Attribute

```
useSeriesShapes="display"
```

Java Methods

```
boolean isUseSeriesShapes();
          throws ServerBloxException
void setUseSeriesShapes(boolean display);
          throws InvalidBloxPropertyValueException,
                  ServerBloxException
```

where:

Argument	Default	Description
display	true	A boolean argument. A value of true specifies that the legend displays shapes for the different data points, a value of false specifies that the shapes are not displayed. The default is true when the application's rendering mode is set to DHTML.

Examples

```
isUseSeriesShapes();
setUseSeriesShapes(true);
```

visible

This is a common Blox property. For a complete description, see "visible" on page 46.

width

This is a common Blox property. For a complete description, see "width" on page 47.

x1AxisTitle

Explicitly defines the title for the X1 axis. This property only applies to bar charts, line charts, and area charts.

Data Sources

All

Syntax

JSP Tag Attribute

```
x1AxisTitle="title"
```

Java Methods

```
String getX1AxisTitle();
          throws ServerBloxException
void setX1AxisTitle(String title);
          throws InvalidBloxPropertyValueException,
                  ServerBloxException
```

where:

Argument	Default	Description
title	null	Any string, indicating the text for the axis title.

Usage
When charting relational data, the chart will not automatically display any axis titles. You must define all titles that you want displayed on the chart.

You can also specify titles with multidimensional data sources, but it is not required. The default value in this case, null, will automatically set axes titles, and an empty string will display no title. A returned value of "null" for the getter methods signifies that the chart automatically determined the axis titles from a multidimensional data source.

Examples
```
getX1AxisTitle();
setX1AxisTitle("This is the X1 Axis");
```

See Also
"o1AxisTitle" on page 233, "pieFeelerTextDisplay" on page 234, "XAxis" on page 253, "XAxisTextRotation" on page 254, "y1AxisTitle" on page 255, "y2AxisTitle" on page 261

x1LogScale
Sets whether or not to use a logarithmic scale for the X1 axis.

Data Sources
All

Syntax
JSP Tag Attribute

```
x1LogScale="width"
```

Java Methods

```
boolean isX1LogScale(); //throws ServerBloxException
void setX1LogScale(boolean logScale);
   //throws InvalidBloxPropertyValueException, ServerBloxException
```

where:

Argument	Default	Description
logScale	false	Specifies whether to use a logarithmic scale for the X1 axis.

Usage
When x1LogScale is set to true, since the chart engine does not automatically calculate the maximum and minimum values for the scale, the following properties have to be specified as well:

- x1ScaleMaxAuto has to be set to false.
- A value for x1ScaleMax has to be specified.
- x1ScaleMinAuto has to be set to false.
- A value for x1ScaleMin has to be specified.
- mustIncludeZero has to be set to false.

Since a log scale cannot start with 0 and the values will never be negative, mustIncludeZero must be set to false (the default is true).

Examples

```
isX1LogScale();
setX1LogScale(true);
```

See Also

"x1ScaleMaxAuto" on page 251, "x1ScaleMax" on page 251, "x1ScaleMinAuto" on page 253, "x1ScaleMin" on page 252, "mustIncludeZero" on page 232, "y1LogScale" on page 257, "y2LogScale" on page 263

x1ScaleMax

Sets the maximum value of the X1 axis.

Data Sources

All

Syntax

JSP Tag Attribute

```
x1ScaleMax="scale"
```

Java Methods

```
double getX1ScaleMax(); //throws ServerBloxException
void setX1ScaleMax(double scale);
   //throws InvalidBloxPropertyValueException, ServerBloxException
```

where:

Argument	Default	Description
scale	null	The maximum value of the X1 axis.

Usage

This property is ignored if x1ScaleMaxAuto is set to `true`. `x1ScaleMax` should always be set to a value larger than `x1ScaleMin`, or the chart may not behave properly.

Examples

```
getX1ScaleMax();
setX1ScaleMax(500000);
```

See Also

"x1ScaleMaxAuto" on page 251, "x1ScaleMin" on page 252

x1ScaleMaxAuto

Sets whether or not to automatically set the maximum value of the X1 axis. When this is `false`, the value of the x1ScaleMax property will be used to set the maximum value of the X1 axis.

Data Sources

All

Syntax

JSP Tag Attribute

```
x1ScaleMaxAuto="auto"
```

Java Methods

```
boolean isX1ScaleMaxAuto(); //throws ServerBloxException
void setX1ScaleMaxAuto(boolean auto)
   //throws InvalidBloxPropertyValueException, ServerBloxException
```

where:

Argument	Default	Description
auto	true	Valid values are true or false. When this is set to false, the maximum value of the X1 axis will be determined by the value of the x1ScaleMax property. By default, this is set to true and the value of the x1ScaleMax property is ignored.

Usage

When using log scale on an axis, all the corresponding *[axis]ScaleMaxAuto* and *[axis]ScaleMinAuto* have to be set to false and a value has to be specified for *[axis]ScaleMax* and *[axis]ScaleMin.* Since a log scale cannot include 0 or negative numbers, mustIncludeZero has to be set to false.

Examples

```
isX1ScaleMaxAuto();
setX1ScaleMaxAuto(false);
```

See Also

"x1LogScale" on page 250, "x1ScaleMax" on page 251

x1ScaleMin

Sets the minimum value of the X1 axis.

Data Sources

All

Syntax

JSP Tag Attribute

```
x1ScaleMin="scale"
```

Java Methods

```
double getX1ScaleMin(); //throws ServerBloxException
void setX1ScaleMin(double scale);
   //throws InvalidBloxPropertyValueException, ServerBloxException
```

where:

Argument	Default	Description
scale	null	The minimum value of the X1 axis.

Usage

This property is ignored if x1ScaleMinAuto is set to true. If mustIncludeZero is set to true, it has priority if x1ScaleMin is greater than zero. x1ScaleMax should always be set to a value larger than x1ScaleMin, or the chart may not behave properly.

Examples

```
getX1ScaleMin();
setX1ScaleMin(10000);
```

See Also

"x1ScaleMinAuto" on page 253, "x1ScaleMax" on page 251, "mustIncludeZero" on page 232

x1ScaleMinAuto

Sets whether or not to automatically set the minimum value of the X1 axis. When this is `false`, the value of the x1ScaleMin property will be used to set the minimum value of the X1 axis.

Data Sources

All

Syntax

JSP Tag Attribute

```
x1ScaleMinAuto="auto"
```

Java Methods

```
boolean isX1ScaleMinAuto(); //throws ServerBloxException
void setX1ScaleMinAuto(boolean auto)
   //throws InvalidBloxPropertyValueException, ServerBloxException
```

where:

Argument	Default	Description
`auto`	`true`	Valid values are `true` or `false`. When this is set to `false`, the minimum value of the X1 axis will be determined by the value of the x1ScaleMin property. By default, this is set to `true` and the value of the x1ScaleMin property is ignored.

Usage

When using log scale on an axis, all the corresponding *[axis]ScaleMaxAuto* and *[axis]ScaleMinAuto* have to be set to `false` and a value has to be specified for *[axis]ScaleMax* and *[axis]ScaleMin*. Since a log scale cannot include 0 or negative numbers, mustIncludeZero has to be set to `false`.

Examples

```
isX1ScaleMinAuto();
setX1ScaleMinAuto(false);
```

See Also

"x1LogScale" on page 250, "x1ScaleMin" on page 252

XAxis

Specifies the dimensions on the X axis. This property only applies to bar charts, line charts, and area charts.

Data Sources

All

Syntax

JSP Tag Attribute

```
XAxis="xAxis"
```

Java Methods

```
String getXAxis();     //throws ServerBloxException

void setXAxis(String xAxis);
void setXAxis(String[] xAxisDimensionNames);
   // throws InvalidBloxPropertyValueException, ServerBloxException
```

where:

Argument	Default	Description
xAxis	empty string	A comma-delimited string of dimension names.
xAxisDimensionNames	empty string	An array containing the dimension names on the X axis.

Usage

When using the default, ChartBlox determines the dimension placement. The `setXAxis()` method automatically refreshes the chart.

See "Chart Axes" on page 195 for details on chart axes

Examples

```
myPresentBlox.getChartBlox().setXAxis("All Products");
```

See Also

"x1AxisTitle" on page 249, "XAxisTextRotation" on page 254, "y1Axis" on page 255, "y2Axis" on page 261

XAxisTextRotation

Specifies the X axis label rotation. This property only applies to bar charts, line charts, and area charts.

Data Sources

All

Syntax

JSP Tag Attribute

```
XAxisTextRotation="type"
```

Java Methods

```
int getXAxisTextRotation();
          throws ServerBloxException
void setXAxisTextRotation(type);
          throws InvalidBloxPropertyValueException,
                  ServerBloxException
```

where:

Argument	Default	Description
type	0	An integer between 0 and 2, inclusive.

Usage

The accepted values are:

`0` = Normal
`1` = Rotated 90 degrees
`2` = Staggered

Examples

```
getXAxisTextRotation();
setXAxisTextRotation(2);
```

See Also

"x1AxisTitle" on page 249, "XAxis" on page 253

y1Axis

Specifies the member names on the Y1 axis.

Data Sources

All

Syntax

JSP Tag Attribute

```
y1Axis="y1Axis"
```

Java Methods

```
String getY1Axis();    //throws ServerBloxException

void setY1Axis(String y1Axis);
void setY1Axis(String[] y1AxisMemberNames);
  //throws InvalidBloxPropertyValueException, ServerBloxException
```

where:

Argument	Default	Description
`y1Axis`	empty string	Comma-delimited string of member display names. The members should be from the same dimension.
`y1AxisMemberNames`	empty string	An array containing the member display names on the Y1 axis.

Usage

See "Chart Axes" on page 195 for details on chart axes.

Under the default, ChartBlox determines the dimension placement. The `setY1Axis()` method automatically refreshes the chart.

Examples

```
myPresentBlox.getChartBlox().setY1Axis("Market");
```

See Also

"y1AxisTitle" on page 255, "y1FormatMask" on page 256, "y2Axis" on page 261

y1AxisTitle

Explicitly defines the title for the Y1 axis.

Data Sources
All

Syntax
JSP Tag Attribute

```
y1AxisTitle="title"
```

Java Methods

```
String getY1AxisTitle();
          throws ServerBloxException
void setY1AxisTitle(String title);
          throws InvalidBloxPropertyValueException,
                  ServerBloxException
```

where:

Argument	Default	Description
title	null	Any string, indicating the text for the axis title.

Usage
See “Chart Axes” on page 195 for details on chart axes.

When charting relational data, the chart will not automatically display any axis titles. You must define all titles that you want displayed on the chart.

You can also specify titles with multidimensional data sources, but it is not required. The default value in this case, null, will automatically set axes titles, and an empty string will display no title. A returned value of “null” for the getter methods signifies that the chart automatically determined the axis titles from a multidimensional data source.

Examples

```
getY1AxisTitle();
setY1AxisTitle("This is the Y1 Axis");
```

See Also
“o1AxisTitle” on page 233, “pieFeelerTextDisplay” on page 234, “x1AxisTitle” on page 249, “y1Axis” on page 255, “y1FormatMask” on page 256, “y2AxisTitle” on page 261

y1FormatMask
Specifies the value of the format mask for the Y1 axis on a chart.

Data Sources
All

Syntax
JSP Tag Attribute

```
y1FormatMask="formatMask"
```

Java Methods

```
String getY1FormatMask();
          throws ServerBloxException
void setY1FormatMask(String formatMask);
          throws InvalidBloxPropertyValueException,
                  ServerBloxException
```

where:

Argument	Default	Description
formatMask	null	See "Numeric Formatting" on page 544 and "formatMask" on page 575.

Usage

The format mask allows you to specify customized formatting for the chart axis values. This format is also used in the dwell labels that appear when the mouse pauses over a chart data item. For example, if the Y1 axis measures a percentage, you can specify `##0.00%` for the format mask. The format masks are set with the same way as the format masks on the grid. For information about how to set the value for these properties, see "Numeric Formatting" on page 544. The keywords `K` and `M` (for thousands and millions) are supported. Division (/) and multiplication (*) are also supported.

Examples

```
getY1FormatMask();
setY1FormatMask("##0.00%");
setY1FormatMask("$#,###/1000");
setY1FormatMask("#,###K");
```

To set the Y1 axis format for dollar values rounded to the nearest 100 million:

```
setY1FormatMask("$###,###,###.##");
```

See Also

"y2FormatMask" on page 262, "Numeric Formatting" on page 544

y1LogScale

Sets whether or not to use a logarithmic scale for the Y1 axis.

Data Sources

All

Syntax

JSP Tag Attribute

```
y1LogScale="width"
```

Java Methods

```
boolean isY1LogScale(); //throws ServerBloxException
void setY1LogScale(boolean logScale);
   //throws InvalidBloxPropertyValueException, ServerBloxException
```

where:

Argument	Default	Description
logScale	false	Specifies whether to use a logarithmic scale for the Y1 axis.

Usage

When `y1LogScale` is set to `true`, since the chart engine does not automatically calculate the maximum and minimum values for the scale, the following properties have to be specified as well:

- y1ScaleMaxAuto has to be set to `false`.
- A value for y1ScaleMax has to be specified.
- y1ScaleMinAuto has to be set to `false`.
- A value for y1ScaleMin has to be specified.
- mustIncludeZero has to be set to `false`.

Since a log scale cannot start with 0 and the values will never be negative, mustIncludeZero must be set to `false` (the default is `true`).

Examples

```
isY1LogScale();
setY1LogScale(true);
```

See Also

"y1ScaleMaxAuto" on page 259, "y1ScaleMax" on page 258, "y1ScaleMinAuto" on page 260, "y1ScaleMin" on page 259, "mustIncludeZero" on page 232, "y2LogScale" on page 263, "x1LogScale" on page 250

y1ScaleMax

Sets the maximum value of the Y1 axis.

Data Sources

All

Syntax

JSP Tag Attribute

```
y1ScaleMax="scale"
```

Java Methods

```
double getY1ScaleMax(); //throws ServerBloxException
void setY1ScaleMax(double scale);
   //throws InvalidBloxPropertyValueException, ServerBloxException
```

where:

Argument	Default	Description
`scale`	`null`	The maximum value of the Y1 axis.

Usage

This property is ignored if y1ScaleMaxAuto is set to `true`. `y1ScaleMax` should always be set to a value larger than `y1ScaleMin`, or the chart may not behave properly.

```
getY1ScaleMax();
setY1ScaleMax(500000);
```

See Also

"y1LogScale" on page 257, "y1ScaleMaxAuto" on page 259, "y1ScaleMin" on page 259.

y1ScaleMaxAuto

Sets whether or not to automatically set the maximum value of the Y1 axis. When this is `false`, the value of the y1ScaleMax property will be used to set the maximum value of the Y1 axis.

Data Sources
All

Syntax
JSP Tag Attribute

```
y1ScaleMaxAuto="auto"
```

Java Methods

```
boolean isY1ScaleMaxAuto(); //throws ServerBloxException
void setY1ScaleMaxAuto(boolean auto)
   //throws InvalidBloxPropertyValueException, ServerBloxException
```

where:

Argument	Default	Description
`auto`	`true`	Valid values are `true` or `false`. When this is set to `false`, the maximum value of the Y1 axis will be determined by the value of the y1ScaleMax property. By default, this is set to `true` and the value of the y1ScaleMax property is ignored.

Usage
When using log scale on an axis, all the corresponding `[axis]ScaleMaxAuto` and `[axis]ScaleMinAuto` have to be set to `false` and a value has to be specified for `[axis]ScaleMax` and `[axis]ScaleMin`. Since a log scale cannot include 0 or negative numbers, mustIncludeZero has to be set to `false`.

Examples

```
isY1ScaleMaxAuto();
setY1ScaleMaxAuto(false);
```

See Also
"y1LogScale" on page 257, "y1ScaleMax" on page 258, "y1ScaleMin" on page 259, "y1ScaleMinAuto" on page 260

y1ScaleMin

Sets the minimum value of the Y1 axis.

Data Sources
All

Syntax
JSP Tag Attribute

```
y1ScaleMin="scale"
```

Java Methods

```
double getY1ScaleMin(); //throws ServerBloxException
void setY1ScaleMin(double scale);
   //throws InvalidBloxPropertyValueException, ServerBloxException
```

where:

Argument	Default	Description
scale	null	The minimum value of the Y1 axis.

Usage
This property is ignored if y1ScaleMinAuto is set to true. y1ScaleMax should always be set to a value larger than y1ScaleMin, or the chart may not behave properly. mustIncludeZeo should be set to false since it has priority if y1ScaleMin is greater than zero

Examples
```
getY1ScaleMin();
setY1ScaleMin(10000);
```

See Also
"y1ScaleMinAuto" on page 260, "y1ScaleMax" on page 258

y1ScaleMinAuto

Sets whether or not to automatically set the minimum value of the Y1 axis. When this is false, the value of the y1ScaleMin property will be used to set the minimum value of the Y1 axis.

Data Sources
All

Syntax
JSP Tag Attribute

```
y1ScaleMinAuto="auto"
```

Java Methods

```
boolean isY1ScaleMinAuto(); //throws ServerBloxException
void setY1ScaleMinAuto(boolean auto)
   //throws InvalidBloxPropertyValueException, ServerBloxException
```

where:

Argument	Default	Description
auto	true	Valid values are true or false. When this is set to false, the minimum value of the Y1 axis will be determined by the value of the y1ScaleMin property. By default, this is set to true and the value of the y1ScaleMin property is ignored.

Usage
When using log scale on an axis, all the corresponding *[axis]ScaleMaxAuto* and *[axis]ScaleMinAuto* have to be set to false and a value has to be specified for *[axis]ScaleMax* and *[axis]ScaleMin*. Since a log scale cannot include 0 or negative numbers, mustIncludeZero has to be set to false.

Examples
```
isY1ScaleMinAuto();
setY1ScaleMinAuto(false);
```

See Also
"y1LogScale" on page 257, "y1ScaleMin" on page 259

y2Axis

Specifies the member display names on the Y2 axis.

Data Sources
All

Syntax
JSP Tag Attribute

```
y2Axis="y2Axis"
```

Java Methods

```
String getY2Axis();    // throws ServerBloxException

void setY2Axis(String y2Axis);
void setY2Axis(String[] y2AxisMemberNames);
//throws InvalidBloxPropertyValueException, ServerBloxException
```

where:

Argument	Default	Description
y2Axis	empty string	Comma-delimited string of member display names. The members should be from the same dimension.
y2AxisMemberNames	empty string	An array containing the member display names on the Y2 axis.

Usage
Under the default, ChartBlox determines the dimension placement. The `setY2Axis()` method automatically refreshes the chart. See "Chart Axes" on page 195 for details on chart axes.

Examples

```
myPresentBlox.getChartBlox().setY2Axis("Market");
```

See Also
"y1Axis" on page 255, "y2AxisTitle" on page 261, "y2FormatMask" on page 262

y2AxisTitle

Explicitly defines the title for the Y2 axis. See "Chart Axes" on page 195 for details on chart axes.

Data Sources
All

Syntax
JSP Tag Attribute

```
y2AxisTitle="title"
```

Java Methods

```
String getY2AxisTitle();
          throws ServerBloxException
void setY2AxisTitle(String title);
          throws InvalidBloxPropertyValueException,
                  ServerBloxException
```

where:

Argument	Default	Description
title	null	Any string, indicating the text for the axis title.

Usage

When charting relational data, the chart will not automatically display any axis titles. You must define all titles that you want displayed on the chart.

You can also specify titles with multidimensional data sources, but it is not required. The default value in this case, null, will automatically set axes titles, and an empty string will display no title. A returned value of "null" for the getter methods signifies that the chart automatically determined the axis titles from a multidimensional data source.

Examples

```
getY2AxisTitle();
setY2AxisTitle("This is the Y2 Axis");
```

See Also

"o1AxisTitle" on page 233, "pieFeelerTextDisplay" on page 234, "x1AxisTitle" on page 249, "y1AxisTitle" on page 255, "y2Axis" on page 261, "y2FormatMask" on page 262

y2FormatMask

Specifies the value of the format mask for the Y2 axis on a chart.

Data Sources

All

Syntax

JSP Tag Attribute

```
y2FormatMask="formatMask"
```

Java Methods

```
String getY2FormatMask();
          throws ServerBloxException
void setY2FormatMask(String formatMask);
          throws InvalidBloxPropertyValueException,
                  ServerBloxException
```

where:

Argument	Default	Description
formatMask	null	See "Numeric Formatting" on page 544 and "formatMask" on page 575.

Usage

The format mask allows you to specify customized formatting for the chart axis values. This format is also used in the dwell labels that appear when the mouse pauses over a chart data item. For example, if the Y2 axis measures a percentage, you can specify `##0.00%` for the format mask. The format masks are set with the same way as the format masks on the grid. For information about how to set the value for these properties, see "Numeric Formatting" on page 544. The keywords `K` and `M` (for thousands and millions) are supported. Division (/) and multiplication (*) are also supported.

Examples

```
getY2FormatMask();
setY2FormatMask("##0.00%");
setY2FormatMask("$#,###/1000");
setY2FormatMask("#,###K");
```

To set the Y2 axis format for dollar values rounded to the nearest 100 million:

```
setY2FormatMask("$###,###,###.##");
```

See Also

"y1FormatMask" on page 256, "Numeric Formatting" on page 544

y2LogScale

Sets whether or not to use a logarithmic scale for the Y2 axis.

Data Sources

All

Syntax

JSP Tag Attribute

```
y2LogScale="width"
```

Java Methods

```
boolean isY2LogScale(); //throws ServerBloxException
void setY2LogScale(boolean logScale);
   //throws InvalidBloxPropertyValueException, ServerBloxException
```

where:

Argument	Default	Description
`logScale`	`false`	Specifies whether to use a logarithmic scale for the Y2 axis.

Usage

When `y2LogScale` is set to `true`, since the chart engine does not automatically calculate the maximum and minimum values for the scale, the following properties have to be specified as well:

- y2ScaleMaxAuto has to be set to `false`.
- A value for y2ScaleMax has to be specified.
- y2ScaleMinAuto has to be set to `false`.
- A value for y2ScaleMin has to be specified.
- mustIncludeZero has to be set to `false`.

Since a log scale cannot start with 0 and the values will never be negative, mustIncludeZero must be set to `false` (the default is `true`).

Examples

```
isY2LogScale();
setY2LogScale(true);
```

See Also

"y2ScaleMaxAuto" on page 264, "y2ScaleMax" on page 264, "y2ScaleMinAuto" on page 266, "y2ScaleMin" on page 265, "mustIncludeZero" on page 232, "x1LogScale" on page 250, "y1LogScale" on page 257

y2ScaleMax

Sets the maximum value of the Y2 axis.

Data Sources

All

Syntax

JSP Tag Attribute

```
y2ScaleMax="scale"
```

Java Methods

```
double getY2ScaleMax(); //throws ServerBloxException
void setY2ScaleMax(double scale);
   //throws InvalidBloxPropertyValueException, ServerBloxException
```

where:

Argument	Default	Description
`scale`	null	The maximum value of the Y2 axis.

Usage

This property is ignored if y2ScaleMaxAuto is set to `true`. `y2ScaleMax` should always be set to a value larger than `y2ScaleMin`, or the chart may not behave properly.

Examples

```
getY2ScaleMax();
setY2ScaleMax(500000);
```

See Also

"y2ScaleMaxAuto" on page 264, "y2ScaleMin" on page 265

y2ScaleMaxAuto

Sets whether or not to automatically set the maximum value of the Y2 axis. When this is `false`, the value of the y2ScaleMax property will be used to set the maximum value of the Y2 axis.

Data Sources

All

Syntax

JSP Tag Attribute

```
y2ScaleMaxAuto="auto"
```

Java Methods

```
boolean isY2ScaleMaxAuto(); //throws ServerBloxException
void setY2ScaleMaxAuto(boolean auto)
   //throws InvalidBloxPropertyValueException, ServerBloxException
```

where:

Argument	Default	Description
auto	true	Valid values are true or false. When this is set to false, the maximum value of the Y2 axis will be determined by the value of the y2ScaleMax property. By default, this is set to true and the value of the y2ScaleMax property is ignored.

Usage

When using log scale on an axis, all the corresponding [axis]ScaleMaxAuto and [axis]ScaleMinAuto have to be set to false and a value has to be specified for [axis]ScaleMax and [axis]ScaleMin. Since a log scale cannot include 0 or negative numbers, mustIncludeZero has to be set to false.

Examples

```
isY2ScaleMaxAuto();
setY2ScaleMaxAuto(false);
```

See Also

"y2LogScale" on page 263, "y2ScaleMax" on page 264

y2ScaleMin

Sets the minimum value of the Y2 axis.

Data Sources

All

Syntax

JSP Tag Attribute

```
y2ScaleMin="scale"
```

Java Methods

```
double getY2ScaleMin(); //throws ServerBloxException
void setY2ScaleMin(double scale);
   //throws InvalidBloxPropertyValueException, ServerBloxException
```

where:

Argument	Default	Description
scale	null	The minimum value of the Y2 axis.

Usage
This property is ignored if y2ScaleMin is set to `true`. `y2ScaleMax` should always be set to a value larger than `y2ScaleMin` or the chart may not behave properly. `mustIncludeZeo` should be set to `false` it has priority if `y2ScaleMin` is greater than zero.

Examples
```
getY2ScaleMin();
setY2ScaleMin(10000);
```

See Also
"y2ScaleMinAuto" on page 266, "y2ScaleMax" on page 264

y2ScaleMinAuto

Sets whether or not to automatically set the minimum value of the Y2 axis. When this is `false`, the value of the y2ScaleMin property will be used to set the minimum value of the Y2 axis.

Data Sources
All

Syntax
JSP Tag Attribute

```
y2ScaleMinAuto="auto"
```

Java Methods

```
boolean isY2ScaleMinAuto(); //throws ServerBloxException
void setY2ScaleMinAuto(boolean auto)
   //throws InvalidBloxPropertyValueException, ServerBloxException
```

where:

Argument	Default	Description
`auto`	`true`	Valid values are `true` or `false`. When this is set to `false`, the minimum value of the Y2 axis will be determined by the value of the y2ScaleMin property. By default, this is set to `true` and the value of the y2ScaleMin property is ignored.

Usage
When using log scale on an axis, all the corresponding *[axis]ScaleMaxAuto* and *[axis]ScaleMinAuto* have to be set to `false` and a value has to be specified for *[axis]ScaleMax* and *[axis]ScaleMin*. Since a log scale cannot include 0 or negative numbers, mustIncludeZero has to be set to `false`.

Examples
```
isY2ScaleMinAuto();
setY2ScaleMinAuto(false);
```

See Also
"y2LogScale" on page 263, "y2ScaleMin" on page 265

ChartBlox Methods

This section describes ChartBlox methods that are not associated with a specific property. For the syntax and descriptions of ChartBlox methods that have a property associated with them, see “ChartBlox Properties and Associated Methods” on page 207. For client-side API common to Blox, see “Client-Side APIs” on page 31.

addEventFilter()

This is a common Blox method that for capturing a server-side event (such as saving and loading bookmarks) and perform custom actions *after* the operation is complete on the server. For details, see “addEventListener()” on page 49.

addEventListener()

This is a common Blox method for capturing an event and performing custom actions *after* the operation is complete on the server. For ChartBlox, this method lets you add a ChartPageEvent to capture the event after a user changes the page filter in the chart. For details, see “addEventListener()” on page 49, “ChartPageEvent Methods” on page 515, and “Example 3: Use the server-side ChartPageListener to set the desired data format on the chart when the chart filter is changed” on page 912.

call()

This is a common client-side Blox method. For a complete description, see “call()” on page 51.

flushProperties()

This is a common client-side Blox method. For a complete description, see “flushProperties()” on page 52.

getDataBlox()

This is a common Blox method. For a complete description, see “getDataBlox()” on page 53.

getProperty()

This is a common Blox method. For a complete description, see “getProperty()” on page 53.

loadBookmark()

This is a common Blox method. For a complete description, see “loadBookmark()” on page 54.

removeEventFilter()

This is a common Blox method that allows you to remove an event filter object added using addEventFilter() for capturing a server-side event (such as saving and loading a bookmark) *before* the event is processed on the server. For details, see “removeEventFilter()” on page 55.

removeEventListener()

This is a common Blox method for capturing an event and performing custom actions *after* the operation is complete on the server. For ChartBlox, this method lets you remove a ChartPageEvent added using `addEventFilter()`. For details, see "removeEventListener()" on page 56.

saveBookmark()

This is a common Blox method. For a complete description, see "saveBookmark()" on page 57.

saveBookmarkHidden()

This is a common Blox method. For a complete description, see "saveBookmarkHidden()" on page 58.

setDataBlox()

This is a common Blox method. For a complete description, see "setDataBlox()" on page 60.

setDataBusy()

This is a common Blox method. For a complete description, see "setDataBusy()" on page 60.

setProperty()

This is a common Blox method. For a complete description, see "setProperty()" on page 61.

updateProperties()

This is a common client-side Blox method. For a complete description, see "updateProperties()" on page 61.

writeChartToFile()

Creates a GIF image file based on the current chart data, and stores it in the location specified in the server file path provided.

Data Sources

All

Syntax

Java Method

```
writeChartToFile(String filepath,
                 int width,
                  int height); // returns boolean

writeChartToFile(String filepath,
                 int width,
                 int height,
                  String renderMode); // returns boolean

writeChartToFile(String filepath,
                 int width,
                 int height,
                 String renderMode
                   String themeName); // returns boolean
```

where:

Argument	Default	Description
filepath	null	File path on server.
width	null	Width of chart in pixels.
height	null	Height of chart in pixels
renderMode	null	The render mode. Use the constant Blox.RENDER_DHTML to use the DHTML render mode; Blox.RENDER_JAVA to use the Java render mode.
themeName	null	The theme to use.

Usage

The method returns true if the chart is written successfully.

Examples

```
<%
  chartBean.writeChartToFile("d://images/smallChart.gif", 100, 100,
Blox.RENDER_DHTML);
  chartBean.writeChartToFile("d://images/mediumChart.gif", 500, 500,
Blox.RENDER_DHTML);
  chartBean.writeChartToFile("d://images/largeChart.gif", 1024, 768,
Blox.RENDER_DHTML);
%>
```

Dial Charts Overview

Custom JSP tags are available for defining a dial chart. A dial chart is a circular chart with one or more dials. They are often used in executive dashboards, flash reports, or Key Performance Indicator (KPI) types of scenarios. Each dial has a scale, a needle, and one or more dial sectors. A dial sector is used to highlight a specified region on a dial chart with a particular color. For instance, you might have a dial plotting inventory with a minimum dial value of 100 and a maximum dial value of 500. There could be a red dial sector for the region between 100 and 200 indicating that if the needle is in this region, there is a danger of a supply inventory shortage.

A dial chart can contain more than one dials; each dial can have its own needle with multiple sectors. Typically, you assign different colors to the regions. Each dial has a start angle, an end angle, a needle, and a radius. A radius of 100% means the greatest possible length allowed given the size of the chart area. The combination of start/end angles and radius makes it possible to put multiple dials on a dial chart. The Blox Sampler contains several dial chart examples that demonstrate the different properties of a dial.

By default, dial charts are plotted based on the first available data value unless you specify otherwise. The API for creating dial charts is available in the com.alphablox.blox.uimodel.core.chart.dial package. Custom JSP tags are available for specifying most of the common properties and are described in “Dial Chart Tag Reference” on page 274.

Creating a Dial Chart

To create a dial chart:

1. Set the ChartBlox chartType property to dial.
2. Add a nested <blox:dial> tag for each dial.

3. For each dial, specify its needle, sector, and scale. The hierarchy of the tags is as follows:

```
<blox:chart chartType="dial">
   <blox:dial ...>
      <blox:needle ...>
      <blox:sector ...>
      <blox:scale ...>
   </blox:dial>
</blox:chart>
```

These tags allow you to specify the attributes for each of the components in a dial chart. For detailed listing on the tags, see "Dial Chart Tag Reference" on page 274. For discussions on each of the dial chart component, see "Dial Chart Components" on page 270.

Note: When you add a nested `dial` tag inside `<blox:chart>`, the chart type will be forced to `dial` even if the `chartType` property is not set or set to something else.

Tip: When you add a dial chart using the tags, the `legendPosition` property of the ChartBlox is automatically set to `none`. Changing the legend position through the ChartBlox dialog in the user interface will have no effect since the legend will not show. You should not reset `legendPosition` to other positions, because the legend will not be meaningful to users.

Dial Chart Components

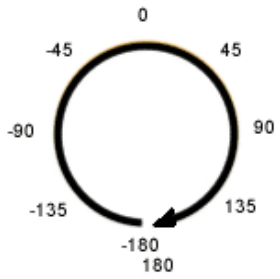
The key components in a dial chart include: Dial, Scale, Sector, and Needle.

Dial

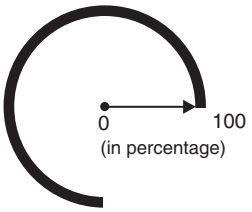
Each dial chart can contain one or more dials. A `<blox:dial>` tag is provided for you to specify the following key properties for each dial in your dial chart:

- `startAngle`: The position at which the dial region will begin.
- `stopAngle`: The position at which the dial region will stop.
- `radius`: The radius of the dial as a percentage of the available space.
- `color`: The fill color for this dial.
- `ticPosition`: The position for the tic marks on the dial. The position can be `inside`, `outside`, or `none`.
- `borderType`: The type of border for the dial. It can be `solid` or `none`.
- `borderColor`: The color for the dial's border.

The following diagram shows the how the values for `startAngle` and `stopAngle` are determined:



The following diagram shows how the radius is determined:



Scale

For each dial, you need to specify its scale. The scale for a dial consists of the minimum value, the maximum value, and the step size between tic marks on the dial. Any needles plotted on this dial are plotted against this scale. The dial's minimum and maximum values are automatically determined based on the first data value unless you specify otherwise. A nested <blox:scale> tag is provided that allows you to specify the following key attributes:

- maximum: The maximum value on the scale.
- minimum: The minimum value on the scale.
- stepSize: The step size of the tic markers.
- scope: The data value used to determine the values on this dial's scale.

The minimum and maximum values can be specified using either percentages or actual values. Using percentages is recommended as this allows the scale to be set dynamically based on the data value. If you specify actual values (without the "%" sign), the data value used for the needle may exceed the maximum value on the scale, resulting in problems plotting the needle. Only specify actual values in a "static" chart where data drilling action is disabled and the value for the needle will not change.

Sector

Using dial sectors, you can divide a dial into different sectors with different colors. This is useful for signalling threshold values. A dial sector is defined with:

- A start value and a stop value.
- An inner radius and an outer radius, which are expressed as percentages of the dial's radius.

A nested <blox:sector> tag is provided that allows you to specify the following key attributes:

- startValue: The beginning value for this sector. This can be either a percentage or actual data value, depending on how the minimum and maximum values are specified in the dial's scale.
- stopValue: The end value for this sector. This can be either a percentage or actual data value, depending on how the minimum and maximum values are specified in the dial's scale.
- color: The color for this sector.
- innerRadius: The inner radius for this sector. The default is 0, the center of the circle.
- outerRadius: The outer radius for this sector. The default is 100, the full length available.
- scope: The cell whose value should be used to determine the values in the sector.

- `tooltip`: The text to display when the mouse hovers over the sector.

The code snippet that generates the above output is available in "Example 1: Specifying Sectors" on page 272.

Needle

A needle is used in a dial chart to indicate what the actual, current data value is against some threshold numbers. Each dial can have one needle. A nested `<blox:needle>` tag is provided that allows you to specify the following key properties:

- `needleWidth`: The width of the needle in pixels.
- `endType`: The type of needle's end. The needle end can be a sharp arrow, a block arrow, a circle, or no needle end.
- `endWidth`: The width of the needle need.
- `color`: The color for this needle.
- `tooltip`: The tooltip to display when users mouse over the needle end.
- `scope` or `value:`The cell whose value, or the actual value, the needle should be pointing to.

Dial Chart Examples

The section includes complete examples that you can run to see the output. Examples include:

Example 1: Specifying Sectors

The following example demonstrates how sectors are created inside a dial. For a live example, see the Blox Sampler.

1. The chart type is first set to "dial." A PresentBlox is used here to include the GridBlox to show you how the dial's scale is determined.
2. There are two dials in this chart. The first one has an angle of -150 to 150, and the second one is a complete circle with an angle of -180 to 180.
3. The scale for this dial goes from 0 to 150% of the first data value returned from the query, with a tic markers for each increment of 2500.
4. Four sectors are created inside this dial, each with a different color (red, yellow, green, and dark green).
5. The second dial in this chart. This one is simply to add a small circle in the center as the base for the needle for better appearance of the chart. Its `needleWidth` is set to `0` and `needleType` to `none`.

The complete code that generates the above output is as follows:

```
      <%@ taglib uri="bloxtld" prefix="blox"%>
      <html>
      <head>
         <blox:header/>
      </head>
      <body>

      <blox:present id="dialExample1" height="90%" width="90%">
         <blox:chart chartType="dial" y1FormatMask="$#,###"
            titleStyle="font-Arial:bold:10"
            title="Milk Chocolate Truffles Sales for Jan 01">
            <blox:dial startAngle="-150" stopAngle="150" color="#CCCC99">
(1)            <blox:scale minimum="0" maximum="150%" stepSize="2500" />
               <blox:sector startValue="0" stopValue="50%"
(2)               color="red" innerRadius="30" outerRadius="80"
(3)               tooltip="Below expectation" />
(4)            <blox:sector startValue="50%" stopValue="80%"
                  color="yellow" outerRadius="80"
                  tooltip="Marginal performance"/>
               <blox:sector startValue="80%" stopValue="120%"
                  color="green" innerRadius="80"
                  tooltip="Satisfactory performance"/>
               <blox:sector startValue="120%" stopValue="150%"
                   color="#009966" innerRadius="80" />
            </blox:dial>

            <%--creating a blue circle in the center --%>
            <blox:dial startAngle="-180" stopAngle="180" color="black"
               radius="10" ticPosition="none" borderType="none">
               <blox:needle needleWidth="0" endType="none" />
(5)         </blox:dial>
         </blox:chart>
         <blox:data dataSourceName="qcc-essbase" useAliases="true"
            query="<ROW (\"All Products\") <CHILD \"Truffles\"
               <COLUMN (\"All Time Periods\") <CHILD \"Qtr 1 01\" !" />
      </blox:present>
      </body>
      </html>
```

Example 2: Specifying Needles and Scope

The following example generates a dial chart with four dials and four different needle types.

1. The chart type is first set to “dial.”
2. The first dial in the chart is simply to create a color border. The start and end angles are set to -135 to 135 with no tic markers or needles.
3. The second dial is the actual dial to convey meaningful data. Its radius is set to 90 so the first dial becomes the border.
4. The scale for the second dial is based on the value of `{scenario:forecast}`, with the minimum value being 0 and the maximum value being 120% of Forecast ($16,828,805 is 120% of the forecasted $14,024,008).
5. The needle for the second dial is based on the value of `{scenario:actual}`.
6. The yellow sector ends and the green sector starts at 100% of the forecasted value. This allows the users to see whether the actual value has achieved the forecasted goal.
7. The third dial in the chart is simply to add a small circle in the center as the base for the needle for better appearance of the chart. Its `needleWidth` is set to `0` and `needleType` to `none`.

The complete code that generates the above output is as follows:

```
       <%@ taglib uri="bloxtld" prefix="blox"%>
       <html>
       <head>
          <blox:header/>
       </head>
       <body>

       <blox:present id="dialExample1" height="90%" width="90%">
          <blox:chart chartType="dial" y1FormatMask="$#,###">
             <blox:dial startAngle="-135" stopAngle="135" color="#CCCC99"
                ticPosition="none" showLabels="false">
                <blox:needle needleWidth="0" endType="none" />
(1)          </blox:dial>
(2)

             <blox:dial startAngle="-135" stopAngle="135" radius="90">
                <blox:scale minimum="0" maximum="120%"
                   scope="{scenario:forecast}" />
(3)             <blox:needle color="blue" needleWidth="3"
(4)                endType="sharpArrow" scope="{scenario:actual}" />
                <blox:sector startValue="0" stopValue="75%"
(5)                color="red" />
                <blox:sector startValue="75%" stopValue="100%"
                   color="yellow"/>
                <blox:sector startValue="100%" stopValue="120%"
(6)                color="green" />
             </blox:dial>

             <%--creating a blue circle in the center --%>
             <blox:dial startAngle="-180" stopAngle="180" color="black"
                radius="10" ticPosition="none" borderType="none">
                <blox:needle needleWidth="0" endType="none" />
(7)          </blox:dial>
          </blox:chart>

          <blox:data dataSourceName="qcc-essbase" useAliases="true"
             query="\"All Time Periods\" <COLUMN (\"Scenario\") <SYM <ICHILD
             \"Scenario\" <ROW (\"All Products\") \"All Products\" !" />

       </blox:present>
       </body>
       </html>
```

Dial Chart Tag Reference

This section describes the tag attributes for the custom JSP tags supporting the creation of dial charts. The information is organized as follows:

<blox:dial> Tag Attributes

The following table lists the attributes for the <blox:dial> tag. For information on what a dial is and what its common properties are, see "Dial" on page 270.

Attribute	Default	Description
borderColor	black	The color of the border for the dial.

Attribute	Default	Description
borderType	solid	The border type for the border around the edge of the dial. Available values are: none and solid. For example: borderType="none"
color	default in the charting engine	The fill color of the dial chart. Specify a Java color or a hexadecimal value (for example, #CCCCFF).
radius	100	The radius of the dial as a percentage of the available space. Valid values are 0 through 100.
showLabels	true	To display the labels for the tics. If there is a scale for this dial, then the labels will be applied to the tics starting from the least value to the greatest value. Any missing labels or excess labels will be blank or ignored.
startAngle	-90	The position at which the dial region will begin. This angle is an upward pointing vertical line and sweeps in a clockwise direction. Values range from -180 to 180. For instance, having a starting angle of -90 and an ending angle of 90 will produce a horizontal half-circle dial.
stopAngle	90	The position at which the dial region will end. This angle is an upward pointing vertical line and sweeps in a clockwise direction. Values range from -180 to 180. For instance, having a starting angle of -90 and an ending angle of 90 will produce a horizontal half-circle dial.
ticPosition	outside	Where to position the tics for the dial axis. Possible values include: • inside: position the tics on the circumference of the dial pointing inward • outside: position the tics on the circumference of the dial pointing outward • none: leave off tics all together

<blox:needle> Tag Attributes

The following table lists the attributes for the <blox:needle> tag. For information on what a dial is and what its common properties are, see "Needle" on page 272. For an example, see "Example 2: Specifying Needles and Scope" on page 273.

Attribute	Default	Description
color	black	The color of the needle (both the needle and the needle end).
endType	sharpArrow	Specifies what will be drawn for the end of the needle. Possible values are: • sharpArrow: a triangle with sharp back corners • blockArrow: a triangle at the end • round: a circle at the end of the needle • none: a plain line needle When using round needle end, endWidth should be set to larger than the needleWidth since a circle with a diameter of the same width as the needle will not be discernible.
endWidth	5	The width of the end of the needle in pixels. If endType is set to none, then endWidth is ignored
needleWidth	2	The width of the needle in pixels.

Attribute	Default	Description

scope	The first data value	The cell whose value the needle should be pointing to. The scope should be specified as a series of dimension and member sets enclosed in braces.
		
		Use either display names or unique names. scope applies only to the row and column axes. If a dimension in the scope is not present, the scope will still match.
		
		For IBM DB2 OLAP Server and Hyperion Essbase data sources, specify the scope as follows:
		`scope="{d0:m0} {d1:m1} ..."`
		
		where d0 denotes a dimension and m0 denotes a member within that dimension. For example, for IBM DB2 OLAP Server and Hyperion Essbase data sources:
		`scope="{scenario:budget}"`
		
		For MS OLAP data sources, specify the scope using unique names as follows:
		`scope="{[Measures]: [Measures].[Profit Ratio]}"`

tooltip	The associated member names and the value	The tooltip to display when end users hover over the needle's end.
value	none	The value the needle points to on the dial.

<blox:scale> Tag Attributes

The following table lists the attributes for the <blox:scale> tag. For information on what a scale is and how it is specified, see "Scale" on page 271.

Attribute	Default	Description
maximum	200%	The maximum value on the dial. You can specify an actual maximum value or a percentage (include the % sign). For example, if the data value is 100,000 and maximum is set to 150%, the maximum value on the scale will be 150,000. In cases where the data value exceeds the maximum value on the dial, the pointer will be pointing at the end of the scale.
minimum	0	The minimum value on the dial. You can specify an actual minimum value, or a percentage (include the % sign). For example, if the data value is 100,000 and minimum is set to 50%, the minimum value on the scale will be 50,000. In cases where the data value is lower than the minimum value on the dial, the pointer will be pointing at the beginning of the scale.

Attribute	Default	Description

scope	The first data value	The cell whose value should be used to determine the values on the scale. scope should be specified as a series of dimension and member sets enclosed in braces.
		
		Use either display names or unique names. scope applies only to the row and column axes. If a dimension in the scope is not present, the scope will still match.
		
		For IBM DB2 OLAP Server and Hyperion Essbase data sources, specify the scope as follows:
		`scope="{d0:m0} {d1:m1}..."`
		
		where d0 denotes a dimension and m0 denotes a member within that dimension. For example, for IBM DB2 OLAP Server and Hyperion Essbase data sources:
		`scope="{scenario:budget}"`
		
		For MS OLAP data sources, specify the scope as follows:
		`scope="{[Measures]: [Measures].[Profit Ratio]}"`

stepSize	Automatically determined by dividing the scale into five regions	The step size between tic marks on the dial.

<blox:sector> Tag Attributes

The following table lists the attributes for the <blox:sector> tag. For information on what a sector is and how to specify a sector, see "Sector" on page 271. For an example, see "Example 1: Specifying Sectors" on page 272.

Attribute	Default	Description
color		The color of the dial sector. Specify a Java color name or a hexadecimal value.
innerRadius	0	The inner radius of this dial sector as a percentage of the dial's radius. Valid values are 0 to 100.
outerRadius	100	The outer radius of this dial sector as a percentage of the dial's radius. Valid values are 0 to 100.

scope	The first data value	The cell whose value should be used to determine the values in the sector. scope should be specified as a series of dimension and member sets enclosed in braces.
		
		Use either display names or unique names. scope applies only to the row and column axes. If a dimension in the scope is not present, the scope will still match.
		
		For IBM DB2 OLAP Server and Hyperion Essbase data sources, specify the scope as follows:
		`scope="{d0:m0} {d1:m1}..."`
		
		where d0 denotes a dimension and m0 denotes a member within that dimension.
		
		For MS OLAP data sources, specify the scope as follows:
		`scope="{[Measures]: [Measures].[Profit Ratio]}"`

Attribute	Default	Description
`startValue`		The start value for this sector. This should be set based on the dial's scale. For example, if the dial's scale has a minimum value of 100 and a maximum value of 500, then a red sector start value of 300 and stop value of 500 will make the region on the dial between 300 and 500 red. This value should be between the dial's scale minimum and maximum.
`stopValue`		The stop value for this sector. This should be set based on the dial's scale. For example, if the dial's scale has a minimum value of 100 and a maximum value of 500, then a red sector start value of 300 and stop value of 500 will make the region on the dial between 300 and 500 red. This value should be between the dial's scale minimum and maximum.
`tooltip`		The text to display when the mouse hovers over the sector.

Chapter 9. CommentsBlox Reference

This chapter contains reference material for CommentsBlox properties, methods and objects. For general reference information about Blox, see Chapter 3, “General Blox Reference Information,” on page 17. For information on how to use this reference, see Chapter 1, “Using This Reference,” on page 1.

CommentsBlox Overview

CommentsBlox allows you to provide cell commenting (also known as cell annotations) functionality to your application. In addition, you can use CommentsBlox for general commenting that are not tied to any other Blox. For example, you can allow users to add comments to a site, an application, a report, or a Web page.

Comments are stored in a JDBC accessible relational database. Supported databases include IBM DB2 UDB, Sybase, Microsoft SQL Server, and Oracle. This data source needs to be defined to DB2 Alphablox. DB2 Alphablox provides a Comments Management page under the Server link in the DB2 Alphablox Administration tab that lets you specify the relational data source to use for storing comments. From that page, you can create “collections” (data tables) to store comments. For cell-level comments, you will need to specify the multidimensional data source used in your GridBlox, the cube to use (for Microsoft Analysis Services), and the dimensions to include. For general comments, you only need to specify the name.

User Interface

When the commentary functionality is set up and enabled on a GridBlox user interface, A Comments menu item becomes available from the right-click menu. A red triangle comment indicator appears in the corner of the cells that has comments.

CommentsBlox is a container for comments that share the same set of fields and the same address scheme. For cell-level comments, the comments have an addressing scheme that incorporates the subset of dimensions and cube information needed to identify the cell. For general comments not tied to data cells, the addressing scheme is simply a string that contains the name of the comment collection. These comments are called “named comments.” Each CommentsBlox can have multiple named comment sets.

When users choose to display comments for a cell, the popped up window shows the address of the cell and all comments made on that cell, and the author and the time the comment was added. Only the author of the comment can delete the comment.

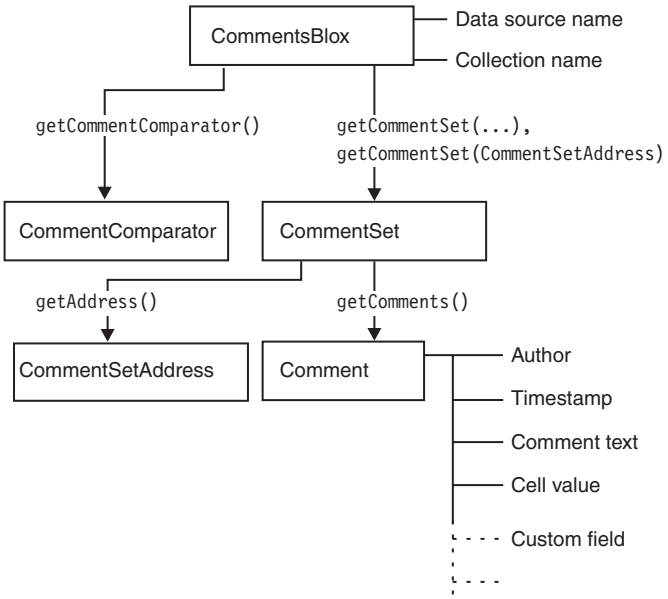
By default the comments are sorted by dates. Users can click the column headers to sort the comments based on the values in that column. Sort order works in toggle mode. Application developers can specify the field to sort on and the sort order using tags.

CommentsBlox Object Hierarchy and API

Through CommentsBlox, you can access a subset of comments associated with a specific cell or a named comment set, and then subsequently set or get more information on individual comment, such as its author, the comment text, and the time the comment was added.

Each comment has four required fields: author, timestamp, cell value, and comment text. You can add your own custom fields when creating the comments collection through the Comments Management page.

The following diagram shows the object hierarchy of CommentsBlox:



The CommentSet object is an interface through which comments are added, updated, and removed from the collection. Each CommentSet also has an address. As described earlier, for cell-level comments, the address of a CommentSet consists of the dimensions and cube information needed to identify the cell. For general comments not tied to data cells, the addressing scheme is simply a string that contains the name of the comment collection. You can access the CommentSet containing comments saved on a cell in a named address through CommentsBlox.

The Comment object has the following static fields that store the information for each comment:

- `FIELD_AUTHOR`
- `FIELD_CELLVALUE`
- `FIELD_COMMENTTEXT`

- `FIELD_TIMESTAMP`

It may also contain other custom fields as defined when the comment collection is created.

The Javadoc for the Comment, CommentSet, CommentSetAddress, and CommentComparator objects are under the `com.alphablox.blox.comments` package.

CommentsBlox Events

CommentsBlox uses CommentsListener to notify an assigned CommentEvent listener (CommentAddedEvent, CommentDeletedEvent, and CommentUpdatedEvent) that a comment was changed in the comments collection. This allows you to perform custom actions such as logging comment changes *after* the events are processed by the server.

Using the CommentsBlox `addCommentsListener()` method, you can add your comment listener. The CommentsListener has a `CommentChanged()` method that lets you specify the comment event to listen for. Each of the CommentAddedEvent, CommentDeletedEvent, and CommentUpdatedEvent lets you access the Comment or CommentSet affected by the associated event. For an example, see "Example 4: Adding a CommentAddedEvent Listener" on page 286.

Database Operations and Permissions

The use of CommentsBlox involves various database operations to support the creation of comments collections, editing a collection, and adding, displaying, and deleting comments. The following table shows the data operations behind the scene depending on the tasks performed. This will help you plan for proper permission setup needed for your application to work.

Task Performed	Data Operations Involved
Create a Comments collection:	Create tables and indexes
Edit an existing Comments collection:	Drop the old tables and create new ones
Delete a Comments collection:	Delete the associated tables
Add a comment:	Update and insert
Delete a comment:	Delete
Display a comment	Select

CommentsBlox JSP Custom Tag Syntax

The DB2 Alphablox Tag Libraries provides custom tags to use in a JSP page for creating each blox. This section describes how to use the custom tag to create a CommentsBlox. For a copy and paste version of the tag with all the attributes, see "CommentsBlox JSP Custom Tag" on page 884. Note that the CommentsBlox tag is a nested tag within the DataBlox custom tag when you want to provide cell-level comments. For named comments, since the comments are not tied to a DataBlox, the CommentsBlox tag is used as a standalone tag.

Syntax

For cell-level comments (associated with a DataBlox):

```
<blox:data id = "myData1"
    dataSourceName = "foodmart"
    query = " <%=myQueryString %>"
    ... >
    <blox:comments
        [attribute="value"] >
       <blox:sortComments
          field="" order="" />
    </blox:comments>
</blox:data>
```

Or you can have a standalone CommentsBlox referenced in a DataBlox:

```
<blox:comments id="myComments"
    [attribute="value"] >
       <blox:sortComments
          field="" order="" />
</blox:comments>

<blox:data id = "myData1"
    dataSourceName = "foodmart"
    query = " <%=myQueryString %>"
    ... >
    <blox:comments
        bloxRef="myComments">
    </blox:comments>
</blox:data>
```

Note: You cannot add a CommentsBlox tag within a DataBlox tag that is using the `bloxRef` attribute. It has to be nested within an actual DataBlox tag where its `dataSourceName` and `query` attributes are defined.

For named comments (not associated with a DataBlox):

```
<blox:comments
    [attribute="value"] >
       <blox:sortComments
          field="" order="" />
</blox:comments>
```

where:

attribute is one of the attributes listed in the attribute table.

value is a valid value for the attribute.

and where the attributes are one of the following:

Attribute
`id`
`bloxName`
`bloxRef`
`collectionName`
`dataSourceName`
`password`
`userName`

The nested <blox:sortComments> tag is optional and has two attributes:

Attribute
field
sort

Usage

Each custom tag can have one or more attributes, separated by one or more space or new line characters. Any extra space or new line characters are ignored. For readability, attributes can each go on a separate line with the same indenting.

You can substitute the closing </blox:comments> tag using the shorthand notation, closing the tag at the end of the attribute list that looks as follows:

```
dataSourceName = "comments_mssql" />
```

Examples

For cell-level comments (associated with a DataBlox):

```
<blox:data id = "myData1"
    dataSourceName = "foodmart"
    query = " <%=myQueryString %>" >
    <blox:comments id = "myComments1"
        collectionName = "sales_comments"
        dataSourceName = "comments_mssql" />
</blox:data>
```

For named comments (not associated with a DataBlox):

```
<blox:comments id = "myComments1"
    collectionName = "sales_comments"
    dataSourceName = "comments_mssql" />
```

Cell-level comments with a specified field to sort on when the Display Comments window pops up:

```
<!--import the following package in order to use the field constants-->
<%@ page import="com.alphablox.blox.comments.*" %>

<blox:data id = "myData2"
    dataSourceName = "QCC-MSAS"
    query = " <%=myQueryString %>" >
    <blox:comments id = "myComments2"
        collectionName = "planning_comments"
        dataSourceName = "comments_mssql" >
        <blox:sortComments
            field="<%= Comment.FIELD_COMMENTTEXT %>"
            order="<%= CommentComparator.DESCENDING %>" />
    </blox:comments>
</blox:data>
```

CommentsBlox Examples

This section provides examples that demonstrate how to use CommentsBlox to enable cell-level comments (associated with a DataBlox and a GridBlox), how to access comments for an individual cell through the MDBResultSet, and how to add an event listener when a comment is added (or deleted or updated).

- Example 1: Enabling cell commenting
- Example 2: Specifying Field to Sort On and Sort Order
- Example 3: Accessing Cell Comments Using MDBResultSet

- Example 4: Adding a CommentAddedEvent Listener

For page-level comments (not associated with a DataBlox) or using custom pages for adding and displaying cell comments, see the Highlighting and Commenting Data chapter in the *Developer's Guide*. For complete, live examples of CommentsBlox, see the Commenting on Data section in Blox Sampler under the Assembly tab on DB2 Alphablox Home Page.

Example 1: Enabling cell commenting

This example demonstrates how to enable cell commenting by setting `commentsEnabled` attribute to `true` on the GridBlox and adding a nested CommentsBlox tag inside the DataBlox. Note that the relational data source used to store comments needs to have been defined to DB2 Alphablox, and the collection name needs to have been created via the Comments Management page under the DB2 Alphablox Administrative tab.

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%
    String query = "your_data_query_here";
%>

<html>
<head>
    <blox:header />
</head>
<body>
    <blox:present id="presentBlox">
        //Enable cell commenting UI in GridBlox
        <blox:grid
            commentsEnabled="true" />

        <blox:data
            dataSourceName="foodmart"
            query="<%=query%>">
            //The datasource and collection names are defined and
            //and created via the DB2 Alphablox Admin Pages
            <blox:comments
                collectionName="sales_comments"
                dataSourceName="comments_mssql" >
            </blox:comments>
        </blox:data>
    </blox:present>
</body>
</html>
```

Example 2: Specifying Field to Sort On and Sort Order

This example is the same as the previous example except that it specifies the default field to sort the comments on when users choose to display comments on a cell. The sort order is set to ascending. The `com.alphablox.blox.comments` package is imported in order to use the constants for the field names and sort order.

```
<%@ page import="com.alphablox.blox.comments.*" %>
<%@ taglib uri="bloxtld" prefix="blox"%>
<%
    String query = "your_data_query_here";
%>
<html>
<head>
    <blox:header />
</head>
<body>
    <blox:present id="presentBlox" mayscriptEnabled="true" >
        //Enable cell commenting UI in GridBlox
```

```
        <blox:grid
            commentsEnabled="true" />
        <blox:data
            dataSourceName="foodmart"
            query="<%=query%>">
            //The datasource and collection names are defined and
            //and created via the DB2 Alphablox Admin Pages
            <blox:comments
                collectionName="sales_comments"
                dataSourceName="comments_mssql" >
                <blox:sortComments
                   field="<%= Comment.FIELD_AUTHOR %>"
                   order="<%= CommentComparator.ASCENDING %>" />
            </blox:comments>
        </blox:data>
    </blox:present>
</body>
</html>
```

Example 3: Accessing Cell Comments Using MDBResultSet

This example demonstrates how to access individual comments associated with a cell through the MDBResultSet object. To access the comments associated with Truffles for FY2001 in the following GridBlox:

Product (Category)	FY2000	FY2001	FY2002
Chocolate Blocks	178,148	3,282,371	3,052,920
Chocolate Nuts		6,686,802	9,390,529
Specialties	295,497	7,769,125	7,909,836
Truffles		749,789	789,908
All Products	473,645	18,488,088	21,143,193

you would access the MDBResultSet of the underlying DataBlox and then get to cell(1,3):

```
MDBResultSet resultSet = (MDBResultSet)
myCommentGrid.getDataBlox().getResultSet();
Cells cells = resultSet.getCells();
Cell cell = cells.getCell(1,3);
```

Now you can get to all comments for that cell:

```
CommentSet truffleCommentSet = cell.getCommentSet();
```

Here is the complete code:

```
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ page import="com.alphablox.blox.data.mdb.*" %>
<%@ page import="com.alphablox.blox.*" %>
<%@ page import="com.alphablox.blox.comments.*" %>
<%@ taglib uri="bloxtld" prefix="blox" %>
<html>
<head>
   <blox:header/>
</head>
<body>
<blox:grid id="myCommentGrid"
   width="60%"
   height="50%"
   commentsEnabled="true"
   defaultCellFormat="#,###"
   bandingEnabled="true">
   <blox:data dataSourceName="QCC-MSAS"
```

```
      query="SELECT {[Time.Fiscal].[All Time Periods].Children} ON COLUMNS,
         {[Products.Category].[All Products].Children,
         [Products.Category].[All Products]} ON ROWS FROM QCC
          WHERE ([Measures].[Sales])">
      <blox:comments
         collectionName="CommentsCollectionMSAS"
         dataSourceName="CommentsCollectionMSAS" />
   </blox:data>
</blox:grid>

<%
   //Access the comments associated with a cell from the result set
   MDBResultSet resultSet = (MDBResultSet)
myCommentGrid.getDataBlox().getResultSet();
   Cells cells = resultSet.getCells();
   Cell cell = cells.getCell(1,3);
   CommentSet truffleCommentSet = cell.getCommentSet();

   //Now get the address of the CommentSet for this cell
   CommentSetAddress truffleAddress = truffleCommentSet.getAddress();
   out.write("<BR>Address of CommentSet for Truffles: "+truffleAddress +
"<br>");

   //Now get the comment text for each comment in the CommentSet
   Comment[] comments = truffleCommentSet.getComments();
   out.write("<BR>The number of comments is: "+comments.length);
   for(int i = 0; i < comments.length; i++) {
      out.write("<BR>Comment Text: "+comments[i].getCommentText()+" for
comment: "+ i + "<br>");
   }

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

The output looks as follows:

```
Address of CommentSet for Truffles: CellCommentAddress: [Locations]:[Locations].[All
Locations];[Measures]:[Measures].[Sales];[Products].[Category]:[Products].[Category].[All
Products].[Truffles];[Products].[Code]:[Products].[Code].[All
Products];[Products].[Seasonal]:[Products].[Seasonal].[All Products];[Scenario]:[All
Scenario];[Seasonal]:[Seasonal].[All Seasonal];[Time].[Calendar]:[Time].[Calendar].[All Time
Periods];[Time].[Fiscal]:[Time].[Fiscal].[All Time Periods].[FY2001];

The number of comments is: 2

Comment Text for comment 0: The sales in the East region were 32% higher than projected,
making up the lost of sales in the West due to machine breakdown.

Comment Text for comment 1: There was a machine breakdown in the west region for two
weeks that impacted the sales of seasonal items.
```

Example 4: Adding a CommentAddedEvent Listener

The following example demonstrates how to capture a CommentAddedEvent and then print the author, comment text, and timestamp to the Alphablox console. To add a comment event listener:

1. Use the CommentsBlox `addCommentsListener()` method to add your comment listener.
2. Your comment listener should implement the CommentsListener interface.

3. Add the action to take when the comment is changed. Specify either the CommentAddedEvent, CommentDeletedEvent, or CommentUpdatedEvent to listen to in the `CommentChanged()` method.
4. You can then use `getComment()` or `getCommentSet()` to access the associated Comment or CommentSet.

```
<%@ taglib uri = "bloxtld" prefix = "blox"%>
<%@ page import="com.alphablox.blox.comments.*,
                 com.alphablox.blox.uimodel.core.MessageBox,
                 com.alphablox.blox.uimodel.BloxModel,
                 com.alphablox.blox.*" %>
<blox:comments id="myComments"
    collectionName="CommentsCollection"
    dataSourceName="CommentsCollection" />

<%! public abstract class CListener implements CommentsListener
   {
   BloxModel model;
   public void commentsChanged(com.alphablox.blox.comments.CommentAddedEvent cadded)
     throws Exception
     {
        Comment comment = cadded.getComment();
        StringBuffer msg = new StringBuffer("---------" + "\n");
        msg.append("Author: " + comment.getAuthor() + "\n");
        msg.append("Comment text: " + comment.getCommentText() + "\n");
        msg.append("Time: " + comment.getTimestamp( ));
        MessageBox msgBox = new MessageBox(msg.toString(),"Comments Added",
           MessageBox.MESSAGE_OK, null);
        model.getDispatcher().showDialog(msgBox);
     }
  } %>
  <blox:present id="CommentsPresentBlox"
    ...
     >
    <blox:grid
       commentsEnabled="true" />
       <blox:data
          dataSourceName="QCC-Essbase"
          query="!">
          <blox:comments
             bloxRef="myComments"/>
       </blox:data>
       <% myComments.addCommentsListener( CListener() );  %>
 </blox:present>
```

CommentsBlox Properties and Methods by Category

The following tables list unique CommentsBlox properties. The tables also list methods for which there are no corresponding properties. For lists of properties and methods common to several Blox, see "Common Blox Properties and Methods by Category" on page 29.

The properties and methods supported by CommentsBlox are organized in the cross reference as follows:

- CommentsBlox—Comment Collection
- CommentsBlox.Query Inner Class
- Comment Object
- CommentComparator Object
- CommentSet Object
- CommentSetAddress Object

CommentsBlox—Comment Collection

Properties	Methods
collectionName	getCollectionName() setCollectionName()
dataSourceName	getDataSourceName() setDataSourceName()
dimensions	getDimensions() setDimensions()
fieldNames	getFieldNames()
namedCommentSets	getNamedCommentSets()
open	isOpen()
password	getPassword() setPassword()
userName	getUserName() setUserName()
	getCommentComparator() setCommentComparator()
	getCommentSet()
	getCollectionName() setCollectionName() getCollectionNames()
	open() close() create() delete()
	addField() clearFields() deleteField() getFieldDescription() isProtectedField()
	performCleanUp()
	hasComments()
	replaceDimensions()

CommentsBlox.Query Inner Class

This inner class has the following Java methods for setting the dimension/member map for querying all cell-level comments set associated with this constraint.

Methods
addDimensionConstraint()
setDimensionConstraint()

Comment Object

The Comment object has the following server-side methods for getting and setting information on individual comments. To use this API, add the following code to the beginning of your JSP:

```
<%@ page import="com.alphablox.blox.comments.*"%>
```

Methods
getAuthor()
getCommentText()
getField()
getFields()
getTimestamp()
getTimestampDate()
isChanged()
setAuthor()
setCommentText()
setField()

CommentComparator Object

The CommentComparator object has the following server-side methods for CommentComparator which allows you to compare the values for the specified field using the specified sort order. To use this API, add the following code to the beginning of your JSP:

```
<%@ page import="com.alphablox.blox.comments.*"%>
```

Methods
CommentComparator()
compare()
getField()
getOrder()

CommentSet Object

The CommentSet object is the interface through which comments can be added, updated, and deleted from the comment collection. It also provides a method for accessing the Comment object. All methods are server-side. To use this API, add the following code to the beginning of your JSP:

```
<%@ page import="com.alphablox.blox.comments.*"%>
```

Methods
addComment()
deleteComment()
getAddress()
getComments()
updateComment()

CommentSetAddress Object

The CommentSetAddress object has the following server-side methods for getting and setting information on a CommentSet's address. To use this API, add the following code to the beginning of your JSP:

```
<%@ page import="com.alphablox.blox.comments.*"%>
```

Methods
getAddressName()
getDimensionMember()
getDimensions()
isNamedAddress()
setDimensionMember()

CommentsBlox Properties and Associated Methods

This section describes the properties supported by CommentsBlox and the methods associated with those properties. The properties are listed alphabetically by property name. For a list of CommentsBlox methods with which no properties are associated, see "CommentsBlox Methods" on page 294. Common Blox properties available from DataBlox are listed but not described. For complete descriptions of common Blox properties, see "Properties and Associated Methods Common to Multiple Blox" on page 32.

id

This is a common Blox tag attribute. For a complete description, see "id" on page 39.

bloxName

This is a common Blox tag attribute. For a complete description, see "bloxName" on page 35.

bloxRef

This is a common Blox tag attribute. For a complete description, see "bloxRef" on page 38

collectionName

The name of the comment collection. Each CommentsBlox needs to be associated with a relational data source and a collection name in that data source.

Data Sources

Relational (for storing comments)

Syntax

JSP Tag Attribute

```
collectionName="name"
```

Java Methods

```
String getCollectionName();
void setCollectionName(String name);
```

where:

Argument	Default	Description
name	null	The name of the comment collection.

dataSourceName

The name of the data source used to stored comments.

Data Sources

Relational (for storing comments)

Syntax

JSP Tag Attribute

```
dataSourceName="name"
```

Java Method

```
String getDataSourceName();
void setDataSourceName(String name);
```

where:

Argument	Default	Description
name	null	The name of the data source.

dimensions

Dimensions defined for the collection used in this CommentsBlox.

Data Sources

Relational (for storing comments)

Syntax

Java Methods

```
String[] getDimensions();    //returns a String array
void setDimensions(String[] dimensions);
```

where:

Argument	Default	Description
dimensions	null	A String array of dimensions (unique names) in your MDB data source defined for the collection of comments for this CommentsBlox.

Usage
If this CommentsBlox represents a named Blox, an array of length 0 is returned. The `setDimensions()` method must be called before calling create(). This is usually handled by the Comments Management page under the Server link in DB2 Alphablox Administration tab.

See Also
"create()" on page 295

fieldNames
The field names available for all comments in this CommentsBlox.

Data Sources
Relational (for storing comments)

Syntax
Java Methods

```
String[] getFieldNames();    //returns a String array;
```

See Also
"addField()" on page 294

namedCommentSets
The list of available named comment sets. Each CommentsBlox needs to be associated with a relational data source and a collection name in that data source.

Data Sources
Relational (for storing comments)

Syntax
Java Methods

```
String[] getNamedCommentSets();
```

Usage
Before calling `getNamedCommentSets()`, make sure that the data source has been set, and, if the data source definition does not contain a username or password, the username and password are properly set in CommentsBlox. The CommentsBlox must be opened before this method is called. For cell-level comment sets, see "getCellCommentsAddresses()" on page 296.

See Also
"dataSourceName" on page 291, "password" on page 293, "userName" on page 293, "open()" on page 299

open
Specifies if the comment collection is open.

Data Sources
Relational (for storing comments)

Syntax
Java Methods

```
boolean isOpen();
```

password

The password to use to connect to the data source used for storing comments.

Data Sources

Relational (for storing comments)

Syntax

JSP Tag Attribute

```
password="password"
```

Java Methods

```
String getPassword();
void setPassword(String password);
```

where:

Argument	Default	Description
password	The password specified in the data source definition via the DB2 Alphablox Admin Pages.	The password to use to connect to the data source

userName

The username to use to connect to the data source used for storing comments.

Data Sources

Relational (for storing comments)

Syntax

JSP Tag Attribute

```
userName="username"
```

Java Methods

```
String getUserName();
void setUserName(String username);
```

where:

Argument	Default	Description
username	The username specified in the data source definition via the DB2 Alphablox Admin Pages.	The username to use to connect to the data source

Usage

When the collection is first created, this username must have sufficient privileges to create tables and indexes. For retrieving and writing comments, this user must have `connect` privileges to the database, and `select` privilege if the user will only be reading comments. The `insert` and `update` privileges are required if the user will be adding or modifying comments.

CommentsBlox Methods

This section describes CommentsBlox methods that are not associated with a specific property. For the syntax and descriptions of CommentsBlox methods that have a property associated with them, see "CommentsBlox Properties and Associated Methods" on page 290. For client-side API common to Blox, see "Client-Side APIs" on page 31.

addField()

Adds fields for this collection one at a time.

Data Sources

Relational (for storing comments)

Syntax

Java Method

```
void addField(String name, String description);
```

where:

Argument	Description
`name`	The name of the field to add to the database for this collection
`description`	The description of the field to add

Usage

Fields can only be added one at a time. This is usually handled by the Comments Management page under the Server link in DB2 Alphablox Administration tab.

See Also

"create()" on page 295

clearFields()

Clears any field definitions that have been added.

Data Sources

Relational (for storing comments)

Syntax

Java Method

```
void clearFields();    // throws CommentsBloxException
```

Usage

This method is useful in cases where you are creating your own comments administrative functionality. `clearFields()` should be called before you save a new collection. It should not be called on an open CommentsBlox. As soon as a comments collection is created, the fields are fixed and cannot be cleared. Therefore, clearing fields should be done before a comments collection is created. This method is also useful when you need to clean up after a comments collection has failed to be created.

See Also

"deleteField()" on page 295

close()

Closes the comment collection.

Data Sources
Relational (for storing comments)

Syntax
Java Method

```
void close();  //throws CommentsBloxException
```

Usage
This is usually handled by the Comments Management page under the Server link in DB2 Alphablox Administration tab.

See Also
"open()" on page 299

create()

Creates the collection tables in the database.

Data Sources
Relational (for storing comments)

Syntax
Java Method

```
void create();   //throws CommentsBloxException
```

Usage
This should be done only once as a setup step for the application, and is usually handled by the Comments Management page under the Server link in DB2 Alphablox Administration tab. Before calling `create()`, add any fields and/or dimensions to this object. After the collection has been created, it cannot be changed.

See Also
"dimensions" on page 291; "addField()" on page 294

delete()

Deletes the comment collection from the database.

Data Sources
Relational (for storing comments)

Syntax
Java Method

```
void delete();
```

Usage
This is usually handled by the Comments Management page under the Server link in DB2 Alphablox Administration tab.

deleteField()

Deletes an existing field from this collection.

Data Sources
Relational (for storing comments)

Syntax
Java Method

```
void deleteField(String name);   // throws CommentsBloxException
```

where:

Argument	Description
name	The name of the field to delete from the database for this collection.

See Also
"clearFields()" on page 294, "addField()" on page 294

getCellCommentsAddresses()
Gets the collection of all cell comments addresses that match the specified dimension and member map.

Data Sources
Relational (for storing comments)

Syntax
Java Method

```
CommentSetAddress[] getCellCommentsAddresses(CommentsBlox.Query query);
                                    // throws CommentsBloxException
```

where:

Argument	Description
query	A Map between the dimension name and the possible members, either as a set or a list of Strings. Passing in an empty Map will result in returning all comment addresses.

Usage
The CommentsBlox.Query inner class has two methods to set and add the dimension/member map with which the cell-level comments collection is associated. See "CommentsBlox.Query Inner Class" on page 301 for its methods. To get the address for named comment sets, use "namedCommentSets" on page 292.

getCollectionName()
Gets the collection name.

Data Sources
Relational (for storing comments)

Syntax
Java Method

```
String getCollectionName();
```

See Also
"setCollectionName()" on page 300

getCollectionNames()

Returns all the collection names for the given data source.

Data Sources

Relational (for storing comments)

Syntax

Java Method

```
String[] getCollectionNames(String dataSourceName,
                            String username,
                             String password);
```

where:

Argument	Description
dataSourceName	The data source name
username	The user name to use to connect to the data source
password	The password to use to connect to the data source

Usage

Set the username and password to null if you wish to use the default username and password found in the data source definition.

getCommentComparator()

Returns the CommentComparator object for this CommentsBlox.

Data Sources

Relational (for storing comments)

Syntax

Java Method

```
CommentComparator getCommentComparator();
```

See Also

"The CommentComparator Object" on page 305, "setCommentComparator()" on page 300

getCommentSet()

Returns the CommentSet object with the comments at this address.

Data Sources

Relational (for storing comments)

Syntax

Java Methods

```
CommentSet getCommentSet(String name);
CommentSet getCommentSet(CommentSetAddress address);
```

where:

Argument	Description
name	The namespace for which to return the CommentSet object.
address	A CommentSetAddress object.

Usage
If there are no comments, `CommentSet.getComments()` will return an array of length 0. This is only used for named comment sets.

See Also
"The CommentSetAddress Object" on page 309

getFieldDescription()
Gets the field description for the given field name.

Data Sources
Relational (for storing comments)

Syntax
Java Method

```
String getFieldDescription(String fieldName);
```

where:

Argument	Description
`fieldName`	The name of the field to get description

See Also
"fieldNames" on page 292

getProperty()
This is a common Blox method. For a complete description, see "getProperty()" on page 53.

hasComments()

Returns `true` if there are comments for the named collection.

Data Sources
Relational (for storing comments)

Syntax
Java Method

```
boolean hasComments(name);
```

where:

Argument	Description
`name`	The name of the comment collection

init()
This is a common Blox method. For a complete description, see "init()" on page 54.

isProtectedField()
Returns true if the field name is one of the reserved comment fields—Author, Timestamp, and CommentText.

Data Sources
Relational (for storing comments)

Syntax

Java Method

```
boolean isProtectedField(String fieldName);
```

where:

Argument	Description
`fieldName`	The name of the comment field. Comment field names are not case-sensitive.

open()

Opens an existing collection.

Data Sources

Relational (for storing comments)

Syntax

Java Method

```
void open();    //throws CommentsBloxException
```

Usage

After setting the data source and optionally the username and password using the `<jsp:useBean>` syntax, you must open the CommentsBlox. It is not bound to a database until this method is called.

Examples

```
<!-- Get the BloxContext from the session -->
<%
BloxContext bc = (BloxContext) Session.getAttribute( BloxContext.BLOX_CONTEXT_ATTR);
%>

<jsp:useBean id="myCommentsBlox"
    class="com.alphablox.blox.CommentsBlox" scope="session" />
    <%
        commentsBlox.init(bc, "myCommentsBlox")
        commentsBlox.setCollectionName(collectionName);
        commentsBlox.setDataSourceName(dataSourceName);
        commentsBlox.setUserName(username);
        commentsBlox.setPassword(password);
        commentsBlox.open();
    %>
</jsp:useBean>
```

performCleanUp()

Performs maintenance on the collection tables in the database.

Data Sources

Relational (for storing comments)

Syntax

Java Method

```
void performCleanUp();   //throws CommentsBloxException
```

Usage

Maintenance should be performed periodically on the tables, especially if large numbers of comments are deleted as this will leave orphan addresses in the

address tables and orphan members in the members table. The database user performing this operation must have `delete` permission on the tables in the database.

An `CommentsBloxException` will be thrown if the collection is closed or an SQL error occurs.

replaceDimensions()

Replaces the defined dimensions in an existing comments collection.

Data Sources

Relational (for storing comments)

Syntax

Java Method

```
void replaceDimensions(String[] dimensions);
                 // throws CommentsBloxException
```

where:

Argument	Description
`dimensions`	A String array of the unique names for dimensions in your MDB data source.

Usage

The CommentsBlox needs to be previously created and the comments collection must not contain comments. Use the `setDimensions()` method (see "dimensions" on page 291) when creating a new comments collection.

setCollectionName()

Sets the collection name.

Data Sources

Relational (for storing comments)

Syntax

Java Method

```
void setCollectionName(String name);   // throws CommentsBloxException
```

where:

Argument	Description
`name`	The name of the comments collection.

See Also

"getCollectionName()" on page 296

setCommentComparator()

Sets a CommentComparator object for this CommentsBlox.

Data Sources

Relational (for storing comments)

Syntax

Java Method

```
void setCommentComparator(CommentComparator commentComparator);
```

where:

Argument	Description
commentComparator	The name of the comments collection.

See Also

"The CommentComparator Object" on page 305, "getCommentComparator()" on page 297

setProperty()

This is a common Blox method. For a complete description, see "getProperty()" on page 53.

CommentsBlox.Query Inner Class

This inner class lets you specify the dimension and members whose cell-level comments set address you want to access. With the constraint set, you can get all the cell comment sets that match this map using the `CommentsBlox.getCellCommentsAddresses()` method.

For non-cell level comments (named comment set), use "namedCommentSets" on page 292 instead.

addDimensionConstraint()

Adds a dimension/member constraint to the map used to find all cell comments that match the constraint set in the map.

Data Sources

Relational (for storing comments)

Syntax

Java Method

```
void addDimensionConstraint(String dimensionName, String memberName);
```

where:

Argument	Description
dimensionName	The name of the dimension the cell-level comments are associated with. This dimension should have been specified in the Comments Management page when the comments collection was created.
memberName	The name of the member in the specified dimensionName the cell-level comments are associated with.

See Also

"getCellCommentsAddresses()" on page 296

setDimensionConstraint()

Sets the dimension/member constraint map for finding cell-level comments that match this constraint.

Data Sources

Relational (for storing comments)

Syntax

Java Method

```
void setDimensionConstraint(String dimensionName, String[] memberNames);
```

where:

Argument	Description
dimensionName	The name of the dimension the cell-level comments are associated with. This dimension should have been specified in the Comments Management page when the comments collection was created.
memberNames	A list of member names in the specified dimensionName the cell-level comments are associated with.

See Also

"getCellCommentsAddresses()" on page 296

The Comment Object

This section describes methods associated with the Comment object. It represents individual comments with associated author, timestamp, comment text, and other information created as custom fields in the collection. To get the individual comments as an array from CommentsBlox, use `CommentsBlox.getCommentSet(name).getComments()`, and add the following code to the beginning of your JSP:

```
<%@ page import="com.alphablox.blox.comments.*"%>
```

The required fields are pre-defined using the following constants:

- `Comment.FIELD_AUTHOR`
- `Comment.FIELD_CELLVALUE`
- `Comment.FIELD_COMMENTTEXT`
- `Comment.FIELD_TIMESTAMP`

getAuthor()

Gets the author of this comment.

Data Sources

Relational (for storing comments)

Syntax

Java Method

```
String getAuthor();
```

Examples

```
String author = comment.getAuthor();
```

getCommentText()

Gets the comment text, the body of the comment.

Data Sources

Relational (for storing comments)

Syntax

Java Methods

```
String getCommentText();
```

getField()

Gets the field value based on the field name.

Data Sources

Relational (for storing comments)

Syntax

Java Methods

```
String getField(String name);
```

where:

Argument	Description
`name`	The name of the field

See Also

"setField()" on page 305

getFields()

Gets an immutable map with the fields attached to this comment.

Data Sources

Relational (for storing comments)

Syntax

Java Methods

```
java.util.Map getFields();    //returns the fields as a Map
```

Usage

The map can not be changed.

getTimestamp()

Gets the comment's timestamp, which is the moment when the comment was saved to the database.

Data Sources

Relational (for storing comments)

Syntax

Java Methods

```
String getTimestamp();
```

getTimestampDate()

Returns the comment's timestamp as a java.util.Date object, which is the moment when the comment was saved to the database.

Data Sources

Relational (for storing comments)

Syntax

Java Methods

```
java.util.Date getTimestampDate();
```

isChanged()

Returns `true` if the comment has been changed since acquired from a CommentSet. New comments are marked as changed.

Data Sources

Relational (for storing comments)

Syntax

Java Methods

```
boolean isChanged();
```

Usage

If the comment has been changed, it should be updated using `CommentSet.updateComment()`.

See Also

"updateComment()" on page 308

setAuthor()

Sets the author field of this comment.

Data Sources

Relational (for storing comments)

Syntax

Java Methods

```
void setAuthor(String name);
```

where:

Argument	Description
`name`	The author name

setCommentText()

Sets the comment text for this comment.

Data Sources

Relational (for storing comments)

Syntax

Java Methods

```
void setCommentText(String commentText);
```

where:

Argument	Description
`commentText`	The comment to add

Usage

The comment text is not allowed to be null. If the comment text is never set or if null is passed in, the value is set to an empty string. There is no limit on the size of the comment.

setField()

Sets a field value.

Data Sources

Relational (for storing comments)

Syntax

Java Methods

```
void setField(String name, String value);
```

where:

Argument	Description
`name`	The field name
`value`	The value for this field

Usage

The `FIELD_TIMESTAMP` field is a special field and its value cannot be set using this method.

The CommentComparator Object

This section describes methods associated with the CommentComparator object. Use this object in conjunction with a CommentsBlox to sort a CommentSet based on a specific field. To get to the CommentComparator object from CommentsBlox, use `CommentsBlox.getCommentComparator()`, and add the following code to the beginning of your JSP:

```
<%@ page import="com.alphablox.blox.comments.*"%>
```

CommentComparator()

Creates a CommentComparator object.

Data Sources

Relational (for storing comments)

Syntax

Java Method

```
CommentComparator(String field, int sortOrder);
        // throws IllegalArgumentException if the sort order is invalid
```

where:

Argument	Description
`field`	A String representing the comment field whose

	values to compare against. The String should be specified using the one of the following constants: • Comment.FIELD_AUTHOR • Comment.FIELD_CELLVALUE • Comment.FIELD_COMMENTTEXT • Comment.FIELD_TIMESTAMP
sortOrder	An integer representing the sort order. Valid constants representing the sort order are: • CommentComparator.DESCENDING • CommentComparator.ASCENDING

Usage

You can use the nested <blox:sortComments> tag to specify the field and sort order:

```
<blox:comments ...>
   <blox:sortComments
       field = "<%= Comment.FIELD_COMMENTTEXT %>"
       order = "<%= CommentComparator.DESCENDING %>" />
</blox:comments>
```

Examples

See "Example 2: Specifying Field to Sort On and Sort Order" on page 284.

compare()

Compares two Comments objects.

Data Sources

Relational (for storing comments)

Syntax

Java Methods

```
int compare(java.lang.Object comment1, java.lang.Object comment2);
```

where:

Argument	Description
comment1	The first Comment object.
comment2	The second Comment object.

Usage

If the specified field value for the first comment comment1 comes before the specified field value for the second comment (comment2):

- Returns > 0 if sortOrder is CommentComparator.DESCENDING
- Returns < 0 if sortOrder is CommentComparator.ASCENDING

If the specified field value for comment1 comes after the specified field value for comments2:

- Returns < 0 if sortOrder is CommentComparator.DESCENDING
- Return > 0 if sortOrder is CommentComparator.ASCENDING

If the specified field value for comment1 equals the specified field value for comment2, this method returns 0. If the specified field value is null, it is represented as a "" for comparison purposes.

getField()

Returns the name of the field whose values are to be compared.

Data Sources

Relational (for storing comments)

Syntax

Java Method

```
String getField();
```

getOrder()

Returns the sort order.

Data Sources

Relational (for storing comments)

Syntax

Java Method

```
int getOrder();
```

Usage

The returned integer should be compared to the constants: `CommentComparator.ASCENDING` and `CommentComparator.DESCENDING`.

The CommentSet Object

This section describes methods associated with the CommentSet object. It is the interface through which comments are added, deleted, and updated to the comment collection. To get to the CommentSet object from CommentsBlox, use `CommentsBlox.getCommentSet(`*name*`)`, and add the following code to the beginning of your JSP:

```
<%@ page import="com.alphablox.blox.comments.*"%>
```

addComment()

Adds a new comment to this CommentSet.

Data Sources

Relational (for storing comments)

Syntax

Java Methods

```
Comment addComment(Comment comment);
        //returns a new Comment object
        //throws CommentsBloxException
```

where:

Argument	Description
`comment`	The comment to add

Usage

The `Comment` object returned is not necessarily the same object as the one added, although the information contained in the new `Comment` object will be identical to that of the input object.

deleteComment()

Deletes the comment passed in.

Data Sources

Relational (for storing comments)

Syntax

Java Methods

```
void deleteComment(Comment comment);
        //throws CommentsBloxException
```

where:

Argument	Description
comment	The Comment object returned from a call to getComments().

See Also

"getComments()" on page 308

getAddress()

Returns the address of this CommentSet as a CommentSetAddress object.

Data Sources

Relational (for storing data)

Syntax

Java Method

```
CommentSetAddress getAddress();   // throws CommentsBloxExcpetion
```

See Also

"The CommentSetAddress Object" on page 309

getComments()

Returns an array of individual Comment objects, one for each comment in this set.

Data Sources

Relational (for storing comments)

Syntax

Java Methods

```
Comment[] getComments();    //returns an array of Comment objects
```

Usage

If there are no comments available, an array of length zero is returned.

See Also

"The Comment Object" on page 302

updateComment()

Updates the `Comment` in this `CommentSet` with new values. The comment must be a reference to a comment retrieved from getComments().

Data Sources
Relational (for storing comments)

Syntax
Java Methods

```
void updateComment(Comment comment);
```

where:

Argument	Description
comment	The comment to modify

The CommentSetAddress Object

This section describes methods associated with the CommentSetAddress object. To get to the CommentSetAddress object from CommentsBlox, use `CommentsBlox.getCommentsSet().getAddress()`, and add the following code to the beginning of your JSP:

```
<%@ page import="com.alphablox.blox.comments.*"%>
```

getAddressName()

Gets the name associated with the named comment set.

Data Sources
Relational (for storing comments)

Syntax
Java Method

```
String getAddressName();
```

Usage
Returns null if it is a cell comment set address.

getDimensionMember()

Gets the member name for the given dimension name from this address.

Data Sources
Relational (for storing comments)

Syntax
Java Method

```
String getDimensionMember(String dimensionName);
```

where:

Argument	Description
dimensionName	The unique name of the dimension to get the member name of.

Usage
Returns null if this is for named comment sets or if the dimension name is not defined in the comments collection.

See Also
"setDimensionMember()" on page 310

getDimensions()

Gets the dimension names defined in the comments collection.

Data Sources

Relational (for storing comments)

Syntax

Java Method

```
String[] getDimensions();
```

isNamedAddress()

Returns `true` if this comment set address represents a named comment set.

Data Sources

Relational (for storing comments)

Syntax

Java Method

```
boolean isNamedAddress();
```

setDimensionMember()

Reassigns the given dimension name member to the given member name.

Data Sources

Relational (for storing comments)

Syntax

Java Method

```
void setDimensionMember(String dimensionName, String memberName);
                        // throws CommentsBloxException
```

where:

Argument	Description
`dimensionName`	The unique dimension name to associate with the member name.
`memberName`	The unique member name to assign to the dimension.

Usage

This does not change the original comment set's address. If the dimension name is not defined in the comments collection, the call is ignored.

See Also

"getDimensionMember()" on page 309

Chapter 10. ContainerBlox Reference

This chapter contains reference material for ContainerBlox properties, methods and objects. For information on how to use this reference, see Chapter 1, "Using This Reference," on page 1.

ContainerBlox Overview

ContainerBlox is an empty Blox with no pre-defined behaviors. DataBlox and all presentation Blox (PresentBlox, GridBlox, ChartBlox, ToolbarBlox, PageBlox, and DataLayoutBlox) are extensions of ViewBlox, which is an extension of ContainerBlox. ContainerBlox allows you to create an area on the page to utilize any user interface components the DB2 Alphablox provides, such as menus and buttons.

You can create your own custom Blox by extending the ContainerBlox to take advantage of the services it provides, such as no page refreshes, full server-side control and logic, use of all core components (for example, the Tree control, menus, and toolbars), same user interface programming model, easy distribution, localizable resources files, and dialogs. Java developers can extend the ContainerBlox to add controls within a Blox, such as adding a drop-down list to the DataLayoutBlox. They can also create reusable, self-contained extensions to the built-in Blox. For example, Java developers may add a color picker dialog to allow users to assign colors to pie slices in a pie chart by utilizing the existing ChartBlox color properties.

ContainerBlox JSP Custom Tag Syntax

The Alphablox Tag Libraries provides custom tags to use in a JSP page for creating each blox. This section describes how to create the custom tag to create a ContainerBlox. For a copy and paste version of the tag with all the attributes, see "ContainerBlox JSP Custom Tag" on page 884.

Syntax

```
<blox:container
    [attribute="value"] >
</blox:container>
```

where:

attribute is one of the attributes listed in the attribute table.

value is a valid value for the attribute.

and where the attributes are one of the following:

Attribute
id
bloxName
enablePoppedOut
height
poppedOut
poppedOutHeight
poppedOutTitle
poppedOutWidth
render
visible
width

Usage

Each custom tag can have one or more attributes, separated by one or more space or new line characters. Any extra space or new line characters are ignored. For readability, attributes can each go on a separate line with the same indenting.

Examples

```
<blox:container id="myContainer"
    width="200"
    height="100"
/>
```

ContainerBlox Properties and Associated Methods

This section describes the properties supported by ContainerBlox and the methods associated with those properties. The properties are listed alphabetically by property name. For a list of ContainerBlox methods with which no properties are associated, see “ContainerBlox Methods” on page 317.

Since most Blox inherits from ContainerBlox, the properties listed here can also be found in “Properties and Associated Methods Common to Multiple Blox” on page 32.

id

This is a common Blox tag attribute. For a complete description, see “id” on page 39.

applicationName

This is a common Blox property. For a complete description, see “applicationName” on page 32.

bloxModel

The current UI model in effect for this Blox.

Data Sources

All

Syntax

Java Method

```
BloxModel getBloxModel();
                    // throws ServerBloxException
```

Usage

The UI model is used by the DHTML client to render the Blox to the browser. This is the base model used to represent the current visual state of the ViewBlox-derived Blox, including PresentBlox, GridBlox, ChartBlox, PageBlox, and DataLayoutBlox. This model is for the Blox frame which includes the built in toolbars, menus, and other components found on all Blox. Use this model to modify the appearance and behavior of the Blox menu, toolbars, and body section. Changes made to the model before and after the render will be reflected to the user. For details, see the com.alphablox.blox.uimodel package in the DB2 Alphablox javadoc. This package is not documented in this book, but its overall object model and potential use for Java developers are discussed in the *Developer's Guide*.

See Also

The com.alphablox.blox.uimodel package in the javadoc; the Blox UI Model in the *Developer's Guide*.

bloxName

This is a common Blox property. For a complete description, see "bloxName" on page 35.

enablePoppedOut

Specifies whether a Blox can be opened in a separate window; that is, "popped out" of the application page.

Data Sources

All

Syntax

JSP Tag Attribute

```
enablePoppedOut="enablePoppedOut"
```

Java Methods

```
boolean isEnablePoppedOut();
       throws ServerBloxException
void setEnablePoppedOut(boolean enablePoppedOut);
       throws ServerBloxException
```

where:

Argument	Default	Description
`enablePoppedOut`	`true`	A boolean argument. A value of `true` specifies that a user can pop out a blox to another window, a value of `false` disables this feature.

Usage

By default, `enablePoppedOut` is set to `true`, and users can have the Blox display in a popped out browser window by clicking the Popped Out button in the toolbar or selecting the "Popped Out" option from the View menu. When `enablePoppedOut` is

set to `false`, this button and menu option are disabled. To remove the button and the menu item, use the Blox UI tags to remove the UI component. See "Menubar, Menu, and MenuItem" on page 816.

poppedOut and its related properties apply to PresentBlox and standalone GridBlox/ChartBlox.

Examples

```
<blox:present id="myPresentBlox"
   enablePoppedOut="false"
   ...>
   ...
</blox:present>
```

See Also

"poppedOut" on page 314, "poppedOutHeight" on page 315, "poppedOutTitle" on page 315, "poppedOutWidth" on page 316

height

This is a common Blox property. For a complete description, see "height" on page 38.

lastAppliedApplicationStateName

This is a common Blox property. For a complete description, see "lastAppliedApplicationStateName" on page 40.

poppedOut

Specifies whether the Blox is to display in a separate window, or "popped out" of the application page when the Blox is loaded.

Data Sources

All

Syntax

JSP Tag Attribute

```
poppedOut="popOut"
```

Java Methods

```
boolean isPoppedOut();
void setPoppedOut(boolean popOut);
```

where:

Argument	Default	Description
`popOut`	`false`	A boolean argument. A value of `true` specifies that the Blox displays in a popped out window when the Blox is loaded. A value of `false` indicates that the Blox displays in the same window as the page.

Usage

Applies to a PresentBlox, a standalone GridBlox, or a standalone ChartBlox.

Examples

```
<blox:present id="myPresentBlox"
   poppedOut="true"
   poppedOutHeight="800"
   poppedOUtWidth="1000"
   poppedOutTitle="Sales Analysis Window"
   ...>
   ...
</blox:present>
```

See Also

"enablePoppedOut" on page 313, "poppedOutHeight" on page 315, "poppedOutTitle" on page 315, "poppedOutWidth" on page 316

poppedOutHeight

Specifies the height (in pixels) of the Blox in the separate, or popped out, window.

Data Sources

All

Syntax

JSP Tag Attribute

```
poppedOutHeight="newHeight"
```

Java Methods

```
int getPoppedOutHeight();
               throws ServerBloxException
void setPoppedOutHeight(int newHeight);
               throws InvalidBloxPropertyValueException,
                       ServerBloxException
```

where:

Argument	Default	Description
newHeight	600	Integer specifying the popped-out window height in pixels.

Examples

```
<blox:present id="myPresentBlox"
   poppedOutHeight="800"
   poppedOUtWidth="1000"
   poppedOutTitle="Sales Analysis Window"
   ...>
   ...
</blox:present>
```

See Also

"enablePoppedOut" on page 313, "poppedOut" on page 314, "poppedOutTitle" on page 315, "poppedOutWidth" on page 316

poppedOutTitle

Specifies the title of the separate, or popped out, window in which the Blox is displayed.

Data Sources

All

Syntax

JSP Tag Attribute

```
poppedOutTitle="title"
```

Java Methods

```
String getPoppedOutTitle();
void setPoppedOutTitle(String title);
```

where:

Argument	Default	Description
title	Popout Blox Window	Any string. The specified string is displayed as the title of the popped out window.

Usage

The default displays the name of the applet as the window title.

Examples

```
<blox:present id="myPresentBlox"
   poppedOutHeight="800"
   poppedOUtWidth="1000"
   poppedOutTitle="Sales Analysis Window"
   ...>
   ...
</blox:present>
```

See Also

"enablePoppedOut" on page 313, "poppedOut" on page 314, "poppedOutHeight" on page 315, "poppedOutWidth" on page 316

poppedOutWidth

Specifies the width, in pixels, of the Blox in the separate, or popped out, window.

Data Sources

All

Syntax

JSP Tag Attribute

```
poppedOutWidth="newWidth"
```

Java Methods

```
int getPoppedOutWidth();
               throws ServerBloxException
void setPoppedOutWidth(int newWidth);
               throws InvalidBloxPropertyValueException,
                       ServerBloxException
```

where:

Argument	Default	Description
newWidth	800	Integer specifying the popped-out window width in pixels.

Usage

The `setPoppedOutWidth` method has no effect if the Blox is already popped out.

Examples

```
<blox:present id="myPresentBlox"
   poppedOutHeight="800"
   poppedOUtWidth="1000"
   poppedOutTitle="Sales Analysis Window"
   ...>
   ...
</blox:present>
```

See Also

"enablePoppedOut" on page 313, "poppedOut" on page 314, "poppedOutHeight" on page 315, "poppedOutTitle" on page 315

propertyNames

This is a common Blox property. For a complete description, see "propertyNames" on page 43.

readEnabled

This is a common Blox property. For a complete description, see "readEnabled" on page 44.

render

This is a common Blox property. For a complete description, see "render" on page 45.

visible

This is a common Blox property. For a complete description, see "visible" on page 46.

width

This is a common Blox property. For a complete description, see "width" on page 47.

writeEnabled

This is a common Blox property. For a complete description, see "writeEnabled" on page 48.

ContainerBlox Methods

This section describes ContainerBlox methods that are not associated with a specific property. For the syntax and descriptions of ContainerBlox methods that have a property associated with them, see "ContainerBlox Properties and Associated Methods" on page 312.

getProperty()

This is a common Blox method. For a complete description, see "getProperty()" on page 53.

getServerContextPath()

This is a common Blox method. For a complete description, see “getServerContextPath()” on page 53.

init()

This is a common Blox method. For a complete description, see “init()” on page 54.

render()

This is a common Blox method. For a complete description, see “render()” on page 56.

renderHtmlHeader()

This is a common Blox method. For a complete description, see “renderHtmlHeader()” on page 57.

setInitialProperty()

This is a common Blox method. For a complete description, see “setInitialProperty()” on page 60.

setProperty()

This is a common Blox method. For a complete description, see “setProperty()” on page 61.

Chapter 11. DataBlox Reference

This chapter contains reference material for the DataBlox properties and methods. For general reference information about Blox, see Chapter 3, "General Blox Reference Information," on page 17. For information on how to use this reference, see Chapter 1, "Using This Reference," on page 1.

DataBlox Overview

DataBlox offers the following functionality:

- provides a representation of a data set (in grid form), either relational or multidimensional, for the assembler to access
- enables application scripting (such as executing a query)
- serves as a data source for other Blox (such as ChartBlox or GridBlox)

DataBlox not only provides a means to access and query data, but also returns a ResultSet object and a MetaData object. The result set returned involves actual data values from a query and enables you to perform tasks such calculations or custom data view. The metadata object contains information on the cubes, dimensions, and members (outline) of the data source. For more information on the DataBlox object model, see "DataBlox—Access to Metadata and Result Sets" on page 7.

To use the APIs associated with RDBMetaData and RDBResultSet, you need to import the com.alphablox.blox.data.rdb package in your JSP page:

```
<%@ page import="com.alphablox.blox.data.rdb.*" %>
```

To use the APIs associated with MDBMetaData and MDBResultSet, you need to import the com.alphablox.blox.data.mdb package in your JSP page:

```
<%@ page import="com.alphablox.blox.data.mdb.*" %>
```

DataBlox JSP Custom Tag Syntax

The Alphablox Tag Libraries provides custom tags to use in a JSP page for creating each blox. This section describes how to create the custom tag to create a DataBlox. For a copy and paste version of the tag with all the attributes, see "DataBlox JSP Custom Tag" on page 885.

Syntax

```
<blox:data
      [attribute="value"] >
</blox:data>
```

where:

attribute is one of the attributes listed in the attribute table.

value is a valid value for the attribute.

and where the attributes are one of the following:

Attribute
id
bloxName
bloxRef
aliasTable
applyPropertiesAfterBookmark
autoConnect
autoDisconnect
bookmarkFilter
calculatedMembers
catalog
columnSort
connectOnStartup
dataSourceName
dimensionRoot
drillDownOption
drillKeepSelectedMember
drillRemoveUnselectedMembers
enableKeepRemove
enableShowHide
hiddenMembers
hiddenTuples
leafDrillDownAvailable
memberNameRemovePrefix
memberNameRemoveSuffix
mergedDimensions
mergedHeaders
onErrorClearResultSet
parentFirst
password
performInAllGroups
query
retainSlicerMemberSet

Attribute
rowSort
schema
selectableSlicerDimensions
showSuppressDataDialog
suppressDuplicates
suppressMissingColumns
suppressMissingRows
suppressNoAccess
suppressZeros
textualQueryEnabled
useAASUserAuthorizationEnabled
useAliases
useOlapDrillOptimization
userName

Usage

Each custom tag can have one or more attributes, separated by one or more space or new line characters. Any extra space or new line characters are ignored. For readability, attributes can each go on a separate line with the same indenting.

You can substitute the closing `</blox:data>` tag using the shorthand notation, closing the tag at the end of the attribute list that looks as follows:

```
useAliases="true" />
```

Examples

```
<blox:data
     dataSourceName="QCC-Essbase"
     query="<ROW (\"All Products\") <ICHILD \"All Products\"
             <COLUMN (\"All Time Periods\") <CHILD \"All Time Periods\"
             <PAGE (Measures) Sales !"
     useAliases="true"
     selectableSlicerDimensions="All Locations" >
</blox:data>
```

When a `data` tag is either nested within the tag of a data presentation Blox (PresentBlox, GridBlox, ChartBlox, DataLayoutBlox, PageBlox, or MemberFilterBlox), it cannot have an `id`. A common practice is to define a standalone DataBlox with an `id` and later reference it in your presentation Blox as follows:

```
<blox:data id="myDataBlox"
     dataSourceName="QCC-Essbase"
     query="<ROW (\"All Products\") <ICHILD \"All Products\"
             <COLUMN (\"All Time Periods\") <CHILD \"All Time Periods\"
             <PAGE (Measures) Sales !"
     useAliases="true"
     selectableSlicerDimensions="All Locations" >
</blox:data>
```

```
<blox:present id="myPresentBlox" >
     <blox:data bloxRef="myDataBlox" />
     ...
</blox:present>
```

This allows you to use the same DataBlox in multiple presentation Blox for synchronized views of the same data, or to access and change the DataBlox property directly using the DataBlox `id` in Java scriptlets.

DataBlox Properties and Methods by Category

The DataBlox properties/Methods category tables list the properties unique to DataBlox, and associated Java methods. For lists of properties and methods common to several Blox, see "Common Blox Properties and Methods by Category" on page 29. The properties and methods supported by DataBlox are organized as follows:

Data Appearance

The following table lists DataBlox properties and methods associated with the appearance of the data.

Properties	Methods
`memberNameRemovePrefix`	`getMemberNameRemovePrefix()` `setMemberNameRemovePrefix()`
`memberNameRemoveSuffix`	`getMemberNameRemoveSuffix()` `setMemberNameRemoveSuffix()`
`mergedDimensions`	`getMergedDimensions()` `setMergedDimensions()`
`mergedHeaders`	`getMergedHeaders()` `setMergedHeaders()`
`showSuppressDataDialog`	`isShowSuppressDataDialog()` `setShowSuppressDataDialog()`
`suppressDuplicates`	`isSuppressDuplicates()` `setSuppressDuplicates()`

suppressMissingColumns	isSuppressMissingColumns() setSuppressMissingColumns()
suppressMissingRows	isSuppressMissingRows() setSuppressMissingRows()
suppressNoAccess	isSuppressNoAccess() setSuppressNoAccess()
suppressZeros	isSuppressZeros() setSuppressZeros()
textualQueryEnabled	isTexutalQueryEnabled() setTextualQueryEnabled()
useOlapDrillOptimization	isUseOlapDrillOptimization() setUseOlapDrillOptimization()

Data Source

The following table lists DataBlox properties and methods associated with data sources.

Property or Method
aliasTable
autoConnect
autoDisconnect
catalog
clearClientCache()
clearResultSet()
connect()
connect(boolean)
dataSourceName
disconnect()
executeNamedDBCalcScript()
generateQuery()
getMetaData()
onErrorClearResultSet
password
query
schema
updateResultSet()
useAASUserAuthorizationEnabled
userName

Data Manipulation

The following table lists DataBlox properties and methods which are used to manipulate data.

Property or Method
clearResultSet()
calculatedMembers
columnSort
dimensionRoot
drillDown()
drillDownOption
drillThrough()
drillKeepSelectedMember
drillRemoveUnselectedMembers
drillToAllDescendants()
drillUp()
enableKeepRemove
enableShowHide
getCalculations()
getDrillThroughReportNames()
hiddenMembers
hiddenTuples
hideMembers()
hiddenTuples
keepOnly()
leafDrillDownAvailable
parentFirst
performInAllGroups
removeColumnSort()
pivot()
retainSlicerMemberSet
removeOnly()
removeRowSort()
rowSort
selectableSlicerDimensions
showMembers()
showTuples()
showOnlyTuples()
getSelectedMembers setSelectedMembers
swapRowAndColumnAxes()
useAliases

Server-side Event Filters and Listeners

The following table lists the methods for capturing events for pre- and post-event processing.

Methods
`addEventFilter()`
`addEventListener()`
`removeEventFilter()`
`removeEventListener()`

Metadata

The following table lists DataBlox methods associated with the metadata (multidimensional and relational) for the result set. To use the APIs associated with the MDBMetaData object, you need to import the com.alphablox.blox.data.mdb package in your JSP page as follows:

```
<%@ page import="com.alphablox.blox.data.mdb.*" %>
```

To use the APIs associated with the RDBMetaData object, you need to import the com.alphablox.blox.data.rdb package in your JSP page as follows:

```
<%@ page import="com.alphablox.blox.data.rdb.*" %>
```

Server-Side Method (Java)
`getMetaData()`
`getMetaData().getDatabaseProductName()`
`getMetaData().getDBVersion()`

Metadata, Multidimensional Database

The following table lists DataBlox methods associated with the multidimensional metadata for the result set.

To use the server-side APIs associated with the MDBMetaData object, you need to import the com.alphablox.blox.data.mdb package in your JSP page as follows:

```
<%@ page import="com.alphablox.blox.data.mdb.*" %>
```

Note: If you are using Microsoft Analysis Services data sources and have merged multiple hierarchies into one dimension (via the mergedDimensions DataBlox property), when working with the MDBMetaData object to access the dimension, you should specify the actual dimension names that are stored in the data source such as [Time].[Calendar] and [Time].[Fiscal]. Since the merged dimension does not actually exist in the data source, using the merged dimension name will result in errors. See “mergedDimensions” on page 367 for more detail.

Method
`getCube()`
`getCube().getDimension()`
`getCube().getDimension().getCube()`
`getCube().getDimension().getDisplayName()`

Method
getCube().getDimension().getRootMember()
getCube().getDimension().getRootMember().getChild()
getCube().getDimension().getRootMember().getChildren()
getCube().getDimension().getRootMember().getDimension()
getCube().getDimension().getRootMember().getDisplayName()
getCube().getDimension().getRootMember(). getGenerationLevel()
getCube().getDimension().getRootMember().getParent()
getCube().getDimension().getRootMember().getUniqueName()
getCube().getDimension().getRootMember().isLeaf()
getCube().getDimension().getRootMembers()
getCube().getDimension().getUniqueName()
getCube().getDimension().getType()
getCube().getDimensions()
getCube().getMultipleHierarchies()
getCube().getMetaData()
getCube().getName()
getCubes()
getNamedDBCalcScriptContents()
getPropertiesOfMember()
resolveDimension()
resolveMember()

Metadata, Relational Database

The following table lists DataBlox methods associated with the relational metadata for the result set.

To use the APIs associated with the RDBMetaData object, you need to import the com.alphablox.blox.data.rdb package in your JSP page as follows:

```
<%@ page import="com.alphablox.blox.data.rdb.*" %>
```

Method
getTable()
getTables()
getTables()
getTable().getColumn()
getTable().getColumns()
getTable().getColumn().getDistinctValues()
getTable().getColumn().getName()
getTable().getColumn().isNumeric()
getTable().getColumn().getType()
getTable().getName()
getTable().getType()

Objects, Result Set and Metadata

This section lists the objects that make up the interfaces to the relational and multidimensional metadata and result sets. The following are the objects cross referenced:

Axis

The following table lists the methods available on the `Axis` object.

Method
`getAxis().getDimension()`
`getAxis().getDimensions()`
`getAxis().getDimensionCount()`
`getAxis().getIndex()`
`getAxis().getResultSet()`
`getAxis().getTupleCount()`
`getAxis().getTuple()`
`getAxis().getTuple()`
`getAxis().getTuples()`

AxisDimension

The following table lists the methods available on the `AxisDimension` object.

Method
getAxis().getDimension().getAxis()
getAxis().getDimension().getDisplayName()
getAxis().getDimension().getIndex()
getAxis().getDimension().getType()
getAxis().getDimension().getUniqueName()

Cells

The following table lists the methods available on the `Cells` object.

Method
getCells().getCell(int).getCommentSet()
getCells().getCell().getCoordinates()
getCells().getCell().getDoubleValue()
getCells().getCell().getIndex()
getCells().getCell().getTuple()
getCells().getCell().getTuples()
getCells().getCell().getValue()
getCells().getCell().hasComments()

Column

The following table lists the methods available on the `Column` object.

Method
getTable().getColumn().getDistinctValues()
getTable().getColumn().getName()
getTable().getColumn().getType()
getTable().getColumn().isNumeric()

Cube

The following table lists the methods available on the `Cube` object.

Method
getCube().getDimension()
getCube().getDimensions()
getCube().getMetaData()
getCube().getMultipleHierarchies()
getCube().getName()

Dimension

The following table lists the methods available on the `Dimension` object.

Method
getCube().getDimension().getCube()
getCube().getDimension().getDisplayName()
getCube().getDimension().getRootMember()
getCube().getDimension().getRootMembers()
getCube().getDimension().getType()
getCube().getDimension().getUniqueName()

Level

The following table lists the methods available on the `Level` object.

Method
getDimension()
getMembers()
getName()
getUniqueName()

MDBMetaData

The following table lists the methods available on the `MDBMetaData` object.

Method
getCube()
getCubes()
getNamedDBCalcScriptContents()
resolveDimension()
resolveMember()

MDBResultSet

The following table lists the methods available on the `MDBResultSet` object.

Method
getAxes()
getAxis()
getAxis()
getAxisCount()
getCells()
getCubes()
getSlicerAxisIndex()
resolveAxisDimension()
resolveTupleMember()

Member

The following table lists the methods available on the `Member` object.

Method
getCube().getDimension().getRootMember(). getAllDescendants()
getCube().getDimension().getRootMember(). getAllLeafDescendants()
getCube().getDimension().getRootMember().getChild()
getCube().getDimension().getRootMember().getChildren()
getCube().getDimension().getRootMember().getDimension()
getCube().getDimension().getRootMember().getDisplayName()
getCube().getDimension().getRootMember(). getGenerationLevel()
getCube().getDimension().getRootMember().getParent()

Method
getCube().getDimension().getRootMember().getUniqueName()
getCube().getDimension().getRootMember().isLeaf()

MetaData

The following table lists the methods available on the `MetaData` object.

Method
getMetaData().getDatabaseProductName()
getMetaData().getDBVersion()

Property

The following table lists the methods available on the `Property` object. Member properties are supported only for Microsoft Analysis Services data sources.

Method
getPropertiesOfMember().getName()
getPropertiesOfMember().getValue()

RDBMetaData

The following table lists the methods available on the `RDBMetaData` object.

Method
getTable()
getTable()
getTables()
getTables()

RDBResultSet

The following table lists the methods available on the `RDBResultSet` object.

Method
exceededMaximumRows()
getColumn()
getColumns()
getNextRow()
getTypes()
getType()
hasMoreRows()
resetCurrentRow()

ResultColumn

The following table lists the methods available on the `ResultColumn` object.

Method
getColumn().getIndex()

Method
getColumn().getName()
getColumn().getType()
getColumn().isNumeric()

ResultSet

The ResultSet object has no methods on it. For details, see “getResultSet()” on page 395.

Table

The following table lists the methods available on the `Table` object.

Method
getTable().getColumn()
getTable().getColumn()
getTable().getColumns()
getTable().getName()
getTable().getType()

Tuple

The following table list the methods available on the `Tuple` object.

Method
getAxis().getTuple().getAxis()
getAxis().getTuple().getIndex()
getAxis().getTuple().getMember()
getAxis().getTuple().getMemberCount()
getAxis().getTuple().getMembers()

TupleMember

The following table list the methods available on the `TupleMember` object.

Method
getAxis().getTuple().getMember().getDimension()
getAxis().getTuple().getMember(). getDisplayName()
getAxis().getTuple().getMember(). getGenerationLevel()
getAxis().getTuple().getMember().getIndex()
getAxis().getTuple().getMember().getSpan()
getAxis().getTuple().getMember().getSpanIndex()
getAxis().getTuple().getMember().getTuple()
getAxis().getTuple().getMember().getUniqueName()
getAxis().getTuple().getMember().isCalculatedMember()
getAxis().getTuple().getMember().isLeaf()

Result Set, Server-Side

The following table lists DataBlox methods associated with the server-side result set containing the data. These methods are only accessible through Java.

Property/Method
clearResultSet()
getRawResultSet()
getResultSet()
“Multidimensional Result Set Methods” on page 407
“Relational Result Set Methods” on page 423
getXMLResultSet()

Result Set, Server-Side—Multidimensional Database

The following table lists DataBlox methods associated with the server-side multidimensional result set. These methods are only accessible through Java.

The following table lists DataBlox methods associated with the server-side multidimensional result set. These methods are only accessible through Java.

To use the server-side APIs associated with the MDBResultSet object, you need to import the com.alphablox.blox.data.mdb package in your JSP page as follows:

```
<%@ page import="com.alphablox.blox.data.mdb.*" %>
```

Method
getAxis()
getAxis()
getAxis().getDimension()
getAxis().getDimension().getAxis()
`getAxis().getDimension().getDisplayName()`
`getAxis().getDimension().getIndex()`
getAxis().getDimension().getType()
getAxis().getDimension().getUniqueName()
`getAxis().getDimensionCount()`
getAxis().getIndex()
getAxis().getResultSet()
getAxis().getTupleCount()
getAxis().getTuple()
getAxis().getTuple()
getAxis().getTuple().getAxis()
getAxis().getTuple().getIndex()
getAxis().getTuple().getMember()
getAxis().getTuple().getMember().getDimension()
`getAxis().getTuple().getMember(). getDisplayName()`
`getAxis().getTuple().getMember(). getGenerationLevel()`

Method
getAxis().getTuple().getMember().getIndex()
getAxis().getTuple().getMember().isLeaf()
getAxis().getTuple().getMember().getSpan()
getAxis().getTuple().getMember().getSpanIndex()
getAxis().getTuple().getMember().getTuple()
getAxis().getTuple().getMember().getUniqueName()
getAxis().getTuple().getMemberCount()
getAxisCount()
getCells()
getCells().getCell()
getCells().getCell(int).getCommentSet()
getCells().getCell().getCoordinates()
getCells().getCell().getDoubleValue()
getCells().getCell().getIndex()
getCells().getCell().getTuple()
getCells().getCell().getTuples()
getCells().getCell().getValue()
getSlicerAxisIndex()
resolveAxisDimension()
resolveTupleMember()

Result Set, Server-Side—Relational Database

The following table lists DataBlox methods associated with the server-side result set for relational data sources. These methods are only accessible through Java.

To use the server-side APIs associated with the RDBResultSet object, you need to import the com.alphablox.blox.data.rdb package in your JSP page as follows:

```
<%@ page import="com.alphablox.blox.data.rdb.*" %>
```

Method
getColumns()
getColumn().getIndex()
getColumn().getName()
getColumn().getType()
getColumn().isNumeric()
getTypes()
getNextRow()
hasMoreRows()

Writeback

The following table lists DataBlox properties and methods associated with creating writeback applications.

Method
commitData()
executeCustomCalc()
executeNamedDBCalcScript()
lockCurrentDataSet()
refresh()
writeback()

Comments

The following table lists the DataBlox method associated with commenting. The method is server-side only.

Method
getCommentsBlox()

Calculations

There are several interfaces associated with the Calculation object. A Calculation object represents a calculated member. These interfaces give you access to parsed calculations defined via DataBlox `calculatedMembers` property. See Calculation Methods for a list of the interfaces.

DataBlox Properties and Associated Methods

This section describes the properties supported by DataBlox and the methods associated with those properties. The properties are listed alphabetically by property name. For a list of DataBlox methods with which no properties are associated, see "DataBlox Methods" on page 385. Common Blox properties available from DataBlox are listed but not described. For complete descriptions of common Blox properties, see "Properties and Associated Methods Common to Multiple Blox" on page 32.

id

This is a common Blox tag attribute. For a complete description, see "id" on page 39.

bloxName

This is a common Blox tag attribute. For a complete description, see "bloxName" on page 35.

bloxRef

This is a common Blox tag attribute. For a complete description, see "bloxRef" on page 38.

aliasTable

Specifies the IBM DB2 OLAP Server or Hyperion Essbase alias table to use with the data source.

Data Sources

IBM DB2 OLAP Server, Hyperion Essbase

Syntax

JSP Tag Attribute

```
aliasTable="aliasTable"
```

Java Methods

```
String getAliasTable();
void setAliasTable(String aliasTable);
```

where:

Argument	Default	Description
aliasTable	empty string	Name of an IBM DB2 OLAP Server or Hyperion Essbase alias table

Usage

The value for `aliasTable` is one of the values provided when defining a data source to DB2 Alphablox. If no `aliasTable` property is specified on a DataBlox, the value is taken from the corresponding data source definition, if the value is present.

The alias tables in the IBM DB2 OLAP Server or Hyperion Essbase database must have names containing only ASCII characters.

See Also

"dataSourceName" on page 356

applyPropertiesAfterBookmark

This is a common Blox property. For a complete description, see "applyPropertiesAfterBookmark" on page 32.

autoConnect

Allows the DataBlox to connect to the database automatically whenever it needs database access.

Data Sources

Relational Only

Syntax

JSP Tag Attribute

```
autoConnect="autoConnect"
```

Java Methods

```
boolean isAutoConnect();
void setAutoConnect(boolean autoConnect);
```

where:

Argument	Default	Description
autoConnect	false	Specify true to enable autoConnect; false to disable it.

Usage

The autoConnect property allows a DataBlox to reconnect to the data source automatically when the application requires a connection. It will not connect to the data source on startup; use the connect() method or the connectOnStartup for this.

You can use autoConnect and autoDisconnect together to create an application where connections are opened only when needed and closed once the requested data is retrieved, without erasing the result set. When autoConnect is enabled and the user performs an operation that requires a data source connection, the DataBlox connects to the data source, restores the current query, and then executes the operation. If autoDisconnect is also enabled, DataBlox disconnects from the data source when the operation is complete.

Important: Connecting to a data source and restoring a query are time-intensive processes. The autoConnect and autoDisconnect properties should be used only in certain cases, such as when the number of connections to a database is limited.

If autoConnect is disabled and autoDisconnect is enabled, the user will not be able to perform operations on the result set after the initial disconnect.

See Also

"autoDisconnect" on page 336, "connect()" on page 387, "connectOnStartup" on page 355, "disconnect()" on page 388

autoDisconnect

Allows the DataBlox to disconnect from the database automatically whenever a data access operation is complete.

Data Sources

Relational; Microsoft Analysis Services

Syntax

JSP Tag Attribute

```
autoDisconnect="autoDisconnect"
```

Java Methods

```
boolean isAutoDisconnect();
void setAutoDisconnect(boolean autoDisconnect);
```

where:

Argument	Default	Description
autoDisconnect	false	Specify true to enable autoDisconnect; false to disable it.

Usage

The autoDisconnect property allows a DataBlox to disconnect from the data source automatically when the application no longer requires a connection, without erasing the current result set. If the connect() method is set to have the DataBlox connect on startup (or the connectOnStartup property is set to true) and autoDisconnect is enabled, the DataBlox will connect, restore a query if applicable, and then disconnect.

You can use autoDisconnect and autoConnect together to create an application where connections are opened only when needed and closed once the requested data is retrieved. When autoDisconnect is enabled, the DataBlox disconnects from the data source whenever an operation requiring a connection is complete. If autoConnect is also enabled, the DataBlox will automatically reconnect to the data source when necessary.

Important: Connecting to a data source and restoring a query are time intensive processes. The autoConnect and autoDisconnect properties should be used only in certain cases, such as when you are experiencing scalability problems with Microsoft Analysis Services due to large client cache memory consumption per connection. In this case, if you have custom code that performs metadata operations such as a for loop with thousands of resolveMember() calls, you should call the clearClientCache() method afterwards to free up the memory. See the Connecting to Data chapter in the *Developer's Guide*.

The autoDisconnect property does not apply to the RDB metadata object on a server-side DataBlox. The DataBlox will not disconnect from an RDB data source after a metadata request, even if autoDisconnect is set to true. The application must explicitly disconnect when it is through with the object. However, you can still use autoConnect to reconnect to the data source on future metadata requests.

If autoDisconnect is enabled and autoConnect is disabled, the user will not be able to perform operations on the result set after the initial disconnect.

See Also

"autoConnect" on page 335, "connect()" on page 387, "connectOnStartup" on page 355, "disconnect()" on page 388

bookmarkFilter

This is a common Blox property. For a complete description, see "bookmarkFilter" on page 33.

calculatedMembers

Specifies a new member that is calculated by DB2 Alphablox using the result set retrieved from the data source. Members used in calculation have to exist in the result set or the calculated member will not show.

Data Sources

All

Syntax

JSP Tag Attribute

```
calculatedMembers="definitionString"
```

Java Methods

```
String getCalculatedMembers();
void setCalculatedMembers(String definitionString);
```

where:

Argument	Default	Description
definitionString	empty string	A comma-delimited list of one or more calculated member definitions. Specify each definition as shown below.

Specify each definition within the definitionString as follows:

dim:*calc*{*refMember*:*gen:*missingIsZero:*drillDownMember*:*drillUpMember*}=
expression{*scopeDim*:*scopeMember*}

where:

definitionString Component	Description
dim	The name of the dimension on which to create a calculated member. **Note:** If you are merging headers using the mergedHeaders property, you should still use the original dimension name rather than the merged dimension header since calculatedMembers is performed before the dimension headers are merged.
calc	The name of calculated member. The name could be the same as an existing member or dimension name since a calculated member has an internal unique name in the form ABXCalc_*dimName_calc*. For example, if a "Total" calculated member is added to the Product dimension, it has a unique name of ABXCalc_Product_Total. This is used internally and the only time it is displayed to the user is when the user turns on unique names in the Dimension Explorer. It never shows in the grid, even when the "use aliases" data option is turned off.
refMember	The name of an existing member, including other calculated members, before which this calculated member is to be placed. Specifying the reference member is optional. You must place double quotes around member names that contain special characters. For example: `calculatedMembers="Product:\"Profit %\"{missingIsZero} = Gross Margin/Sales*100"` The calculated member *calc* will be placed before *refMember* in the grid. If you do not specify a reference member, the calculated member *calc* will be placed in the last row or column. If the user drills on or hides the reference member, the calculated member retains its position. However, if the user removes the reference member, the calculated member moves to the last row or column. See "Example 2: Specifying the position of the calculated member" on page 349.

definitionString Component	**Description**
`gen`	The generation number of the calculated member. Specifying a generation number is optional. Generation can be defined as either absolute or relative to the `refMember`, and it determines the indentation of the calculated member on the axis. The default generation number is 1. To specify absolute generation, define *gen* as a positive integer. A colon is required before the generation number even if there is no reference member defined. To specify relative generation, define *gen* as a + (plus) or - (minus) operator followed by an integer. The calculated member's generation will be the reference member's generation number plus or minus the integer defined in `gen`. A reference member must be defined in order to use relative generation, and the two must be separated by a colon.
`missingIsZero`	Optional keyword (case-insensitive) to use if you want all missing values for members involved in the calculation to be treated as zero. By default, all missing values in the calculation are treated as missing. To change the default behavior, use this special keyword. The following example will treat all missing values in Product1, Product2, and Product3 as zero. `Product:Total Sales{missingIsZero} = Product1 + Product2 + Product3` **Note:** This keyword only affects calculations using member variables. It has no effect on calculation functions.
`drillDownMember`	Optional. The real member to drill down when users try to drill down on this calculated member.
`drillUpMember`	Optional. The real member to drill up when users try to drill up from this calculated member. For example: `Year:Total{:::Qtr3:Jul} = sum()` This would drill down on Qtr3 and drill up on Jul from the calculated member "Total." Notice that when the optional *refMember*, *gen*, and `MissingIsZero` are not specified inside the curly braces, you should include the colons so the *drillDownMember* will not be taken as the *refMember*.

definitionString Component	**Description**

`expression`	The arithmetic expression involving members of *dim* or values from other dimensions. For example,
	
	`calculatedMembers="All Products:Products 1 and 2 = Product1 + Product2"`
	
	adds a calculated member called "Products 1 and 2" in the "All Products" dimension. This expression involves only members from the same dimension where the calculated member is added.
	
	If the calculation involves values from the intersection of multiple dimensions, you should specify the dimension where each member is from by separating the dimension and member name with a colon, and then separate each *dimesnion:member* pair with a semi-colon:
	
	dim1:member1;dim2:member2;...;dimN:memberN
	
	where:
	• at least one dimension is from the row axis
	• at least one dimension is from the column axis
	• for each semicolon delimited *dimension:member* pair, the dimension name has to be supplied, followed by a colon and the member of that dimension
	
	In the following example, a calculated member "Percentage of Total" is added to the All Products dimension with values from the intersection of All Products and All Locations as dividers:
	
	`calculatedMembers="All Products: Percentage of Total= All Products / All Locations:All Locations; All Products:All Products`
	
	See "Example 5: Calculations involving members from different dimensions" on page 350 for more details. For functions supported in calculations, see "Functions for `CalculatedMembers`" on page 341 for a detailed listing and description.

definitionString Component	Description
scopeDim: scopeMember	Defines the dimension and members for which the calculated member is displayed. Additional scope members are separated by commas. Additional pairs are divided within the braces by a semicolon. Specifying a scope is optional. Member names that contain special characters should be enclosed in double quotes (note that you need to escape inner double quotes). When a scope is defined, the calculated member appears only for the members specified in the scope. However, if the calculated member and a scope member are on different axes, intersecting cells that do not fall in the scope are still drawn. Their value is determined by the missingValueString GridBlox property. If no scope is defined, the calculated member is displayed wherever the members involved in the *expression* appear. The following member search functions are available for specifying the level of members the calculated member should be displayed for: • `Leaf()`: the leaf-level descendants of the specified member. Example: `{Market: leaf(East)}` • `Child()`: the children of the specified member. Example: `{Market: child(East)}` • `Descendants()`: all descendants of the specified member. Only one member can be specified in the function. Example: `scope="{Market:descendants(East)}"` • `Gen()`: all members of the specified generation. Example: `{Market: gen(2)}` • `Not()`: members to which the calculation should not be applied. Example: `{Market: not(East, West)}` There is another `Find()` function for finding members that meet the specified criteria. For example, `Find(Sales < 10000)`. The function names are case-insensitive. See the usage discussion of "Scoping" on page 347 and "Example 3: Adding a generation number and scope" on page 349.

Functions for `CalculatedMembers`: You can use functions in your calculation expression. Each function, except `Abs`, `Sqrt`, `Round`, `ifNotNumber`, `Power`, `Rank`, `RunningTotal`, and searching related functions, has the following syntax:

- *functionName(gen(generation))*. Calculate the value based on all the item members that are at a specified generation. Generation 1 means calculating the value based on root members. The example below creates a calculated member called "Standard Deviation" on the Product dimension with its value being the standard deviation of all members at generation 2.

  ```
  Product: Standard Deviation = Stdev(gen(2))
  ```

  Generation 0 means all generations are included in the calculation.
- *functionName(member1, member2, ..., memberN)*. Calculate the value based on the specified members in the dimension where the calculated member is added to. The example below adds a calculated member called "Standard Deviation" on the Product dimension with its value being the standard deviation of CD, Cassette, and TV

  ```
  Product: Standard Deviation = Stdev(CD, Cassette, TV).
  ```

- *functionName(searchFunction(member)).* Calculate the value based on the results from the search functions (`Child`, `Descendants`, `Leaf`, and `find`). The example below adds a calculated member called "Standard Deviation" on the Product dimension with its value being the standard deviation of all children of Audio.

  ```
  Product: Standard Deviation = Stdev(Child(Audio))
  ```

- A function can also take the result from another calculation. The example below adds a calculated member called "Absolute Total Values" on the Product dimension with its value being the absolute value of the total for generation 2 members.

  ```
  Product: Absolute Total Values = Abs(Sum(gen(2)))
  ```

The functions supported are as follows:

- "Arithmetic Functions" on page 342
- "Search Functions" on page 343
- "Special Calculation Functions" on page 344
- "Conditional and Missing Value Related Function" on page 346

Arithmetic Functions:

Arithmetic Functions	Description

`Abs`	Returns the absolute value of a number. This can only be used on a single number item such as the result of another calculation or a single member. For example:
	`Product: Average for Audio = Abs(Average(Audio)`
	`Product: Absolute Value for CD Sales = Abs(CD)`

`Average`	Returns the average of all the numbers in the definition, which is the `sum` divided by `count`. If the `count` is zero, the average is returned as a missing value.
`Count`	Returns the count of all numbers in the definition. Missing values are ignored. If there are no values to count, zero is returned.
`Max`	Returns the highest value in all the numbers in the definition.
`Median`	Returns the value of the number in the middle of the set; that is, half the numbers have values that are greater than the median, and half have values that are less.
`Min`	Returns the lowest value in all the numbers in the definition.
`Power`	Calculates the first parameter raised to the power of the second parameter. The parameters can either be member names or numeric constants.
`Product`	Returns the multiplication of all the values in the definition.
`Round`	Returns the integer part of the number rounded to the nearest whole number. This can only be used

	on a single number item such as the result of another calculation or a single member. For example:
	`Product: Total Sales = Round(Sum(gen(2)))`
	`Product: Rounding value for CD Sales = Round(CD)`

`Sqrt`	Returns the square root of a number. This can only be used on a single number item such as the result of another calculation or a single member. For example:
	`Product: My Calculation 2 = Sqrt(Average(gen(2)))`
	`Product: My Calculatioin 1 = Sqrt(CD)`

`Stdev`	Returns the standard deviation of all the numbers in the definition. The standard deviation is a measure of how widely values are dispersed from the average value (the mean).
`Sum`	Returns the addition of all the numbers in the definition. Missing values are ignored. If there are no values to add, zero will be returned.
`Var`	Returns the variance, which the average squared deviation of each number in the set from the average.

Note: For Microsoft Analysis Services data sources, specify the unique names when using the search functions.

Search Functions:

Search Functions	**Description**
`Child`	Returns all children of the specified member. For example: `Product: Average = Average(Child(Visual))` will calculate the average of all children of Visual. For Microsoft Analysis Services data sources, you must use the full path to the member names. For example, use `child([Time].[Calendar].[All Time Periods].[2000])` rather than `child([2000])`.
`Child`	Returns all descendants of the specified member. For example: `Product: Average = Average(Descendants(Visual))` will calculate the average of all descendants of Visual. For Microsoft Analysis Services data sources, you must use the full path to the member names. For example, use `descendants([Time].[Calendar].[All Time Periods].[2000])` rather than `descendants([2000])`.
`Leaf`	Returns all leaf-level descendants of the specified member. For example: `Product: Average = Average(Leaf(Visual))`

will calculate the average of all leaf members of Visual.

For Microsoft Analysis Services data sources, you must use the full path to the member names. For example, use `leaf([Time].[Calendar].[All Time Periods].[2000])` rather than `leaf([2000])`.

`Find` — Returns all members that meet the search criteria. For example:

```
Sum(Find(All Products:Rank>5)
```

calculates the sum of all whose ranking is lower than 5. Valid comparisons are `>`, `<`, `>=`, `<=`, `=`, and `!=` (`<>` can also be used for not equal tests).

For Microsoft Analysis Services data sources, you must use the full path to the member names.

Special Calculation Functions: There are three special calculation functions: `LookupCount`, `Rank`, and `RunningTotal`.

Special Calculation Functions	Description
`LookupCount`	Similar to Excel's LOOKUP function, this function accepts two parameters (two strings) containing lists of comma separated numbers. The function looks in the first list for the specified value and returns a value from the same position in the second list. This function can be used to create custom statistical functions based on fixed value lookups. For example: D3() could be defined as `LookupCount("2,3,4,5", "8.3, 6.4, 4.3, 2.1")` If there is a count of 2 items with values in the D3() column, then the result would be 8.3. If there are 5 items in the count, then the result of the calculation would be 2.1. If there are more than 5 or less than two, then the result would be NaN (Not a Number). This function can be used to implement statistical functions required in calculated members to create control charts. Control charts are usually employed to visually look for variations to figure out whether a process is in control or out of control.
`Rank`	Returns the values from the specified dimensions in ascending or descending order for the specified member. The syntax for `Rank` is: Rank(*member*, *dimension*, *generation*, *order*, *grouping*, *number*) where: • *member* is the member in this dimension which you want to rank

- *dimension* is the dimension on the opposite axis whose members will be used to generate the rank
- *generation* is the generation of the members of the dimension to be ranked. A generation of 0 means that all members are ranked.
- *order* is either `ASC` for ascending order or `DESC` for descending order. In descending order, the largest number will be ranked 1. In ascending order, the smallest number will be ranked 1.
- *grouping* is optional. If it is present and set to `GROUPDIM`, when there are multiple dimensions on the opposite axis, separate ranking will be done for each grouping of the dimension that is not part of the ranking definition. If it is not set or set to `NOGROUP`, ranking will be performed across groups. This will only have an effect when there is more than one dimension on the axis.
- *number* is optional. Specifies the number of items to rank. For example, specifying a number of 5 means to rank the top 5 members. Note that when this parameter is specified, the grouping parameter (`GROUPDIM` or `NOGROUP`) has to be specified as well.

See "Example 6: Adding ranking" on page 350 and "Example 7: Adding a separate ranking within each group" on page 351.

`RunningTotal`

Returns the accumulative sum of values from the specified dimension for the specified member. The syntax for `RunningTotal` is:

RunningTotal(*member*, *dimension*, *generation*, *grouping*)

where:

- *member* is the member in this dimension which you want to calculate the running totals for
- *dimension* is the dimension on the opposite axis whose members will be used to calculate the running totals
- *generation* is the generation of the members of the dimension to be calculated. A generation of 1 means to only include the root members in the calculation. A generation of 0 means that all members are to be included.
- *grouping* is optional, and if it is present and set to `GROUPDIM`, when there are multiple dimensions on the opposite axis, separate running totals will be done for each grouping of the dimension that is not part of the running total definition. This will only have an effect when there is more than one dimension on the axis.

See “Example 8: Adding running totals within each group” on page 352.

Conditional and Missing Value Related Function:

Conditional and Missing Value Related Functions

`If`

This function takes three parameters, separated by commas:

if(*condition*, *result_if_true*, *result_if_false*)

where *condition* has a left part and a right part, separated by one of the following operators: `<=`, `>=`, `=`, `<`, `>`, or `!=` (`<>` can also be used for not equal tests). For example:

```
Scenario:Act/Bdgt{MissingisZero} =
    If(Budget=0, 0, Actual - Budget*100 / Budget)
```

This will create a calculated member named “Act/Bdgt” in the Scenario dimension. The value for the calculated member will be zero if Budget is zero (or if Budget is missing as the `MissingIsZero` keyword indicates). If Budget is not zero, the value for “Act/Bdgt” will be Actual-Budget*100/Budget.

`ifNotNumber`

By default, missing or null values are treated as missing. You can substitute the function `ifNotNumber` for a member value to provide special case logic to handle missing or null values in the result set used in the calculation. The `ifNotNumber` function has the following syntax:

ifNotNumber(*memberName*,*value*)

where:

- *memberName* is the name of the member in which the function operates on.
- *value* is the numeric value which replaces the missing or null member value. The value specified must contain no commas.

Note: This function works on one member at a time. If you want missing values to be treated as zero for all members involved in calculations, use the keyword `missingIsZero`. See the usage discussion on “Calculation Involving Missing Values” on page 348. For an example, see “Example 4: Replacing missing or null values with the value 0” on page 349.

Usage

The following restrictions apply to calculated members:

- You must place double quotes around member names that contain special characters such as commas.
- You cannot specify relative positions for calculated members in Expand/Collapse mode.

- The values used in the calculation need to be in the result set in order for the added calculated member to display. For example, if a calculation involves generation 3 members, the calculated member will not display unless generation 3 members are in the result set.
- With Microsoft OLAP/Analysis Services and DB2 Alphablox cubes, the syntax must use unique member names.
- With relational data sources, the available "dimensions" are Record # and Columns.
- You cannot use the Keep/Remove option on calculated members, but you can use the Show/Hide option.

A unique name (base name in IBM DB2 OLAP Server or Hyperion Essbase) or display name can be used for the dimension and member names specified in the property's value. This allows you to differentiate between different members or dimensions with the same display names. In IBM DB2 OLAP Server or Hyperion Essbase, you can specify a member, regardless of the alias table in use, by using the base name.

To clear a calculated member, pass an empty string to the `setCalculatedMembers()` method.

Scoping: If you are creating multiple calculated members and want the scoping to either include or exclude other calculated members, the ordering of the calculated members is important and will affect the scope of the displayed results. For example, if you provide a scope to limit a calculated member to calculate only on a specific set of members, any calculated members that are created after the scoped calculated member (that is, calculated members that appear to the right in the *definitionString*) will be included in the initial scoped calculated member. This is because at the time the scoped calculated member is evaluated, the other calculated members do not exist and are therefore not removed from the scope. In these cases, you might want to put the definition for the non-scoped members used in other calculated members before the scoped calculated member. The following two definitions are not equivalent and would produce the results shown:

```
calculatedMembers="All Products: Product1 and Product2=Product1 + Product2
{Measures:Sales,COGS}, Measures:Gross Margin=Sales - COGS"
```

produces the following output:

	Sales	COGS	Gross Margin
Product1	500.00	300.00	200.00
Product2	1500.00	1000.00	500.00
Product1 + Product2	2000.00	1300.00	700.00

whereas:

```
calculatedMembers="Measures:Gross Margin=Sales - COGS, All Products:
Product1 and Product2=Product1 + Product2 {Measures:Sales,COGS}"
```

produces the following:

	Sales	COGS	Gross Margin
Product1	500.00	300.00	200.00
Product2	1500.00	1000.00	500.00
Product1 + Product2	2000.00	1300.00	

The difference between these two examples is in the first example, the scope of the calculated member `Product1 + Product2` will include the calculated member `Gross Margin` (because at the time of scoping, it did not exist); in the second example, the scope of the calculated member `Product1 + Product2` will not include the calculated member `Gross Margin`.

You can also use the member search functions to specify for whether the calculated member should be displayed for all children or all leaf-level descendants of a member or for members of a specific generation. The following example creates a calculated member Difference (the difference between Actual and Budget) on the Scenario dimension for all children of Products.

```
Scenario:Difference  = Actual - Budget {Products: Child(Products)}
```

Calculation Involving Missing Values: When the calculation involves missing values, by default they are treated as missing data and the calculation will fail. When missing data is displayed in a GridBlox, the grid cell will be blank by default. This is because GridBlox has a property called missingValueString, whose default value is an empty string. To treat all missing values as zero in your calculation, use the `missingIsZero` keyword. For example,

```
All Locations:East+Central {West:missingIsZero} = East + Central
```

adds a calculated member called "East+Central" before the member West, and all missing values are treated as 0 in the calculation. This keyword, however, only applies to member variables and has no effects on calculation functions.

If you want to treat missing values as missing for some of the members but not all, use the `ifNotNumber` function for each member that you want missing values to be treated as zero. For example,

```
All Locations:East+Central {West} = ifNotNumber(East,0) + Central
```

adds a calculated member called "East+Central" before the member West, and missing values in East will be treated as 0 while missing values in Central will be treated as missing. As a result, when Central contains missing values, the calculation will fail and return missing, and an empty string will be displayed in those grid cells unless the GridBlox' `missingValueString` property is set otherwise.

Note: If a calculation results in missing values in the entire row or column, then the row or column will not appear at all.

See "Example 4: Replacing missing or null values with the value 0" on page 349.

Examples

Example 1: Adding a calculated member named Profit at the end of the Measures dimension: The value in each cell of the Profit member is derived by subtracting the values in the corresponding cells of the Expenses and Sales members.

```
setCalculatedMembers("Measures : Profit = Sales - Expenses");
```

	Actual		**Budget**	
	East	West	East	West
Sales	*value*	*value*	*value*	*value*
Expenses	*value*	*value*	*value*	*value*

	Actual		Budget	
Profit	*calc value*	*calc value*	*calc value*	*calc value*

Example 2: Specifying the position of the calculated member: You can position the calculated member Profit before the member Expenses in the grid by adding Expenses as a reference member:

```
setCalculatedMembers("Measures : Profit {Expenses} = Sales - Expenses");
```

	Actual		Budget	
	East	West	East	West
Sales	*value*	*value*	*value*	*value*
Profit	*calc value*	*calc value*	*calc value*	*calc value*
Expenses	*value*	*value*	*value*	*value*

Example 3: Adding a generation number and scope:

```
setCalculatedMembers("Measures : Profit {Expenses:2} = Sales - Expenses
{Actual:West}");
```

	Actual		Budget	
	East	West	East	West
Sales	*value*	*value*	*value*	*value*
Profit	*#missing*	*calc value*	*#missing*	*#missing*
Expenses	*value*	*value*	*value*	*value*

Example 4: Replacing missing or null values with the value 0: Assume the following JSP tag:

```
calculatedMembers = "All Locations:East+Central {West:2} = East + Central"
```

This code produces a calculated member called “East+Central” that will be positioned before the member West, with the same level of indentation as generation 2 members. The output is as follows:

All Time Periods	All Locations	Truffles	Chocolate Blocks	Chocolate Nuts
2000	**Central**	6,119	29,068	
	East	883	3,679	
	East+Central	7,002	32,747	
	West	44,029	268,398	
	All Locations	51,031	301,145	

Since there is no data for Chocolate Nuts, there is also no data for the calculated member “East+Central” for Chocolate Nuts.

If we add the `missingIsZero` keyword to treat missing values as zero:

```
calculatedMembers = "All Locations:East+Central {West:3:missingIsZero} =
East + Central"
```

The output generated is as follows:

All Time Periods	All Locations	Truffles	Chocolate Blocks	Chocolate Nuts
2000	**Central**	6,119	29,068	
	East	883	3,679	
	East+Central	7,002	32,747	0
	West	44,029	268,398	
	All Locations	51,031	301,145	

Example 5: Calculations involving members from different dimensions: In the following JSP tag example, a calculated member "Percent Total" is added to the All Products dimension with values calculated by dividing the value of All Products by the intersection of All Products and All Locations:

```
calculatedMembers="All Products: Percent Total=
All Products / All Locations:All Locations; All Products:All Products"
```

The generated output is as follows:

All Time Periods	All Locations	Specialties	Seasonal	All Products	Percent Total
2000	**Central**	72455.09	6849.592	107642.24	0.105
	East	10381.12	1620.36	14943.32	0.015
	West	593387.76	47641	905814.55	0.881
	All Locations	676223.97	56110.95	1028400.11	1
2001	**Central**	855542.96	29691.17	2384096.835	0.183
	East	6288893.64	15333.12	177250.755	0.136
	West	3266694.67	97033.65	8887288.195	0.681
	All Locations	4751131.27	142057.94	13043886.785	1
All Time Periods	**Central**	927998.05	36540.76	2491739.075	0.177
	East	639274.76	16953.48	1787445.075	0.127
	West	3860082.43	144674.65	9793102.745	0.695
	All Locations	5427355.24	198168.89	14072286.895	1

Each value in the Percent Total column is calculated using the values from All Products as the dividends and the values at the intersection of All Products and All Locations (1028400.11 for Year 2000 and 13043886.785 for year 2001) as the dividers.

Example 6: Adding ranking: In this example, a calculated member "Rank" is added to the All Products dimension, with the largest number for generation 2 members ranked first:

```
All Products:Rank = Rank(All Products, All Locations, 2, DESC)
```

The output generated is as follows:

All Time Periods	All Locations	Specialties	Seasonal	All Products	Rank
2000	Central	72455.09	6849.592	107642.24	8
	East	10381.12	1620.36	14943.32	9
	West	593387.76	47641	905814.55	7
	All Locations	676223.97	56110.95	1028400.11	
2001	Central	855542.96	29691.17	2384096.835	4
	East	6288893.64	15333.12	177250.755	6
	West	3266694.67	97033.65	8887288.195	2
	All Locations	4751131.27	142057.94	13043886.785	
All Time Periods	Central	927998.05	36540.76	2491739.075	3
	East	639274.76	16953.48	1787445.075	5
	West	3860082.43	144674.65	9793102.745	1
	All Locations	5427355.24	198168.89	14072286.895	

To rank only the top N or bottom N members, add an integer as the sixth parameter at the end. Note that in order to specify the number of ranking, the fifth parameter (NOGROUP or GROUPDIM) needs to be specified:

```
calculatedMembers="All Products:Rank = Rank(All Products, All Locations, 2,
DESC, NOGROUP, 5)"
```

Example 7: Adding a separate ranking within each group:

```
calculatedMembers = "All Products:Rank =
Rank(All Products, All Locations, 2, DESC, GROUPDIM)"
```

The above example generates the following output:

All Time Periods	All Locations	Specialties	Seasonal	All Products	Rank
2000	Central	72455.09	6849.592	107642.24	2
	East	10381.12	1620.36	14943.32	3
	West	593387.76	47641	905814.55	1
	All Locations	676223.97	56110.95	1028400.11	
2001	Central	855542.96	29691.17	2384096.835	2
	East	6288893.64	15333.12	177250.755	3
	West	3266694.67	97033.65	8887288.195	1
	All Locations	4751131.27	142057.94	13043886.785	
All Time Periods	Central	927998.05	36540.76	2491739.075	2
	East	639274.76	16953.48	1787445.075	3
	West	3860082.43	144674.65	9793102.745	1
	All Locations	5427355.24	198168.89	14072286.895	

A calculated member “Rank” is added to the All Products dimension based on the values of generation 2 members in the All Locations dimension, with the largest number ranked first and separate ranking within each group in the dimension.

Example 8: Adding running totals within each group:

```
calculatedMembers = "All Products:Running Totals =
RunningTotal(All Products, All Locations, 2, GROUPDIM)"
```

The above example generates the following output:

All Time Periods	All Locations	Specialties	Seasonal	All Products	Running Totals
2000	Central	72455	6850	107642	107642
	East	10381	1620	14943	122586
	West	593388	47641	905815	1028400
	All Locations	676224	56111	1028400	
2001	Central	855543	29691	2384097	2384097
	East	6288894	15333	1772502	4156599
	West	3266695	97034	8887288	13043887
	All Locations	4751131	142058	13043887	
All Time Periods	Central	927998	36541	2491739	2491739
	East	639275	16953.48	1787445	4279184
	West	3860082	144675	9793102.745	14072287
	All Locations	5427355	198169	14072287	

A calculated member called "Running Totals" is added to the All Products dimension, which contains the accumulative sum for each group of the generation 2 members on the All Locations dimension.

See Also

"Inputting and Modifying Data" in the *Developer's Guide*.

catalog

Specifies the database catalog for this DataBlox.

Data Sources

All

Syntax

JSP Tag Attribute

```
catalog=catalog
```

Java Methods

```
String getCatalog();
void setCatalog(String catalog);
```

where:

Argument	Default	Description
`catalog`	empty string	Name of a database catalog.

Usage

The value for `catalog` is one of the values provided when defining a data source to Analysis Server. If no `catalog` property is specified on a DataBlox, the value is taken from the corresponding data source definition, if the value is present.

A catalog is known as an "application" in IBM DB2 OLAP Server or Hyperion Essbase terminology.

columnSort

Specifies how to sort data values for members on the column axis.

Data Sources

All

Syntax

JSP Tag Attribute

```
columnSort=sortString
```

Java Methods

```
String getColumnSort();   //returns String of 4 comma-separated items

void   setColumnSort(ResultColumn column, boolean ascending)
void   setColumnSort(Tuple tuple, AxisDimension dimension,
                     boolean ascending);
void   setColumnSort(Tuple tuple, AxisDimension dimension,
                     boolean ascending, boolean preserveHierarchy);
void   setColumnSort(String sortString);
```

where:

Argument	Default	Description
`column`	none	A column in a relational result set.
`ascending`	none	Specify `true` to sort ascending; `false` to sort descending.
`tuple`	none	The tuple on the column axis that specifies the column to be sorted.
`dimension`	none	The dimension on the row axis for which grouping is preserved. Specify `null` to indicate no grouping is to be preserved on the row axis.
`preserveHierarchy`	`false`	Specify `true` to preserve the hierarchy in the row axis, keeping members with their parents after the sort operation; `false` to not preserve hierarchy.

Argument	Default	Description

sortString	none	A comma-delimited string in one of the following formats:
		• *tupleIndex*, *direction*
		• *tupleIndex*, *groupingNestLevel*, *direction*
		• *tupleIndex*, *groupingNestLevel*, *direction*, *preserveHierarchy*
		
		tupleIndex —string representation of an integer, representing the zero-based tuple index member (column) to sort, where `0` indicates the leftmost column.
		
		groupingNestLevel —string representation of an integer, representing the dimension on the row axis for which grouping is preserved. For example, if Product and Market are on the row axis, a value of 1 sorts into sequence within the Market dimension. Specify `-1` to sort without regard to row groupings. The default is `-1`.
		
		direction —a case-insensitive string of either `"Ascending"`, `"Asc"`, `"Descending"` or `"Desc"`.
		
		preserveHierarchy —string representation of a boolean. see the `preserveHierarchy` argument. Defaults to `false`.
		
		For example:
		`setColumnSort("1,0,asc,true");`
		`setColumnSort("1,0,asc");`
		`setColumnSort("0,descending");`

Usage

The `getColumnSort` method returns a string of four comma-separated items: *tupleIndex*, *groupingNestLevel*, *direction*, and *preserveHierarchy*.

The following screen shots show the results of an ascending sort operation on the Qtr1 column depending on 1) whether the hierarchy is preserved , and 2) whether the grouping within the a specified level/dimension is preserved. The first example preserves the hierarchy in the Market dimension, yet not the grouping within the Product dimension.

Product	Market	Qtr1
200	East	562
	New Mexico	-14
	Louisiana	336
	Oklahoma	468
	Texas	675
	South	1465
	West	2325
	Central	2369
	Market	6721
100	West	1042
	New Mexico	-27
	Oklahoma	171
	Louisiana	212
	Texas	695
	South	1051
	Central	2208
	East	2747
	Market	7048

The following example preserves neither the hierarchy in the Market dimension nor the grouping in the Product dimension.

Product	Market	Qtr1
100	New Mexico	-27
200	New Mexico	-14
100	Oklahoma	171
	Louisiana	212
200	Louisiana	336
	Oklahoma	468
	East	562
	Texas	675
100	Texas	695
	West	1042
	South	1051
200	South	1465
100	Central	2208
200	West	2325
	Central	2369
100	East	2747
200	Market	6721
100	Market	7048

Examples

The following example demonstrates the use of the `columnSort` tag attribute:

```
columnSort="1, 0, asc"
```

See Also

"rowSort" on page 374, "removeColumnSort()" on page 401

connectOnStartup

Specifies whether the DataBlox will automatically connect to its data source upon Blox instantiation.

Data Sources

All

Syntax

JSP Tag Attribute

```
connectOnStartup="connectOnStartup"
```

Java Methods

```
boolean isConnectOnStartup();
void setConnectOnStartuo(boolean connectOnStartup);
```

where:

Argument	Default	Description
`connectOnStartup`	`true`	A boolean argument. Specify `true` to automatically connect to the data source when the Blox is instantiated; `false` to not connect.

Usage

To prevent a DataBlox from connecting to a data source, set this property to `false`. When the query property is set later, you need to call the `connect()` method to connect to the data source.

When connectOnStartup is set to true, it overrides the autoConnect property. The connectOnStartup property causes a database connection, even if no query is defined.

See Also

"query" on page 373, "autoConnect" on page 335, "autoDisconnect" on page 336

dataSourceName

Identifies the external data source that this DataBlox accesses.

Data Sources

All

Syntax

JSP Tag Attribute

```
dataSourceName="dataSourceName"
```

Java Methods

```
String getDataSourceName();
void   setDataSourceName(String dataSourceName);
```

where:

Argument	Default	Description
dataSourceName	empty string	A data source name as defined in DB2 Alphablox.

Usage

The data source name must be defined in DB2 Alphablox. If you do not specify a data source, the Blox loads with no initial data source. This feature enables the data source to be set programmatically, perhaps based on user properties or actions. However, to prevent an error message appearing to the user, be sure to set the value of the autoConnect property to false.)

This setDataSourceName() method also reads in the properties of the data source such as username, password, catalog, schema, query, and dimensions on page axis. Therefore, if you want to set any of these properties using the Java methods such as setUserName() and setPassword(), set them after the setDataSourceName() method.

Tip: The order these data source properties are set is not an issue if you use the Blox tags. The tags are designed to enforce that the data source is set before the call to set the other data source properties, the side effect is taken care for you.

Tip: For security reasons, it is not a good idea to provide values for the userName and password properties on an HTML page. For more information on these properties, see "userName" on page 384 property and "password" on page 371. For related information, see also "query" on page 373, "schema" on page 376, "useAASUserAuthorizationEnabled" on page 382, and "catalog" on page 352.

Examples

The following example shows the custom tag attributes with values for an IBM DB2 OLAP Server or Hyperion Essbase data source.

```
dataSourceName="MyEssbaseDataSource"
schema="Basic"
catalog="Demo"
query="<SYM <ROW (Product) <ICHILD Product <COL (Market) <ICHILD Market !"
```

The following example shows the properties with values for an Alphablox cube.

```
dataSourceName="MyAlphabloxCube"
query="SELECT Measures.MEMBERS ON COLUMNS,
         {[Store].[Store State].[CA], [Store].[Store State].[WA]}
         ON ROWS
       FROM [Sales]"
```

The following example shows the properties with values for a Microsoft OLAP Services data source.

```
dataSourceName="MyOLAPDataSource"
catalog="MySchema"
schema="MyCatalog"
query="SELECT Measures.MEMBERS ON COLUMNS,
         {[Store].[Store State].[CA], [Store].[Store State].[WA]}
         ON ROWS
       FROM [Sales]"
```

The following example shows the properties with values for an IBM DB2 UDB data source.

```
dataSourceName="MyDB2"
catalog="MyCatalog"
query="SELECT Actual.SalesQty, Actual.ProductID FROM MyCatalog"
```

The following example shows the properties with values for an Oracle data source.

```
dataSourceName="MyOracleDataSource"
schema="MySchema"
catalog="MyCatalog"
query="SELECT Actual.SalesQty, Actual.ProductID,
          Projected.SalesQty, Projected.ProductID
       FROM Actual, Projected
       WHERE Actual.SalesQty <Projected.SalesQty"
```

The value for the `dataSourceName` string must be a data source already defined to DB2 Alphablox. For information on setting up data sources, see the *Administrator's Guide*.

See Also

"catalog" on page 352, "query" on page 373, "schema" on page 376, "useAASUserAuthorizationEnabled" on page 382

dimensionRoot

Specifies the dimension and a single member to use as the root.

Data Sources

Multidimensional

Syntax

JSP Tag Attribute

```
dimensionRoot="dimensionNameAndNewRootMember"
```

Java Methods

```
String   getDimensionRoot();
Member[] getDimensionRoot(Dimension dimension);
void     setDimensionRoot(String dimensionNameAndNewRootMember);
void     setDimensionRoot(Dimension dimension, Member member);
```

where:

Argument	Default	Description
dimensionNameAndNewRootMember	empty string	A String which specifies the new root member for the specified dimension or dimensions. The String is in the form: `"DimA:NewRootMemberA;DimB:NewRootMemberB;"` If you specify a dimension without a new root member, the dimension root is reset to the database default root member for that dimension.
dimension	none	A Dimension object. If `dimension` appears in a page filter, in the Member Filter, or if a member is `null`, then the database uses its default dimension root. acts as the root of the dimension.
member	none	A Member object. If `member` is `null`, the dimension root resets to the database default.

Usage

If the named dimension appears in a page filter or the Member Filter, the selected member acts as the root of the dimension. If there is conflict between this property value and the query, the query overrides the property.

A unique name (base name in IBM DB2 OLAP Server or Hyperion Essbase) or display name can be used for the dimension and member name string specified in the property's value. This allows assemblers to differentiate between different members or dimensions with the same display names. In IBM DB2 OLAP Server or Hyperion Essbase, an assembler can specify a member, regardless of the alias table in use, by using the base name.

Multiple dimensions can be specified, but only one member per dimension.

The `String getDimensionRoot()` method returns a String of the form:

```
"DimA:RootMemberA;DimB:RootMemberB;"
```

Examples

The following example specifies, as an attribute to the DataBlox custom tag, that the root of the `Products` dimension is `Tools` and the root of the `Market` dimension is `East`.

```
dimensionRoot="Products: Tools;Markets: East"
```

See Also

"query" on page 373, "Multidimensional Metadata Methods" on page 427

drillDownOption

Specifies the type of drill operation to perform.

Data Sources

Multidimensional

Syntax

JSP Tag Attribute

```
drillDownOption="drillDown"
```

Java Methods

```
int     getDrillDownOption();
boolean setDrillDownOption(int drillDown);
```

where:

Argument	Default	Description
drillDown	1	An integer from 1 to 5 specifying the level to drill down to. Possible values are: 1: Drill down to next generation 2: Drill down to all descendants (same as "Expand All") 3: Drill down to bottom generation 4: Drill to siblings 5: Drill to same generation

Usage

For explanations of the terms such as descendants, siblings, and generation, see "OLAP Terms and Concepts" in the *Administrator's Guide*.

Examples

```
setDrillDownOption(4);
```

See Also

"drillDown()" on page 388, "drillKeepSelectedMember" on page 359, "drillRemoveUnselectedMembers" on page 360, "drillToAllDescendants()" on page 390.

drillKeepSelectedMember

Specifies whether the member being drilled on should be retained or removed when the display repaints.

Data Sources

Multidimensional

Syntax

JSP Tag Attribute

```
drillKeepSelectedMember="keepSelected"
```

Java Methods

```
boolean isDrillKeepSelectedMember();
void    setDrillKeepSelectedMember(boolean keepSelected);
```

where:

Argument	Default	Description
keepSelected	true	A boolean argument. Specify true to retain the member being drilled on; false to remove it.

Examples

```
setDrillKeepSelectedMember(false);
```

See Also

"drillRemoveUnselectedMembers" on page 360

drillRemoveUnselectedMembers

Specifies whether to remove all members that are not being drilled on when the display repaints.

Data Sources

Multidimensional

Syntax

JSP Tag Attribute

```
drillRemoveUnselectedMembers="removeUnselected"
```

Java Methods

```
boolean isDrillRemoveUnselectedMembers();
void    setDrillRemoveUnselectedMembers(boolean removeUnselected);
```

where:

Argument	Default	Description
removeUnselected	false	Specify true to remove all members that are not being drilled on, false to keep them.

Examples

```
setDrillRemoveUnselectedMembers(true);
```

See Also

"drillKeepSelectedMember" on page 359

enableKeepRemove

Specifies whether the Keep Only and Remove Only options are available to the end user in the context menus of both the GridBlox and ChartBlox.

Data Sources

All

Syntax

JSP Tag Attribute

```
enableKeepRemove="enable"
```

Java Methods

```
boolean isEnableKeepRemove();
void    setEnableKeepRemove(boolean enable);
```

where:

Argument	Default	Description
enable	false	Specify true to enable the Keep Only and Remove Only options; false to disable them.

Usage
The Keep Only and Remove Only options give end users the ability to control which members and columns are visible in the grid.

Even if you have enabled or disabled the Remove Only and Keep Only option using this property, the end user can enable or disable it by using the "Enable keep and remove" checkbox under the Data tab of the Options dialog.

Examples
```
setEnableKeepRemove(true);
```

See Also
"enableShowHide" on page 361

enableShowHide
Specifies whether the Show Only, Show All, and Hide Only options are available to end users in the context menus of both the GridBlox and ChartBlox.

Data Sources
All

Syntax
JSP Tag Attribute
```
enableShowHide="enable"
```

Java Methods
```
boolean isEnableShowHide();
void    setEnableShowHide(boolean enable);
```

where:

Argument	Default	Description
enable	true	Specify true to enable the Show Only, Show All, and Hide Only options; false to disable them.

Usage
The Show/Hide options give you and the end user the ability to control which members are visible in the grid. It does not replace the Keep/Remove options, although its behavior is similar. You can use the Show/Hide feature in the same places as Keep/Remove.

The end user can turn the Show/Hide feature on and off through the Options dialog box under the Data tab. The options available when it is enabled are to Show or Hide members individually, and to Show All members of a dimension. The user cannot selectively show members once they are hidden, or hide all members. Each dimension must contain one member at all times.

Hidden members continue to appear in the Member Filter. Additionally, saving a bookmark when certain members are hidden preserves their hidden state.

Show/Hide vs. Keep/Remove

The Show/Hide feature differs from the Keep/Remove feature in several ways. The Show/Hide feature preserves the hidden state of members through subsequent GUI operations until they are unhidden through Show All. For example, the user can drill up and back down to the hidden member's level, and the member will stay hidden. This is in contrast to the Keep/Remove functionality, where under the same circumstances the removed member would be revealed.

Show/Hide does not interfere with the output of DB2 Alphablox calculated members. Although a member may be hidden from view, the data is still accessible for use in calculations. When a member is removed using Keep/Remove, the calculated member can no longer access the removed data and will return the value "#missing" (or whatever missing value string you have specified).

Examples

```
setEnableShowHide(true);
```

See Also

"enableKeepRemove" on page 360

hiddenMembers

Specifies which members to hide using the Show/Hide functionality.

Data Sources

All

Syntax

JSP Tag Attribute

```
hiddenMembers="membersToHide"
```

Java Methods

```
String   getHiddenMembers();
void     setHiddenMembers(String membersToHide);

Member[] getHiddenMembers(MDBMetaData MDBMetaData, int i);
Member   getHiddenMembers(MDBMetaData MDBMetaData);
void     setHiddenMembers(Member[] members);

Column[] getHiddenMembers(RDBMetaData RDBMetaData, int i);
Column   getHiddenMembers(RDBMetaData RDBMetaData);
void      setHiddenMembers(Column[] columns);
```

where:

Argument	Default	Description

membersToHide	empty string	List of initially hidden members. Format the string as follows:
		`DimensionName1:MemberNameA,MemberNameB;`
		`DimensionName2:MemberNameD,MemberNameD;`
		`...`
		`DimensionName`*N*`:MemberNameX,MemberNameY`
		
		The semicolon is required to separate members in different dimensions.
		Note: If you merge multiple dimension headers into one using "mergedHeaders" on page 368, since mergedHeaders are performed first, you should use the newly merged headers when specifying the members to hide.

MDBMetaData	none	The MDBMetaData Object.
members	none	An array of members (the Member interface).
RDBMetaData	none	The RDBMetaData Object.
columns	none	An array of columns (the Columns interface).
i	none	Gets or sets the *i*th member in the array.

Usage
One member of each dimension must remain in the grid. If you specify for all members of a dimension to be hidden, the last member specified will not be hidden. It is best to make sure that you do not remove all members of any dimension.

Examples
If you want to hide the members *East* and *North* on the *Market* dimension and *Audio* on the *Product* dimension, you would define the `hiddenMembers` property as follows:

```
hiddenMembers="Market:East,North; Product:Audio"
```

See Also
"enableShowHide" on page 361, "hideMembers()" on page 397, "showMembers()" on page 403

hiddenTuples
Specifies which tuples in the result set to hide using the Show/Hide functionality.

Data Sources
Multidimensional

Syntax
JSP Tag Attribute

```
hiddenTuples="selectedTuples"
```

Java Methods

```
String getHiddenTuples();   // throws ServerBloxException;
void setHiddenTuples(String selectedTuples);
                 // throws ServerBloxException
```

where:

Argument	Default	Description

`selectedTuples`	empty string	List of initially hidden tuples. Format the string as follows:
		`Dimension1,Dimension2,...,DimensionN:`
		`Dim1Member1, Dim2Member1,..,DimNMember1;`
		`Dim1Member2,Dim2Member2,...,DimNMember2;...`
		`Dim1MemberM,Dim2MemberM,...,DimNMemberM`
		
		Each tuple list contains the dimension names separated by commas, followed by a colon, followed by the list of tuples. Each tuple consists of one member from each of the dimensions specified and is separated by a semicolon. Each member of the tuple is separated by a comma.
		
		You can also specify multiple tuple lists, with each list enclosed in curly braces and separated by a comma. This allows you specify tuples from dimensions on the opposite axis in the same syntax. See the Usage section below for details.
		Note: If you merge multiple dimension headers into one using "mergedHeaders" on page 368, since mergedHeaders are performed first, you should use the newly merged headers when specifying the tuples to hide.

Usage

In order for a specified tuple to be hidden, the tuple must exist in the result set.

You can specify tuples from the opposite dimension in the same syntax by enclosing the tuple list from dimensions on one axis in curly braces and separate the list by a comma:

```
{Period,Product:Q1,Audio;Q2,Visual},{Accounts,Market:Profit,East}
```

The above example will generate an output as follows:

		Margin				Profit		
Period	Product	East	West	South	Market	West	South	Market
Q1	Visual	4950.9	23995	15344	44289.9	8042	4474	4373.9
	Product	1461	37890	15344	54495	12917	4474	834
Q2	Audio	9331	13354		22685	4320	0	6852
	Product	28232	36785	14895	79912	11449	4199	22924
Q3	Audio	9390	13745		23135	5324	0	8300
	Visual	22282	24740	15675	62697	8946	5430	22680
	Product	31672	38485	15675	85832	14270	5430	30980

Note: The `setHiddenTuples()` method sets the new list of hidden tuples, overriding any hidden tuple list set earlier. To hide additional tuples, use the hideTuples() method.

See Also

"hideTuples()" on page 398

leafDrillDownAvailable

Specifies whether the user should be allowed to drill down on a leaf member.

Data Sources

Multidimensional

Syntax

JSP Tag Attribute

```
leafDrillDownAvailable="available"
```

Java Methods

```
boolean isLeafDrillDownAvailable();
void    setLeafDrillDownAvailable(boolean available);
```

where:

Argument	Default	Description
`available`	`false`	Specify `true` to enable leaf drill downs; `false` to disable them.

Usage

This property is useful only when you want to perform custom actions when a user drills down on a leaf member. In this case only, you need to set the value to `true`. If you do not need to enable drilling down on a leaf member to call a function, leave this property at its default value of `false`.

memberNameRemovePrefix

Specifies the start point of a member name when returned from the data source.

Data Sources

Multidimensional

Syntax

JSP Tag Attribute

```
memberNameRemovePrefix="prefix"
```

Java Methods

```
String getMemberNameRemovePrefix();
void   setMemberNameRemovePrefix(String prefix);
```

where:

Argument	Default	Description
`prefix`	empty string	Start point of the member name.

Usage

The `memberNameRemovePrefix` property removes the text from a member name string returned from a data source beginning with and including the specified string.

Note: This method only affects the result set, not the metadata. That is, subsequent metadata calls to get the display name of a member will still include the prefix.

This property is of particular use with IBM DB2 OLAP Server or Hyperion Essbase data sources where member names must be unique. Unique names are often created by adding unique strings as suffixes or prefixes on member names. Using this property enables stripping off the prefix strings before displaying the member names.

The removal can only be applied to member names; it cannot be applied to dimension names. Additionally, properties and methods that take member names as arguments will use the unique member name prior to the prefix or suffix removal.

Examples

If the `memberNameRemovePrefix property` is ″##″, the member ″123##Year″ will be displayed as ″Year″.

This property can be used with the `memberNameRemoveSuffix` property. For example, if the `memberNameRemovePrefix` string is ″$$″ and the `memberNameRemoveSuffix` string is ″##″, then the member ″123$$Year##978-9″ will be displayed as ″Year″.

See Also

“memberNameRemoveSuffix” on page 366

memberNameRemoveSuffix

Specifies the end point of a member name when returned from the data source.

Data Sources

Multidimensional

Syntax

JSP Tag Attribute

```
memberNameRemoveSuffix="suffix"
```

Java Methods

```
String getMemberNameRemoveSuffix();
void   setMemberNameRemoveSuffix(String suffix);
```

where:

Argument	Default	Description
`suffix`	empty string	End point of the member name.

Usage

The `memberNameRemoveSuffix` property removes the text from a member name string returned from a data source beginning with and including the specified string.

Note: This method only affects the result set, not the metadata. That is, subsequent metadata calls to get the display name of a member will still include the suffix.

This property is of particular use with IBM DB2 OLAP Server or Hyperion Essbase data sources where member names must be unique. Unique names are often created by adding unique strings as suffixes or prefixes on member names. Using this property enables stripping off the suffix strings before displaying the member names.

The removal can only be applied to member names; it cannot be applied to dimension names. Additionally, properties and methods that take member names as arguments will use the unique member name prior to the prefix or suffix removal.

Examples

If the memberNameRemoveSuffix is ″##″, the member ″Year##978-9″ will be displayed as ″Year″.

This property can be used with the `memberNameRemovePrefix` property. For example, if the `memberNameRemovePrefix` string is ″$$″ and the `memberNameRemoveSuffix` string is ″##″, then the member ″123$$Year##978-9″ will be displayed as ″Year″.

See Also

“memberNameRemovePrefix” on page 365

mergedDimensions

Specifies whether multiple hierarchies of a dimension should be merged in the Other axis in the Data Layout panel and in Member Filter.

Data Sources

Microsoft Analysis Services

Syntax

JSP Tag Attribute

```
mergedDimensions="dimensionString"
```

Java Methods

```
String getMergedDimensions();
void setMergedDimensions(String dimensionString);
```

where:

Argument	Default	Description
`dimensionString`	`null`	A comma-delimited String representing the prefix of the dimensions to merge in the Other axis of the Data Layout panel

Usage

Microsoft Analysis Services supports multiple hierarchies, allowing alternate views of cube data. Multiple hierarchies are two or more dimensions with names that share the same prefix followed by a period but have different suffixes. For example, Time.Calendar and Time.Fiscal are two different dimensions, but if you merge them into a “logical” dimension (which does not actually exist), you can enhance the usability of your application as your users are less likely to be confused.

Once multiple hierarchies are merged, they appear as one dimension in the user interface in the Other axis in the Data Layout panel. For example, if you specify to merge all dimensions with the prefix "Time," Time.Calendar and Time.Fiscal will appear as a "Time" dimension in the Data Layout panel. When a user drags the Time dimension to the Row, Column, or Page axis, a dialog automatically pops up, asking the user to select one of the two hierarchies she wants to use. In Member Filter, all corresponding hierarchies are displayed under the Time dimension, but users can only select from one hierarchy.

Note: When you access the dimensions through methods such as the MDBMetaData object's `resolveDimension()` method, you should specify the actual dimension names ([Time].[Calendar] and [Time].[Fiscal], for example) that are actually stored in the data source. Since the merged dimension does not actually exist in the data source, using the merged dimension name will result in an error. To find out the dimensions that make up the merged dimension, use the getCube().getMultipleHierarchies() method.

Examples

The following example shows how to merge all hierarchies with the prefix "Time" into a non-existing dimension called Time, and all hierarchies with the prefix "Products" into a non-existing dimension called Products:

```
myDataBlox.setMergedDimensions("Time,Products");
```

See Also

"getCube().getMultipleHierarchies()" on page 435

mergedHeaders

Specifies the dimensions on the same axis whose headers are to be merged.

Data Sources

All

Syntax

JSP Tag Attribute

```
mergedHeaders="mergedString"
```

Java Methods

```
String getMergedHeaders();
void setMergedHeaders(String mergedString);
```

where:

Argument	Default	Description
`mergedString`	`null`	A colon-separated string of *dimensionString:matchPatterns:drillableDim* See the Usage section below for details.

Usage

- *dimensionString*—This string specifies the dimensions whose headers are to be merged and the new member name for the merged header, in the format of:

  dimensionList = newMemberName

  dimensionList is a comma-separated list of dimensions whose headers are to be merged. *newMemberName* is the name of the merged header. This is the name to

use if you want to hide a member, row, or tuple (using DataBlox's `hiddenMembers` or `hiddenTuples` property). By default, the merged header adds a space as a separator among the merged dimension headers. For example, if the headers for Scenario and Measures are merged, the new header is ″Scenario Measures,″ with a space in between.

Note: The order of the dimension specification has to be the same as that in the result set, and they must be consecutive. For example, if you have a query that returns All Time Periods, Measures, and Scenario in that sequence, then the following examples are valid:

```
mergedHeaders="All Time Periods, Measures, Scenario = Measures and
Scenario by Year" mergedHeaders="Measures, Scenario = Measures &
Scenario"
```

But the following are not:

```
mergedHeaders="All Time Periods, Scenario = Scenario by Period"
(dimensions are not consecutive) mergedHeaders="Measures, All Time
Periods = Measures by Period" (order is not correct)
```

Note: Calculated member (specified using the calculatedMembers property) is performed before the headers are merged. Therefore, when you need to add a calculated member or members, use the original dimension names. hiddenMembers and hiddenTuples, on the contrary, are performed after `mergedHeaders`, and therefore the newly merged header should be used.

- *matchPatterns*—Optional; A comma-separated list of pairs of the header pattern to match and the replacing header. Each pair of old header and new header should be in the format of *olderHeader = newHeader.* The following example merges the headers for Measures and All Time Periods, and replaces the string "Qtr 1" found in any header with the string "Q1," "Qtr 2" with "Q2 ," "Qtr 3" with "Q3," and "Qtr 4" with "Q4."

  ```
  mergedHeaders="All Time Periods, Measures = Measures by Period:  Qtr 1 =
  Q1, Qtr 2 = Q2, Qtr 3 = Q3, Qtr 4 = Q4"
  ```

  As a result, Qtr 1 00 Sales becomes Q1 00 Sales and Qtr 1 01 Forecast becomes Q1 01 Forecast.
- *drillableDim*—Optional; The dimension that is drillable when users drill on the merged header. If a drillable dimension is not specified, the first dimension listed in the *dimensionString* is by default the drillable dimension.

  Note: If a drillable dimension is specified without any match patterns, the colon separating the two strings should still be included. For example:

  ```
  mergedHeaders="Measures, Scenario::Scenario"
  ```

  Note: There can only be one drillable dimension when headers are merged. Setting a drillable dimension makes the other dimensions in the *dimensionString* not drillable.

Examples

The following example merges the headers for dimensions All Time Periods and Measures, with the new merged dimension name being Measures by Period. Four header name replacement match patterns are specified. The drillable dimension is set to All Time Periods (which is also the default if not specified, since it is the first dimension in the list).

```
mergedHeaders="All Time Periods, Measures= Measures by Period:
Qtr 1 = Q1, Qtr 2 = Q2, Qtr 3 = Q3, Qtr 4 = Q4:
All Time Periods"/>
```

onErrorClearResultSet

Specifies whether the existing result set should be cleared if a subsequent database operation fails.

Data Sources

All

Syntax

JSP Tag Attribute

```
onErrorClearResultSet="clearResultSet"
```

Java Methods

```
boolean isOnErrorClearResultSet();
void    setOnErrorClearResultSet(boolean clearResultSet);
```

where:

Argument	Default	Description
clearResultSet	false	Specify true to clear the result set; false not to clear it.

See Also

"clearResultSet()" on page 386

parentFirst

Specifies how the parents are returned relative to the children.

Data Sources

Multidimensional

Syntax

JSP Tag Attribute

```
parentFirst="parentFirst"
```

Java Methods

```
int getParentFirst();
void setParentFirst(int parentFirst);
```

where:

Argument	Default	Description
parentFirst	Respect the order of the members returned from the query	In Blox tags, specify true to place the parents before the children; false to place children first. If this tag attribute is not specified, the order of the members returned from the query is respected. The related Java methods take and return integers. setParentFirst() takes the following values: • DataBlox.PARENT_DEFAULT— the order of the members returned from the query should be respected • DataBlox.PARENT_FIRST— parent members should come before their child members regardless of the order of the members returned from the query • DataBlox.PARENT_LAST— parent members should come after their child members regardless of the order of the members returned from the query

Usage

In Blox tags, setting the value to true will return the data with the parent first (above or to the left of the children); setting the value to false will return the data with the parent last (below or to the right of the children). If this attribute is not specified in your DataBlox, the order of the members returned from the query is respected. Note that if you want to use the GridBlox's expand/collapse mode (expandCollapseMode ="true") and want parents to display first, set parentFirst to true rather than do so in the query. This is to ensure the expand/collapse mode can search through the result set correctly to determine the base members and shared members.

Important: If you set to have parent members come before or after the members returned, you cannot reset to respect the default order.

Examples

The following example demonstrates how to set the parent members to come before the children using JSP tags and Java method:

```
<blox:data ..
   parentFirst="true" />
<% myDataBlox.setParentFirst(DataBlox.PARENT_FIRST); %>
```

The following example demonstrates how to get the current order of parents and their children:

```
<% String message;
   int order;
   order = myDataBlox.getParentFirst();
   if (order == myDataBlox.PARENT_FIRST) {
      message = "Parent First";
   } else if (order == myDataBlox.PARENT_LAST) {
      message = "Parent Last";
   } else message="Default Order";
   out.write("The current parent-child order is: " + message);
%>
```

password

Specifies the database password to use when accessing the data source.

Data Sources
All

Syntax
JSP Tag Attribute

```
password="password"
```

Java Methods

```
void setPassword(String password);
```

where:

Argument	Default	Description
password	empty string	Password for accessing the data source.

Usage
A default password is one of the values provided when defining a data source to DB2 Alphablox. If no `password` property is specified on a DataBlox, the value is taken from the data source definition, unless `AASUserAuthorizationEnabled` is set to `true` (in which case, the password the user entered is used).

If you use this method in conjunction with `setDataSourceName()` method, you should set the password after calling `setDataSourceName()`. Otherwise, the DataBlox will connect with all the properties in the data source specified and override any properties set earlier. This is because the `setDataSourceName()` method also reads in the properties of the data source such as username, password, catalog, schema, query, and dimensions on page axis. Therefore, if you want to set any of these properties using the Java methods, set them after calling `setDataSourceName()`.

Tip: The order these data source properties are set is not an issue if you use the Blox tags. The tags are designed to enforce that the data source is set before the call to set the other data source properties, the side effect is taken care for you.

Examples

```
setPassword("secret");
```

See Also
"dataSourceName" on page 356, "userName" on page 384

performInAllGroups
Specifies whether a drill operation is performed on all occurrences of the selected member in each outer nested group containing the dimension, or only on the single selected occurrence of the member.

Data Sources
Multidimensional

Syntax
JSP Tag Attribute

```
performInAllGroups="perform"
```

Java Methods

```
boolean isPerformInAllGroups();
void setPerformInAllGroups(boolean perform);
```

where:

Argument	Default	Description
perform	true	Specify true to make a drill apply to all occurrences of the selected member; false to drill only the single occurrence of the member.

Usage

Even when this property is set to true, the member name must be the same in the other groups for the drill to occur. For example, assume Period is nested within Product. A drill on Qtr1 in VCR expands Qtr1 in TV because the member names are the same. However, Qtr2 through Qtr4 in VCR and TV are not expanded because the member names are different.

query

Specifies the initial query string that is passed to the data source.

Data Sources

All

Syntax

JSP Tag Attribute

```
query="queryString"
```

Java Methods

```
String getQuery();
void   setQuery(String queryString);
```

where:

Argument	Default	Description
queryString	empty string	Query statement in the language understood by the data source. For relational data sources, use a SQL SELECT statement. For multidimensional data sources, use the appropriate language, such as Microsoft MDX or Essbase Report Specifications.

Usage

The getQuery() method returns the last query string that has been set for the current data source. User actions, such as sorts or drills, performed since the last query are not reflected in the returned value.

The setQuery() method sets the query string. The query is executed when the connect() method is called.

Examples

For an example of a query using the Microsoft MDX query language, see the MDX Statement in "Retrieving Data" of the *Developer's Guide*. For specific information from Microsoft on MDX,, see the following web links:

```
http://www.microsoft.com/data/oledb/
```

and

```
http://msdn.microsoft.com/library/techart/intromdx.htm
```

For an example of a query using an Essbase Report Specification, see Essbase Report Specifications in "Retrieving Data" of the *Developer's Guide*. For specifics, see the online documentation in the Essbase installation directory:

```
\docs\techref\RPTIND.HTM
```

See Also

"generateQuery()" on page 392, "selectableSlicerDimensions" on page 376

retainSlicerMemberSet

Specifies whether to retain the member selections made in the grid.

Data Sources

Multidimensional

Syntax

JSP Tag Attribute

```
retainSlicerMemberSet="persistMemberSelection"
```

Java Methods

```
boolean isRetainSlicerMemberSet();
void setRetainSlicerMemberSet(boolean persistMemberSelection);
```

where:

Argument	Default	Description
`persistMemberSelection`	`true`	Specify `true` to persist member selections made in the grid.

Usage

When `true` (default) the member selections made in the grid are retained and the page filters will show the children of the selected member. When `false`, the member selections made in the grid are not retained when users move a dimension back and forth between the Page dimension and the Row or Column dimension.

rowSort

Specifies how to sort data values for members on the row axis.

Data Sources

All

Syntax

JSP Tag Attribute

```
rowSort="sortString"
```

Java Methods

```
string getRowSort();   //returns String of 4 comma-separated items

void setRowSort(Tuple tuple, AxisDimension dimension,
                boolean ascending);
void setRowSort(Tuple tuple, AxisDimension dimension,
                boolean ascending, boolean preserveHierarchy);
void setRowSort(String sortString);
```

where:

Argument	Default	Description
`tuple`	none	The tuple on the row axis that specifies the row to be sorted.
`dimension`	none	The dimension on the column axis for which grouping is preserved. Specify `null` to indicate no grouping is to be preserved on the column axis.
`ascending`	none	Specify `true` to sort ascending; `false` to sort descending.
`preserveHierarchy`	`false`	Specify `true` to preserve the hierarchy in the column axis, keeping members with their parents after the sort operation; `false` to not preserve hierarchy. This argument is only valid in the Java method.

`sortString`	none	A comma-delimited string in one of the following formats:
		• *tupleIndex*, *direction*
		• *tupleIndex*, *groupingNestLevel*, *direction*
		• *tupleIndex*, *groupingNestLevel*, *direction*, *preserveHierarchy*
		
		tupleIndex — string representation of an integer representing the zero-based tuple index member (row) to sort, where `0` indicates the topmost row.
		
		groupingNestLevel — string representations of an integer representing the dimension on the column axis for which grouping is preserved. For example, if Time and Measures are on the column axis, a value of 1 sorts into sequence within the Measures dimension. Specify `-1` to sort without regard to column groupings. The default is `-1`.
		
		direction — a case-insensitive string of either `"Ascending"`, `"Asc"`, `"Descending"` or `"Desc"`.
		
		preserveHierarchy — string representation of a boolean. See the `preserveHierarchy` argument. Defaults to `false`.
		
		For example:
		`setRowSort("1,0,asc");`
		`setRowSort("1,0,asc,true");`
		`setRowSort("0,descending");`

Usage

The `getRowSort` method returns a string of four comma-separated items: *tupleIndex*, *groupingNestLevel*, *direction*, and *preserveHierarchy*.

Examples

The following example demonstrates the use of the `rowSort` tag attribute:

```
rowSort="1, 0, asc"
```

See Also

"removeRowSort()" on page 402, "columnSort" on page 353

schema

Specifies the name of the schema to access.

Data Sources

All

Syntax

JSP Tag Attribute

```
schema="schema"
```

Java Methods

```
String getSchema();
void   setSchema(String schema);
```

where:

Argument	Default	Description
schema	empty string	Name of the schema

Usage

The value for schema is one of the values provided when defining a data source to DB2 Alphablox. If you do not specify the schema property for a DataBlox, the value is taken from the data source definition.

A schema is a "database" in IBM DB2 OLAP Server or Hyperion Essbase terminology.

See Also

"catalog" on page 352

selectableSlicerDimensions

Specifies the dimensions that appear on the page (slicer) axis. The slicer dimensions act as filters on the data.

Data Sources

Multidimensional

Syntax

JSP Tag Attribute

```
selectableSlicerDimensions="dimensionString"
```

Java Methods

```
Dimension[] getSelectableSlicerDimensions(MDBMetaData metadata);
Dimension   getSelectableSlicerDimensions(MDBMetaData metadata,
                                          int i);
void        setSelectableSlicerDimensions(String dimensionString);
void        setSelectableSlicerDimensions(Dimension[] dimensions);
```

where:

Argument	Default	Description
metadata	If null, creates a new instance of Dimension	A valid specification to the MDBMetaData interface.
i	none	The index number indicating the page axis dimension. Typically the index number for the page axis is 2.
dimensionString	empty string	A comma-separated string of unique dimension names.
dimensions	If null, creates a new instance of Dimension	A valid specification to the Dimension interface.

Usage

The selectableSlicerDimensions property has no effect on dimensions that already reside on the row or column axis. It can only operate on dimensions that are currently on the "Other" (unused) axis. This method is relevant for multidimensional data sources only.

showSuppressDataDialog

When the useOlapDrillOptimization property is set to true, if either suppressMissingColumns or suppressMissingRows is also set to true, this property specifies whether a warning dialog should pop up. This dialog alerts users of the possibility of incomplete data when they drill down and then perform further data analysis operations.

Data Sources

Microsoft Analysis Services

Syntax

JSP Tag Attribute

```
showSuppressDataDialog="showDialog"
```

Java Methods

```
boolean isShowSuppressDataDialog();
void    setShowSuppressDataDialog(boolean showDialog);
```

where:

Argument	Default	Description
showDialog	true	Specify true to pop up an alert dialog; false to suppress the pop-up dialog.

Usage

When the property (set either by assemblers through Blox tags, a Java method or by users through the Blox user interface) and the useOlapDrillOptimization property are both set to true, users may only see partial data. This happens when they drill down and then perform other actions such as changing page filters, drilling up, or using Member Filter. When this sequence of user actions occur, a

dialog will pop up that alerts the users of this possibility and recommends the users to turn off Suppress Missing. When this `showSuppressDataDialog` property is set to false, the dialog will not pop up.

See Also

"useOlapDrillOptimization" on page 383, "suppressMissingColumns" on page 378, "suppressMissingRows" on page 379

suppressDuplicates

Specifies whether to remove from the grid those rows or columns containing duplicate header values.

Data Sources

Multidimensional

Syntax

JSP Tag Attribute

```
suppressDuplicates="suppress"
```

Java Methods

```
boolean isSuppressDuplicates();
void    setSuppressDuplicates(boolean suppress);
```

where:

Argument	Default	Description
`suppress`	`true`	Specify `true` to suppress duplicate header values; `false` to leave them.

Usage

To suppress duplicate shared members in IBM DB2 OLAP Server or Hyperion Essbase result sets, use the `SUPSHARE` command in your report script query. To learn more about this command, refer to your IBM DB2 OLAP Server or Hyperion Essbase documentation.

See Also

"suppressMissingColumns" on page 378, "suppressMissingRows" on page 379, "suppressNoAccess" on page 380, "suppressZeros" on page 380

suppressMissingColumns

Specifies whether to remove from the grid those columns containing no data at all.

Data Sources

Multidimensional

Syntax

JSP Tag Attribute

```
suppressMissingColumns="suppress"
```

Java Methods

```
boolean isSuppressMissingColumns();
void    setSuppressMissingColumns(boolean suppress);
```

where:

Argument	Default	Description
suppress	false	Specify true to suppress columns containing no data; false to leave them.

Usage

Use the missingValueStringproperty on GridBlox to specify what to display in cells with no value.

When the data source is Microsoft Analysis Services, this property should be used with care in conjunction with the useOlapDrillOptimization property. When both properties are set to true, users may only see partial data when they drill down and then perform other actions such as changing page filters, drilling up, or using Member Filter. See "useOlapDrillOptimization" on page 383 for more information.

To suppress duplicate shared members in IBM DB2 OLAP Server or Hyperion Essbase result sets, use the SUPSHARE command in your report script query. To learn more about this command, refer to your IBM DB2 OLAP Server or Hyperion Essbase documentation.

See Also

"suppressMissingRows" on page 379, "suppressDuplicates" on page 378, "suppressNoAccess" on page 380, "suppressZeros" on page 380

suppressMissingRows

Specifies whether to remove from the grid those rows containing no data at all.

Data Sources

Multidimensional

Syntax

JSP Tag Attribute

```
suppressMissingRows="suppress"
```

Java Methods

```
boolean isSuppressMissingRows();
void    setSuppressMissingRows(boolean suppress);
```

where:

Argument	Default	Description
suppresss	false	Specify true to suppress rows containing no data; false to leave them.

Usage

Use the missingValueString property on GridBlox to specify what to display in cells with no value.

When the data source is Microsoft Analysis Services, this property should be used with care in conjunction with the useOlapDrillOptimization property. When both properties are set to true, users may only see partial data when they drill down

and then perform other actions such as changing page filters, drilling up, or using Member Filter. See “useOlapDrillOptimization” on page 383 for more information.

To suppress duplicate shared members in IBM DB2 OLAP Server or Hyperion Essbase result sets, use the `SUPSHARE` command in your report script query. To learn more about this command, refer to your IBM DB2 OLAP Server or Hyperion Essbase documentation.

See Also

“suppressMissingColumns” on page 378, “suppressMissingRows” on page 379, “suppressDuplicates” on page 378, “suppressNoAccess” on page 380, “suppressZeros” on page 380

suppressNoAccess

Specifies whether to remove from the grid those rows or columns containing data the users cannot access.

Data Sources

Multidimensional

Syntax

JSP Tag Attribute

```
suppressNoAccess="suppress"
```

Java Methods

```
boolean isSuppressNoAccess();
void    setSuppressNoAccess(boolean suppress);
                        throws InvalidBloxPropertyValueException,
                                ServerBloxException
```

where:

Argument	Default	Description
`suppress`	`false`	Specify `true` to suppress data the users cannot access; `false` to leave it.

Usage

To suppress duplicate shared members in IBM DB2 OLAP Server or Hyperion Essbase result sets, use the `SUPSHARE` command in your report script query. To learn more about this command, refer to your IBM DB2 OLAP Server or Hyperion Essbase documentation.

See Also

“suppressDuplicates” on page 378, “suppressMissingColumns” on page 378, “suppressMissingRows” on page 379, “suppressZeros” on page 380

suppressZeros

Specifies whether to remove from the grid those rows or columns containing all zeros.

Data Sources

Multidimensional

Syntax

JSP Tag Attribute

```
suppressZeros="suppress"
```

Java Methods

```
boolean isSuppressZeros();
void    setSuppressZeros(boolean suppress);
```

where:

Argument	Default	Description
suppress	false	Specify true to suppress columns or rows containing all zeros; false to leave them.

Usage

To suppress duplicate shared members in IBM DB2 OLAP Server or Hyperion Essbase result sets, use the SUPSHARE command in your report script query. To learn more about this command, refer to your IBM DB2 OLAP Server or Hyperion Essbase documentation.

See Also

"suppressDuplicates" on page 378, "suppressMissingColumns" on page 378, "suppressMissingRows" on page 379, "suppressNoAccess" on page 380

textualQueryEnabled

Specifies whether the data query should be restored using the textual query rather than the serialized query.

Data Sources

All

Syntax

Java Methods

```
boolean isTextualQueryEnabled();    //throws ServerBloxException
void setTextualQueryEnabled(boolean textualQuery);
                    //throws InvalidBloxPropertyValueException,
                              ServerBloxException
```

where:

Argument	Default	Description
textualQuery	false	Specify whether the data query should be loaded using the query string stored in the textual query file. By default, a bookmark is loaded using the serialized query.

Usage

When a bookmark is first added, the *delta* between the query set in the DataBlox and the query that generates the current data view is saved in a <bookmark_name>.data text file and a <bookmark_name>.query file that stores the query as serialized object. Later when the bookmark is loaded, the serialized query is used unless this property is set to true. This is useful in cases where there is a change of data outline or member names and you want to modify the textual

query accordingly. Manipulation of the textual query is more efficient since DB2 Alphablox does not need to manipulate the result set to match the serialized object.

Note, however, that the textual query is not updated when the bookmark is resaved with a different data view. If you anticipate the need to use textual queries, you may want to ensure the textual query is up-to-date. In this case, you can capture a bookmark save event and get the current textual query using the DataBlox `generateQuery()` method to update the query in the bookmark. This involves the use of the `addEventFilter()` common Blox method to add a method that implements the `BookmarkSaveFilter` interface.

See Also

"Serialized Query and Textual Query" on page 129, "Bookmark Events and Event Filters" on page 128, "generateQuery()" on page 392.

useAASUserAuthorizationEnabled

Specifies whether to use the user name and password entered during DB2 Alphablox login for authentication to an IBM DB2 OLAP Server or Hyperion Essbase data source.

Data Sources

IBM DB2 OLAP Server, Hyperion Essbase

Syntax

JSP Tag Attribute

```
useAASUserAuthorizationEnabled="useIt"
```

Java Methods

```
boolean isAASUserAuthorizationEnabled();
void    setAASUserAuthorizationEnabled(boolean useIt);
```

where:

Argument	Default	Description
`useIt`	`false`	Specify `true` to use the DB2 Alphablox login information for IBM DB2 OLAP Server or Hyperion Essbase authentication; `false` to use the normal authentication process.

Usage

This property is valid only when using DB2 Alphablox in a standalone configuration without using external web server security.

When set to `true`, the data source uses the values entered when the user logged in for access to the data source. When set to `false`, the data source uses the normal authentication process.

useAliases

Specifies whether to use aliases or database member values in row and column headings.

Data Sources

IBM DB2 OLAP Server, Hyperion Essbase

Syntax

JSP Tag Attribute

```
useAliases="useAliases"
```

Java Methods

```
boolean isUseAliases();
void    setUseAliases(boolean useAliases);
```

where:

Argument	Default	Description
useAliases	false	Specify true to use aliases; false to use member names

Usage

Database member values are typically codes (such as 001-200); aliases are names (such as Diet Cola).

This property overrides the use of the {OUTALTNAMES} command in an IBM DB2 OLAP Server or Hyperion Essbase report script.

useOlapDrillOptimization

Specifies whether drill optimization should be enabled for Microsoft Analysis Services data sources.

Data Sources

Microsoft Analysis Services

Syntax

JSP Tag Attribute

```
useOlapDrillOptimization="optimize"
```

Java Methods

```
boolean isUseOlapDrillOptimization();
void    setUseOlapDrillOptimization(boolean optimize);
```

where:

Argument	Default	Description
optimize	true	Specify true to use the query optimization

Usage

By default, this property is set to true for Microsoft Analysis Services data sources for better query performance. Use this property with care in conjunction with the suppressMissing property. When both properties are set to true, users may only see partial data when they drill down and then perform other actions such as changing page filters, drilling up, or using Member Filter. When this sequence of user actions occur, a dialog will pop up that alerts the users of this possibility and recommends the users to turn off Suppress Missing. This dialog can be turned off with the showSuppressDataDialog property.

See Also

"showSuppressDataDialog" on page 377, "suppressMissingColumns" on page 378, "suppressMissingRows" on page 379

userName

Specifies the database user name to use when accessing the data source.

Data Sources

All

Syntax

JSP Tag Attribute

```
userName="userName"
```

Java Methods

```
String getUserName();
void   setUserName(String userName);
```

where:

Argument	Default	Description
userName	empty string	Database user name.

Usage

A default user name is one of the values provided when defining a data source to DB2 Alphablox. If no `userName` property is specified for a DataBlox, the value is taken from the data source definition.

If you use this method in conjunction with `setDataSourceName()` method, you should set the user name after calling `setDataSourceName()`. Otherwise, the DataBlox will connect with all the properties in the data source specified and override any properties set earlier. This is because the `setDataSourceName()` method also reads in the properties of the data source such as username, password, catalog, schema, query, and dimensions on page axis. Therefore, if you want to set any of these properties using the Java methods, set them after calling `setDataSourceName()`.

Tip: The order these data source properties are set is not an issue if you use the Blox tags. The tags are designed to enforce that the data source is set before the call to set the other data source properties, the side effect is taken care for you.

Examples

```
myDataBlox.setDataSourceName("myDataSource");
myDataBlox.setUserName("secretName");
myDataBlox.connect();
```

See Also

"dataSourceName" on page 356

DataBlox Methods

This section describes DataBlox methods that are not associated with a specific property. For the syntax and descriptions of DataBlox methods that have a property associated with them, see "DataBlox Properties and Associated Methods" on page 334.

For the methods on the result set and metadata objects, see "Multidimensional Result Set Methods" on page 407, "Relational Result Set Methods" on page 423.

addEventFilter()

This is a common Blox method that for capturing an event and perform custom actions *after* the operation is complete on the server. For details, see "addEventListener()" on page 49.

addEventListener()

This is a common Blox method that allows you to capture an event such as drilling down and pivoting *after* the operation is complete on the server. For details, see "addEventListener()" on page 49.

addSelectedMembers()

Adds members to the selected members for the given dimension.

Data Sources

Multidimensional

Syntax

Java Method

```
void addSelectedMembers(Member[] members);
```

where:

Argument	Description
`members`	An array of members derived from the Members interface.
`dimensionName`	Name of the dimension from which to add members.
`memberNames`	Comma-delimited list of member names.

See Also

"getSelectedMembers setSelectedMembers" on page 396. For an example of a class that uses an event filter, see "A Complete drillDownEventFilter Example" on page 465.

clearClientCache()

Clears the memory and reconnects to the Microsoft Analysis Services data source using the same connection parameters.

Data Sources

Microsoft Analysis Services

Syntax

Java Method

```
void clearClientCache();
  // throws ServerBloxException, com.alphablox.util.BadConnectionException
```

Usage

This method will disconnect (autoDisconnect has to be set to `true`) to free up the memory and then reconnect to the data source using the current connection parameters.

Setting autoDisconnect to `true` will cause a disconnect to the data source whenever a query is executed (this includes user data analysis actions such as drilling and pivoting). The connection will automatically reconnect when it needs to and operations will continue to work seamlessly. If you have custom code that performs metadata operations such as a for loop with thousands resolveMember() calls, you should call this `clearClientCache` method afterwards to free up the memory.

This method will not work if autoDisconnect is set to `false.`

See Also

"autoDisconnect" on page 336

clearResultSet()

Deletes the current result set for this DataBlox.

Data Sources

All

Syntax

Java Method

```
void clearResultSet();
```

See Also

"getResultSet()" on page 395, "getXMLResultSet()" on page 397

commitData()

Writes the current data set back to the database.

Data Sources

All

Syntax

Java Method

```
void commitData();
```

Usage

The data set must have been previously locked using the lockCurrentDataSet() method. Once the data set is committed, it is automatically unlocked. Any further calls to `commitData()` require the data set to be relocked.

Note: IBM DB2 OLAP Server and Hyperion Essbase queries do not support the use of attribute dimensions in writeback operations.

See Also

"lockCurrentDataSet()" on page 399

connect()

Connects to a data source.

Data Sources
All

Syntax
Java Method

```
void connect();
       // throws DataBloxCannotConnectException, ServerBloxException
```

Usage
The `connect()` method connects to the data source using the current user name, password, schema, and catalog. The value for these properties is retrieved from the data source definition maintained by DB2 Alphablox. If the data source is currently connected, this method will disconnect, clear the current result set, and connect using any data properties on the Data Peer that have changed.

Note: If you already have a server-side object created (such as a MDBMetaData object), after calling `connect()`, since the object is still pointing to the original connection that no longer exists, you will need to recreate the server-side object.

If the value for the `query` property is missing, no query is executed when the connection is made. Use the `setQuery()` method to set the value for the query property.

You can also use `connect(true)` to execute the defined textual query. See "connect(boolean)" on page 387.

If you only want to update the result set after applying result set property changes (e.g., after setting `useAliases` to `true` or `false`) without reapplying the data source name, user name, password, schema, use updateResultSet().

See Also
"connect(boolean)" on page 387, "disconnect()" on page 388, "query" on page 373, "updateResultSet()" on page 406, the Connecting to Data chapter in the *Developer's Guide*.

connect(boolean)

Connects to a data source.

Data Sources
All

Syntax
Java Method

```
void connect(boolean executeTextualQuery);
       // throws DataBloxCannotConnectException, ServerBloxException
```

where:

Argument	Description

executeTextualQuery	If true, execute the textual query if the query property has been set. If false, connect without executing the query.

Usage
With connect(true), the connection is made, the defined textual query is executed and the result set is retrieved. With connect(false), the connection is made, the defined textual query is not executed.

If you only want to update the result set after applying result set property changes (e.g., after setting useAliases to true or false) without reapplying the data source name, user name, password, schema, use updateResultSet().

See Also
"connect()" on page 387, "disconnect()" on page 388, "query" on page 373, "updateResultSet()" on page 406, the Connecting to Data chapter in the *Developer's Guide*.

disconnect()
Disconnects from the current data source.

Data Sources
All

Syntax
Java Method

```
void disconnect(boolean clearResultSet);
```

where:

Argument	Default	Description
clearResultSet	none	Specify true to clear the result set on disconnect; false to leave it. If no argument is specified, clearResultSet defaults to true.

Usage
If the clearResultSet argument is set to false and the user invokes an operation that requires a connection, an exception is raised and displayed to the user.

If you specify true for the argument, it disconnects and clears the result set.

See Also
"connect()" on page 387, "autoDisconnect" on page 336, "autoConnect" on page 335

drillDown()
Causes a drill down on the specified member in the current data set.

Data Sources
Multidimensional

Syntax
Java Method

```
void drillDown(TupleMember member);
```

where:

Argument	Description
member	A TupleMember object.
axisIndex	Specify 0 for column, 1 for row.
nestIndex	The number of the dimension on the axis specified by axisIndex. Dimension numbers begin with zero and are numbered from left to right on the row axis, and from top to bottom on the column axis.
memberIndex	A 0-based value of the member within the dimension.

Examples

The following example drills down on the second member of the first dimension of the row axis:

```
drillDown(1,0,1);
```

See Also

"drillToAllDescendants()" on page 390, "drillUp()" on page 390

drillThrough()

Performs a drillthrough operation at the specified cell.

Data Sources

Microsoft Analysis Services; IBM DB2 OLAP Server; Hyperion Essbase

Syntax

Java Methods

```
RDBResultSet drillThrough(int columnCoordinate,
                          int rowCoordinate);
                                  // throws ServerBloxException

RDBResultSet drillThrough(Tuple[] coordinates);
                                  // throws ServerBloxException

RDBResultSet drillThrough(String reportName,
                          int columnCoordinate,
                          int rowCoordinate);
                                  // throws ServerBloxException
```

where:

Argument	Description
columnCoordinate	The column coordinate of the specified cell
rowCoordinate	The row coordinate of the specified cell.
coordinates	The first tuple's index is the column coordinate of the specified cell. The second tuple's index is the row coordinate of the specified cell.
reportName	The name of the drillthrough report to use. For IBM DB2 OLAP Server or Hyperion Essbase, this is for Essbase Analytics Services or Essbase Deployment Services data sources which have drillthrough reports set up through Essbase Integration Services (EIS).

Usage
The coordinates are used to determine the unique names of the current members for the dimensions on the column and row axis. For MSAS data sources, a `DRILLTHROUGH` MDX statement will be generated and executed as a result of this method. The DataBlox will determine the unique names of the current members on the slicer axis. The relational data returned from the drillthrough is encapsulated in a RDBResultSet.

See Also
"drillThroughEnabled" on page 569, "drillThroughWindow" on page 570

drillToAllDescendants()
Causes a drill down to all descendants from the specified member in the current data set.

Data Sources
Multidimensional

Syntax
Java Method

```
void drillToAllDescendants(TupleMember member);
```

where:

Argument	Description
`member`	A TupleMember object.
`axisIndex`	Specify `0` for column, `1` for row.
`nestIndex`	The number of the dimension on the axis specified by `axisIndex`. Dimension numbers begin with zero and are numbered from left to right on the row axis, and from top to bottom on the column axis.
`memberIndex`	A 0-based value of the member within the dimension.

See Also
"drillDown()" on page 388, "drillUp()" on page 390

drillUp()
Causes a drill up on the specified member in the current data set.

Data Sources
Multidimensional

Syntax
Java Method

```
void drillUp(TupleMember member);
```

where:

Argument	Description
`member`	A TupleMember object.
`axisIndex`	Specify `0` for column, `1` for row.

`nestIndex`	The number of the dimension on the axis specified by `axisIndex`. Dimension numbers begin with zero and are numbered from left to right on the row axis, and from top to bottom on the column axis.
`memberIndex`	A 0-based value of the member within the dimension.

See Also
"drillDown()" on page 388, "drillToAllDescendants()" on page 390

executeCustomCalc()

Executes a calculation script on an IBM DB2 OLAP Server or Hyperion Essbase database.

Data Sources
IBM DB2 OLAP Server, Hyperion Essbase

Syntax
Java Method

```
void executeCustomCalc(String command);
```

where:

Argument	Description
`command`	Calculation command to submit to the database, such as `CALC ALL;`.

Usage
The IBM DB2 OLAP Server or Hyperion Essbase database does not need to be locked to perform a recalculation.

Note: IBM DB2 OLAP Server and Hyperion Essbase queries do not support the use of attribute dimensions in writeback operations.

This method is silently ignored by other data sources.

See Also
"executeNamedDBCalcScript()" on page 391

executeNamedDBCalcScript()

Executes the named IBM DB2 OLAP Server or Hyperion Essbase calc script.

Data Sources
IBM DB2 OLAP Server, Hyperion Essbase

Syntax
Java Method

```
void executeNamedDBCalcScript(String calcScriptName);
```

where:

Argument	Description
`calcScriptName`	Name of calc script to execute. The named calc script must reside on the IBM DB2 OLAP Server or Hyperion Essbase server.

Usage
The application name and database name used in the calc script must exactly match the values for the catalog and schema properties on the DataBlox.

See Also
"executeCustomCalc()" on page 391

generateQuery()
Generates and returns a query string reflecting the current state of the result set.

Data Sources
Multidimensional

Syntax
Java Method

```
String generateQuery();
```

Usage
The application must be connected to the data source at the time the method is called. For MSAS data sources, this returns an optimized query. For example, if you drill down on the member 2003, the query would contain [2003].children as opposed to listing the individual child members for 2003.

See Also
"query" on page 373

getCalculations()
Gets an array of calculated members.

Data Sources
All

Syntax
Java Method

```
Calculation[] getCalculations();  // throws ServerBloxException
```

Usage
Each member in the returned array is of type Calculation. The type Calculation and all of its related interfaces and classes are in com.alphablox.blox.data.calculation package in the Javadoc at:

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

getCommentsBlox()
Gets the CommentsBlox that was set.

Data Sources
Multidimensional

Syntax
Java Method

```
getCommentsBlox();       //returns the CommentsBlox object
```

See Also
Chapter 9, "CommentsBlox Reference," on page 279

getDrillThroughReportNames()

Returns the list of drillthrough reports found at the specified cell.

Data Sources

IBM DB2 OLAP Server, Hyperion Essbase

Syntax

Java Method

```
String[] getDrillThroughReportNames(int columnCoordinate,
                                    int rowCoordinate);
                                   //throws ServerBloxException
```

where:

Argument	Description
`columnCoordinate`	The column coordinate for the specified cell.
`rowCoordinate`	The row coordinate for the specified cell.

Usage

This method is only relevant to IBM DB2 OLAP Server, Hyperion Essbase Analytic Services, and Essbase Deployment Services data sources which have drillthrough reports set up through IBM DB2 OLAP Server Integration Services or Hyperion Essbase Integration Services.

getMetaData()

Returns an interface to the MetaData Object.

Data Sources

All

Syntax

Java Method

```
MetaData getMetaData();
```

Usage

Enables access to the metadata for an underlying data source. The `getMetaData()` method will connect to the data source if necessary. The MetaData object returned is read-only.

For the server-side `getMetaData()` method, you typically do not use this it by itself; instead, you access either the multidimensional metadata ("Multidimensional Metadata Methods" on page 427) or the relational metadata ("Relational Database Metadata Methods" on page 440), depending on what type of data source you are connected to. To access the server-side multidimensional or relational metadata objects, you must cast this method to one of those objects as shown in the following examples.

Examples

The following example casts the server-side `getMetaData()` method to the multidimensional metadata object (`MDBMetaData`). In this example, `myMetadata` is a variable defined as type `MDBMetaData`, and `myDataBlox` is the name of the DataBlox whose metadata you are accessing.

```
MDBMetaData myMetaData=(MDBMetaData) myDataBlox.getMetaData();
```

The following example casts the server-side getMetaData() method to the relational metadata object (RDBMetaData). In this example, myMetaData is a variable defined as type RDBMetaData, and myDataBlox is the name of the DataBlox whose result set you are accessing.

```
RDBResultSet myMetaData=(RDBMetaMeta) myDataBlox.getMetaData();
```

After casting the getMetaData() method to the needed object, you can then access the methods available through the MDBMetaData or RDBMetaData interfaces by using the myMetaData variable you defined. In the case of MDBMetaData, you can access its methods as in the following example:

```
myMetaData.getCubes().getDimensions().getChildren();
```

See Also

"Multidimensional Metadata Methods" on page 427, "Relational Database Metadata Methods" on page 440.

getMetaData().getDatabaseProductName()

Returns the database product name (for example, "IBM DB2 OLAP Server").

Data Sources

All

Syntax

Java Method

```
String getMetaData().getDatabaseProductName();
```

Usage

The getMetaData().getDatabaseProductName() and getMetaData().getDBVersion() methods are useful when application logic requires different processing for different data sources. For example, use the getDatabaseProductName method to determine if a user has drilled from a multidimensional data source into the supporting detail in a relational data source.

getMetaData().getDBVersion()

Returns the database version number (such as "6.1").

Data Sources

All

Syntax

Java Method

```
String getMetaData().getDBVersion();
```

Usage

The getMetaData().getDatabaseProductName() and getMetaData().getDBVersion() methods are useful when application logic requires different processing for different data sources. For example, use the getDatabaseProductName method to determine if a user has drilled from a multidimensional data source into the supporting detail in a relational data source.

getRawResultSet()

Returns a read-only copy of the result set.

Data Sources
All

Syntax
Java Method

```
ResultSet getRawResultSet();
```

Usage
This method will connect to the database if needed, and it never returns a null result set. This result set corresponds to the result set which is returned by the database *before* any calculated members (calculatedMembers) are calculated, headers are merged (mergedHeaders), or members are hidden (hiddenMembers). Therefore, calculated members and the results of merged headers will not appear in this result set. Hidden members, on the contrary, will appear in this result set since they have not yet been hidden. To get the result set reflecting the results of these property settings, use getResultSet().

Because there are no methods directly on the `ResultSet` object, you typically do not use this method by itself; instead, you access either the multidimensional result set ("Multidimensional Result Set Methods" on page 407) or the relational result set ("Relational Result Set Methods" on page 423), depending on what type of data source you are connected to. See "getResultSet()" on page 395 for examples.

See Also
"getResultSet()" on page 395, "getXMLResultSet()" on page 397, "Result Set, Server-Side" on page 332, "Multidimensional Result Set Methods" on page 407, "Relational Result Set Methods" on page 423.

getResultSet()
Returns a read-only of the result set.

Data Sources
All

Syntax
Java Method

```
ResultSet getResultSet();
```

Usage
This method will connect to the database if needed, and it never returns a null result set. This result set corresponds to the result set which is displayed by the DataBlox *after* calculated members (calculatedMembers) are calculated, headers are merged (mergedHeaders), and members are hidden (hiddenMembers). Therefore, calculated members and the results of merged headers will appear in this result set. Hidden members, on the contrary, will not appear in this result set since they have already been hidden. To get the raw result set before these property settings are applied, use getRawResultSet().

Because there are no methods directly on the `ResultSet` object, you typically do not use this method by itself; instead, you access either the multidimensional result set ("Multidimensional Result Set Methods" on page 407) or the relational result set ("Relational Result Set Methods" on page 423), depending on what type of data source you are connected to. To access the multidimensional or relation ResultSet Objects, you must cast this method to the one of those objects as shown in the following examples.

Examples

The following example casts the `getResultSet()` method to the multidimensional result set object (`MDBResultSet`). In this example, `myResultSet` is a variable defined as type `MDBResultSet`, and `myDataBlox` is the name of the DataBlox whose result set you are accessing.

```
MDBResultSet myResultSet=(MDBResultSet) myDataBlox.getResultSet();
```

The following example casts the `getResultSet()` method to the relational result set object (`RDBResultSet`). In this example, `myResultSet` is a variable defined as type `RDBResultSet`, and `myDataBlox` is the name of the DataBlox whose result set you are accessing.

```
RDBResultSet myResultSet=(RDBResultSet) myDataBlox.getResultSet();
```

After casting the `getResultSet()` method to the needed object, you can then access the methods available through the `MDBResultSet` or `RDBResultSet` interfaces by using the `myResultSet` variable you defined, as in the following example:

```
myResultSet.getAxis(1);
```

See Also

"getRawResultSet()" on page 394, "getXMLResultSet()" on page 397, "Result Set, Server-Side" on page 332, "Multidimensional Result Set Methods" on page 407, "Relational Result Set Methods" on page 423.

getSelectedMembers
setSelectedMembers

Specifies or returns the members for the dimensions currently in the data set.

Data Sources

Multidimensional

Syntax

Java Methods

```
Member[] getSelectedMembers(Dimension dimension);
void     setSelectedMembers(Member[] members);
```

where:

Argument	Description
`dimension`	A valid specification to the Dimension interface.
`members`	An array of member names to be selected.

Usage

For dimensions on the row and column axes, the selected members are displayed in the grid. For dimensions on the page and other axes, the selected member is the member to be set as the filter.

When the specified dimension is the page or other axis, the first member returned by `getSelectedMembers()` is always the currently selected member.

When the specified dimension is the page or other axis, you can only pass a single member to `setSelectedMembers()`. If you pass multiple members the request is ignored.

See Also
"clearResultSet()" on page 386

getXMLResultSet()

Returns an interface to the result set as XML DOM.

Data Sources
NA

Syntax
Java Method

```
AASCubeXMLDocument getXMLResultSet();
```

Usage
Enables access to the result set as XML document object model (DOM). The `getXMLResultSet()` method will connect to the data source if necessary. The AASCubeXMLDocument object returned is read-only.

See Also
"getResultSet()" on page 395, "Result Set, Server-Side" on page 332

hideMembers()

Hides the specified member(s) in the data set. The members specified are added to any that are already hidden by the `hiddenMembers` property.

Data Sources
All

Syntax
Java Method

```
void hideMembers(Column[] columnNames);
void hideMembers(Member[] members);
void hideMembers(String membersToHide);
                     // throws ServerBloxException
```

where:

Argument	Description
`columnNames`	An array of column names. The array must be built using the `getColumns()` APIs.
`members`	An array of member names. The array must be built using the `getDimensions()` APIs.
`membersToHide`	List of members to hide. Format the string as follows: DimensionName1:MemberNameA,MemberNameB; DimensionName2:MemberNameD,MemberNameD; ... DimensionName*N*:MemberNameX,MemberNameY The members should be grouped by dimension. Separate the groups in the list with a semicolon (;). Within each group, separate the members by commas.

See Also

"hiddenMembers" on page 362, "showMembers()" on page 403

For the methods on the `Member` interface, see "getCube().getDimension().getCube()" on page 428, "getCube().getDimension().getDisplayName()" on page 429, "getCube().getDimension().getRootMember(). getGenerationLevel()" on page 432, and "getCube().getDimension().getRootMember().isLeaf()" on page 433.

For the methods on the `Column` interface, see "getTable().getColumns()" on page 442, "getTable().getColumn().getDistinctValues()" on page 443, "getTable().getColumn().getName()" on page 443, "getTable().getColumn().isNumeric()" on page 444, "getTable().getColumn().getType()" on page 444.

hideTuples()

Hides the specified tuple(s) in the result set. The tuples specified are added to any that are already hidden by the hiddenTuples property.

Data Sources

Multidimensional

Syntax

Java Method

```
void hideTuples(String selectedTuples);
                        // throws ServerBloxException
```

where:

Argument	Description

`selectedTuples`	List of hidden tuples to hide. Format the string as follows:
	`Dimension1,Dimension2,...,DimensionN:`
	`Dim1Member1, Dim2Member1,..,DimNMember1;`
	`Dim1Member2,Dim2Member2,...,DimNMember2;...`
	`Dim1MemberM,Dim2MemberM,...,DimNMemberM`
	Each tuple list contains the dimension names separated by commas, followed by a colon, followed by the list of tuples. Each tuple consists of one member from each of the dimensions specified and is separated by a semicolon. Each member of the tuple is separated by a comma.
	You can also specify multiple tuple lists, with each list enclosed in curly braces and separated by a comma. This allows you specify tuples from dimensions on the opposite axis in the same syntax.

Usage

If there are already hidden tuples in the result set, this method adds to the list of hidden tuples. It does not reset the hidden tuple list.

Examples

```
myDataBlox.hideTuples("{Period,Product:Qtr1,Audio;Qtr2,Visual},{Accounts,Marke
t:Profit,East}");
```

See Also

"hiddenTuples" on page 363, "showTuples()" on page 404, "showOnlyTuples()" on page 404

keepOnly()

Keeps only the specified member (and its associated tuples) on the chart or grid.

Data Sources

Multidimensional

Syntax

Java Method

```
void keepOnly(TupleMember member);
void keepOnly(TupleMember[] members);
```

where:

Argument	Description
`member`	A member name.
`members`	A vector of member names.
`axisIndex`	Specify `0` for column, `1` for row.
`nestIndex`	The number of the dimension on the axis specified by `axisIndex`. Dimension numbers begin with zero and are numbered from left to right on the row axis, and from top to bottom on the column axis.
`memberIndex`	A 0-based value of the member within the dimension.

Usage

If there is a conflict between the action specified by this method and the value set by the performInAllGroups property, `performInAllGroups` takes precedence over `keepOnly()`.

See Also

"enableKeepRemove" on page 360, "removeOnly()" on page 401.

loadBookmark()

This is a common Blox method. For a complete description, see "loadBookmark()" on page 54.

lockCurrentDataSet()

Locks the called-upon result set; does not lock the entire database.

Data Sources

Multidimensional

Syntax

Java Method

```
void lockCurrentDataSet();
```

Usage
The result set must be locked before committing data to a database. This method must be called prior to calling the commitData() method.

Note: IBM DB2 OLAP Server and Hyperion Essbase queries do not support the use of attribute dimensions in writeback operations.

See Also
"commitData()" on page 386

pivot()
Pivots a single dimension in the current result set along a specified axis.

Data Sources
Multidimensional

Syntax
Java Method

```
void pivot(int oldAxisIndex, int oldNestIndex, int newAxisIndex,
           int newNestIndex);
```

where:

Argument	Description
`newAxisIndex`	Specify `0` for column, `1` for row.
`newNextIndex`	The number of the dimension on the axis specified by `newAxisIndex`. Dimension numbers begin with zero and are numbered from left to right on the row axis, and from top to bottom on the column axis.
`oldAxisIndex`	Specify `0` for column, `1` for row.
`oldNestLevel`	The number of the dimension on the axis specified by `oldAxisIndex`. Dimension numbers begin with zero and are numbered from left to right on the row axis, and from top to bottom on the column axis.

Usage
This method pivots the specified dimension to another position in the schema, either from axis to axis, slicer to axis, or axis to slicer.

refresh()
Refreshes the current data set.

Data Sources
Multidimensional

Syntax
Java Method

```
void refresh();
```

Usage
This method is useful following a call to the lockCurrentDataSet() method to ensure the data is fresh before any modifications are made.

When calling the method from a Blox other than a DataBlox, use the following syntax:

```
getDataBlox().refresh();
```

If it is likely that a query is running, issue an explicit `waitOnBusy()` method before issuing a `refresh()`.

Note: IBM DB2 OLAP Server and Hyperion Essbase queries do not support the use of attribute dimensions in writeback operations.

See Also

"lockCurrentDataSet()" on page 399

removeColumnSort()

Removes the sort settings specified by `ColumnSort()`.

Data Sources

All

Syntax

Java Method

```
void removeColumnSort();
```

See Also

"columnSort" on page 353, "rowSort" on page 374

removeEventFilter()

This is a common Blox method that allows you to remove an event filter object added using addEventFilter() for capturing a server-side event (such as drilling down and pivoting) *before* it is processed on the server. For details, see "removeEventFilter()" on page 55.

removeEventListener()

This is a common Blox method that allows you to remove an event listener object created using addEventListener() for capturing a server-side event (such as drilling down and pivoting) *after* that operation is complete on the server. For details, see "removeEventListener()" on page 56.

removeOnly()

Removes only the defined member (and its associated Tuples) on the chart or grid.

Data Sources

Multidimensional

Syntax

Java Methods

```
void removeOnly(TupleMember member);
void removeOnly(TupleMember[] members);
```

where:

Argument	Description
`member`	A member name.

`members`	A vector of member names.
`axisIndex`	Specify `0` for column, `1` for row.
`nestIndex`	The number of the dimension on the axis specified by `axisIndex`. Dimension numbers begin with zero and are numbered from left to right on the row axis, and from top to bottom on the column axis.
`memberIndex`	A 0-based value of the member within the dimension.

See Also
"enableKeepRemove" on page 360, "keepOnly()" on page 399

removeRowSort()

Removes the sort settings specified by `RowSort()`.

Data Sources
Multidimensional

Syntax
Java Method

```
void removeRowSort();
```

See Also
"rowSort" on page 374, "columnSort" on page 353

saveBookmark()

This is a common Blox method. For a complete description, see "saveBookmark()" on page 57.

saveBookmarkHidden()

This is a common Blox method. For a complete description, see "saveBookmarkHidden()" on page 58.

setDataValues()

Changes data values in the result set at the coordinates specified.

Data Sources
Multidimensional (but not DB2 Alphablox cubes)

Syntax
Java Method

```
void setDataValues(Tuple[][] coordinates,
                   String[] values);
```

where:

Argument	Description
`coordinates`	An array of coordinates. A single coordinate is itself an array of tuple that specifies a cell in the data cube.
`values`	An array of data values to be written back. If only

one value appears in the value string, it is applied to all specified cell coordinates. Non-numeric values such as a blank string or `#MISSING` are submitted to the data source as a missing value.

Usage

Note that IBM DB2 OLAP Server and Hyperion Essbase queries do not support the use of attribute dimensions in writeback operations.

setSelectedMembers()

For a description of this method, see "getSelectedMembers setSelectedMembers" on page 396.

showMembers()

Shows the specified member(s) in the data set. The members specified are removed from any that are hidden by the `hiddenMembers` property.

Data Sources

All

Syntax

Java Method

```
void showMembers(Column[] columnNames)
     throws ServerBloxException
void showMembers(Member[] selectedMembersArray)
     throws ServerBloxException
```

where:

Argument	Description
`columnNames`	An array of column names. The array must be built using the `getColumns()` APIs.
`selectedMembersArray`	An array of member names. The array must be built using the `getDimensions()` APIs.
`selectedMembers`	A list of members which will be added to the list of `showMembers`. Each member in the list must be separated from the next with a semicolon (;). Format the string as follows: DimensionName1:MemberName1,MemberName2; DimensionName2:MemberName3,MemberName4;... DimensionName*N*:MemberName*N*,MemberName*N* The semicolon is required to separate members in different dimensions.

See Also

"hiddenMembers" on page 362, "hideMembers()" on page 397

For the methods on the `Member` interface, see "getCube().getDimension().getCube()" on page 428, "getCube().getDimension().getDisplayName()" on page 429, "getCube().getDimension().getRootMember(). getGenerationLevel()" on page 432, and "getCube().getDimension().getRootMember().isLeaf()" on page 433.

For the methods on the Column interface, see “getTable().getColumns()” on page 442, “getTable().getColumn().getDistinctValues()” on page 443, “getTable().getColumn().getName()” on page 443, “getTable().getColumn().isNumeric()” on page 444, “getTable().getColumn().getType()” on page 444.

showTuples()

Shows/unhides the specified tuple(s) in the result set if they are hidden.

Data Sources

Multidimensional

Syntax

Java Method

```
void showTuples(String selectedTuples);
                        // throws ServerBloxException
```

where:

Argument	Default	Description
selectedTuples	none	List of hidden tuples to show/unhide. Format the string as follows: Dimension1,Dimension2,...,Dimension*N*: Dim1Member1, Dim2Member1,..,DimNMember1; Dim1Member2,Dim2Member2,...,DimNMember2;... Dim1Member*M*,Dim2Member*M*,...,DimNMember*M* Each tuple list contains the dimension names separated by commas, followed by a colon, followed by the list of tuples. Each tuple consists of one member from each of the dimensions specified and is separated by a semicolon. Each member of the tuple is separated by a comma. You can also specify multiple tuple lists, with each list enclosed in curly braces and separated by a comma. This allows you specify tuples from dimensions on the opposite axis in the same syntax.

Usage

This method adds tuples to the list of tuples to show. To specify the only tuples to show and hide all others, use showOnlyTuples(). The tuples to show need to exist in the result set.

Examples

```
myDataBlox.showTuples("{Period,Product:Qtr1,Audio;Qtr2,Visual},{Accounts,Marke
t:Profit,East}");
```

See Also

“hiddenTuples” on page 363, “showOnlyTuples()” on page 404

showOnlyTuples()

Shows only the specified tuple(s) in the result set.

Data Sources

Multidimensional

Syntax

Java Method

```
void showOnlyTuples(String selectedTuples);
                          // throws ServerBloxException
void showOnlyTuples(Tuple[] tuples);
                          // throws ServerBloxException
```

where:

Argument	Description
selectedTuples	List of tuples to show. Format the string as follows: Dimension1,Dimension2,...,Dimension*N*: Dim1Member1, Dim2Member1,..,DimNMember1; Dim1Member2,Dim2Member2,...,DimNMember2;... Dim1Member*M*,Dim2Member*M*,...,DimNMember*M* Each tuple list contains the dimension names separated by commas, followed by a colon, followed by the list of tuples. Each tuple consists of one member from each of the dimensions specified and is separated by a semicolon. Each member of the tuple is separated by a comma. You can also specify multiple tuple lists, with each list enclosed in curly braces and separated by a comma. This allows you specify tuples from dimensions on the opposite axis in the same syntax.
tuple	The list of tuple objects to show.

Usage

Shows only tuples specified, hiding others not in the list. To show additional tuples rather than reset the list of tuples to show, use showTuples(). The tuples to show need to exist in the result set.

Examples

```
myDataBlox.showOnlyTuples("{Period,Product:Qtr1,Audio;Qtr2,Visual},{Accounts,M
arket:Profit,East}");
```

See Also

"hiddenTuples" on page 363, "showTuples()" on page 404, "hideTuples()" on page 398. For accessing the tuple object, see "getAxis().getTuple()" on page 412

swapRowAndColumnAxes()

Swaps the axes of the current result set displayed in the grid and chart.

Data Sources

Multidimensional

Syntax

Java Method

```
void swapRowAndColumnAxes();    // throws ServerBloxException
```

Usage

Unlike the pivot() method, this method moves all dimensions on the row and column axes to the opposite axes in a single action.

See Also
"pivot()" on page 400

updateResultSet()

Applies the DataBlox properties to the textual or serialized query (the query object), depending on which query is most recent, and creates a new result set.

Data Sources
All

Syntax
Java Method

```
void updateResultSet();
   // throws ServerBloxException, com.alphablox.blox.DataException
```

Usage
All Blox using this DataBlox will be notified that the result set has been updated and update themselves accordingly. Before calling `updateResultSet()`, if the DataBlox is not connected, it will automatically connect.

See Also
"connect()" on page 387; the Connecting to Data chapter in the *Developer's Guide*.

unlockAll()

Unlocks any data that was previously locked in the IBM DB2 OLAP Server or Hyperion Essbase database.

Data Sources
IBM DB2 OLAP Server, Hyperion Essbase

Syntax
Java Method

```
void unlockAll();
```

Usage
Any data that is still locked at the end of a user session is automatically unlocked as a safeguard.

Note: IBM DB2 OLAP Server and Hyperion Essbase queries do not support the use of attribute dimensions in writeback operations.

writeback()

Combines into a single method the actions performed by the lockCurrentDataSet(), commitData(), setDataValues(), executeCustomCalc(), unlockAll(), and refresh() methods. For more information, see the *Developer's Guide*.

Data Sources
IBM DB2 OLAP Server, Hyperion Essbase, Microsoft Analysis Services

Syntax
Java Methods

```
void writeback(Tuple[][] coordinates,
               Object[] values,
               String command);      // throws ServerBloxException
```

where:

Argument	Description
`coordinates`	An array of `Tuple` object coordinates. A single coordinate is itself an array of tuples that specifies a cell in the data cube. For a list of the members on the Tuple interface, see "Tuple" on page 331.
`values`	New values for the cells, in the sequence specified by `coordinates`. If one `value` is specified, it is applied to all specified cell coordinates. Non-numeric values such as a blank string or #MISSING are submitted to the data source as a missing value. For the Java method, specify a vector of Object type. If the object is not of Number type, the argument will be parsed as a string to get the number.
`command`	The command to execute on the database. For IBM DB2 OLAP Server or Hyperion Essbase, the acceptable values are an empty string (""), null, an Essbase calc script (for example, "CALC ALL;"), or an Essbase named calc script. If the string includes a ";" in it, then a check is done to determine whether to execute `executeDBCalcCommand` or executeNamedDBCalcScript().

Usage

- The writeback method supports writeback to IBM DB2 OLAP Server or Hyperion Essbase cubes (including both leaf and non-leaf members) and Microsoft Analysis Services 2000 cubes (leaf members only); DB2 Alphablox cubes do not support writeback.
- For Microsoft Analysis Services 2000, the `command` string is ignored, but must be present (use an empty string). For more complex applications, which include writing back to non-leaf members, you can use the MDX UPDATE CUBE command in a DataBlox `setQuery()` method to update cubes. For details on UPDATE CUBE command, see your Microsoft documentation.
- When using the writeback method with Microsoft Analysis Services 2000, you cannot write null values. Numbers must be of the type double (i.e., numbers with 64 bits).

Note: IBM DB2 OLAP Server and Hyperion Essbase queries do not support the use of attribute dimensions in writeback operations.

Multidimensional Result Set Methods

The server-side multidimensional ResultSet object (`MDBResultSet`) provides an interface to the result set for multidimensional data sources such as IBM DB2 OLAP Server, Hyperion Essbase, Microsoft Analysis Services and DB2 Alphablox cubes. To access the methods on the `MDBResultSet` object, you must cast the `getResultSet()` or `getRawResultSet()` method to the `MDBResultSet` object as described in "getResultSet()" on page 395.

To use the APIs associated with the MDBResultSet, you need to import the com.alphablox.blox.data.mdb package in your JSP page as follows:

```
<%@ page import="com.alphablox.blox.data.mdb.*" %>
```

This section describes all the methods available in the `MDBResultSet` object. This includes methods on the `Axis`, `AxisDimension`, `Tuple`, and `TupleMember` objects. The methods in this section are organized alphabetically by their fully qualified object syntax. For a cross referenced list of methods on each object, see "Objects, Result Set and Metadata" on page 327.

For the methods available on DataBlox, see "DataBlox Methods" on page 385. For the syntax and descriptions of DataBlox methods that have a property associated with them, see "DataBlox Properties and Associated Methods" on page 334.

Note: The object syntax shown for the methods in this section represents only one way to access the methods. There are other possibilities, depending on the outline of your data and the way you access different objects, for how to access a given method. For example, the following two method calls will both access the `getDisplayName` method:

```
getAxis(n).getDimension(n).getDisplayName();
getAxes()[n].getDimensions()[n].getDisplayName();
```

getAxes()

Returns an array containing all the axis within this result set.

Data Sources

Multidimensional

Syntax

Java Method

```
Axis[] getAxes();
```

Usage

This method returns `null` if there are no axes in the result set.

This method is part of the `MDBResultSet` interface.

getAxis()

Returns the axis element for the axis specified. Returns null if no axis element exists for the axis specified.

Data Sources

Multidimensional

Syntax

Java Method

```
Axis getAxis(String axisName);
```

where:

Argument	Description
`axisName`	One of the following constants from the Axis object: `CHAPTER_AXIS`, `COLUMN_AXIS`, `PAGE_AXIS`, `ROW_AXIS`, `SECTION_AXIS`, `SLICER_AXIS`.

Usage
This method is part of the `MDBResultSet` interface.

Pages, chapters and sections are not valid axis names when the data source is IBM DB2 OLAP Server or Hyperion Essbase. With IBM DB2 OLAP Server or Hyperion Essbase data sources, you only have access to the row and column axes (and slicers). You do not have access to the other axes, as you do with other data sources (for example, Microsoft Analysis Services and DB2 Alphablox cubes).

getAxis()

Returns an interface to the `Axis` object.

Data Sources
Multidimensional

Syntax
Java Method

```
Axis getAxis(int index);
```

where:

Argument	Description
`index`	An integer corresponding to the axis number: • `0` - column axis • `1` - row axis • `2` - page axis

Usage
This method is part of the `MDBResultSet` interface.

getAxis().getDimension()
Returns an interface to the dimension corresponding to the index specified

Data Sources
Multidimensional

Syntax
Java Method

```
AxisDimension getAxis(index).getDimension(int index);
```

where:

Argument	Description
`index`	An integer corresponding to the dimension number.

Usage
This method is part of the `Axis` interface.

getAxis().getDimension().getAxis()
Returns an interface to the Axis for the dimension.

Data Sources
Multidimensional

Syntax
Java Method

```
Axis getAxis(index).getDimension(index).getAxis();
```

Usage
This method is part of the AxisDimension interface.

getAxis().getDimension().getDisplayName()
Returns the display name for the dimension.

Data Sources
Multidimensional

Syntax
Java Method

```
String getAxis(index).getDimension(index).getDisplayName();
```

Usage
This method is part of the AxisDimension interface.

getAxis().getDimension().getIndex()
Returns the index number for the dimension relative to the other dimensions in the axis.

Data Sources
Multidimensional

Syntax
Java Method

```
int getAxis(index).getDimension(index).getIndex();
```

Usage
This method is part of the AxisDimension interface.

getAxis().getDimension().getType()
Returns a constant indicating the type of dimension.

Data Sources
Multidimensional

Syntax
Java Method

```
int getAxis(index).getDimension(index).getType();
```

Usage
This method is part of the AxisDimension interface.

The constants returned are as follows:

Constant Returned	
UNKNOWN_DIMENSION_TYPE	The dimension type cannot be determined.

NORMAL_DIMENSION	A normal dimension type (not measures, time, etc.).
MEASURES_DIMENSION	The measures dimension.
TIME_DIMENSION	The time dimension.
ATTR_DIMENSION	The IBM DB2 OLAP Server or Hyperion Essbase attribute dimension.
CALC_ATTR_DIMENSION	The internal calculated IBM DB2 OLAP Server or Hyperion Essbase attribute dimension.

getAxis().getDimension().getUniqueName()

Returns the unique name for the dimension.

Data Sources

Multidimensional

Syntax

Java Method

```
String getAxis(index).getDimension(index).getUniqueName();
```

Usage

This method is part of the AxisDimension interface.

getAxis().getDimensionCount()

Returns the number of dimensions on the axis.

Data Sources

Multidimensional

Syntax

Java Method

```
int getAxis(index).getDimensionCount();
```

Usage

This method is part of the Axis interface.

getAxis().getDimensions()

Returns an array of all the dimensions within this axis.

Data Sources

Multidimensional

Syntax

Java Method

```
AxisDimension[] getAxis(index).getDimensions();
```

Usage

This method is part of the Axis interface.

getAxis().getIndex()

Returns the index number for the axis.

Data Sources
Multidimensional

Syntax
Java Method

```
int getAxis(index).getIndex();
```

Usage
This method is part of the `Axis` interface.

getAxis().getResultSet()
Returns an interface to the MDBResultSet object for the axis.

Data Sources
Multidimensional

Syntax
Java Method

```
MDBResultSet getAxis(index).getResultSet();
```

Usage
This method is part of the `Axis` interface.

See Also
"getResultSet()" on page 395

getAxis().getTupleCount()
Returns the number of tuples in the axis.

Data Sources
Multidimensional

Syntax
Java Method

```
int getAxis(index).getTupleCount();
```

Usage
This method is part of the `Axis` interface.

getAxis().getTuple()
Returns the tuple for the member on the specified index.

Data Sources
Multidimensional

Syntax
Java Method

```
Tuple getAxis(index).getTuple(int index);
```

Usage
This method is part of the `Axis` interface.

getAxis().getTuple()
Returns the tuple for the specified members.

Data Sources
Multidimensional

Syntax
Java Method

```
Tuple getAxis(index).getTuple(String[] members);
```

where:

Argument	Description
members	An array of member names.

Usage
This method is part of the Axis interface.

getAxis().getTuple().getAxis()
Returns the axes associated with the specified tuple.

Data Sources
Multidimensional

Syntax
Java Method

```
Axis getAxis(index).getTuple(members).getAxis();
```

Usage
This method is part of the Tuple interface.

getAxis().getTuple().getIndex()
Returns the index number of the specified tuple relative to the other tuples in the Axis.

Data Sources
Multidimensional

Syntax
Java Method

```
int getAxis(index).getTuple(members).getIndex();
```

Usage
This method is part of the Tuple interface.

getAxis().getTuple().getMember()
Returns the member at the specified index number for the specified tuple.

Data Sources
Multidimensional

Syntax
Java Method

```
TupleMember getAxis(index).getTuple(members).getMember(int index);
```

Usage
This method is part of the Tuple interface.

getAxis().getTuple().getMember().getDimension()

Returns an interface to the dimension for the specified member.

Data Sources
Multidimensional

Syntax
Java Method

```
AxisDimension getAxis(index).getTuple(members).getMember(index).getDimension();
```

Usage
This method is part of the TupleMember interface.

getAxis().getTuple().getMember(). getDisplayName()

Returns the display name for the specified member.

Data Sources
Multidimensional

Syntax
Java Method

```
String getAxis(index).getTuple(members).getMember(index).getDisplayName();
```

Usage
This method is part of the TupleMember interface.

getAxis().getTuple().getMember(). getGenerationLevel()

Returns an integer representing the number of parent generations to which this member belongs. If the return value is 0, then the member has no parents.

Data Sources
Multidimensional

Syntax
Java Method

```
int getAxis(index).getTuple(members).getMember(index).getGenerationLevel();
```

Usage
This method is part of the TupleMember interface.

getAxis().getTuple().getMember().getIndex()

Returns the index of the member relative to other members in the tuple.

Data Sources
Multidimensional

Syntax
Java Method

```
int getAxis(index).getTuple(members).getMember(index).getIndex();
```

Usage
This method is part of the TupleMember interface.

getAxis().getTuple().getMember().getSpan()

Returns an integer representing the span (the number of tuple members that this tuple's parent member spans) for the specified member.

Data Sources

Multidimensional

Syntax

Java Method

```
int getAxis(index).getTuple(members).getMember(index).getSpan();
```

Usage

In the following example, the member named `First Quarter` would return a span of 3 (Actual, Budget, and Variance), and Q1 would return a span of 2 (Actual and Budget):

First Quarter			Q1		Q2	
Actual	Budget	Variance	Actual	Budget	Actual	Budget

This method is part of the `TupleMember` interface.

getAxis().getTuple().getMember().getSpanIndex()

Returns the 0-based index of the member in a series of spanned members. For example, if `First Quarter` has children `January`, `February`, and `March`, then the member `January` has a `spanIndex` of `0`.

Data Sources

Multidimensional

Syntax

Java Method

```
int getAxis(index).getTuple(members).getMember(index).getSpanIndex();
```

Usage

This method is part of the `TupleMember` interface.

getAxis().getTuple().getMember().getTuple()

Returns an interface to the tuple for the member.

Data Sources

Multidimensional

Syntax

Java Method

```
Tuple getAxis(index).getTuple(members).getMember(index).getTuple();
```

Usage

This method is part of the `TupleMember` interface.

getAxis().getTuple().getMember().getUniqueName()

Returns the unique name of the member

Data Sources
Multidimensional

Syntax
Java Method

```
String getAxis(index).getTuple(members).getMember(index).getUniqueName();
```

Usage
This method is part of the `TupleMember` interface.

getAxis().getTuple().getMember().isCalculatedMember()

Returns `true` if the member is an Alphablox calculated member or a Microsoft Analysis Services (session-based) calculated member.

Data Sources
Multidimensional

Syntax
Java Method

```
boolean getAxis(index).getTuple(members).getMember(index).isCalculatedMember();
```

Usage
This method is part of the `TupleMember` interface.

getAxis().getTuple().getMember().isLeaf()

Returns `true` if the member is a leaf node (that is, if the member has no children). Otherwise returns `false`.

Data Sources
Multidimensional

Syntax
Java Method

```
boolean getAxis(index).getTuple(members).getMember(index).isLeaf();
```

Usage
This method is part of the `TupleMember` interface.

getAxis().getTuple().getMemberCount()

Returns the number of members for the specified tuple.

Data Sources
Multidimensional

Syntax
Java Method

```
int getAxis(index).getTuple(members).getMemberCount();
```

Usage
This method is part of the `Tuple` interface.

getAxis().getTuple().getMembers()

Returns an array containing all the members within this tuple. The length of this array is equal to the int returned by the `getMemberCount` method.

Data Sources
Multidimensional

Syntax
Java Method

```
TupleMember[] getAxis(index).getTuple(members).getMembers();
```

Usage
This method is part of the `Tuple` interface.

See Also
"getAxis().getTuple().getMemberCount()" on page 416

getAxis().getTuples()

Returns an array of tuples found in the axis.

Data Sources
Multidimensional

Syntax
Java Method

```
Tuple[] getAxis().getTuples();
```

Usage
This method is part of the `Axis` interface.

getAxisCount()

Returns the number of axes in the cube, excluding the slicer axis.

Data Sources
Multidimensional

Syntax
Java Method

```
int getAxisCount();
```

Usage
This method is part of the `MDBResultSet` interface.

getCells()

Returns an interface to the cells of the cube.

Data Sources
Multidimensional

Syntax
Java Method

```
Cells getCells();
```

Usage
This method is part of the MDBResultSet interface.

getCells().getCell()
Returns the cell element for the cell specified in each method.

Data Sources
Multidimensional

Syntax
Java Methods

```
Cell getCells().getCell();
```

This is an overloaded method. Other forms are:

```
getCell(int col);
getCell(int[] coordinates);
getCell(int col, int row);
getCell(int col, int row, int page);
getCell(int col, int row, int page, int section);
getCell(int col, int row, int page, int section, int chapter);
getCell(Tuple columnTuple);
getCell(Tuple[] tuples);
getCell(Tuple columnTuple, Tuple rowTuple);
getCell(Tuple columnTuple, Tuple rowTuple, Tuple pageTuple);
getCell(Tuple columnTuple, Tuple rowTuple, Tuple pageTuple,
        Tuple sectionTuple);
getCell(Tuple columnTuple, Tuple rowTuple, Tuple pageTuple,
        Tuple sectionTuple, Tuple chapterTuple);
```

where:

Argument	Description
column	The column number, indexed starting from 0.
coordinates	The cell coordinates.
row	The row number, indexed starting from 0.
page	The page number, indexed starting from 0.
section	The section number, indexed starting from 0.
chapter	The chapter number, indexed starting from 0.
columnTuple	The column tuple.
pageTuple	The page tuple.
sectionTuple	The section tuple.
chapterTuple	The chapter tuple.

Usage
These methods are part of the Cells interface.

getCells().getCell(int).getCommentSet()
Returns the CommentSet object associated with the comment saved on the cell.

Data Sources
Multidimensional

Syntax
Java Methods

```
CommentSet getCommentSet();
   //throws CommentsBloxException,CommentsNoAccessException;
```

Usage
This method is part of the `Cell` interface.

See Also
"getCells().getCell().hasComments()" on page 420, "The CommentSet Object" on page 307

getCells().getCell().getCoordinates()
Returns the cell coordinates.

Data Sources
Multidimensional

Syntax
Java Methods

```
int[] getCells().getCell(int).getCoordinates();
```

Usage
This method is part of the `Cell` interface.

getCells().getCell().getDoubleValue()
Returns the cell value as a double.

Data Sources
Multidimensional

Syntax
Java Methods

```
double getCells().getCell(int).getDoubleValue();
```

Usage
This method is part of the `Cell` interface.

getCells().getCell().getIndex()
Returns the index number for the specified axis in the cell.

Data Sources
Multidimensional

Syntax
Java Method

```
int getCells().getCell(int).getIndex(int axisIndex);
```

where:

Argument	Description
`axisIndex`	The indexed number of the axis, indexed starting from 0.

Usage
This method is part of the `Cell` interface.

getCells().getCell().getTuple()
Returns an interface to the tuple for the axis specified.

Data Sources
Multidimensional

Syntax
Java Methods

```
Tuple getCells().getCell(int).getTuple(int axisIndex);
```

where:

Argument	Description
`axisIndex`	The indexed number of the axis, indexed starting from 0.

Usage
This method is part of the `Cell` interface.

getCells().getCell().getTuples()
Returns the coordinates for the cell as an array of tuples.

Data Sources
Multidimensional

Syntax
Java Method

```
Tuple[] getCells().getCell(int).getTuple(int axisIndex);
```

Usage
This method is part of the `Cell` interface.

getCells().getCell().getValue()

Returns the cell value as a string. The value is either a normal value or one of the following constants: `VALUE_MISSING`, `VALUE_NO_ACCESS`, or `VALUE_ERROR`

Data Sources
Multidimensional

Syntax
Java Method

```
String getCells().getCell(int).getValue();
```

Usage
This method is part of the `Cell` interface.

getCells().getCell().hasComments()
Identifies if there are comments stored on this cell.

Data Sources
Multidimensional

Syntax
Java Method

```
boolean hasComments();   //throws CommentsBloxException
```

Usage
This method is part of the `Cell` interface.

See Also
"getCells().getCell(int).getCommentSet()" on page 418, "The CommentSet Object" on page 307

getCubes()
Returns an array of Cube objects.

Data Sources
Multidimensional

Syntax
Java Method

```
Cube[] getCubes();
```

Usage
In IBM DB2 OLAP Server or Hyperion Essbase, the method always returns an array of size 1 made up of a cube with the name of the database. In Microsoft Analysis Services, the method usually returns an array of size 1 made up of a cube with the name from the FROM clause of the MDX query unless more than one cube is specified. This method is part of the `MDBResultSet` interface. For each Cube object, you can then call its `getDimension()`, `getDimensions()`, `getMetaData()`, `getMultipleHierarchies()`, and `getName()` methods. For methods available on the Cube object, see "Multidimensional Metadata Methods" on page 427.

getSlicerAxisIndex()
Returns the index number for the slicer axis of the cube.

Data Sources
Multidimensional

Syntax
Java Method

```
int getSlicerAxisIndex();
```

Usage
This method is part of the `MDBResultSet` interface.

resolveAxisDimension()
Returns an interface to the AxisDimension object for the dimension specified.

Data Sources
Multidimensional

Syntax

Java Method

```
AxisDimension resolveAxisDimension(String uniqueDimensionName);
```

where:

Argument	Description
uniqueDimensionName	The unique name of the dimension.

Usage

This method is part of the MDBResultSet interface.

The dimension name must be unique: a unique name in IBM DB2 OLAP Server or Hyperion Essbase, or a fully qualified name in data sources that use MDX (Microsoft Analysis Services and DB2 Alphablox cubes).

If a dimension in Microsoft Analysis Services data source has multiple hierarchies such as [Time].[Fiscal] and [Time].[Calendar], you need to specify the dimension for the hierarchy as follows:

```
resolveAxisDimension("[Time].[Fiscal]");
```

or

```
resolveAxisDimension("Time (Fiscal)");
```

This will return the AxisDimension object with the display name Time (Fiscal) and the unique name [Time].[Fiscal]. If you only specify `resolveAxisDimension("Time")` or `resolveAxisDimension("[Time]")`, null will be returned.

resolveTupleMember()

Returns an interface to the tuple member array for the member specified.

Data Sources

Multidimensional

Syntax

Java Method

```
TupleMember[]
   resolveTupleMember(String uniqueMemberName, boolean findAll);
```

where:

Argument	Description
uniqueDimensionName	The unique name of the member.
findAll	A boolean argument specifying whether to find all tuple members with the specified name (true) or to find the first tuple member that matches the specified name (false).

Usage

This method is part of the MDBResultSet interface.

The member name must be unique: a unique name in IBM DB2 OLAP Server or Hyperion Essbase, or a fully qualified name in data sources that use MDX (Microsoft Analysis Services and DB2 Alphablox cubes).

Relational Result Set Methods

The server-side relational ResultSet object (RDBResultSet) provides an interface to the result set for relational data sources such as IBM DB2 UDB, Oracle, Microsoft SQL Server, and Sybase. To access the methods on the RDBResultSet object, you must cast the getResultSet() or getRawResultSet() method to the RDBResultSet object as described in "getResultSet()" on page 395.

To use the APIs associated with RDBResultSet, you need to import the com.alphablox.blox.data.rdb package in your JSP page as follows:

```
<%@ page import="com.alphablox.blox.data.rdb.*" %>
```

This section describes all the methods available in the RDBResultSet object. This includes methods on the ResultColumn object. The methods in this section are organized alphabetically by their fully qualified object syntax. For a cross referenced list of methods on each object, see "Objects, Result Set and Metadata" on page 327.

For the methods available on DataBlox, see "DataBlox Methods" on page 385. For the syntax and descriptions of DataBlox methods that have a property associated with them, see "DataBlox Properties and Associated Methods" on page 334.

exceededMaximumRows()

Identifies if the returned result set from a relational drillthrough has exceeded the specified maximum rows.

Data Sources

Relational drillthrough from IBM DB2 OLAP Server, Hyperion Essbase, or Microsoft Analysis Services data sources

Syntax

Java Method

```
int exceededMaximumRows();
```

getColumn()

Returns the specified column for the result set.

Data Sources

Relational

Syntax

Java Method

```
ResultColumn getColumn(int index);
```

where:

Argument	Description
index	The index into the array of columns, specifying from which to get the result set.

Usage

This method is part of the RDBResultSet interface.

getColumn().getIndex()

Returns a 0-based index of the column.

Data Sources
Relational

Syntax
Java Method

```
int getColumn(index).getIndex();
```

Usage
This method is part of the `ResultColumn` interface.

getColumn().getName()

Returns the name of the column.

Data Sources
Relational

Syntax
Java Method

```
String getColumn(index).getName();
```

Usage
This method is part of the `ResultColumn` interface.

getColumn().getType()

Returns a constant indicating the data type of the column.

Data Sources
Relational

Syntax
Java Method

```
int getColumn(index).getType();
```

Usage
This method is part of the `ResultColumn` interface.

The name of the data type returned is from `java.sql.type`: `BIGINT`, `BINARY`, `BIT`, `CHAR`, `DATE`, `DECIMAL`, `DOUBLE`, `FLOAT`, `INTEGER`, `LONGVARBINARY`, `LONGVARCHAR`, `NUMERIC`, `REAL`, `SMALLINT`, `TIME`, `TIMESTAMP`, `TINYINT`, `VARBINARY`, `VARCHAR`.

getColumn().isNumeric()

Returns `true` if the column has a numeric data type, otherwise returns `false`.

Data Sources
Relational

Syntax
Java Method

```
boolean getColumn(index).isNumeric();
```

Usage
This method is part of the `ResultColumn` interface.

The `isNumeric()` method returns `true` if `getType()` equals one of the following types: `NUMERIC` , `DECIMAL`, `BIT`, `TINYINT`, `SMALLINT`, `INTEGER`, `BIGINT`, `REAL`, `FLOAT`, `DOUBLE`, otherwise it returns `false`.

getColumns()
Returns an array of columns for the result set.

Data Sources
Relational

Syntax
Java Method

```
ResultColumn[] getColumns();
```

Usage
This method is part of the `RDBResultSet` interface.

Examples
"Example 1: Walk Through a Relational Result Set" on page 909

getNextRow()
Returns the next row of data.

Data Sources
Relational

Syntax
Java Method

```
Object[] getNextRow(boolean nullsAsObject)
     throws RDBDataException
```

where:

Argument	Description
`nullsAsObject`	A boolean argument. When set to `true`, the method populates an element of the Object array with `com.alphablox.server.data.utils.MissingValue` when there is no value for the Object; when set to `false`, the element of the Object array is populated with a `null`.

Usage
This method is part of the `RDBResultSet` interface.

Examples
"Example 1: Walk Through a Relational Result Set" on page 909

getType()
Returns the data type of the specified data element within the column.

Data Sources
Relational

Syntax

Java Method

```
int getType(int index);
```

where:

Argument	Description
index	The index into the array of columns, specifying from which to get the data type.

Usage

This method is part of the RDBResultSet interface.

getTypes()

Returns an array of data types for each element in a row of data.

Data Sources

Relational

Syntax

Java Method

```
int[] getTypes();
```

Usage

This method is part of the RDBResultSet interface.

The names of the data types returned are from java.sql.type: BIGINT, BINARY, BIT, CHAR, DATE, DECIMAL, DOUBLE, FLOAT, INTEGER, LONGVARBINARY, LONGVARCHAR, NUMERIC, REAL, SMALLINT, TIME, TIMESTAMP, TINYINT, VARBINARY, VARCHAR.

Examples

"Example 1: Walk Through a Relational Result Set" on page 909

hasMoreRows()

Returns true if the result set has more data, otherwise returns false.

Data Sources

Relational

Syntax

Java Method

```
boolean hasMoreRows();
```

Usage

This method is part of the RDBResultSet interface.

resetCurrentRow()

Resets the current row pointer. The current row will reset to the first row.

Data Sources

Relational

Syntax
Java Method

```
void resetCurrentRow();
```

Usage
Resets the current row pointer, making the current row the first row. The next call to getNextRow() will return the first row. This method is part of the RDBResultSet interface.

Multidimensional Metadata Methods

The server-side multidimensional metadata object (MDBMetaData) provides an interface to the metadata for multidimensional data sources such as IBM DB2 OLAP Server, Hyperion Essbase, Microsoft Analysis Services and DB2 Alphablox cubes. To access the methods on the MDBMetaData object, you must cast the getMetaData() method to the MDBMetaData object as described in "getMetaData()" on page 393.

To use the APIs associated with the MDBMetaData object, you need to import the com.alphablox.blox.data.mdb package in your JSP page as follows:

```
<%@ page import="com.alphablox.blox.data.mdb.*" %>
```

This section describes all the methods available in the MDBMetaData object. This includes methods on the Cube, Dimension, and Member objects. The methods in this section are organized alphabetically by their fully qualified object syntax. For a cross referenced list of methods on each object, see "Objects, Result Set and Metadata" on page 327.

For the methods available on DataBlox, see "DataBlox Methods" on page 385. For the syntax and descriptions of DataBlox methods that have a property associated with them, see "DataBlox Properties and Associated Methods" on page 334.

Note: The object syntax shown for the methods in this section represents only one way to access the methods. There are other possibilities, depending on the outline of your data and the way you access different objects, for how to access a given method. For example, the following two method calls will both access the getChild method:

```
getCube(n).getDimension(n).getRootMember(n).getChild(n);
getCubes()[n].getDimensions()[n].getRootMembers()[n].getChild(n);
```

Tip: With Microsoft Analysis Services data sources, you may experience scalability problems due to large client cache memory consumption per connection. This can happen after, for example, a for loop with thousands of resolveMember() calls. In this case, you should call the clearClientCache() method afterwards (and set autoDisconnect to true) to free up the memory.

getCube()
Returns the cube with the specified index in the array of cubes in the database described by this metadata.

Data Sources
Multidimensional

Syntax
Java Method

```
Cube getCube(int index);
    // throws DataBloxBadConnectionException
```

where:

Argument	Description
index	The index into the array of cubes, specifying from which cube to get the metadata.

Usage

This method is part of the MDBMetaData interface.

Returns null if the cube with the specified index does not exist. Enables read-only access to the cube that is part of an underlying multidimensional data source. The getMDBMetaData() method will connect to the data source if necessary.

getCube().getDimension()

Returns the dimension for the specified index from the specified Cube object.

Data Sources

Multidimensional

Syntax

Java Method

```
Dimension getCube(index).getDimensions(int index);
     // throws DataBloxBadConnectionException, MDBDataException
```

where:

Argument	Description
index	The index into the array of dimensions, specifying from which dimension to get the metadata.

Usage

This method is part of the Cube interface.

Enables read-only access to the dimension for an underlying multidimensional data source. The getMDBMetaData() method will connect to the data source if necessary.

getCube().getDimension().getCube()

Returns the members cube for the specified dimension.

Data Sources

Multidimensional

Syntax

Java Method

```
Cube getCube(index).getDimension(index).getCube()
    throws DataBloxBadConnectionException
           MDBDataException
```

Usage

This method is part of the Dimension interface.

Enables read-only access to all of the members for a dimension of an underlying multidimensional data source. The `getMDBMetaData()` method will connect to the data source if necessary.

getCube().getDimension().getDisplayName()

Returns the display name for this dimension.

Data Sources

Multidimensional

Syntax

Java Method

```
String getCube(index).getDimension(index).getDisplayName();
```

Usage

This method is part of the `Dimension` interface.

Enables read-only access to the dimensions for an underlying multidimensional data source. The `getMDBMetaData()` method will connect to the data source if necessary.

getCube().getDimension().getLevels()

Returns an array of Level objects for the specified dimension.

Data Sources

Multidimensional

Syntax

Java Method

```
Level[] getCube(index).getDimension(index).getLevels();
```

Usage

A level is an element of a dimension hierarchy. Levels describe the hierarchy from the highest (most summarized) level to the lowest (most detailed) level of data. The first element of the array contains the level for the root members, while the last element in the array contains the level for the leaf members.

Methods available from the Level interface is described in "Level API" on page 440.

getCube().getDimension().getRootMember()

Returns the member (root level only) for the dimension specified.

Data Sources

Multidimensional

Syntax

Java Method

```
Member getCube(int index).getDimension(int index).getRootMember(int index);
    // throws DataBloxBadConnectionException, MDBDataException
```

where:

Argument	Description

index	The index into the root member array of the dimension.

Usage

This method is part of the Dimension interface.

Enables read-only access to the root-level member for a dimension of an underlying multidimensional data source. The getMDBMetaData() method will connect to the data source if necessary.

getCube().getDimension().getRootMember(). getAllDescendants()

Returns a Member array containing the descendants of this member.

Availability

Render Modes	All
Data Sources	Multidimensional

Syntax

Java Method

```
Member[]
getCube(index).getDimension(index).getRootMember(index).getAllDescendants();
          // throws DataBloxBadConnectionException, MDBDataException
```

Usage

This method is part of the Member interface.

Enables read-only access to all of the specified child member of an underlying multidimensional data source. The getMDBMetaData() method will connect to the data source if necessary.

getCube().getDimension().getRootMember(). getAllLeafDescendants()

Returns a Member array containing the leaf descendants of this member.

Availability

Render Modes	All
Data Sources	Multidimensional

Syntax

Java Method

```
Member[]
getCube(index).getDimension(index).getRootMember(index).getAllLeafDescendan
ts(); //throws DataBloxBadConnectionException, MDBDataException
```

Usage

This method is part of the Member interface.

Enables read-only access to all of the specified child member of an underlying multidimensional data source. The getMDBMetaData() method will connect to the data source if necessary.

getCube().getDimension().getRootMember().getChild()

Returns the child member with the specified index in the array of child members.

Data Sources

Multidimensional

Syntax

Java Method

```
Member getCube(index).getDimension(index).getRootMember(index).getChild(int
index);   // throws DataBloxBadConnectionException, MDBDataException
```

where:

Argument	Description
index	The index into the child of, for example, a root member.

Usage

This method is part of the Member interface.

Enables read-only access to all of the specified child member of an underlying multidimensional data source. The getMDBMetaData() method will connect to the data source if necessary.

getCube().getDimension().getRootMember().getChildren()

Returns an array containing the child members for the member specified.

Data Sources

Multidimensional

Syntax

Java Method

```
Member[] getCube(index).getDimension(index).getRootMember(index).getChildren();
```

Usage

This method is part of the Member interface.

Enables read-only access to all of the members for a dimension of an underlying multidimensional data source. The getMDBMetaData() method will connect to the data source if necessary.

getCube().getDimension().getRootMember().getDimension()

Returns the dimension to which the specified member belongs.

Data Sources

Multidimensional

Syntax

Java Method

```
Dimension
getCube(index).getDimension(index).getRootMember(index).getDimension();
```

Usage

This method is part of the Member interface.

Enables read-only access to all of the members for a dimension of an underlying multidimensional data source. The `getMDBMetaData()` method will connect to the data source if necessary.

getCube().getDimension().getRootMember().getDisplayName()

Returns the display name for the member specified.

Data Sources

Multidimensional

Syntax

Java Method

```
Member[]
getCube(index).getDimension(index).getRootMember(index).getDisplayName();
```

Usage

This method is part of the `Member` interface.

Enables read-only access to all of the members for a dimension of an underlying multidimensional data source. The `getMDBMetaData()` method will connect to the data source if necessary.

getCube().getDimension().getRootMember(). getGenerationLevel()

Returns the generation level for the member specified. The number is the distance from the top of the hierarchy; root members have the lowest generation level of 1, direct descendants of the root members have a generation level of 2, and so on.

Data Sources

Multidimensional

Syntax

Java Method

```
int
getCube(index).getDimension(index).getRootMember(index).getGenerationLevel(
);
```

Usage

This method is part of the `Member` interface.

Enables read-only access to all of the members for a dimension of an underlying multidimensional data source. The `getMDBMetaData()` method will connect to the data source if necessary.

getCube().getDimension().getRootMember().getParent()

Returns the parent member for the member specified.

Data Sources

Multidimensional

Syntax

Java Method

```
Member getCube(index).getDimension(index).getRootMember(index).getParent();
```

Usage
This method is part of the `Member` interface.

Enables read-only access to all of the members for a dimension of an underlying multidimensional data source. The `getMDBMetaData()` method will connect to the data source if necessary.

getCube().getDimension().getRootMember().getUniqueName()
Returns the unique name for the member specified.

Data Sources
Multidimensional

Syntax
Java Method

```
String getCube(index).getDimension(index).getRootMember(index).getUniqueName();
```

Usage
This method is part of the `Member` interface.

Enables read-only access to all of the members for a dimension of an underlying multidimensional data source. The `getMDBMetaData()` method will connect to the data source if necessary.

getCube().getDimension().getRootMember().isLeaf()

Returns `true` if the specified member has no children, otherwise returns `false`.

Data Sources
Multidimensional

Syntax
Java Method

```
boolean getCube(index).getDimension(index).getRootMember(index).isLeaf();
```

Usage
This method is part of the `Member` interface.

Enables read-only access to all of the members for a dimension of an underlying multidimensional data source. The `getMDBMetaData()` method will connect to the data source if necessary.

getCube().getDimension().getRootMembers()
Returns an array of the members (root level only) for the dimension(s) specified.

Data Sources
Multidimensional

Syntax
Java Method

```
Member[] getCube(index).getDimension(index).getRootMembers();
    //throws DataBloxBadConnectionException, MDBDataException
```

Usage
This method is part of the `Dimension` interface.

Enables read-only access to the root-level members for a dimension of an underlying multidimensional data source. The `getMDBMetaData()` method will connect to the data source if necessary.

getCube().getDimension().getUniqueName()

Returns the unique name for this dimension.

Data Sources

Multidimensional

Syntax

Java Method

```
String getCube(index).getDimension(index).getUniqueName();
```

Usage

This method is part of the `Dimension` interface.

Enables read-only access to all of the members for a dimension of an underlying multidimensional data source. The `getMDBMetaData()` method will connect to the data source if necessary.

getCube().getDimension().getType()

Returns the type of the dimension. The dimension types are listed below.

Data Sources

Multidimensional

Syntax

Java Method

```
int getCube(index).getDimension(index).getType();
```

Usage

This method is part of the `Dimension` interface.

Enables read-only access to the dimensions for an underlying multidimensional data source. The `getMDBMetaData()` method will connect to the data source if necessary.

The valid dimension types are represented by the following constants:

Constant Returned	
`UNKNOWN_DIMENSION_TYPE`	The dimension type cannot be determined.
`NORMAL_DIMENSION`	A normal dimension type (not measures, time, etc.).
`MEASURES_DIMENSION`	The measures dimension.
`TIME_DIMENSION`	The time dimension.
`ATTR_DIMENSION`	The IBM DB2 OLAP Server or Hyperion Essbase attribute dimension.
`CALC_ATTR_DIMENSION`	The internal calculated IBM DB2 OLAP Server or Hyperion Essbase attribute dimension.

getCube().getDimensions()

Returns an array of the dimensions from the Cube object.

Data Sources
Multidimensional

Syntax
Java Method

```
Dimension[] getCube(index).getDimensions();
    //throws DataBloxBadConnectionException, MDBDataException
```

Usage
This method is part of the `Cube` interface.

Enables read-only access to the dimensions for an underlying multidimensional data source. The `getMDBMetaData()` method will connect to the data source if necessary.

If a dimension in Microsoft Analysis Services data source has multiple hierarchies such as [Time].[Fiscal] and [Time].[Calendar], both will be returned and will be counted as two dimensions (in Microsoft Analysis Services, multiple hierarchies are actually dimensions with names that share the same prefix followed by a period but have different suffixes).

getCube().getMetaData()

Returns the MDBMetaData object for the cube or cubes specified.

Data Sources
Multidimensional

Syntax
Java Method

```
MDBMetaData getCube(index).getMetaData();
```

Usage
This method is part of the `Cube` interface.

Enables read-only access to the metadata for a specified cube in an underlying multidimensional data source. The `getMDBMetaData()` method will connect to the data source if necessary.

getCube().getMultipleHierarchies()

Returns an array of the Dimension objects which make up the multiple hierarchies under the specified dimension.

Data Sources
Microsoft Analysis Services

Syntax
Java Method

```
Dimension[] getMultipleHierarchies(String dimensionName);
     //throws DataBloxBadConnectionException,MDBDataException
```

where:

Argument	Description
dimensionName	The name of the dimension that has multiple hierarchies.

Usage

Microsoft Analysis Services supports dimensions with multiple hierarchies that provide similar yet alternate views of cube data. A dimension with multiple hierarchies is defined as two or more dimensions with names that share the same prefix followed by a period but have different suffixes. If your cube has the dimensions [Time].[Fiscal] and [Time].[Calendar], calling `getMultipleHierarchies("[Time]")` or `getMultipleHierarchies("Time")` returns an array of length 2 with the Dimension objects corresponding to the dimensions [Time].[Fiscal] and [Time].[Calendar]. If you use this method on a dimension that has only one hierarchy, only one dimension will be returned.

See Also

"mergedDimensions" on page 367

getCube().getName()

Returns the name of the specified cube.

Data Sources

Multidimensional

Syntax

Java Method

```
String getCube(index).getName();
```

Usage

This method is part of the `Cube` interface.

Enables access to the name of the cube in an underlying multidimensional data source. The `getMDBMetaData()` method will connect to the data source if necessary.

getCubes()

Returns an array of the cubes from the MDBMetaData object.

Data Sources

Multidimensional

Syntax

Java Method

```
Cube[] getCubes()
    throws DataBloxBadConnectionException
```

Usage

This method is part of the `MDBMetaData` interface.

In IBM DB2 OLAP Server or Hyperion Essbase, there is only one cube and it is always called `Cube 1`.

Enables read-only access to the cubes that are part of an underlying multidimensional data source. The `getMDBMetaData()` method will connect to the data source if necessary.

getNamedDBCalcScriptContents()

Returns the contents of a named calculation script from the IBM DB2 OLAP Server or Hyperion Essbase server.

Data Sources

IBM DB2 OLAP Server or Hyperion Essbase only

Syntax

Java Method

```
String getNamedDBCalcScriptContents(String calcScript);
```

where:

Argument	Description
`calcScript`	A String containing the named calculation script stored in the IBM DB2 OLAP Server or Hyperion Essbase database.

Usage

This method is part of the `MDBMetaData` interface.

The application name and database name used in the IBM DB2 OLAP Server or Hyperion Essbase calculation script must exactly match the values for the catalog and schema parameters on DataBlox. This method is silently ignored by non-Essbase data sources.

getPropertiesOfMember()

Returns an array of Property object representing the properties of the specified member.

Data Sources

Microsoft Analysis Services

Syntax

Java Method

```
Property[] getPropertiesOfMember(String uniqueMemberName);
     // throws DataBloxBadConnectionException, MDBDataException
```

where:

Argument	Description
`uniqueMemberName`	The unique name of the member to find.

Usage

Returns null if no such property exists. A member property is an attribute of a dimension member. It is optional, but is often used to provide end users with additional information about the member. To access the property name and value, see “getPropertiesOfMember().getName()” on page 437 and “getPropertiesOfMember().getValue()” on page 438.

getPropertiesOfMember().getName()

Returns the name of the member property.

Data Sources

Microsoft Analysis Services

Syntax
Java Method

```
String getName();
```

See Also
"getPropertiesOfMember()" on page 437.

getPropertiesOfMember().getValue()
Returns the value of the member property.

Data Sources
Microsoft Analysis Services

Syntax
Java Method

```
java.lang.Object getValue();
```

Usage
Returns the value for the member property as an `Object`. You can then use methods available to java.lang.Object such as `toString()` to convert it to a String or `equals()` for comparison.

See Also
"getPropertiesOfMember()" on page 437.

resolveDimension()
Finds and returns the dimension with the specified unique dimension name within the database.

Data Sources
Multidimensional

Syntax
Java Method

```
Dimension resolveDimension(String uniqueDimensionName);
     //throws DataBloxBadConnectionException, MDBDataException
```

where:

Argument	Description
`uniqueDimensionName`	The unique name of the dimension.

Usage
This method is part of the `MDBMetaData` interface.

The dimension name must be unique: a unique name in IBM DB2 OLAP Server or Hyperion Essbase, or a fully qualified name in data sources that use MDX (Microsoft Analysis Services and DB2 Alphablox cubes).

Microsoft Analysis Services supports multiple hierarchies, which are dimensions with names that share the same prefix followed by a period but have different suffixes. If multiple hierarchies such as [Time].[Fiscal] and [Time].[Calendar] in a Microsoft Analysis Services data source are merged into a "Time" dimension, you need to specify the dimension for the hierarchy as shown in either of the following examples:

```
resolveDimension("[Time].[Fiscal]");  //returns a Dimension object
resolveDimension("Time (Fiscal)");     //returns a Dimension object
```

The above examples will return a Dimension object with the unique name [Time].[Fiscal] or the display name Time (Fiscal). Both will have the same dimension root.

Since the merged "Time" dimension does not actually exist, the following code will result in a MDBException:

```
resolveDimension("[Time]");   //results in a MDBException
```

resolveMember()

Finds and returns the member with the specified unique member name within the database.

Data Sources

Multidimensional

Syntax

Java Method

```
Member resolveMember(String uniqueMemberName);
           throws DataBloxBadConnectionException,
                  MDBDataException

Member resolveMember(String uniqueMemberName,
                     boolean searchAgainstDisplayName);
           throws DataBloxBadConnectionException,
                  MDBDataException
```

where:

Argument	Description
`uniqueMemberName`	The unique name of the member.
`searchAgainstDisplayName`	Whether to also search against display names

Usage

This method is part of the `MDBMetaData` interface. It searches and returns the first member with a matching unique or display name. If `searchAgainstDisplayName` is set to `true`, this method also searches against display names. If `searchAgainstDisplayName` is set to `false`, it only searches against unique names. This results in a faster search.

The member name specified is case-sensitive and must be unique: a unique name in IBM DB2 OLAP Server or Hyperion Essbase, or a fully qualified name in data sources that use MDX (Microsoft Analysis Services and DB2 Alphablox cubes).

For data sources that support multiple cubes, you can pre-pend the cube name to the member name to ensure that the member is resolved against only the specific cube. For example, if the cube name is QCC, and:

- the member you are looking for has the unique name of [Product].[All Products].[L0].[All Products], the search string should be [QCC].[Product].[All Products].[L0].[All Products]
- the member you are looking for has the display name of All Products, the search string should be [QCC].All Products

If the member is not found in the specified cube, `resolveMember()` will not search against the other cubes.

Level API

This section lists the methods available to the Level interface. You can access the Level object using the `getCube().getDimension().getLevels()` method from a DataBlox.

getDimension()

Returns the dimension that contains this level.

Data Sources: Multidimensional

Syntax: Java Method

```
Dimension getDimension();
```

getMembers()

Returns an array of Members containing all the members on this level.

Data Sources: Multidimensional

Syntax: Java Method

```
Member[] getMembers();  // throws NotFoundException, BadConnectionException,
                                  DataBloxBadConnectionException,
                                  MDBDataException
```

getName()

Returns the display name for this level.

Data Sources: Multidimensional

Syntax: Java Method

```
String getName();
```

getUniqueName()

Returns the unique name for this level.

Data Sources: Multidimensional

Syntax: Java Method

```
String getUniqueName();
```

Relational Database Metadata Methods

The server-side relational metadata object (`RDBMetaData`) provides an interface to the metadata for relational data sources such as IBM DB2 UDB, Oracle, Microsoft SQL Server, and Sybase. To access the methods on the `RDBMetaData` object, you must cast the `getMetaData()` method to the `RDBMetaData` object as described in "getMetaData()" on page 393.

To use the APIs associated with the RDBMetaData object, you need to import the com.alphablox.blox.data.rdb package in your JSP page as follows:

```
<%@ page import="com.alphablox.blox.data.rdb.*" %>
```

This section describes all the methods available in the RDBMetaData object. This includes methods on the Table and Column objects. The methods in this section are organized alphabetically by their fully qualified object syntax. For a cross referenced list of methods on each object, see "Objects, Result Set and Metadata" on page 327.

For the methods available on DataBlox, see "DataBlox Methods" on page 385. For the syntax and descriptions of DataBlox methods that have a property associated with them, see "DataBlox Properties and Associated Methods" on page 334.

Note: The object syntax shown for the methods in this section represents only one way to access the methods. There are other possibilities for how to access a given method. For example, the following two method calls will both access the getName method:

```
getTable(n).getColumn(n).getName();
getTables()[n].getColumns()[n].getName();
```

getTable()

Returns an interface to the specified table.

Data Sources

Relational

Syntax

Java Method

```
Table getTable(String tableName)
     throws RDBDataException
```

where:

Argument	Description
tableName	The name of a specified table.

Usage

This method is part of the RDBMetaData interface.

Enables access to the metadata for a table in the underlying relational data source. The DataBlox getMetaData() method will connect to the data source if necessary. The RDBMetaData object returned is read-only.

getTable()

Returns an interface to the table specified by the index.

Data Sources

Relational

Syntax

Java Method

```
Column getTable(int index)
     throws RDBDataException
```

where:

Argument	Description
index	The index into the specified table.

Usage
This method is part of the RDBMetaData interface.

Enables access to the metadata for a table in the underlying relational data source. The DataBlox getMetaData() method will connect to the data source if necessary. The RDBMetaData object returned is read-only.

getTable().getColumn()
Returns a specified column object for the specified table.

Data Sources
Relational

Syntax
Java Method

```
Column getTable(index).getColumn(String columnName)
     throws RDBDataException
```

where:

Argument	Description
columnName	The name of a specified column.

Usage
This method is part of the Table interface.

Enables access to the metadata for an underlying relational data source. The DataBlox getMetaData() method will connect to the data source if necessary. The RDBMetaData object returned is read-only.

getTable().getColumn()
Returns an interface to the column specified by the index.

Data Sources
Relational

Syntax
Java Method

```
Column getTable(index).getColumn(int index)
     throws RDBDataException
```

where:

Argument	Description
index	The index into the specified column.

Usage
This method is part of the Table interface.

getTable().getColumns()
Returns an array of columns for the specified table(s).

Data Sources
Relational

Syntax
Java Method

```
Column[] getTable(index).getColumns()
     throws RDBDataException
```

Usage
This method is part of the `Table` interface.

Enables access to the metadata for an underlying relational data source. The DataBlox `getMetaData()` method will connect to the data source if necessary. The RDBMetaData object returned is read-only.

getTable().getColumn().getDistinctValues()
Returns an array of columns for the specified table(s) or the value of a specified column (for the indexed method).

Data Sources
Relational

Syntax
Java Method

```
Object[] getTable(index).getColumn(index).getDistinctValues()
     throws RDBDataException
Object[] getTables().getColumns().getDistinctValues(int index)
     throws RDBDataException
```

Usage
This method is part of the `Column` interface.

Enables access to the metadata for an underlying relational data source. The DataBlox `getMetaData()` method will connect to the data source if necessary. The RDBMetaData object returned is read-only.

getTable().getColumn().getName()
Returns the name of a specified column.

Data Sources
Relational

Syntax
Java Method

```
String getTable(index).getColumn(index).getName()
     throws RDBDataException
```

Usage
This method is part of the `Column` interface.

Enables access to the metadata for an underlying relational data source. The DataBlox `getMetaData()` method will connect to the data source if necessary. The RDBMetaData object returned is read-only.

Examples
"Example 1: Walk Through a Relational Result Set" on page 909

getTable().getColumn().isNumeric()

Returns a boolean indicating if a column is a numeric data type. If a column is numeric, it returns `true`, otherwise, it returns `false`.

Data Sources
Relational

Syntax
Java Method

```
boolean getTable(index).getColumn(index).isNumeric()
```

Usage
This method is part of the `Columns` interface.

The `isNumeric()` method returns `true` if `getType()` equals one of the following types: `NUMERIC` , `DECIMAL`, `BIT`, `TINYINT`, `SMALLINT`, `INTEGER`, `BIGINT`, `REAL`, `FLOAT`, `DOUBLE`, otherwise it returns `false`.

Enables access to the metadata for an underlying relational data source. The DataBlox `getMetaData()` method will connect to the data source if necessary. The RDBMetaData object returned is read-only.

getTable().getColumn().getType()

Returns the name of the data type of a specified column.

Data Sources
Relational

Syntax
Java Method

```
String getTable(index).getColumn(index).getName()
```

Usage
This method is part of the `Columns` interface.

The names of the data types returned are from `java.sql.type`: `BIGINT`, `BINARY`, `BIT`, `CHAR`, `DATE`, `DECIMAL`, `DOUBLE`, `FLOAT`, `INTEGER`, `LONGVARBINARY`, `LONGVARCHAR`, `NUMERIC`, `REAL`, `SMALLINT`, `TIME`, `TIMESTAMP`, `TINYINT`, `VARBINARY`, `VARCHAR`.

getTable().getName()

Returns the name of the specified table(s).

Data Sources
Relational

Syntax
Java Method

```
String getTable(index).getName();
```

Usage
This method is part of the `Table` interface.

Enables access to the metadata for an underlying relational data source. The DataBlox `getMetaData()` method will connect to the data source if necessary. The RDBMetaData object returned is read-only.

getTable().getType()

Returns a string indicating if the specified table is a table, a view, or a synonym.

Data Sources

Relational

Syntax

Java Method

```
String getTable(index).getType();
```

Usage

This method is part of the `Table` interface.

Returns one of the following indicating what the database object is: `TABLE`, `VIEW`, or `SYNONYM`.

getTables()

Returns an array of all the tables for the data source.

Data Sources

Relational

Syntax

Java Method

```
Table[] getTables()
     throws RDBDataException
```

Usage

This method is part of the `RDBMetaData` interface.

Enables access to the metadata for an underlying relational data source. The DataBlox `getMetaData()` method will connect to the data source if necessary. The RDBMetaData object returned is read-only.

getTables()

Returns all the tables of the specified type.

Data Sources

Relational

Syntax

Java Method

```
String getTables(String type)
     throws RDBDataException
```

where:

Argument	Description
`type`	One of the following types of tables: `TABLE`, `VIEW`, or `SYNONYM`.

Usage

This method is part of the RDBMetaData interface.

Enables access to the metadata for a table in the underlying relational data source. The DataBlox getMetaData() method will connect to the data source if necessary. The RDBMetaData object returned is read-only.

Calculation Methods

The calculatedMembers property allows you to define calculated members by specifying the calculation expression and the dimension on which to create the calculated member. Optionally, you can specify the generation of the calculated member, the scope for which the calculated member is displayed, whether missing values should be treated as zero, and more. The calculation expression supports various arithmetic functions (such as Abs, Average, Round, Sqrt, Stdev and Sum), search functions (such as Child and Leaf), two-pass calculation functions (such as Rank and RunningTotal), and missing value function (ifNotNumber). See "calculatedMembers" on page 337 for detailed syntax, usage, and examples.

Through the DataBlox's getCalculations() method you can programmatically identify the individual component or function involved in the calculation. This method returns an array of calculated members, each of which is of type Calculation. The type Calculation and all of its related classes can be found in the com.alphablox.blox.data.calculation package. The interfaces in the package include:

- "Calculation Interface" on page 446
- "CalcBinaryExpression Interface" on page 449
- "CalcConstant Interface" on page 450
- "CalcError Interface" on page 450
- "CalcFunction Interface" on page 450
- "CalcFunctionMultiDim Interface" on page 451
- "CalcNotNumberFunction Interface" on page 452
- "CalcOperand Interface" on page 452
- "CalcScope Interface" on page 453
- "CalcScopeItem Interface" on page 453
- "CalcUnaryExpression Interface" on page 454
- "CalcVariable Interface" on page 454

Note: Only setter methods are available on the Calculation. See the *Administrator's Guide* for instruction on how to set the classpath when you write your own Java classes extending this package.

Calculation Interface

The Calculation interface is a representation of a calculation in the data. It contains the following parts: Name, Dimension, Scope, IsMissingZero, RelativeMemberName, Generation, Relative Generation and an expression, which can either be a string representation or a CalcOperand.

This section describes all methods in the base class Calculation. Use DataBlox().getCalculations() to get at the parsed calculations.

getDimension()

Gets the dimension name on which the calculation is being done.

Data Sources
All

Syntax
Java Method

```
String getDimension();
```

getExpression()
Gets the unparsed expression for this calculation.

Data Sources
All

Syntax
Java Method

```
String getExpression();
```

Usage
The returned string is exactly what the right hand side of the calculation is. Another alternative is to use the getOperand() method to access the details of the type of Calculation.

getGeneration()
Gets the generation of the calculation.

Data Sources
All

Syntax
Java Method

```
int getGeneration();
```

Usage
The default value for generation is 1.

getName()
Gets the name of the calculated member as displayed in the user interface.

Data Sources
All

Syntax
Java Method

```
String getName();
```

getOperand()
Gets the parsed expression of the calculation as an operand.

Data Sources
All

Syntax
Java Method

```
CalcOperand getOperand();
```

See Also
"CalcOperand Interface" on page 452

getRelativeGeneration()

Gets the generation relative to the specified relative member name.

Data Sources
All

Syntax
Java Method

```
int getRelativeGeneration();
```

getRelativeMemberName()

Gets the member name before which the calculation will be placed.

Data Sources
All

Syntax
Java Method

```
String getRelativeMemberName();
```

getScope()

Gets the scope of the calculation.

Data Sources
All

Syntax
Java Method

```
CalcScope getScope();
```

See Also
"CalcScope Interface" on page 453

isMissingIsZero()

Gets whether or not `missingIsZero` is set for this calculation.

Data Sources
All

Syntax
Java Method

```
boolean isMissingIsZero();
```

toString()

Returns the calculation as a string.

Data Sources
All

Syntax
Java Method
```
String toString();
```

Usage
The returned string is basically the value of the `calculatedMember` property except that there may be differences with spacing, case, and parentheses.

CalcBinaryExpression Interface
This interface is a representation of a binary expression in a calculation. A binary expression consists of a left operand, a right operand and an operator. The left and right operands could be any CalcOperand type, as the calculations are nested. The operator will be one of `+`, `-`, `*`, or `/`. The `toString()` method will give a string representation of the binary expression.

getLeftOperand()
Gets the left operand of the expression.

Data Sources
All

Syntax
Java Method
```
CalcOperand getLeftOperand();
```

See Also
"CalcOperand Interface" on page 452

getOperator()
Gets the operator for this expression.

Data Sources
All

Syntax
Java Method
```
char getOperator();
```

Usage
The operator will be one of `+`, `-`, `*`, or `/`.

getRightOperand()
Gets the right operand of this expression.

Data Sources
All

Syntax
Java Method
```
CalcOperand getRightOperand();
```

See Also
"CalcOperand Interface" on page 452

CalcConstant Interface

If a CalcOperand is of type CalcConstant, then the value is a constant. There is only one method in this interface.

getValue()

Gets the value of the constant as a double.

Data Sources

All

Syntax

Java Method

```
double getValue();
```

CalcError Interface

This class is a representation of a calculation error. It simply contains a `toString()` method to get a string representation of the syntax error. This section describes the method in the CalcError interface.

toString()

Returns the syntax error as a string.

Data Sources

All

Syntax

Java Method

```
String toString();
```

CalcFunction Interface

If a CalcOperand is of type CalcFunction, then it is a function, such as `Sum`, `Average`, `Min`, `Max`, `Stdev`, or others. It contains a function name and a list of operands which are the sources of the function.

getFunctionName()

Returns the name of the function.

Data Sources

All

Syntax

Java Method

```
String getFunctionName();
```

Usage

Returns the name of the function, such as `Abs`, `Average`, `Round`, `Sqrt`, `Min`, `Max`, `Sum`, and `Var`.

getOperands()

Returns a list of the operands that make up the source operands for this function

Data Sources

All

Syntax
Java Method
```
CalcOperand[] getOperands();
```

Usage
When a calculation function contains a list of source operands (for example, `Sum(East, West, Central)`), this method returns all the source operands of type CalcOperand. Otherwise, the array will be empty.

See Also
"getParam()" on page 451, "CalcOperand Interface" on page 452

getParam()
Returns the unparsed string of the parameters of the function.

Data Sources
All

Syntax
Java Method
```
String getParam();
```

Usage
When a calculation function contains not a list of operands but only one parameter (for example, `Sum(gen(2))`), this method returns the parameter as a string. In the example given, the string returned will be `gen(2).`

See Also
"getOperands()" on page 450

CalcFunctionMultiDim Interface
CalcFunctionMultiDim represents a function which works on multiple dimensions. If a CalcFunction is of type CalcFunctionMultiDim, then it is a calculation that involves multiple dimensions. Example of these are `Rank` and `RunningTotal`. This interface extends CalcFunction, so it contains all the methods available in CalcFunction.

getFunctionName()
This is a method inherited from the CalcFunction interface. See "getFunctionName()" on page 450.

getMinimumParameterCount()
Returns the minimum number of parameters that are required for this function to be valid.

Data Sources
All

Syntax
Java Method
```
int getMinimumParameterCount();
```

getOperands()

This is a method inherited from the CalcFunction interface. See “getOperands()” on page 450.

getParam()

This is a method inherited from the CalcFunction interface. See “getParam()” on page 451.

getParams()

Returns string representations of the parameters.

Data Sources

All

Syntax

Java Method

```
String[] getParams();
```

CalcNotNumberFunction Interface

CalcNotNumberFunction is a representation of the `ifNotNumber` calculation function. It extends CalcVariable and contains just one additional method to get the value to substitute if the member is not a number.

getName()

This is a method inherited from the CalcVariable interface. See “getName()” on page 454.

getSubstituteValue()

Gets the substitute value to be used if the member name is not a number.

Data Sources

All

Syntax

Java Method

```
double getSubstituteValue();
```

CalcOperand Interface

CalcOperand is the base class for all operands which are used in a calculation. It includes the following sub-interfaces:

- “CalcBinaryExpression Interface” on page 449
- “CalcConstant Interface” on page 450
- “CalcError Interface” on page 450
- “CalcFunction Interface” on page 450
- “CalcFunctionMultiDim Interface” on page 451
- “CalcNotNumberFunction Interface” on page 452
- “CalcUnaryExpression Interface” on page 454
- “CalcVariable Interface” on page 454

You should use `instanceof` to check the actual type of the operand. The `toString()` method will give a string representation of the operand.

CalcScope Interface

CalcScope contains the scope part of the calculation syntax. It contains an array of individual scope items of type CalcScopeItem. You can access this interface via the `DataBlox().getCalculations().getScope()` method.

getScopeItems()

Gets an array of scope items that make up the scope.

Data Sources

All

Syntax

Java Method

```
CalcScopeItem[] getScopeItems();
```

toString()

Returns the string of the scope.

Data Sources

All

Syntax

Java Method

```
String toString();
```

CalcScopeItem Interface

You can access this interface via the `DataBlox().getCalculations().getScope().getScopeItems()` method. .

getDimension()

Gets the dimension of this scope item.

Data Sources

All

Syntax

Java Method

```
String getDimension();
```

getMembers()

Gets the members of this dimension in the scope.

Data Sources

All

Syntax

Java Method

```
String[] getMembers();
```

toString()

Returns a string for this scope item.

Data Sources
All

Syntax
Java Method

```
String toString();
```

CalcUnaryExpression Interface

If the CalcOperand is of type CalcUnaryExpression, then it is a value that has only one operand. A unary expression is typically used to negate a constant. For example, if a calculation involves multiplying a number by -1, then -1 would be represented as a CalcUnaryExpression which has an operand of 1 and an operator of "-". This section describes all methods in the CalcUnaryExpression interface.

getOperand()

Returns the CalcOperand.

Data Sources
All

Syntax
Java Method

```
CalcOperand getName();
```

See Also
"CalcOperand Interface" on page 452

getOperator()

Returns the operator.

Data Sources
All

Syntax
Java Method

```
char getOperator();
```

Usage
The returned operator is usually "-", which is used to negate a constant.

CalcVariable Interface

If a CalcOperand is of type CalcVariable, then it is a variable name.

getName()

Gets the name of the variable.

Data Sources
All

Syntax

Java Method

```
String getName();
```

Chapter 12. DataLayoutBlox Reference

This chapter contains reference material for DataLayoutBlox. For general reference information about Blox, see Chapter 3, "General Blox Reference Information," on page 17. For information on how to use this reference, see Chapter 1, "Using This Reference," on page 1.

DataLayoutBlox Overview

DataLayoutBlox provides a graphical representation of multidimensional database dimensions, organized by the axis on which they appear (row, column, or page). A fourth axis, other, lists dimensions that do not appear on any other axis, but are available in the current result set.

Graphical User Interface

The DataLayoutBlox GUI allows users to perform the following tasks:
- view and move dimensions between axes
- drag-and-drop dimension into or out of GridBlox
- access the Member Filter

DataLayoutBlox has two interface types: tree (default) and drop list. The tree interface contains expandable and collapsible tree menus and supports drag-and-drop operations. The drop list interface uses drop lists to support moving dimensions among axes.

For instructions on using the DataLayoutBlox user interface, see DataLayoutBlox user help. You can access the user help by clicking the help button on the toolbar in the Blox user interface.

DataLayoutBlox JSP Custom Tag Syntax

The Alphablox Tag Libraries provides custom tags to use in a JSP page for creating each Blox. This section describes how to create the custom tag to create a DataLayoutBlox. For a copy and paste version of the tag with all the attributes, see "DataLayoutBlox JSP Custom Tag" on page 886.

```
<blox:dataLayout
       [attribute="value"] >
</blox:dataLayout>
```

where:

attribute is one of the attributes listed in the attribute table.

value is a valid value for the attribute.

and where the attributes are one of the following:

Attribute
id
applyPropertiesAfterBookmark
bloxEnabled
bloxName
bookmarkFilter
height
helpTargetFrame
hiddenDimensionsOnOtherAxis
interfaceType
localeCode
maximumUndoSteps
noDataMessage
render
visible
width

Usage

Each custom tag can have one or more attributes, separated by one or more space or new line characters. Any extra space or new line characters are ignored. For readability, attributes can each go on a separate line with the same indenting.

You can substitute the closing `</blox:dataLayout>` tag using the shorthand notation, closing the tag at the end of the attribute list that looks as follows:

```
width="650" />
```

Examples

```
<blox:dataLayout
     id="namedDataLayoutBlox"
     width="100"
     height="400" />
```

DataLayoutBlox Properties/Tag Attributes

This section lists the properties and tag attributes available through DataLayoutBlox alphabetically. For complete descriptions of common Blox properties, see “Properties and Associated Methods Common to Multiple Blox” on page 32. For a list of DataLayoutBlox methods with which no properties are associated, see “DataLayoutBlox Methods” on page 461.

id

This is a common Blox property. For a complete description, see “id” on page 39.

applyPropertiesAfterBookmark

This is a common Blox property. For a complete description, see “applyPropertiesAfterBookmark” on page 32.

bloxEnabled

This is a common Blox property. For a complete description, see “bloxEnabled” on page 35.

bloxModel

This is a common Blox property. For a complete description, see “bloxModel” on page 38

bloxName

This is a common Blox property. For a complete description, see “bloxName” on page 35.

bookmarkFilter

This is a common Blox property. For a complete description, see “bookmarkFilter” on page 33.

height

This is a common Blox property. For a complete description, see “height” on page 38.

helpTargetFrame

This is a common Blox property. For a complete description, see “helpTargetFrame” on page 39.

hiddenDimensionsOnOtherAxis

Specifies the dimensions to be hidden on the Other axis in the Data Layout panel.

Data Sources

Multidimensional

Syntax

JSP Tag Attribute

```
hiddenDimensionsOnOtherAxis="dimensionsToHide"
```

Java Methods

```
String getHiddenDimensionsOnOtherAxis();
void setHiddenDimensionsOnOtherAxis(String dimensionsToHide);
                    // throws ServerBloxException
```

where:

Argument	Description
dimensionsToHide	A comma-delimited String representing the dimensions to hide on the Other axis.

Usage

By default, any dimensions in the cube not specified in the data query are placed in the Other axis in the Data Layout panel. By hiding the dimensions in the Other axis, you can prevent users from pivoting these dimensions or changing the currently selected member for these dimensions. For example, you may want to hide IBM DB2 OLAP Server or Hyperion Essbase Attribute dimensions or prevent your user from changing the Measures filter. Or in your Microsoft Analysis

Services data source, you may have multiple hierarchies [Time].[Fiscal] and [Time].[Calendar]. Your query already specifies to have [Time].[Fiscal] on the Column axis and need to hide [Time].[Calendar] in the Other axis to avoid confusion.

The dimensions are only hidden on the UI; they still exist in the data. If a dimension is specified in your query to appear, for example, on the Column axis and you hide it again on the Other axis, the dimension will still show on the Column axis. However, once the users pivot the dimension into the Other axis, it will become hidden from then on. To unhide a dimension, the `hiddenDimensionsOnOtherAxis` property needs to be reset.

Examples

The following example hides the [Time].[Calendar], Measures, and [Attribute Calcs] dimensions from on the Other axis.

```
hiddenDimensionsOnOtherAxis="[Time].[Calendar], Measures,
[Attribute Calcs]"
```

interfaceType

Sets the interface type for the Data Layout panel.

Data Sources

Multidimensional

Syntax

JSP Tag Attribute

```
interfaceType="type"
```

Java Methods

```
String getInterfaceType();
   // throws ServerBloxException
void setInterfaceType(String type);
   // throws InvalidBloxPropertyValueException, ServerBloxException
```

where:

Argument	Description
`type`	Valid values are `tree` and `dropList`. The default is `tree`. When set to `tree`, the Data Layout panel has a tree navigation interface with tree menus that expand and collapse. See "DataLayoutBlox Overview" on page 457.

localeCode

This is a common Blox property. For a complete description, see "localeCode" on page 40.

maximumUndoSteps

This is a common Blox property. For a complete description, see "maximumUndoSteps" on page 41.

noDataMessage

This is a common Blox property. For a complete description, see "noDataMessage" on page 42.

render

This is a common Blox property. For a complete description, see “render” on page 45.

visible

This is a common Blox property. For a complete description, see “visible” on page 46.

width

This is a common Blox property. For a complete description, see “width” on page 47.

DataLayoutBlox Methods

DataLayoutBlox supports no unique methods other than those that access the ResultSet object. This section lists the DataLayoutBlox ResultSet object methods. For client-side API common to Blox, see “Client-Side APIs” on page 31.

addEventFilter()

This is a common Blox method that for capturing a server-side event (such as saving and loading bookmarks) and perform custom actions *after* the operation is complete on the server. For details, see “addEventListener()” on page 49.

addEventListener()

This is a common Blox method that allows you to capture a server-side event (such as saving and loading bookmarks) and perform custom actions *after* the operation is complete on the server. For details, see “addEventListener()” on page 49.

call()

This is a common client-side Blox method. For a complete description, see “call()” on page 51.

flushProperties()

This is a common client-side Blox method. For a complete description, see “flushProperties()” on page 52.

getDataBlox()

This is a common Blox method. For a complete description, see “setDataBlox()” on page 60.

loadBookmark()

This is a common Blox method. For a complete description, see “loadBookmark()” on page 54.

removeEventFilter()

This is a common Blox method that allows you to remove an event filter object added using addEventFilter() for capturing a server-side event (such as saving and loading a bookmark) *before* the event is processed on the server. For details, see “removeEventFilter()” on page 55.

removeEventListener()

This is a common Blox method that allows you to remove an event listener object created using addEventListener() for capturing a server-side event (such as saving and loading a bookmark) *after* that operation is complete on the server. For details, see “removeEventListener()” on page 56.

saveBookmark()

This is a common Blox method. For a complete description, see “saveBookmark()” on page 57.

saveBookmarkHidden()

This is a common Blox method. For a complete description, see “saveBookmarkHidden()” on page 58.

setDataBlox()

This is a common Blox method. For a complete description, see “setDataBlox()” on page 60.

setDataBusy()

This is a common client-side Blox method. For a complete description, see “setDataBusy()” on page 60.

updateProperties()

This is a common client-side Blox method. For a complete description, see “updateProperties()” on page 61.

Chapter 13. Event Filter Objects

This chapter describes the event filter objects and the methods used with them. The common Blox methods addEventFilter() and removeEventFilter() take the event listener objects as arguments and let you perform custom actions *before* the event has been processed on the server. For capturing events after they have been processed, see Chapter 14, "Event Listener Objects," on page 501.

Event Filter Objects Overview

You can capture an event and perform custom actions either *before* or *after* the event is processed on the server. The event filter objects are server-side objects that allow you to capture some user event such as drilling down or pivoting and perform some actions *before* the event is actually processed. To use the event filters, you need to first add the specific event filter object using the `addEventFilter()` method.

There are two types of event filter objects.

- DataBlox related: You can capture the following data analysis operations: collapse, drill down, drill through, drill up, expand, hide only, keep only, member select, pivot, remove only, show all, show only, swap axis, and data query.
- Bookmark related: You can capture the following bookmark related events: delete bookmark, load bookmark, rename bookmark, and save bookmark.

Once you add an event filter to DataBlox, PresentBlox, or other user interface Blox using the addEventFilter() method, you can then write your own class that

implements the corresponding event filter object and specify the actions you want to take before the event is actually processed. The event filter objects are server-side objects, and the methods on the objects are all server-side Java methods. The processing for the events therefore occurs on the server.

To perform post-event processing, you should use the event listeners. For details on event listeners and a comparison of the usage of the two, see Chapter 14, "Event Listener Objects," on page 501.

Scenarios for Using Event Filters

You can use the event filter objects to perform custom application logic based on a user action such as drilling down, expanding or collapsing an outline, and so on. For example, you can catch each time a user drills down on a member and then throw an exception for all users except those named Howard.

The actions you perform during an event can be very simple or very complex. You might need to perform some checks, for example, to see if a user has the authority to perform the action. Or you might want to write a class that sends mail to the finance department each time someone tries to drill down on certain sensitive parts of the database.

An important aspect of the event filters is that they are triggered when an action happens, but *before* the event is actually processed, thus allowing your application to cancel the action before it happens. For example, the `DrillDownEvent` occurs when a user clicks on a member to drill down, but before the drill down is executed on the database and new data is returned to the client.

Using Event Filters and Events

The event filter objects are part of the `com.alphablox.blox.filter` package. You must use the following JSP import statement at the beginning of any JSP file that uses these objects:

```
<%@ page import="com.alphablox.blox.filter.*" %>
```

This package includes interfaces for listeners of the various events. You will need to define a class which implements these interfaces in order to intercept the specific event you want to capture. The name of these interfaces all end with the word `Filter`, such as `BookmarkDeleteFilter`, `DrillDownFilter`, `ExpandFilter`, and `HideOnlyFilter`. These filters have a corresponding method such as `bookmarkDelete()`, `drillDown()`, `expand()`, and `hideOnly()` that you can implement to specify your own actions. All these methods require a corresponding event object as the input to act on. These event object names all end with the word `Event`, such as `BookmarkDeleteEvent`, `DrillDownEvent`, `ExpandEvent`, and `HideOnlyEvent`.

For example, if you want to check if the user performing a drill down operation should be allowed to, you need to:

1. Add a server-side drill down event filter to your DataBlox using the method `addEventFilter(`*YourDrillDownEventFilter*`)`:

```
<blox:present id="myPresent">
   ...
<%
   myPresent.getDataBlox().addEventFilter(new DDFilter() );
%>

</blox:present>
```

In the above example, DDFilter is the name of your drill down event filter object.

2. Have your drill down event filter object implement the DrillDownFilter interface:

```
<%!
public class DDFilter implements DrillDownFilter
{
   //more code here....
}
%>
```

3. Add actions to take when the drillDown method is called. The method takes a DrillDownEvent object as input.

```
<%!
public class DDFilter implements DrillDownFilter
{
   BloxModel model;

// drillDown is the method to implement to capture a drilldown
// events. It takes a DrillDownEvent object as input.
   public void drillDown( DrillDownEvent dde ) throws Exception
    {
      DataBlox blox = dde.getDataBlox();
      StringBuffer msg = new StringBuffer("DRILL DOWN event on " +
blox.getBloxName() + "\n");
      msg.append("With Axis ID: " + dde.getAxisIndex() + ", ");
      msg.append("Nest level: " + dde.getNestLevel() + ", ");
      msg.append("Member index: " + dde.getMemberIndex() + ", and ");
      msg.append("TupleMember: " + dde.getMember().getDisplayName());
      MessageBox msgBox = new MessageBox(msg.toString(), "DrillDown Filter
Message", MessageBox.MESSAGE_OK, null);
      model.getDispatcher().showDialog(msgBox);
    }
}
%>
```

Place add/removeEventFilter Methods Inside Blox Custom Tags

To ensure that a new event is not added each time the page is reloaded, place the code using the addEventFilter() methods inside of the Blox custom tags on your JSP page. For example, the following code creates a Blox and adds a filter that is called whenever a user drills down on a member:

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

<blox:present id="myPresent">
   <blox:data .../>

<%
   myPresent.getDataBlox().addEventFilter(new DDFilter() );
%>

</blox:present>
```

A Complete drillDownEventFilter Example

This complete example shows how to intercept a drill down action and write output using the MessageBox UI model component when the drill down event is triggered.

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ page import="com.alphablox.blox.filter.*,
                 com.alphablox.blox.uimodel.core.MessageBox,
```

```
                com.alphablox.blox.uimodel.BloxModel,
                com.alphablox.blox.DataBlox" %>

<html>
<head>
<blox:header/>
</head>

<body>
<blox:present id="myPresent">
  <blox:data dataSourceName="QCC-Essbase" query="!" />
  <% myPresent.getDataBlox().addEventFilter(new
DDFilter(myPresent.getBloxModel()) ); %>
</blox:present>
</body>
</html>

<%!
  public class DDFilter implements DrillDownFilter
  {
    BloxModel model;
    public DDFilter(BloxModel model) {
      this.model = model;
    }

    // drillDown is the method to implement to capture a drilldown
    // event. It takes a DrillDownEvent object as input.
    public void drillDown( DrillDownEvent dde ) throws Exception
    {
      DataBlox blox = dde.getDataBlox();
      StringBuffer msg = new StringBuffer("DRILL DOWN event on " +
blox.getBloxName() + "\n");
      msg.append("With Axis ID: " + dde.getAxisIndex() + ", ");
      msg.append("Nest level: " + dde.getNestLevel() + ", ");
      msg.append("Member index: " + dde.getMemberIndex() + ", and ");
      msg.append("TupleMember: " + dde.getMember().getDisplayName());
      MessageBox msgBox = new MessageBox(msg.toString(), "DrillDown Filter
Message", MessageBox.MESSAGE_OK, null);
      model.getDispatcher().showDialog(msgBox);
    }
  }
%>
```

By placing the `addEventFilter()` method within the Blox custom tags, it ensures that you will not add multiple filters each time the page is reloaded. In this example, the class created displays a message dialog box containing information about the current drill down action before the drill down event occurs.

You can add as many filters on the same event as you like, and they will be processed in the order in which they are added or until the event is canceled.

This example is available in Blox Sampler, under the Interacting with Data section.

Methods to Implement for Event Filters

To create an event filter, you must write a class that implements one or more event filter methods listed below. The following table lists the events to capture, the method to implement in order to catch that event, and a link to the supporting methods for that filter event.

Event to capture (when a user performs the action)	Interface to implement	Available Event Methods
bookmark:delete	`bookmarkDelete(BookmarkDeleteEvent)` in `BookmarkDeleteFilter`	"BookmarkDeleteEvent Methods" on page 474
bookmark: load	`bookmarkLoad(BookmarkLoadEvent)` in `BookmarkLoadFilter`	"BookmarkLoadEvent Methods" on page 475
bookmark: rename	`bookmarkRename(BookmarkRenameEvent)` in `BookmarkRenameFilter`	"BookmarkRenameEvent Methods" on page 476
bookmark: save	`bookmarkSave(BookmarkSaveEvent)` in `BookmarkSaveFilter`	"BookmarkSaveEvent Methods" on page 477
collapse	`collapse(CollapseEvent)` in `CollapseFilter`	"CollapseEvent Methods" on page 479
drill down/expand all	`drillDown(DrillDownEvent)` in `DrillDownFilter`	"DrillDownEvent Methods" on page 481
drill through	`drillThrough(DrillThroughEvent)` in `DrillThroughFilter`	"DrillThroughEvent Methods" on page 482
drill up	`drillUp(DrillUpEvent)` in `DrillUpFilter`	"DrillUpEvent Methods" on page 484
expand	`expand(ExpandEvent)` in `ExpandFilter`	"ExpandEvent Methods" on page 484
hide only	`hideOnly(HideOnlyEvent)` in `HideOnlyFilter`	"HideOnlyEvent Methods" on page 485
keep only	`keepOnly(KeepOnlyEvent)` in `KeepOnlyFilter`	"KeepOnlyEvent Methods" on page 489
select a member (for example, in Member Filter)	`memberSelect(MemberSelectEvent)` in `MemberSelectFilter`	"MemberSelectEvent Methods" on page 490
pivot	`pivot(PivotEvent)` in `PivotFilter`	"PivotEvent Methods" on page 491
data query	`query(QueryEvent)`	"QueryEvent Methods" on page 494
remove only	`removeOnly(RemoveOnlyEvent)` in `RemoveOnlyEvent`	"RemoveOnlyEvent Methods" on page 496
show all	`showAll(ShowAllEvent)` in `ShowAllFilter`	"ShowAllEvent Methods" on page 497
show only	`showOnly(ShowOnlyEvent)` in `ShowOnlyFilter`	"ShowOnlyEvent Methods" on page 499
swap axis	`swapAxis(SwapAxisEvent)` in `SwapAxisFilter`	"SwapAxisEvent Methods" on page 500

bookmarkDelete(BookmarkDeleteEvent)

To capture when a user performs a bookmark delete action, you must implement a method with the following signature.

Data Sources

All

Syntax

Java Method

```
public void bookmarkDelete(BookmarkDeleteEvent event);
                  throws java.lang.Exception
```

Usage

This method is on the `BookmarkDeleteFilter` interface in the package `com.alphablox.blox.filter`.

See Also

"BookmarkDeleteEvent Methods" on page 474

bookmarkLoad(BookmarkLoadEvent)

To capture when a user performs a bookmark load action, you must implement a method with the following signature.

Data Sources

All

Syntax

Java Method

```
public void bookmarkLoad(BookmarkLoadEvent event);
                  throws java.lang.Exception
```

Usage

This method is on the `BookmarkLoadFilter` interface in the package `com.alphablox.blox.filter`.

See Also

"BookmarkLoadEvent Methods" on page 475

bookmarkRename(BookmarkRenameEvent)

To capture when a user performs a bookmark rename action, you must implement a method with the following signature.

Data Sources

All

Syntax

Java Method

```
public void bookmarkRename(BookmarkRenameEvent event);
                  throws java.lang.Exception
```

Usage

This method is on the `BookmarkRenameFilter` interface in the package `com.alphablox.blox.filter`.

See Also

"BookmarkRenameEvent Methods" on page 476

bookmarkSave(BookmarkSaveEvent)

To capture when a user performs a bookmark save action, you must implement a method with the following signature.

Data Sources

All

Syntax

Java Method

```
public void bookmarkSave(BookmarkSaveEvent event);
                  throws java.lang.Exception
```

Usage
This method is on the `BookmarkSaveFilter` interface in the package `com.alphablox.blox.filter`.

See Also
"BookmarkSaveEvent Methods" on page 477

collapse(CollapseEvent)
To capture when a user performs a collapse action on the data, you must implement a method with the following signature.

Data Sources
Multidimensional

Syntax
Java Method

```
public void collapse(CollapseEvent event)
      throws java.lang.Exception
```

Usage
This method is on the `CollapseFilter` interface in the package `com.alphablox.blox.filter`.

See Also
"CollapseEvent Methods" on page 479

drillDown(DrillDownEvent)
To capture when a user performs a drill down operation on the data, you must implement a method with the following signature.

Data Sources
Multidimensional

Syntax
Java Method

```
public void drillDown(DrillDownEvent event)
      throws java.lang.Exception
```

Usage
This method is on the `DrillDownFilter` interface in the package `com.alphablox.blox.filter`.

See Also
"DrillDownEvent Methods" on page 481

drillThrough(DrillThroughEvent)
To capture when a user performs a drillthrough operation on the data, you must implement a method with the following signature.

Data Sources
IBM DB2 OLAP Server; Hyperion Essbase; Microsoft Analysis Services

Syntax
Java Method

```
public void drillThrough(DrillThroughEvent event)
      throws java.lang.Exception
```

Usage

This method is on the `DrillThroughFilter` interface in the package `com.alphablox.blox.filter`. For IBM DB2 OLAP Server, IBM DB2 OLAP Server Deployment Services, Hyperion Essbase, or Essbase Deployment Services, this is for data sources which have drillthrough reports set up through IBM DB2 OLAP Server Integration Services or Essbase Integration Services.

See Also

"DrillThroughEvent Methods" on page 482

drillUp(DrillUpEvent)

To capture when a user performs a drillup operation on the data, you must implement a method with the following signature.

Data Sources

Multidimensional

Syntax

Java Method

```
public void drillUp(DrillUpEvent event)
      throws java.lang.Exception
```

Usage

This method is on the `DrillUpFilter` interface in the package `com.alphablox.blox.filter`.

See Also

"DrillUpEvent Methods" on page 484

expand(ExpandEvent)

To capture when a user performs an expand operation on the data, you must implement a method with the following signature.

Data Sources

Multidimensional

Syntax

Java Method

```
public void expand(ExpandEvent event)
      throws java.lang.Exception
```

Usage

The expand operation can be performed when the grid in set to display in expand/collapse mode. This is different from an Expand All operation, which is to drill to all descendants. To capture an Expand All operation, see "DrillDownEvent Methods" on page 481.

This method is on the `ExpandFilter` interface in the package `com.alphablox.blox.filter`.

See Also

"ExpandEvent Methods" on page 484

hideOnly(HideOnlyEvent)

To capture when a user performs a Hide Only operation on the data, you must implement a method with the following signature.

Data Sources

Multidimensional

Syntax

Java Method

```
public void hideOnly(HideOnlyEvent event)
      throws java.lang.Exception
```

Usage

This method is on the `HideOnlyFilter` interface in the package `com.alphablox.blox.filter`.

See Also

"HideOnlyEvent Methods" on page 485

keepOnly(KeepOnlyEvent)

To capture when a user performs a Keep Only operation on the data, you must implement a method with the following signature.

Data Sources

Multidimensional

Syntax

Java Method

```
public void keepOnly(KeepOnlyEvent event)
      throws java.lang.Exception
```

Usage

This method is on the `KeepOnlyFilter` interface in the package `com.alphablox.blox.filter`.

See Also

"KeepOnlyEvent Methods" on page 489

memberSelect(MemberSelectEvent)

To capture when a user selects a member (for example, in the Member Filter), you must implement a method with the following signature.

Data Sources

Multidimensional

Syntax

Java Method

```
public void memberSelect(MemberSelectEvent event)
      throws java.lang.Exception
```

Usage

This method is on the `MemberSelectFilter` interface in the package `com.alphablox.blox.filter`.

See Also

"MemberSelectEvent Methods" on page 490

pivot(PivotEvent)

To capture when a user performs a pivot operation on the data, you must implement a method with the following signature.

Data Sources

Multidimensional

Syntax

Java Method

```
public void pivot(PivotEvent event)
      throws java.lang.Exception
```

Usage

This method is on the `PivotFilter` interface in the package `com.alphablox.blox.filter`.

See Also

"PivotEvent Methods" on page 491

query(QueryEvent)

To capture a query operation, you must implement a method with the following signature.

Data Sources

Multidimensional

Syntax

Java Method

```
public void query(QueryEvent event)
      throws java.lang.Exception
```

Usage

This method is on the `QueryFilter` interface in the package `com.alphablox.blox.filter`.

See Also

"QueryEvent Methods" on page 494

removeOnly(RemoveOnlyEvent)

To capture when a user performs a Remove Only operation on the data, you must implement a method with the following signature.

Data Sources

Multidimensional

Syntax

Java Method

```
public void removeOnly(RemoveOnlyEvent event)
      throws java.lang.Exception
```

Usage
This method is on the `RemoveOnlyFilter` interface in the package `com.alphablox.blox.filter`.

See Also
"RemoveOnlyEvent Methods" on page 496

showAll(ShowAllEvent)
To capture when a user performs a Show All operation on the data, you must implement a method with the following signature.

Data Sources
Multidimensional

Syntax
Java Method

```
public void showAll(ShowAllEvent event)
      throws java.lang.Exception
```

Usage
This method is on the `ShowAllFilter` interface in the package `com.alphablox.blox.filter`.

See Also
"ShowAllEvent Methods" on page 497

showOnly(ShowOnlyEvent)
To capture when a user performs a Show Only operation on the data, you must implement a method with the following signature.

Data Sources
Multidimensional

Syntax
Java Method

```
public void showOnly(ShowOnlyEvent event)
      throws java.lang.Exception
```

Usage
This method is on the `ShowOnlyFilter` interface in the package `com.alphablox.blox.filter`.

See Also
"ShowOnlyEvent Methods" on page 499

swapAxis(SwapAxisEvent)
To capture when a user performs a swap axis operation on the data, you must implement a method with the following signature.

Data Sources
Multidimensional

Syntax
Java Method

```
public void swapAxis(SwapAxisEvent event)
      throws java.lang.Exception
```

Usage
This method is on the `SwapAxisFilter` interface in the package `com.alphablox.blox.filter`.

See Also
"SwapAxisEvent Methods" on page 500

BookmarkDeleteEvent Methods

This section lists the Java methods available to the `BookmarkDeleteEvent` interface.

cancelEvent()

Sets the processed flag in the server so that the event is not processed, effectively cancelling the event.

Data Sources
All

Syntax
Java Method

```
void cancelEvent();
```

Usage
This method is on the `FilterEvent` class.

getBlox()

Gets the Blox that generates this event.

Data Sources
All

Syntax
Java Method

```
Blox getBlox();
```

Usage
Returns a Blox object.

getBookmark()

Gets the bookmark involved in this event.

Data Sources
All

Syntax
Java Method

```
public Bookmark getBookmark();
```

Usage
Returns a Bookmark object.

See Also
"Bookmark Object Properties and Associated Methods" on page 149

getSource()

Returns the object which is the source of the event.

Data Sources
All

Syntax
Java Method

```
java.lang.Object getSource();
```

Usage
Overrides the `getSource()` method in `java.util.EventObject`.

isCanceled()

Returns `true` if an event has been canceled, otherwise returns `false`.

Data Sources
All

Syntax
Java Method

```
boolean isCanceled();
```

Usage
This method is on the `FilterEvent` class.

BookmarkLoadEvent Methods

This section lists the Java methods available to the `BookmarkLoadEvent` interface.

cancelEvent()

Sets the processed flag in the server so that the event is not processed, effectively cancelling the event.

Data Sources
All

Syntax
Java Method

```
void cancelEvent();
```

Usage
This method is on the `FilterEvent` class.

getBlox()

The same as "getBlox()" on page 474 in "BookmarkDeleteEvent Methods" on page 474.

getBookmark()

Gets the bookmark involved in this event.

Data Sources
All

Syntax
Java Method

```
public Bookmark getBookmark();
```

Usage
Returns a Bookmark object.

See Also
"Bookmark Object Properties and Associated Methods" on page 149.

getSource()

Returns the object which is the source of the event.

Data Sources
All

Syntax
Java Method

```
java.lang.Object getSource();
```

Usage
Overrides the `getSource()` method in `java.util.EventObject`.

isCanceled()

Returns `true` if an event has been canceled, otherwise returns `false`.

Data Sources
All

Syntax
Java Method

```
boolean isCanceled();
```

Usage
This method is on the `FilterEvent` class.

BookmarkRenameEvent Methods

This section lists the Java methods available to the `BookmarkRenameEvent` interface.

cancelEvent()

Sets the processed flag in the server so that the event is not processed, effectively cancelling the event.

Data Sources
All

Syntax
Java Method

```
void cancelEvent();
```

Usage
This method is on the `FilterEvent` class.

getBlox()
The same as "getBlox()" on page 474 in "BookmarkDeleteEvent Methods" on page 474.

getBookmark()
Gets the bookmark involved in this event.

Data Sources
All

Syntax
Java Method
```
public Bookmark getBookmark();
```

Usage
Returns a Bookmark object.

See Also
"Bookmark Object Properties and Associated Methods" on page 149.

getSource()
Returns the object which is the source of the event.

Data Sources
All

Syntax
Java Method
```
java.lang.Object getSource();
```

Usage
Overrides the `getSource()` method in `java.util.EventObject`.

isCanceled()

Returns `true` if an event has been canceled, otherwise returns `false`.

Data Sources
All

Syntax
Java Method
```
boolean isCanceled();
```

Usage
This method is on the `FilterEvent` class.

BookmarkSaveEvent Methods

This section lists the Java methods available to the `BookmarkSaveEvent` interface.

cancelEvent()

Sets the processed flag in the server so that the event is not processed, effectively cancelling the event.

Data Sources
All

Syntax
Java Method

```
void cancelEvent();
```

Usage
This method is on the `FilterEvent` class.

getBlox()

The same as "getBlox()" on page 474 in "BookmarkDeleteEvent Methods" on page 474.

getBookmark()

Gets the bookmark involved in this event.

Data Sources
All

Syntax
Java Method

```
public Bookmark getBookmark();
```

Usage
Returns a Bookmark object.

See Also
"Bookmark Object Properties and Associated Methods" on page 149.

getSource()

Returns the object which is the source of the event.

Data Sources
All

Syntax
Java Method

```
java.lang.Object getSource();
```

Usage
Overrides the `getSource()` method in `java.util.EventObject`.

isCanceled()

Returns `true` if an event has been canceled, otherwise returns `false`.

Data Sources
All

Syntax
Java Method
```
boolean isCanceled();
```

Usage
This method is on the `FilterEvent` class.

CollapseEvent Methods

This section lists the Java methods available to the `CollapseEvent` interface.

cancelEvent()

Sets the processed flag in the server so that the event is not processed, effectively cancelling the event.

Data Sources
Multidimensional

Syntax
Java Method
```
void cancelEvent();
```

Usage
This method is on the `FilterEvent` class.

getAxisIndex()

Returns the axis index for this operation (for example, 0 for the column axis, 1 for the row axis, 2 for the page axis, etc.).

Data Sources
Multidimensional

Syntax
Java Method
```
int getAxisIndex();
```

Usage
This method is on the `SingleDataFilterEvent` class.

getBlox()

The same as “getBlox()” on page 474 in “BookmarkDeleteEvent Methods” on page 474.

getDataBlox()

Returns the DataBlox that was the source of the event.

Data Sources
Multidimensional

Syntax
Java Method
```
DataBlox getDataBlox();
```

Usage
This method is on the `DataFilterEvent` class.

getMember()

Returns the `TupleMember` object for the event.

Data Sources
Multidimensional

Syntax
Java Method

```
TupleMember getMember()
        throws ServerBloxException
```

Usage
This method is on the `SingleDataFilterEvent` class.

See Also
"getAxis().getTuple().getMember()" on page 413

getMemberIndex()

Returns the zero-based index for the member. The member index is the index of the member selected for this operation in the result set for the chosen dimension.

Data Sources
Multidimensional

Syntax
Java Method

```
int getMemberIndex();
```

Usage
This method is on the `SingleDataFilterEvent` class.

See Also
"getAxis().getTuple().getMember().getIndex()" on page 414

getNestLevel()

Return the nest level for this operation. The nest level is the offset of the dimension in the axis where the first dimension in an axis is 0.

Data Sources
Multidimensional

Syntax
Java Method

```
int getNestLevel();
```

Usage
This method is on the `SingleDataFilterEvent` class.

See Also
"getAxis().getTuple().getMember(). getGenerationLevel()" on page 414

getSource()

Returns the object which is the source of the event.

Data Sources
Multidimensional

Syntax
Java Method

```
java.lang.Object getSource();
```

Usage
Overrides the `getSource()` method in `java.util.EventObject`.

isCanceled()

Returns `true` if an event has been canceled, otherwise returns `false`.

Data Sources
Multidimensional

Syntax
Java Method

```
boolean isCanceled();
```

Usage
This method is on the `FilterEvent` class.

DrillDownEvent Methods

This section lists the Java methods available to the `DrillDownEvent` interface.

cancelEvent()

The same as "cancelEvent()" on page 479 in "CollapseEvent Methods" on page 479.

getAxisIndex()

The same as "getAxisIndex()" on page 479 in "CollapseEvent Methods" on page 479.

getBlox()

The same as "getBlox()" on page 474 in "BookmarkDeleteEvent Methods" on page 474.

getDataBlox()

The same as "getDataBlox()" on page 479 in "CollapseEvent Methods" on page 479.

getDrillDownOption()

Returns the drill down option that would be used for this drill operation.

Data Sources
Multidimensional

Syntax

Java Method

```
int getDrillDownOption();
```

Usage

Returns an integer from 1 to 5 indicating the level to drill down to. Possible values are:

1: Drill down to next generation
2: Drill down to all descendants (the same as an Expand All operation)
3: Drill down to bottom generation
4: Drill to siblings
5: Drill to same generation

The default is 1.

See Also

"drillDownOption" on page 358

getMember()

The same as "getMember()" on page 480 in "CollapseEvent Methods" on page 479.

getMemberIndex()

The same as "getMemberIndex()" on page 480 in "CollapseEvent Methods" on page 479.

getNestLevel()

The same as "getNestLevel()" on page 480 in "CollapseEvent Methods" on page 479.

getSource()

The same as "getSource()" on page 481 in "CollapseEvent Methods" on page 479.

isCanceled()

The same as "isCanceled()" on page 481 in "CollapseEvent Methods" on page 479.

DrillThroughEvent Methods

This section lists the Java methods available to the `DrillThroughEvent` interface.

cancelEvent()

The same as "cancelEvent()" on page 479 in "CollapseEvent Methods" on page 479.

However, this only cancels the data operation, but the pop-up window triggered by the drillthrough event still shows up. To cancel the entire operation, including the pop-ip window, use the client-side event `"ClickEvent" on page 73`.

getBlox()

The same as "getBlox()" on page 474 in "BookmarkDeleteEvent Methods" on page 474.

getColumnIndex()

Returns the column coordinate of the selected cell in which to perform the drillthrough at.

Data Sources

Relational

Syntax

Java Method

```
int getColumnIndex();
```

Usage

Returns the column coordinate of the selected cell in which to perform the drill through at.

getDataBlox()

The same as “getDataBlox()” on page 479 in “CollapseEvent Methods” on page 479.

getRowIndex()

Returns the row coordinate of the selected cell in which to perform the drill through at.

Data Sources

Relational

Syntax

Java Method

```
int getRowIndex();
```

Usage

Returns the row coordinate of the selected cell in which to perform the drill through at.

getSource()

The same as “getSource()” on page 481 in “CollapseEvent Methods” on page 479.

getTuples()

Returns a tuple array corresponding to the selected cell in which to perform the drillthrough at.

Data Sources

Relational

Syntax

Java Method

```
Tuple[] getTuples();     // throws ServerBloxException
```

Usage

Returns a tuple array corresponding to the selected cell in which to perform the drillthrough at. The first tuple in the array corresponds to the column tuple of the selected cell. The second tuple in the array corresponds to the row tuple of the selected cell.

isCanceled()

The same as “isCanceled()” on page 481 in “CollapseEvent Methods” on page 479.

DrillUpEvent Methods

This section lists the Java methods available to the `DrillUpEvent` interface.

cancelEvent()

The same as “cancelEvent()” on page 479 in “CollapseEvent Methods” on page 479.

getAxisIndex()

The same as “getAxisIndex()” on page 479 in “CollapseEvent Methods” on page 479.

getBlox()

The same as “getBlox()” on page 474 in “BookmarkDeleteEvent Methods” on page 474.

getDataBlox()

The same as “getDataBlox()” on page 479 in “CollapseEvent Methods” on page 479.

getMember()

The same as “getMember()” on page 480 in “CollapseEvent Methods” on page 479.

getMemberIndex()

The same as “getMemberIndex()” on page 480 in “CollapseEvent Methods” on page 479.

getNestLevel()

The same as “getNestLevel()” on page 480 in “CollapseEvent Methods” on page 479.

getSource()

The same as “getSource()” on page 481 in “CollapseEvent Methods” on page 479.

isCanceled()

The same as “isCanceled()” on page 481 in “CollapseEvent Methods” on page 479.

ExpandEvent Methods

This section lists the Java methods available to the `ExpandEvent` interface.

cancelEvent()

The same as “cancelEvent()” on page 479 in “CollapseEvent Methods” on page 479.

getAxisIndex()

The same as “getAxisIndex()” on page 479 in “CollapseEvent Methods” on page 479.

getBlox()

The same as “getBlox()” on page 474 in “BookmarkDeleteEvent Methods” on page 474.

getDataBlox()

The same as “getDataBlox()” on page 479 in “CollapseEvent Methods” on page 479.

getMember()

The same as “getMember()” on page 480 in “CollapseEvent Methods” on page 479.

getMemberIndex()

The same as “getMemberIndex()” on page 480 in “CollapseEvent Methods” on page 479.

getNestLevel()

The same as “getNestLevel()” on page 480 in “CollapseEvent Methods” on page 479.

getSource()

The same as “getSource()” on page 481 in “CollapseEvent Methods” on page 479.

isCanceled()

The same as “isCanceled()” on page 481 in “CollapseEvent Methods” on page 479.

HideOnlyEvent Methods

This section lists the Java methods available to the `HideOnlyEvent` interface.

cancelEvent()

The same as “cancelEvent()” on page 479 in “CollapseEvent Methods” on page 479.

getAxisIndex()

Returns an array of integers defining all axis indexes for this operation (for example, 0 for the column axis, 1 for the row axis, 2 for the page axis, etc.).

Data Sources

Multidimensional

Syntax

Java Method

```
int[] getAxisIndex();
```

Usage

This method is on the `MultipleDataFilterEvent` class.

getAxisIndex(coordset)

Returns an integer defining the axis index for this operation.

Data Sources

Multidimensional

Syntax

Java Method

```
int getAxisIndex(int coordset);
```

where:

Argument	Default	Description
coordset	none	An integer representing the specified coordinate set (0-based). Valid values are from 0 to *event*.getSize()-1.

Usage

A coordinate consists of the axis index, the nesting level, and the member index on the same level. For axis index, 0 is for the column axis, 1 for the row axis, and 2 for the page axis. In the following example, West is on the column axis (0), nested under 2004 and Sales (nesting level = 2), and the third member on the level (2). The coordinate for West is [0, 2, 2].

	2004		
	Sales		
Products	East	Central	West
Truffles	[data]	[data]	[data]
Specialties	[data]	[data]	[data]

getBlox()

The same as “getBlox()” on page 474 in “BookmarkDeleteEvent Methods” on page 474.

getDataBlox()

The same as “getDataBlox()” on page 479 in “CollapseEvent Methods” on page 479.

getMember()

Returns all of the TupleMember objects for the event.

Data Sources

Multidimensional

Syntax

Java Method

```
TupleMember[] getMember()
        throws ServerBloxException
```

Usage

This method is on the MultipleDataFilterEvent class.

See Also
"getAxis().getTuple().getMember()" on page 413

getMember(coordset)

Returns the `TupleMember` object for the event.

Data Sources
Multidimensional

Syntax
Java Method

```
TupleMember getMember(int coordset)
        throws ServerBloxException
```

where:

Argument	Default	Description
`coordset`	none	An integer representing the specified coordinate set (0-based). Valid values are from 0 to *event*`.getSize()-1`.

Usage
This method is on the `MultipleDataFilterEvent` class. See "getAxisIndex(coordset)" on page 486 for more information on coordinate sets.

See Also
"getAxis().getTuple().getMember()" on page 413

getMemberIndex()

Returns all of the zero-based indexes for the member. The member index is the index of the member selected for this operation in the result set for the chosen dimension.

Data Sources
Multidimensional

Syntax
Java Method

```
int[] getMemberIndex();
```

Usage
This method is on the `MultipleDataFilterEvent` class.

See Also
"getAxis().getTuple().getMember().getIndex()" on page 414

getMemberIndex(coordset)

Returns all of the zero-based indexes for the member. The member index is the index of the member selected for this operation in the result set for the chosen dimension.

Data Sources
Multidimensional

Syntax
Java Method

```
int getMemberIndex(coordset);
```

where:

Argument	Default	Description
coordset	none	An integer representing the specified coordinate set (0-based). Valid values are from 0 to *event*.getSize()-1.

Usage
This method is on the MultipleDataFilterEvent class.

See Also
"getAxis().getTuple().getMember().getIndex()" on page 414. See "getAxisIndex(coordset)" on page 486 for more information on coordinate sets

getNestLevel()
Return all of the nest level for this operation. The nest level is the offset of the dimension in the axis where the first dimension in an axis is 0.

Data Sources
Multidimensional

Syntax
Java Method

```
int[] getNestLevel();
```

Usage
This method is on the MultipleDataFilterEvent class.

See Also
"getAxis().getTuple().getMember(). getGenerationLevel()" on page 414

getNestLevel(coordset)
Return the nest level for this operation. The nest level is the offset of the dimension in the axis where the first dimension in an axis is 0.

Data Sources
Multidimensional

Syntax
Java Method

```
int getNestLevel(int coordset);
```

where:

Argument	Default	Description
coordset	none	An integer representing the specified coordinate set (0-based). Valid values are from 0 to *event*.getSize()-1.

Usage
This method is on the `MultipleDataFilterEvent` class.

See Also
"getAxis().getTuple().getMember(). getGenerationLevel()" on page 414. See "getAxisIndex(coordset)" on page 486 for more information on coordinate sets.

getSize()
Return the count of the number of available result set coordinate sets.

Data Sources
Multidimensional

Syntax
Java Method

```
int getSize();
```

Usage
This method is on the `MultipleDataFilterEvent` class.

getSource()
The same as "getSource()" on page 481 in "CollapseEvent Methods" on page 479.

isCanceled()
The same as "isCanceled()" on page 481 in "CollapseEvent Methods" on page 479.

KeepOnlyEvent Methods
This section lists the Java methods available to the `KeepOnlyEvent` interface.

cancelEvent()
The same as "cancelEvent()" on page 479 in "CollapseEvent Methods" on page 479.

getAxisIndex()
The same as "getAxisIndex()" on page 485 in "HideOnlyEvent Methods" on page 485.

getAxisIndex(coordset)
The same as "getAxisIndex(coordset)" on page 486 in "HideOnlyEvent Methods" on page 485.

getBlox()
The same as "getBlox()" on page 474 in "BookmarkDeleteEvent Methods" on page 474.

getDataBlox()
The same as "getDataBlox()" on page 479 in "CollapseEvent Methods" on page 479.

getMember()

The same as “getMember()” on page 486 in “HideOnlyEvent Methods” on page 485.

getMember(coordset)

The same as “getMember(coordset)” on page 487 in “HideOnlyEvent Methods” on page 485.

getMemberIndex()

The same as “getMemberIndex()” on page 487 in “HideOnlyEvent Methods” on page 485.

getMemberIndex(coordset)

The same as “getMemberIndex(coordset)” on page 487 in “HideOnlyEvent Methods” on page 485.

getNestLevel()

The same as “getNestLevel()” on page 488 in “HideOnlyEvent Methods” on page 485.

getNestLevel(coordset)

The same as “getNestLevel(coordset)” on page 488 in “HideOnlyEvent Methods” on page 485.

getSize()

The same as “getSize()” on page 489 in “HideOnlyEvent Methods” on page 485.

getSource()

The same as “getSource()” on page 481 in “CollapseEvent Methods” on page 479.

isCanceled()

The same as “isCanceled()” on page 481 in “CollapseEvent Methods” on page 479.

MemberSelectEvent Methods

This section lists the Java methods available to the `MemberSelectEvent` interface.

cancelEvent()

The same as “cancelEvent()” on page 479 in “CollapseEvent Methods” on page 479.

getBlox()

The same as “getBlox()” on page 474 in “BookmarkDeleteEvent Methods” on page 474.

getDimension()

Return an interface to the metadata for the dimension associated with this member selection event.

Data Sources
Multidimensional

Syntax
Java Method

```
Dimension getDimension();
```

See Also
"getAxis().getDimension()" on page 409

getNewMemberSelections()
Returns an array of members which comprises the new list of selected members for the dimension.

Data Sources
Multidimensional

Syntax
Java Method

```
Member[] getNewMemberSelections();
```

getOldMemberSelections()
Returns an array of members which comprises the current list of selected members for the dimension.

Data Sources
Multidimensional

Syntax
Java Method

```
Member[] getOldMemberSelections();
```

getSource()
The same as "getSource()" on page 481 in "CollapseEvent Methods" on page 479.

isCanceled()
The same as "isCanceled()" on page 481 in "CollapseEvent Methods" on page 479.

PivotEvent Methods
This section lists the Java methods available to the `PivotEvent` interface.

cancelEvent()
The same as "cancelEvent()" on page 479 in "CollapseEvent Methods" on page 479.

getBlox()
The same as "getBlox()" on page 474 in "BookmarkDeleteEvent Methods" on page 474.

getNewAxis()

Returns the new axis index (in the MDBResultSet) which the dimension is being pivoted to.

Data Sources
Multidimensional

Syntax
Java Method

```
int getNewAxis();
```

Usage
Returns the new axis index from the MDBResultSet. Use this method instead of `getNewDisplayAxis()` to get the index to use with server-side objects.

getNewDisplayAxis()

Returns the new axis index (in the displayed result set) which the dimension is being pivoted to.

Data Sources
Multidimensional

Syntax
Java Method

```
int getNewDisplayAxis();
```

Usage
Returns the new axis index (0=column, 1=row, 2=page, 3=other) in the displayed result set. Use this method instead of `getNewAxis()` to get the index to use with the `DataBlox.pivot()` method.

getNewDisplayNestLevel()

Returns the new nesting level (in the displayed result set) which the dimension is being pivoted to.

Data Sources
Multidimensional

Syntax
Java Method

```
int getNewDisplayNestLevel();
```

Usage
Returns the new nesting level in the displayed result set. Use this method instead of `getNewNestLevel()` to get the index to use with the `DataBlox.pivot()` method.

getNewNestLevel()

Returns the new nesting level (in the MDBResultSet) which the dimension is being pivoted to.

Data Sources
Multidimensional

Syntax
Java Method
```
int getNewNestLevel();
```

Usage
Returns the new nesting level in the MDBResultSet. Use this method instead of `getNewDisplayNestAxis()` to get the index to use with server-side objects.

getOldAxis()
Returns the old axis index (in the MDBResultSet) which the dimension was pivoted from.

Data Sources
Multidimensional

Syntax
Java Method
```
int getOldAxis();
```

Usage
Returns the old axis index from the MDBResultSet. Use this method instead of `getOldDisplayAxis()` to get the index to use with server-side objects.

getOldDisplayAxis()
Returns the old axis index (in the displayed result set) which the dimension was pivoted from.

Data Sources
Multidimensional

Syntax
Java Method
```
int getOldDisplayAxis();
```

Usage
Returns the old axis index (0=column, 1=row, 2=page, 3=other) in the displayed result set. Use this method instead of `getOldAxis()` to get the index to use with the `DataBlox.pivot()` method.

getOldDisplayNestLevel()
Returns the old nesting level (in the displayed result set) which the dimension was pivoted from.

Data Sources
Multidimensional

Syntax
Java Method
```
int getOldDisplayNestLevel();
```

Usage
Returns the old nesting level in the displayed result set. Use this method instead of `getOldNestLevel()` to get the index to use with the `DataBlox.pivot()` method.

getOldNestLevel()

Returns the old nesting level (in the MDBResult) which the dimension was pivoted from.

Data Sources

Multidimensional

Syntax

Java Method

```
int getOldNestLevel();
```

Usage

Returns the old nesting level in the MDBResultSet. Use this method instead of `getOldDisplayNestAxis()` to get the index to use with server-side objects.

getSource()

The same as "getSource()" on page 481 in "CollapseEvent Methods" on page 479.

isCanceled()

The same as "isCanceled()" on page 481 in "CollapseEvent Methods" on page 479.

QueryEvent Methods

This section lists the Java methods available to the `QueryEvent` interface.

cancelEvent()

The same as "cancelEvent()" on page 479 in "CollapseEvent Methods" on page 479.

getAxes()

Returns an array containing all the axis within this result set.

Data Sources

Multidimensional

Syntax

Java Method

```
Axis[] getAxes();
```

Usage

Returns an array containing all the axes with this result set or null if there are no axes or if this is a text-based query.

getAxisCount()

Returns the number of axes in the cube excluding the slicer axis.

Data Sources

Multidimensional

Syntax

Java Method

```
int getAxisCount();
```

Usage
Returns the number of axes in the cube, excluding the slicer axis or -1 if this is a text-based query.

getBlox()
The same as "getBlox()" on page 474 in "BookmarkDeleteEvent Methods" on page 474.

getDimensionsOnPageAxis()
Return a list of dimensions that will be placed on the page axis.

Data Sources
Multidimensional

Syntax
Java Method
```
String getDimensionsOnPageAxis();
```

getQuery()
Return the query that is about to be executed. This only returns a value for text-based queries. Internal queries return null.

Data Sources
Multidimensional

Syntax
Java Method
```
String getQuery();
```

getSlicerAxisIndex()
Returns the index of the slicer axis for internal queries.

Data Sources
Multidimensional

Syntax
Java Method
```
int getSlicerAxisIndex();
```

Usage
Returns the index of the slicer axis in this result set or -1 if this is a text-based query. Use `getAxis(int` *index*`)` with the returned integer from this method to return the slicer axis.

getSource()
The same as "getSource()" on page 481 in "CollapseEvent Methods" on page 479.

isCanceled()
The same as "isCanceled()" on page 481 in "CollapseEvent Methods" on page 479.

isInternalQuery()

Returns true if the current query is an internal query. Internal queries are generated when bookmarks are restored.

Data Sources

Multidimensional

Syntax

Java Method

```
boolean isInternalQuery();
```

Usage

boolean `true` for internal queries; `false` if this is a text-based query.

RemoveOnlyEvent Methods

This section lists the Java methods available to the `RemoveOnlyEvent` interface.

cancelEvent()

The same as “cancelEvent()” on page 479 in “CollapseEvent Methods” on page 479.

getAxisIndex()

The same as “getAxisIndex()” on page 485 in “HideOnlyEvent Methods” on page 485.

getAxisIndex(coordset)

The same as “getAxisIndex(coordset)” on page 486 in “HideOnlyEvent Methods” on page 485.

getBlox()

The same as “getBlox()” on page 474 in “BookmarkDeleteEvent Methods” on page 474.

getDataBlox()

The same as “getDataBlox()” on page 479 in “CollapseEvent Methods” on page 479.

getMember()

The same as “getMember()” on page 486 in “HideOnlyEvent Methods” on page 485.

getMember(coordset)

The same as “getMember(coordset)” on page 487 in “HideOnlyEvent Methods” on page 485.

getMemberIndex()

The same as “getMemberIndex()” on page 487 in “HideOnlyEvent Methods” on page 485.

getMemberIndex(coordset)

The same as “getMemberIndex(coordset)” on page 487 in “HideOnlyEvent Methods” on page 485.

getNestLevel()

The same as “getNestLevel()” on page 488 in “HideOnlyEvent Methods” on page 485.

getNestLevel(coordset)

The same as “getNestLevel(coordset)” on page 488 in “HideOnlyEvent Methods” on page 485.

getSize()

The same as “getSize()” on page 489 in “HideOnlyEvent Methods” on page 485.

getSource()

The same as “getSource()” on page 481 in “CollapseEvent Methods” on page 479.

isCanceled()

The same as “isCanceled()” on page 481 in “CollapseEvent Methods” on page 479.

ShowAllEvent Methods

This section lists the Java methods available to the `ShowAllEvent` interface.

cancelEvent()

The same as “cancelEvent()” on page 479 in “CollapseEvent Methods” on page 479.

getAxisIndex()

The same as “getAxisIndex()” on page 485 in “HideOnlyEvent Methods” on page 485.

getAxisIndex(coordset)

The same as “getAxisIndex(coordset)” on page 486 in “HideOnlyEvent Methods” on page 485.

getBlox()

The same as “getBlox()” on page 474 in “BookmarkDeleteEvent Methods” on page 474.

getDataBlox()

The same as “getDataBlox()” on page 479 in “CollapseEvent Methods” on page 479.

getDimension()

Returns an array of `AxisDimension` objects for this operation.

Data Sources
Multidimensional

Syntax
Java Method

```
AxisDimension[] getDimension();
```

See Also
"getAxis().getDimension()" on page 409

getDimension(coordset)

Return the `AxisDimension` object for this event.

Data Sources
Multidimensional

Syntax
Java Method

```
AxisDimension getDimension(int coordset);
```

where:

Argument	Default	Description
`coordset`	none	An integer representing the specified coordinate set.

See Also
"getAxis().getDimension()" on page 409. See "getAxisIndex(coordset)" on page 486 for more information on coordinate sets.

getNestLevel()

Returns all of the nesting level from which the user performed a Show All operation on a dimension for this event.

Data Sources
Multidimensional

Syntax
Java Method

```
int[] getNestLevel();
```

getNestLevel(coordset)

Returns the nesting level from which the user performed a Show All operation on a dimension for this event.

Data Sources
Multidimensional

Syntax
Java Method

```
int getNestLevel(int coordset);
```

where:

Argument	Default	Description
coordset	none	An integer representing the specified coordinate set.

See Also

See “getAxisIndex(coordset)” on page 486 for more information on coordinate sets.

getSize()

Return the count of the number of available result set coordinate sets.

Data Sources

Multidimensional

Syntax

Java Method

```
int getSize();
```

isCanceled()

The same as “isCanceled()” on page 481 in “CollapseEvent Methods” on page 479.

ShowOnlyEvent Methods

This section lists the Java methods available to the ShowOnlyEvent interface.

cancelEvent()

The same as “cancelEvent()” on page 479 in “CollapseEvent Methods” on page 479.

getAxisIndex()

The same as “getAxisIndex()” on page 485 in “HideOnlyEvent Methods” on page 485.

getAxisIndex(coordset)

The same as “getAxisIndex(coordset)” on page 486 in “HideOnlyEvent Methods” on page 485.

getBlox()

The same as “getBlox()” on page 474 in “BookmarkDeleteEvent Methods” on page 474.

getDataBlox()

The same as “getDataBlox()” on page 479 in “CollapseEvent Methods” on page 479.

getMember()

The same as “getMember()” on page 486 in “HideOnlyEvent Methods” on page 485.

getMember(coordset)

The same as “getMember(coordset)” on page 487 in “HideOnlyEvent Methods” on page 485.

getMemberIndex()

The same as “getMemberIndex()” on page 487 in “HideOnlyEvent Methods” on page 485.

getMemberIndex(coordset)

The same as “getMemberIndex(coordset)” on page 487 in “HideOnlyEvent Methods” on page 485.

getNestLevel()

The same as “getNestLevel()” on page 488 in “HideOnlyEvent Methods” on page 485.

getNestLevel(coordset)

The same as “getNestLevel(coordset)” on page 488 in “HideOnlyEvent Methods” on page 485.

getSize()

The same as “getSize()” on page 489 in “HideOnlyEvent Methods” on page 485.

getSource()

The same as “getSource()” on page 481 in “CollapseEvent Methods” on page 479.

isCanceled()

The same as “isCanceled()” on page 481 in “CollapseEvent Methods” on page 479.

SwapAxisEvent Methods

This section lists the Java methods available to the `SwapAxisEvent` interface.

cancelEvent()

The same as “cancelEvent()” on page 479 in “CollapseEvent Methods” on page 479.

getBlox()

The same as “getBlox()” on page 474 in “BookmarkDeleteEvent Methods” on page 474.

getSource()

The same as “getSource()” on page 481 in “CollapseEvent Methods” on page 479.

isCanceled()

The same as “isCanceled()” on page 481 in “CollapseEvent Methods” on page 479.

Chapter 14. Event Listener Objects

This chapter describes the event listener objects and the methods used with them. The common Blox methods addEventListener() and removeEventListener() take the event listener objects as arguments and let you perform custom actions *after* the event has been processed on the server. For capturing events before they are actually processed, see Chapter 13, "Event Filter Objects," on page 463.

Event Listener Objects Overview

The event listener objects are server-side objects that allow you to be notified when some user event such as drilling down or pivoting and perform some actions after the event has been processed. To use the event listener, you need to first add the specific event listener object using the addEventListener() method. There are three types of event listener objects.

- DataBlox related. You can capture the completion of the following data analysis operations: collapse, drill down, drill through, drill up, expand, hide only, keep only, member select, pivot, remove only, show all, show only, swap axis, and data query.
- Bookmark related. You can capture the completion of the following bookmark related events: delete bookmark, load bookmark, rename bookmark, and save bookmark.
- ChartBlox related. You can capture the event when users change the page filter.

When a user-triggered event, such as swapping axis from the Blox user interface, is completed, the corresponding event listener will be notified.

Once you add an event listener to DataBlox, PresentBlox, or other user interface Blox using the addEventListener() method, you can then write your own class that implements the corresponding event listener object and specify the actions you want to take when the event is completed. The event listener objects are server-side objects, and the methods on the objects are all server-side Java methods. The processing for the events therefore occurs on the server.

To perform pre-event processing, you should use the event filters. For a comparison of the usage of the two, see "Event Listeners VS. Event Filters" on page 503.

Scenarios for Using Event Listeners

You can use the event listener objects to perform custom application logic based on a user action such as drilling down, expanding or collapsing an outline, and so on. For example, after a hide-only event is completed, you may want to update another Blox, handle an exception that is a side-effect of the event, or send messages back to the client based on the results of the event. Event listeners will only be triggered when the event is completed with no errors.

Using Event Listeners

The event listener objects are part of the `com.alphablox.blox.event` package. You must use the following JSP import statement at the beginning of any JSP file that uses these objects:

```
<%@ page import="com.alphablox.blox.event.*" %>
```

This package includes interfaces for listeners of the various events. The way to use event listeners is very similar to that for event filters. You will need to define a class which implements these interfaces in order to intercept the specific event whose completion you want to be notified of. The name of these interfaces all end with the word `Listener`, such as `BookmarkDeleteListener`, `DrillDownListener`, `ExpandListener`, and `HideOnlyListener`. These listeners have a corresponding method such as `bookmarkDelete()`, `drillDown()`, `expand()`, and `hideOnly()` that you can implement to specify your own actions. All these methods require a corresponding event object as the input to act on. These event object names all end with the word `Event`, such as `BookmarkDeleteEvent`, `DrillDownEvent`, `ExpandEvent`, and `HideOnlyEvent`.

For example, if you want to perform custom actions after the user performs a drilldown operation:

1. Add a server-side drill down event listener to the DataBlox:

```
<blox:present id="myPresentBlox">
   <blox:data bloxRef="myData"/>
   ...

<%
  myPresentBlox.getDataBlox().addEventListener( new DDHandler());
%>
...
</blox:present>
```

   In the above example, `DDHandler` is the name of your event listener object. Notice that the listener is added inside the Blox tag to ensure that a new event listener is not added each time the page is loaded.

2. Have your event listener object implement the appropriate event listener interface:

```
<%!
  public static class DDHandler implements DrillDownListener
  {
    ...
  }
%>
```

3. Add actions to take after the `drillDown` method is called. The `drillDown` method must be implemented, and the method takes a `DrillDownEvent` object as input:

```
<%!
public class DDHandler implements DrillDownListener
{
// drillDown is the method to implement to capture a drilldown
// events. It takes a DrillDownEvent object as input.
   public void drillDown( DrillDownEvent dde ) throws Exception
   {
      DataBlox blox = dde.getDataBlox();
      // do something here
   }
}
%>
```

Event Listeners VS. Event Filters

Event listeners are used to be notified of a successful completion of an event while event filters are used to intercept an event on the server as the server receives it, yet before the event is processed. Implementation of an event listener and that of an event filter are very similar. The following table provides a summary of the similarity and differences of the two.

	Event Listeners	**Event Filters**
When notification is received	After event is processed	Before event is processed
Package	`com.alphablox.blox.event`	`com.alphablox.blox.filter`
Interfaces in the package	All interfaces end with the word `Listener`, such as `DrillDownListener` and `RemoveOnlyListener`	All interfaces end with the word `Filter`, such as `DrillDownFilter`, and `RemoveOnlyFilter`
Methods to implement	These listeners have a corresponding method, such as `drillDown()`, and `removeOnly()`, that takes the corresponding event object as argument: `drillDown(DrillDownEvent)` `removeOnly(RemoveOnlyEvent)`	These filters have a corresponding method such as `bookmarkLoad()`, `drillDown()`, and `removeOnly()`, that takes the corresponding event object as argument: `drillDown(DrillDownEvent)` `removeOnly(RemoveOnlyEvent)`
Events	Same event object names as those in event filters.	Same event object names events as those in event listeners. However, these events have a `cancelEvent()` and a `isCanceled()` method that those in event listeners don't.

You can create an event handler that handles both pre- and post-event processing for a specified event. For example:

```
<%!
  public class DDHandler implements DrillDownFilter, DrillDownListener
  {
    public void drillDown(DrillDownEvent event) throws Exception {
      // actions to take before the event is processed
    }
```

```
    public void drillDown(com.alphablox.blox.event.DrillDownEvent event) {
      // actions to take after the event has been processed
    }
  }
%>
```

However, since the event objects have the same name in both the event filters and the event listeners packages, if you want to use the same class to handle both pre- and post-event processing, you should specify the complete class names that include the package information.

Place addEventListener Method Inside Blox Custom Tags

To ensure that a new event is not added each time the page is reloaded, place the code using the `addEventListener()` methods inside of the Blox custom tags on your JSP page. For example, the following code creates a Blox and adds a listener that is called whenever a user hides (only) a member:

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

<blox:present id="myPresent">
   <blox:data .../>
...
<%
   myPresent.getDataBlox().addEventListener(new HideOnlyHandler() );
%>
</blox:present>

<%!
  public class HideOnlyHandler implements HideOnlyListener
  {
    public void hideOnly( HideOnlyEvent hoe)
    {
    ...// custom actions here
    }
  }
%>
```

A Complete drillDownEventListener Example

This complete example shows how to be notified when a drill down action has occurred and write the output using the MessageBox UI model component:

```
<%@ page import="com.alphablox.blox.event.*,
                 com.alphablox.blox.uimodel.core.MessageBox,
                 com.alphablox.blox.uimodel.BloxModel" %>
<%@ page import="com.alphablox.blox.DataBlox" %>
<%@ taglib uri="bloxtld" prefix="blox" %>

<html>
<head>
  <blox:header/>
</head>

<body>
<blox:present id="myPresent2">
  <blox:data
    dataSourceName="QCC-Essbase"
    query="!"  />
  <% myPresent2.getDataBlox().addEventListener( new
SimpleListener(myPresent2.getBloxModel()) );    %>
</blox:present>

</body>
```

```
</html>

<%!
  public class SimpleListener implements DrillDownListener
  {
    BloxModel model;
    public SimpleListener(BloxModel model) {
      this.model = model;
    }

    public void drillDown( DrillDownEvent event ) throws Exception
    {
      DataBlox blox = event.getDataBlox();
      StringBuffer msg = new StringBuffer("DRILL DOWN event on " +
blox.getBloxName() + "\n");
      msg.append("With Axis ID: " + event.getAxisIndex() + ", ");
      msg.append("Nest level: " + event.getNestLevel() + ", ");
      msg.append("Member index: " + event.getMemberIndex() );

      MessageBox msgBox = new MessageBox(msg.toString(), "DrillDown
Listener Message", MessageBox.MESSAGE_OK, null);
      model.getDispatcher().showDialog(msgBox);
    }
  }
%>
```

You can add as many listeners on the same event as you like, and they will be processed in the order in which they are added.

Methods to Implement for Event Listener Objects

To create an event listener, you must write a class that implements one or more event listener methods listed below. The following table lists the events to capture, the method to implement in order to catch that event, and a link to the supporting methods for that filter event.

Event to capture (when a user performs the action)	Interface to implement	Available Event Methods
bookmark:delete	`bookmarkDelete(BookmarkDeleteEvent)`in `BoomarkDeleteListener`	"BookmarkDeleteEvent Methods" on page 513
bookmark: load	`bookmarkLoad(BookmarkLoadEvent)`in `BoomarkLoadListener`	"BookmarkLoadEvent Methods" on page 514
bookmark: rename	`bookmarkRename(BookmarkRenameEvent)`in `BoomarkRenameListener`	"BookmarkRenameEvent Methods" on page 514
bookmark: save	`bookmarkSave(BookmarkSaveEvent)`in `BoomarkSaveListener`	"BookmarkSaveEvent Methods" on page 514
filter change in ChartBlox	`changePage(ChartPageEvent)`in `ChartPageListener`	"ChartPageEvent Methods" on page 515
collapse	`collapse(CollapseEvent)`in `CollapseListener`	"CollapseEvent Methods" on page 516
drill down/expand all	`drillDown(DrillDownEvent)`in `DrillDownListener`	"DrillDownEvent Methods" on page 517
drill through	`drillThrough(DrillThroughEvent)`in `DrillThroughEvent`	"DrillThroughEvent Methods" on page 518
drill up	`drillUp(DrillUpEvent)`in `DrillUpListener`	"DrillUpEvent Methods" on page 519
expand	`expand(ExpandEvent)`in `ExpandListener`	"ExpandEvent Methods" on page 520
hide only	`hideOnly(HideOnlyEvent)`in `HideOnlyListener`	"HideOnlyEvent Methods" on page 521

Event to capture (when a user performs the action)	Interface to implement	Available Event Methods
keep only	`keepOnly(KeepOnlyEvent)` in `KeepOnlyListener`	"KeepOnlyEvent Methods" on page 524
select a member (for example, in Member Filter)	`memberSelect(MemberSelectEvent)` in `MemberSelectListener`	"MemberSelectEvent Methods" on page 525
export data to PDF	`pdf(PdfEvent)` in `PdfListener`	"PivotEvent Methods" on page 527
pivot	`pivot(PivotEvent)` in `PivotListener`	"PivotEvent Methods" on page 527
data query	`query(QueryEvent)` in `QueryListener`	"QueryEvent Methods" on page 529
remove only	`removeOnly(RemoveOnlyEvent)` in `RemoveOnlyListener`	"RemoveOnlyEvent Methods" on page 531
show all	`showAll(ShowAllEvent)` in `ShowAllListener`	"ShowAllEvent Methods" on page 532
show only	`showOnly(ShowOnlyEvent)` in `ShowOnlyListener`	"ShowOnlyEvent Methods" on page 534
swap axis	`swapAxis(SwapAxisEvent)` in `SwapAxisListener`	"SwapAxisEvent Methods" on page 535

bookmarkDelete(BookmarkDeleteEvent)

To be notified after a user performs a bookmark delete action, you must implement a method with the following signature.

Data Sources

All

Syntax

Java Method

```
public void bookmarkDelete(BookmarkDeleteEvent event)
```

Usage

This method is on the `BookmarkDeleteListener` interface in the package `com.alphablox.blox.event`.

See Also

"BookmarkDeleteEvent Methods" on page 513

bookmarkLoad(BookmarkLoadEvent)

To be notified after a user performs a bookmark load action, you must implement a method with the following signature.

Data Sources

All

Syntax

Java Method

```
public void bookmarkLoad(BookmarkLoadEvent event)
```

Usage

This method is on the `BookmarkLoadListener` interface in the package `com.alphablox.blox.event`.

See Also

"BookmarkLoadEvent Methods" on page 514

bookmarkRename(BookmarkRenameEvent)

To be notified after a user performs a bookmark rename action, you must implement a method with the following signature.

Data Sources
All

Syntax
Java Method

```
public void bookmarkRename(BookmarkRenameEvent event)
```

Usage
This method is on the `BookmarkRenameListener` interface in the package `com.alphablox.blox.event`.

See Also
"BookmarkRenameEvent Methods" on page 514

bookmarkSave(BookmarkSaveEvent)

To be notified after a user performs a bookmark save action, you must implement a method with the following signature.

Data Sources
All

Syntax
Java Method

```
public void bookmarkSave(BookmarkSaveEvent event)
```

Usage
This method is on the `BookmarkSaveListener` interface in the package `com.alphablox.blox.event`.

See Also
"BookmarkSaveEvent Methods" on page 514

changePage(ChartPageEvent)

To be notified after a user changes the page filter in a ChartBlox, you must implement a method with the following signature.

Data Sources
Multidimensional

Syntax
Java Method

```
public void changePage(ChartPageEvent event)
      throws java.lang.Exception
```

Usage
This method is on the `ChartPageListener` interface in the package `com.alphablox.blox.event`.

See Also
"ChartPageEvent Methods" on page 515

collapse(CollapseEvent)

To be notified after a user performs a collapse action on the data, you must implement a method with the following signature.

Data Sources

Multidimensional

Syntax

Java Method

```
public void collapse(CollapseEvent event)
      throws java.lang.Exception
```

Usage

This method is on the `CollapseListener` interface in the package `com.alphablox.blox.event`.

See Also

"CollapseEvent Methods" on page 516

drillDown(DrillDownEvent)

To be notified after a user performs a "drill down" operation on the data, you must implement a method with the following signature.

Data Sources

Multidimensional

Syntax

Java Method

```
public void drillDown(DrillDownEvent event)
      throws java.lang.Exception
```

Usage

This method is on the `DrillDownListener` interface in the package `com.alphablox.blox.event`.

See Also

"DrillDownEvent Methods" on page 517

drillThrough(DrillThroughEvent)

To be notified after a user performs a drillthrough operation on the data, you must implement a method with the following signature.

Data Sources

IBM DB2 OLAP Server, Hyperion Essbase, Microsoft Analysis Services

Syntax

Java Method

```
public void drillThrough(DrillThroughEvent event)
      throws java.lang.Exception
```

Usage

This method is on the `DrillThroughListener` interface in the package `com.alphablox.blox.event`. For IBM DB2 OLAP Server, IBM DB2 OLAP Server Deployment Services, Hyperion Essbase, or Essbase Deployment Services, this is

for data sources which have drillthrough reports set up through IBM DB2 OLAP Server Integration Services or Essbase Integration Services.

See Also
"DrillThroughEvent Methods" on page 518

drillUp(DrillUpEvent)

To be notified after a user performs a drillup operation on the data, you must implement a method with the following signature.

Data Sources
Multidimensional

Syntax
Java Method

```
public void drillUp(DrillUpEvent event)
      throws java.lang.Exception
```

Usage
This method is on the `DrillUpListener` interface in the package `com.alphablox.blox.event`.

See Also
"DrillUpEvent Methods" on page 519

expand(ExpandEvent)

To be notified after a user performs an expand operation on the data, you must implement a method with the following signature.

Data Sources
Multidimensional

Syntax
Java Method

```
public void expand(ExpandEvent event)
      throws java.lang.Exception
```

Usage
The expand operation can be performed when the grid in set to display in expand/collapse mode. This is different from an Expand All operation, which is to drill to all descendants. To capture an Expand All operation, see "DrillDownEvent Methods" on page 517.

This method is on the `ExpandListener` interface in the package `com.alphablox.blox.event`.

See Also
"ExpandEvent Methods" on page 520

hideOnly(HideOnlyEvent)

To be notified after a user performs a Hide Only operation on the data, you must implement a method with the following signature.

Data Sources
Multidimensional

Syntax
Java Method

```
public void hideOnly(HideOnlyEvent event)
      throws java.lang.Exception
```

Usage
This method is on the `HideOnlyListener` interface in the package `com.alphablox.blox.event`.

See Also
"HideOnlyEvent Methods" on page 521

keepOnly(KeepOnlyEvent)
To be notified after a user performs a Keep Only operation on the data, you must implement a method with the following signature.

Data Sources
Multidimensional

Syntax
Java Method

```
public void keepOnly(KeepOnlyEvent event)
      throws java.lang.Exception
```

Usage
This method is on the `KeepOnlyListener` interface in the package `com.alphablox.blox.event`.

See Also
"KeepOnlyEvent Methods" on page 524

memberSelect(MemberSelectEvent)
To be notified after a user selects a member (for example, in the Member Filter), you must implement a method with the following signature.

Data Sources
Multidimensional

Syntax
Java Method

```
public void memberSelect(MemberSelectEvent event)
      throws java.lang.Exception
```

Usage
This method is on the `MemberSelectListener` interface in the package `com.alphablox.blox.event`.

See Also
"MemberSelectEvent Methods" on page 525

pdf(PdfEvent)
To be notified after a user exports the data to PDF, you must implement a method with the following signature.

Data Sources
All

Syntax
Java Method

```
public void pdf(PdfEvent event)
      throws java.lang.Exception
```

Usage
This method is on the `PdfListener` interface in the package `com.alphablox.blox.event`.

See Also
"PivotEvent Methods" on page 527

pivot(PivotEvent)
To be notified after a user performs a pivot operation on the data, you must implement a method with the following signature.

Data Sources
Multidimensional

Syntax
Java Method

```
public void pivot(PivotEvent event)
      throws java.lang.Exception
```

Usage
This method is on the `PivotListener` interface in the package `com.alphablox.blox.evnet`.

See Also
"PivotEvent Methods" on page 527

query(QueryEvent)
To be notified after a query operation, you must implement a method with the following signature.

Data Sources
Multidimensional

Syntax
Java Method

```
public void query(QueryEvent event)
      throws java.lang.Exception
```

Usage
This method is on the `QueryListener` interface in the package `com.alphablox.blox.event`.

See Also
"QueryEvent Methods" on page 529

removeOnly(RemoveOnlyEvent)

To be notified after a user performs a Remove Only operation on the data, you must implement a method with the following signature.

Data Sources

Multidimensional

Syntax

Java Method

```
public void removeOnly(RemoveOnlyEvent event)
      throws java.lang.Exception
```

Usage

This method is on the `RemoveOnlyListener` interface in the package `com.alphablox.blox.event`.

See Also

"RemoveOnlyEvent Methods" on page 531

showAll(ShowAllEvent)

To be notified after a user performs a Show All operation on the data, you must implement a method with the following signature.

Data Sources

Multidimensional

Syntax

Java Method

```
public void showAll(ShowAllEvent event)
      throws java.lang.Exception
```

Usage

This method is on the `ShowAllListener` interface in the package `com.alphablox.blox.event`.

See Also

"ShowAllEvent Methods" on page 532

showOnly(ShowOnlyEvent)

To be notified after a user performs a Show Only operation on the data, you must implement a method with the following signature.

Data Sources

Multidimensional

Syntax

Java Method

```
public void showOnly(ShowOnlyEvent event)
      throws java.lang.Exception
```

Usage

This method is on the `ShowOnlyListener` interface in the package `com.alphablox.blox.event`.

See Also
"ShowOnlyEvent Methods" on page 534

swapAxis(SwapAxisEvent)

To be notified after a user performs a swap axis operation on the data, you must implement a method with the following signature.

Data Sources
Multidimensional

Syntax
Java Method

```
public void swapAxis(SwapAxisEvent event)
      throws java.lang.Exception
```

Usage
This method is on the `SwapAxisListener` interface in the package `com.alphablox.blox.event`.

See Also
"SwapAxisEvent Methods" on page 535

BookmarkDeleteEvent Methods

This section lists the Java methods available to the `BookmarkDeleteEvent` object.

getBlox()

Gets the Blox that generates this event.

Data Sources
All

Syntax
Java Method

```
Blox getBlox();
```

Usage
Returns a Blox object.

getBookmark()

Gets the bookmark involved in this event.

Data Sources
All

Syntax
Java Method

```
Bookmark getBookmark();
```

Usage
Returns a Bookmark object.

See Also
"Bookmark Object Properties and Associated Methods" on page 149

getSource()

Returns the object which is the source of the event.

Data Sources

All

Syntax

Java Method

```
java.lang.Object getSource();
```

Usage

Overrides the `getSource()` method in `java.util.EventObject`.

BookmarkLoadEvent Methods

This section lists the Java methods available to the `BookmarkLoadEvent` object.

getBlox()

The same as “getBlox()” on page 513 in “BookmarkDeleteEvent Methods” on page 513.

getBookmark()

The same as “getBookmark()” on page 513 in “BookmarkDeleteEvent Methods” on page 513.

getSource()

The same as “getSource()” on page 514 in “BookmarkDeleteEvent Methods” on page 513.

BookmarkRenameEvent Methods

This section lists the Java methods available to the `BookmarkRenameEvent` object.

getBlox()

The same as “getBlox()” on page 513 in “BookmarkDeleteEvent Methods” on page 513.

getBookmark()

The same as “getBookmark()” on page 513 in “BookmarkDeleteEvent Methods” on page 513.

getSource()

The same as “getSource()” on page 514 in “BookmarkDeleteEvent Methods” on page 513.

BookmarkSaveEvent Methods

This section lists the Java methods available to the `BookmarkSaveEvent` object.

getBlox()

The same as “getBlox()” on page 513 in “BookmarkDeleteEvent Methods” on page 513.

getBookmark()

The same as “getBookmark()” on page 513 in “BookmarkDeleteEvent Methods” on page 513.

getSource()

The same as “getSource()” on page 514 in “BookmarkDeleteEvent Methods” on page 513.

ChartPageEvent Methods

This section lists the Java methods available to the `ChartPageEvent` object. For a complete example, see “Example 3: Use the server-side ChartPageListener to set the desired data format on the chart when the chart filter is changed” on page 912.

getBlox()

The same as “getBlox()” on page 513 in “BookmarkDeleteEvent Methods” on page 513.

getChartBlox()

Gets the ChartBlox which fired this event.

Data Sources

Multidimensional

Syntax

Java Method

```
ChartBlox getChartBlox();
```

getDimension()

Gets the dimension that the chart filter is set to as a String.

Data Sources

Multidimensional

Syntax

Java Method

```
String getDimension();
```

getSelection()

Gets the member selected to be on the chart filter as a String.

Data Sources

Multidimensional

Syntax

Java Method

```
String getSelection();
```

getSource()

The same as “getSource()” on page 514 in “BookmarkDeleteEvent Methods” on page 513.

CollapseEvent Methods

This section lists the Java methods available to the `CollapseEvent` object.

getAxisIndex()

Returns the axis index for this operation (0 for the column axis, 1 for the row axis, and 2 for the page axis).

Data Sources
Multidimensional

Syntax
Java Method

```
int getAxisIndex();
```

Usage
This method is on the `SingleDataListenerEvent` class.

getBlox()

The same as "getBlox()" on page 513 in "BookmarkDeleteEvent Methods" on page 513.

getDataBlox()

Returns the DataBlox that was the source of the event.

Data Sources
Multidimensional

Syntax
Java Method

```
DataBlox getDataBlox();
```

Usage
This method is on the `DataEvent` class.

getMemberIndex()

Returns the zero-based index for the member. The member index is the index of the member selected for this operation in the result set for the chosen dimension.

Data Sources
Multidimensional

Syntax
Java Method

```
int getMemberIndex();
```

Usage
This method is on the `SingleDataListenerEvent` class.

See Also
"getAxis().getTuple().getMember().getIndex()" on page 414

getMemberName()

Returns the unique name of all the members for this operation.

Data Sources
Multidimensional

Syntax
Java Method

```
String[] getMemberName();
        //throws ServerBloxException
```

Usage
This method is on the `MultipleDataEvent` class.

getNestLevel()

Return the nest level for this operation. The nest level is the offset of the dimension in the axis where the first dimension in an axis is 0.

Data Sources
Multidimensional

Syntax
Java Method

```
int getNestLevel();
```

Usage
This method is on the `SingleDataFilterEvent` class.

See Also
"getAxis().getTuple().getMember(). getGenerationLevel()" on page 414

getSource()

The same as "getSource()" on page 514 in "BookmarkDeleteEvent Methods" on page 513.

DrillDownEvent Methods

This section lists the Java methods available to the `DrillDownEvent` object.

getAxisIndex()

The same as "getAxisIndex()" on page 516 in "CollapseEvent Methods" on page 516.

getBlox()

The same as "getBlox()" on page 513 in "BookmarkDeleteEvent Methods" on page 513.

getDataBlox()

The same as "getDataBlox()" on page 516 in "CollapseEvent Methods" on page 516.

getDrillDownOption()

Returns the drill down option that would be used for this drill operation.

Data Sources
Multidimensional

Syntax

Java Method

```
int getDrillDownOption();
```

Usage

Returns an integer from 1 to 5 indicating the level to drill down to. Possible values are:

1: Drill down to next generation
2: Drill down to all descendants (the same as an Expand All operation)
3: Drill down to bottom generation
4: Drill to siblings
5: Drill to same generation

The default is 1.

See Also

"drillDownOption" on page 358

getMemberIndex()

The same as "getMemberIndex()" on page 516 in "CollapseEvent Methods" on page 516.

getNestLevel()

The same as "getNestLevel()" on page 517 in "CollapseEvent Methods" on page 516.

getSource()

The same as "getSource()" on page 514 in "BookmarkDeleteEvent Methods" on page 513.

DrillThroughEvent Methods

This section lists the Java methods available to the `DrillThroughEvent` object.

getBlox()

The same as "getBlox()" on page 513 in "BookmarkDeleteEvent Methods" on page 513.

getColumnIndex()

Returns the column coordinate of the selected cell in which to perform the drillthrough at.

Data Sources

Relational

Syntax

Java Method

```
int getColumnIndex();
```

Usage

Returns the column coordinate of the selected cell in which to perform the drill through at.

getDataBlox()

The same as “getDataBlox()” on page 516 in “CollapseEvent Methods” on page 516.

getRowIndex()

Returns the row coordinate of the selected cell in which to perform the drill through at.

Data Sources

Relational

Syntax

Java Method

```
int getRowIndex();
```

Usage

Returns the row coordinate of the selected cell in which to perform the drill through at.

getSource()

The same as “getSource()” on page 514 in “BookmarkDeleteEvent Methods” on page 513.

getTuples()

Returns a tuple array corresponding to the selected cell in which to perform the drillthrough at.

Data Sources

Relational

Syntax

Java Method

```
Tuple[] getTuples();     // throws ServerBloxException
```

Usage

Returns a tuple array corresponding to the selected cell in which to perform the drillthrough at. The first tuple in the array corresponds to the column tuple of the selected cell. The second tuple in the array corresponds to the row tuple of the selected cell.

DrillUpEvent Methods

This section lists the Java methods available to the `DrillUpEvent` object.

getAxisIndex()

The same as “getAxisIndex()” on page 516 in “CollapseEvent Methods” on page 516.

getBlox()

The same as “getBlox()” on page 513 in “BookmarkDeleteEvent Methods” on page 513.

getDataBlox()

The same as “getDataBlox()” on page 516 in “CollapseEvent Methods” on page 516.

getMemberName()

The same as “getMemberName()” on page 516 in “CollapseEvent Methods” on page 516.

getMemberIndex()

The same as “getMemberIndex()” on page 516 in “CollapseEvent Methods” on page 516.

getNestLevel()

The same as “getNestLevel()” on page 517 in “CollapseEvent Methods” on page 516.

getSource()

The same as “getSource()” on page 514 in “BookmarkDeleteEvent Methods” on page 513.

ExpandEvent Methods

This section lists the Java methods available to the `ExpandEvent` object.

getAxisIndex()

The same as “getAxisIndex()” on page 516 in “CollapseEvent Methods” on page 516.

getBlox()

The same as “getBlox()” on page 513 in “BookmarkDeleteEvent Methods” on page 513.

getDataBlox()

The same as “getDataBlox()” on page 516 in “CollapseEvent Methods” on page 516.

getMemberName()

The same as “getMemberName()” on page 516 in “CollapseEvent Methods” on page 516.

getMemberIndex()

The same as “getMemberIndex()” on page 516 in “CollapseEvent Methods” on page 516.

getNestLevel()

The same as “getNestLevel()” on page 517 in “CollapseEvent Methods” on page 516.

getSource()

The same as “getSource()” on page 514 in “BookmarkDeleteEvent Methods” on page 513.

HideOnlyEvent Methods

This section lists the Java methods available to the `HideOnlyEvent` object.

getAxisIndex()

Returns an array of integers defining all axis indexes for this operation (for example, 0 for the column axis, 1 for the row axis, 2 for the page axis, etc.).

Data Sources

Multidimensional

Syntax

Java Method

```
int[] getAxisIndex();
```

Usage

This method is on the `MultipleDataEvent` class.

getAxisIndex(coordset)

Returns an integer defining the axis index for this operation

Data Sources

Multidimensional

Syntax

Java Method

```
int getAxisIndex(int coordset);
```

where:

Argument	Default	Description
`coordset`	none	An integer representing the specified coordinate set (0-based). Valid values are from 0 to `event.getSize()-1`.

Usage

A coordinate consists of the axis index, the nesting level, and the member index on the same level. For axis index, 0 is for the column axis, 1 for the row axis, and 2 for the page axis. In the following example, West is on the column axis (0), nested under 2004 and Sales (nesting level = 2), and the third member on the level (2). The coordinate for West is [0, 2, 2].

	2004		
	Sales		
Products	East	Central	West
Truffles	[data]	[data]	[data]
Specialties	[data]	[data]	[data]

getBlox()

The same as “getBlox()” on page 513 in “BookmarkDeleteEvent Methods” on page 513.

getDataBlox()

The same as “getDataBlox()” on page 516 in “CollapseEvent Methods” on page 516.

getMemberIndex()

Returns the all of the zero-based indexes for the member. The member index is the index of the member selected for this operation in the result set for the chosen dimension.

Data Sources

Multidimensional

Syntax

Java Method

```
int[] getMemberIndex();
```

Usage

This method is on the `MultipleDataFilterEvent` class.

See Also

“getAxis().getTuple().getMember().getIndex()” on page 414

getMemberIndex(coordset)

Returns the all of the zero-based indexes for the member. The member index is the index of the member selected for this operation in the result set for the chosen dimension.

Data Sources

Multidimensional

Syntax

Java Method

```
int getMemberIndex(int coordset);
```

where:

Argument	Default	Description
`coordset`	none	An integer representing the specified coordinate set (0-based). Valid values are from 0 to *event*`.getSize()-1`.

Usage

This method is on the `MultipleDataEvent` class.

See Also

“getAxis().getTuple().getMember().getIndex()” on page 414. See “getAxisIndex(coordset)” on page 521 for more information on coordinate sets.

getMemberName()

Returns the unique name of all the members for this operation.

Data Sources

Multidimensional

Syntax

Java Method

```
String[] getMemberName();
        //throws ServerBloxException
```

Usage

This method is on the `MultipleDataEvent` class.

getMemberName(coordset)

Returns the unique name for this operation.

Data Sources

Multidimensional

Syntax

Java Method

```
String getMemberName(int coordset);
        // throws ServerBloxException
```

where:

Argument	Default	Description
`coordset`	none	An integer representing the specified coordinate set (0-based). Valid values are from 0 to *event*`.getSize()-1`.

See Also

"getAxis().getTuple().getMember()" on page 413. See "getAxisIndex(coordset)" on page 521 for more information on coordinate sets.

getNestLevel()

Returns all of the nest level for this operation. The nest level is the offset of the dimension in the axis where the first dimension in an axis is 0.

Data Sources

Multidimensional

Syntax

Java Method

```
int[] getNestLevel();
```

Usage

This method is on the `MultipleDataEvent` class.

See Also

"getAxis().getTuple().getMember(). getGenerationLevel()" on page 414

getNestLevel(coordset)

Returns the nest level for this operation. The nest level is the offset of the dimension in the axis where the first dimension in an axis is 0.

Data Sources

Multidimensional

Syntax

Java Method

```
int getNestLevel(int coordset);
```

where:

Argument	Default	Description
coordset	none	An integer representing the specified coordinate set (0-based). Valid values are from 0 to *event*.getSize()-1.

See Also

"getAxis().getTuple().getMember(). getGenerationLevel()" on page 414. See "getAxisIndex(coordset)" on page 521 for more information on coordinate sets.

getSize()

Return the count of the number of available result set coordinate sets.

Data Sources

Multidimensional

Syntax

Java Method

```
int getSize();
```

Usage

This method is on the MultipleDataEvent class.

getSource()

The same as "getSource()" on page 514 in "BookmarkDeleteEvent Methods" on page 513.

KeepOnlyEvent Methods

This section lists the Java methods available to the KeepOnlyEvent object.

getAxisIndex()

The same as "getAxisIndex()" on page 521 in "HideOnlyEvent Methods" on page 521.

getAxisIndex(coordset)

The same as "getAxisIndex(coordset)" on page 521 in "HideOnlyEvent Methods" on page 521.

getBlox()

The same as “getBlox()” on page 513 in “BookmarkDeleteEvent Methods” on page 513.

getDataBlox()

The same as “getDataBlox()” on page 516 in “CollapseEvent Methods” on page 516.

getMemberName()

The same as “getMemberName()” on page 523 in “HideOnlyEvent Methods” on page 521.

getMemberName(coordset)

The same as “getMemberName(coordset)” on page 523 in “HideOnlyEvent Methods” on page 521.

getMemberIndex()

The same as “getMemberIndex()” on page 522 in “HideOnlyEvent Methods” on page 521.

getMemberIndex(coordset)

The same as “getMemberIndex(coordset)” on page 522 in “HideOnlyEvent Methods” on page 521.

getNestLevel()

The same as “getNestLevel()” on page 523 in “HideOnlyEvent Methods” on page 521.

getNestLevel(coordset)

The same as “getNestLevel(coordset)” on page 524 in “HideOnlyEvent Methods” on page 521.

getSize()

The same as “getSize()” on page 524 in “HideOnlyEvent Methods” on page 521.

getSource()

The same as “getSource()” on page 514 in “BookmarkDeleteEvent Methods” on page 513.

MemberSelectEvent Methods

This section lists the Java methods available to the `MemberSelectEvent` object.

getBlox()

The same as “getBlox()” on page 513 in “BookmarkDeleteEvent Methods” on page 513.

getDimension()

Return an interface to the metadata for the dimension associated with this member selection event.

Data Sources
Multidimensional

Syntax
Java Method

```
Dimension getDimension();
```

See Also
"getAxis().getDimension()" on page 409

getNewMembers()

Returns the array of new members as a String array.

Data Sources
Multidimensional

Syntax
Java Method

```
String[] getNewMembers();
```

getNewMemberSelections()

Returns an array of members which comprises the new list of selected members for the dimension.

Data Sources
Multidimensional

Syntax
Java Method

```
Member[] getNewMemberSelections();
```

getOldMembers()

Returns the array of old members as a String array.

Data Sources
Multidimensional

Syntax
Java Method

```
String[] getOldMembers();
```

getOldMemberSelections()

Returns an array of members which comprises the current list of selected members for the dimension.

Data Sources
Multidimensional

Syntax
Java Method

```
Member[] getOldMemberSelections();
```

getSource()

The same as “getSource()” on page 514 in “BookmarkDeleteEvent Methods” on page 513.

PdfEvent Methods

This section lists the Java methods available to the `PdfEvent` object.

getBlox()

The same as “getBlox()” on page 513 in “BookmarkDeleteEvent Methods” on page 513.

getSource()

The same as “getSource()” on page 514 in “BookmarkDeleteEvent Methods” on page 513.

PivotEvent Methods

This section lists the Java methods available to the `PivotEvent` object.

getBlox()

The same as “getBlox()” on page 513 in “BookmarkDeleteEvent Methods” on page 513.

getNewAxis()

Returns the new axis index (in the MDBResultSet) which the dimension is being pivoted to.

Data Sources

Multidimensional

Syntax

Java Method

```
int getNewAxis();
```

Usage

Returns the new axis index from the MDBResultSet. Use this method instead of `getNewDisplayAxis()` to get the index to use with server-side objects.

getNewDisplayAxis()

Returns the new axis index (in the displayed result set) which the dimension is being pivoted to.

Data Sources

Multidimensional

Syntax

Java Method

```
int getNewDisplayAxis();
```

Usage
Returns the new axis index (0=column, 1=row, 2=page, 3=other) in the displayed result set. Use this method instead of `getNewAxis()` to get the index to use with the `DataBlox.pivot()` method.

getNewDisplayNestLevel()
Returns the new nesting level (in the displayed result set) which the dimension is being pivoted to.

Data Sources
Multidimensional

Syntax
Java Method

```
int getNewDisplayNestLevel();
```

Usage
Returns the new nesting level in the displayed result set. Use this method instead of `getNewNestLevel()` to get the index to use with the `DataBlox.pivot()` method.

getNewNestLevel()
Returns the new nesting level (in the MDBResultSet) which the dimension is being pivoted to.

Data Sources
Multidimensional

Syntax
Java Method

```
int getNewNestLevel();
```

Usage
Returns the new nesting level in the MDBResultSet. Use this method instead of `getNewDisplayNestAxis()` to get the index to use with server-side objects.

getOldAxis()
Returns the old axis index (in the MDBResultSet) which the dimension was pivoted from.

Data Sources
Multidimensional

Syntax
Java Method

```
int getOldAxis();
```

Usage
Returns the old axis index from the MDBResultSet. Use this method instead of `getOldDisplayAxis()` to get the index to use with server-side objects.

getOldDisplayAxis()
Returns the old axis index (in the displayed result set) which the dimension was pivoted from.

Data Sources
Multidimensional

Syntax
Java Method

```
int getOldDisplayAxis();
```

Usage
Returns the old axis index (0=column, 1=row, 2=page, 3=other) in the displayed result set. Use this method instead of `getOldAxis()` to get the index to use with the `DataBlox.pivot()` method.

getOldDisplayNestLevel()

Returns the old nesting level (in the displayed result set) which the dimension was pivoted from.

Data Sources
Multidimensional

Syntax
Java Method

```
int getOldDisplayNestLevel();
```

Usage
Returns the old nesting level in the displayed result set. Use this method instead of `getOldNestLevel()` to get the index to use with the `DataBlox.pivot()` method.

getOldNestLevel()

Returns the old nesting level (in the MDBResult) which the dimension was pivoted from.

Data Sources
Multidimensional

Syntax
Java Method

```
int getOldNestLevel();
```

Usage
Returns the old nesting level in the MDBResultSet. Use this method instead of `getOldDisplayNestAxis()` to get the index to use with server-side objects.

getSource()

The same as “getSource()” on page 514 in “BookmarkDeleteEvent Methods” on page 513.

QueryEvent Methods

This section lists the Java methods available to the `QueryEvent` object.

getAxes()

Returns an array containing all the axis within this result set.

Data Sources
Multidimensional

Syntax
Java Method

```
Axis[] getAxes();
```

Usage
Returns an array containing all the axes with this result set or null if there are no axes or if this is a text-based query.

getAxisCount()
Returns the number of axes in the cube excluding the slicer axis.

Data Sources
Multidimensional

Syntax
Java Method

```
int getAxisCount();
```

Usage
Returns the number of axes in the cube, excluding the slicer axis or -1 if this is a text-based query.

getBlox()
The same as “getBlox()” on page 513 in “BookmarkDeleteEvent Methods” on page 513.

getDimensionsOnPageAxis()
Return a list of dimensions that will be placed on the page axis.

Data Sources
Multidimensional

Syntax
Java Method

```
String getDimensionsOnPageAxis();
```

getQuery()
Return the query that is about to be executed. This only returns a value for text-based queries. Internal queries return null.

Data Sources
Multidimensional

Syntax
Java Method

```
String getQuery();
```

getSlicerAxisIndex()
Returns the index of the slicer axis for internal queries.

Data Sources
Multidimensional

Syntax
Java Method

```
int getSlicerAxisIndex();
```

Usage
Returns the index of the slicer axis in this result set or -1 if this is a text-based query. Use `getAxis(int` *index*`)` with the returned integer from this method to return the slicer axis.

getSource()

The same as “getSource()” on page 514 in “BookmarkDeleteEvent Methods” on page 513.

isInternalQuery()

Returns true if the current query is an internal query. Internal queries are generated when bookmarks are restored.

Data Sources
Multidimensional

Syntax
Java Method

```
boolean isInternalQuery();
```

Usage
boolean `true` for internal queries; `false` if this is a text-based query.

RemoveOnlyEvent Methods

This section lists the Java methods available to the `RemoveOnlyEvent` object.

getAxisIndex()

The same as “getAxisIndex()” on page 521 in “HideOnlyEvent Methods” on page 521.

getAxisIndex(coordset)

The same as “getAxisIndex(coordset)” on page 521 in “HideOnlyEvent Methods” on page 521.

getBlox()

The same as “getBlox()” on page 513 in “BookmarkDeleteEvent Methods” on page 513.

getDataBlox()

The same as “getDataBlox()” on page 516 in “CollapseEvent Methods” on page 516.

getMemberName()

The same as “getMemberName()” on page 523 in “HideOnlyEvent Methods” on page 521.

getMemberName(coordset)

The same as “getMemberName(coordset)” on page 523 in “HideOnlyEvent Methods” on page 521.

getMemberIndex()

The same as “getMemberIndex()” on page 522 in “HideOnlyEvent Methods” on page 521.

getMemberIndex(coordset)

The same as “getMemberIndex(coordset)” on page 522 in “HideOnlyEvent Methods” on page 521.

getNestLevel()

The same as “getNestLevel()” on page 523 in “HideOnlyEvent Methods” on page 521.

getNestLevel(coordset)

The same as “getNestLevel(coordset)” on page 524 in “HideOnlyEvent Methods” on page 521.

getSize()

The same as “getSize()” on page 524 in “HideOnlyEvent Methods” on page 521.

getSource()

The same as “getSource()” on page 514 in “BookmarkDeleteEvent Methods” on page 513.

ShowAllEvent Methods

This section lists the Java methods available to the `ShowAllEvent` object.

getAxisIndex()

The same as “getAxisIndex()” on page 521 in “HideOnlyEvent Methods” on page 521

getAxisIndex(coordset)

The same as “getAxisIndex(coordset)” on page 521 in “HideOnlyEvent Methods” on page 521.

getBlox()

The same as “getBlox()” on page 513 in “BookmarkDeleteEvent Methods” on page 513.

getDataBlox()

The same as “getDataBlox()” on page 516 in “CollapseEvent Methods” on page 516.

getDimension()

Returns all AxisDimensions object involved in this operation.

Data Sources
Multidimensional

Syntax
Java Method

```
AxisDimension[] getDimension();
```

See Also
"getAxis().getDimension()" on page 409

getDimension(coordset)

Returns the AxisDimension object for this event.

Data Sources
Multidimensional

Syntax
Java Method

```
AxisDimension getDimension(int coordset);
```

where:

Argument	Default	Description
coordset	none	An integer representing the specified coordinate set (0-based). Valid values are from 0 to *event*.getSize()-1.

See Also
See "getAxisIndex(coordset)" on page 521 for more information on coordinate sets.

getNestLevel()

Returns all nesting levels from which the user performed a Show All operation on a dimension for this event.

Data Sources
Multidimensional

Syntax
Java Method

```
int[] getNestLevel();
```

Usage
The nesting level is the offset of the dimension in the axis, where the first dimension in an axis is 0.

getNestLevel(coordset)

Returns all of the nesting level from which the user performed a Show All operation on a dimension.

Data Sources
Multidimensional

Syntax
Java Method

```
int getNestLevel(int coordset);
```

where:

Argument	Description
`coordset`	An integer representing the specified coordinate set (0-based). Valid values are from 0 to *event*`.getSize()-1`.

Usage
The nesting level is the offset of the dimension in the axis, where the first dimension in an axis is 0.

See Also
See "getAxisIndex(coordset)" on page 521 for more information on coordinate sets.

getSize()
Return the count of the number of available result set coordinate sets.

Data Sources
Multidimensional

Syntax
Java Method

```
int getSize();
```

getSource()
The same as "getSource()" on page 514 in "BookmarkDeleteEvent Methods" on page 513.

ShowOnlyEvent Methods
This section lists the Java methods available to the `ShowOnlyEvent` object.

getAxisIndex()
The same as "getAxisIndex()" on page 521 in "HideOnlyEvent Methods" on page 521.

getAxisIndex(coordset)
The same as "getAxisIndex(coordset)" on page 521 in "HideOnlyEvent Methods" on page 521

getBlox()
The same as "getBlox()" on page 513 in "BookmarkDeleteEvent Methods" on page 513.

getDataBlox()

The same as “getDataBlox()” on page 516 in “CollapseEvent Methods” on page 516.

getMemberName()

The same as “getMemberName()” on page 523 in “HideOnlyEvent Methods” on page 521.

getMemberName(coordset)

The same as “getMemberName(coordset)” on page 523 in “HideOnlyEvent Methods” on page 521.

getMemberIndex()

The same as “getMemberIndex()” on page 522 in “HideOnlyEvent Methods” on page 521.

getMemberIndex(coordset)

The same as “getMemberIndex(coordset)” on page 522 in “HideOnlyEvent Methods” on page 521.

getNestLevel()

The same as “getNestLevel()” on page 523 in “HideOnlyEvent Methods” on page 521.

getNestLevel(coordset)

The same as “getNestLevel(coordset)” on page 524 in “HideOnlyEvent Methods” on page 521.

getSize()

The same as “getSize()” on page 524 in “HideOnlyEvent Methods” on page 521.

getSource()

The same as “getSource()” on page 514 in “BookmarkDeleteEvent Methods” on page 513.

SwapAxisEvent Methods

This section lists the Java methods available to the `SwapAxisEvent` object.

getBlox()

The same as “getBlox()” on page 513 in “BookmarkDeleteEvent Methods” on page 513.

getSource()

The same as “getSource()” on page 514 in “BookmarkDeleteEvent Methods” on page 513.

Chapter 15. GridBlox Reference

This chapter contains reference material for GridBlox properties, methods and objects. For general reference information about Blox, see Chapter 3, "General Blox Reference Information," on page 17. For information on how to use this reference, see Chapter 1, "Using This Reference," on page 1.

GridBlox Overview

The user interface for GridBlox consists of a tabular data display area consisting of rows and columns of data cells and optional grid controls. Users can view multidimensional data on row, column, and page axes. Users can manipulate the data presentation by drilling up and down through data hierarchies, move data dimensions to different axes, include or omit data dimensions, and so forth.

GridBlox displays data from both relational and multidimensional data sources. It displays relational data in a two-dimensional row-and-column format. It displays multidimensional data in an interactive, multidimensional grid format, enabling users to perform multidimensional analysis. DB2 Alphablox includes a cube server that transforms relational data into multidimensional cubes, enabling GridBlox to display the data in multidimensional format.

Note: See "OLAP Terms and Concepts" in the *Administrator's Guide* for more information on multidimensional analysis.

The DHTML client displays a GridBlox as an HTML element with the `id` specified in the `<blox:grid>` tag. Each grid in the HTML element is a `DIV` element, and has all the attributes, methods and events typically associated with a `DIV`. In addition, a `Selection` object is available that represents the cells that are currently selected. For details on the DHTML Client API, see the *Developer's Guide*.

GridBlox JSP Custom Tag Syntax

The Alphablox Tag Libraries provides custom tags to use in a JSP page for creating each Blox. This section describes how to create the custom tag to create a GridBlox. For a copy and paste version of the tag with all the attributes, see "GridBlox JSP Custom Tag" on page 886.

Syntax

```
<blox:grid
   [attribute="value"] >
      [<blox:cellAlert
           [attribute="value"] />]
      [<blox:cellEditor
           [attribute="value"] />]
      [<blox:cellFormat
```

```
          [attribute="value"] />]
      [<blox:cellLink
          [attribute="value"] />]
      [<blox:drillThroughWindow
          [attribute="value"] />]
      [<blox:editableCellStyle
          [attribute="value"] />]
      [<blox:editedCellStyle
          [attribute="value"] />]
      [<blox:formatMask
          [attribute="value"] />]
      [<blox:formatName
          [attribute="value"] />]
</blox:grid>
```

where:

attribute is one of the attributes listed in the attribute table.

value is a valid value for the attribute.

and where the attributes are one of the following:

<blox:grid> tag
Attribute
id
autosizeEnabled
applyPropertiesAfterBookmark
bandingEnabled
bloxEnabled
bloxName
bookmarkFilter
columnHeadersWrapped
columnWidths
commentsEnabled
defaultCellFormat
drillThroughEnabled
drillThroughWindow
editableCellStyle
editedCellStyle
enablePoppedOut
expandCollapseMode
gridLinesVisible
headingsEnabled
height
helpTargetFrame
informationWindowName
localeCode
maximumUndoSteps
menubarVisible

<blox:grid> tag
Attribute
missingValueString
noAccessValueString
noDataMessage
poppedOut
poppedOutHeight
poppedOutTitle
poppedOutWidth
relationalRowNumbersOn
removeAction
render
rightClickMenuEnabled
rowHeadersWrapped
rowHeadingsVisible
rowHeadingWidths
rowHeight
rowIndentation
showColumnDataGeneration
showColumnHeaderGeneration
showRowDataGeneration
showRowHeaderGeneration
toolbarVisible
visible
width
writebackEnabled

<blox:cellAlert> nested tag **See "cellAlert" on page 548.**
Attribute
index
apply
background
condition
description
enabled
font
foreground
format
group
link

<blox:cellAlert> nested tag **See "cellAlert" on page 548.**
Attribute
image_align
image
scope
value
value2

<blox:cellEditor> nested tag **See "cellEditor" on page 555.**
Attribute
index
scope

<blox:cellFormat> nested tag **See "cellFormat" on page 558.**
Attribute
index
background
font
foreground
format
group
scope

<blox:cellLink> nested tag **See "cellLink" on page 562.**
Attribute
index
description
link
scope
image_align
image

<blox:drillThroughWindow> nested tag **See "drillThroughWindow" on page 570.**
Attribute
height

<blox:drillThroughWindow> nested tag **See "drillThroughWindow" on page 570.**
Attribute
locationbarVisible
menubarVisible
name
resizable
scrollbarsVisible
statusbarVisible
toolbarVisible
url
width

<blox:editableCellStyle> nested tag **See "editableCellStyle" on page 572.**
Attribute
background
font
foreground

<blox:editedCellStyle> nested tag **See "editedCellStyle" on page 573.**
Attribute
background
font
foreground

<blox:formatMask> nested tag **See "formatMask" on page 575.**
Attribute
index
mask

<blox:formatName> nested tag **See "formatName" on page 577.**
Attribute
index
name

Usage

Each custom tag can have one or more attributes, separated by one or more space or new line characters. Any extra space or new line characters are ignored. For readability, attributes can each go on a separate line with the same indenting.

When there are no nested tags (such as the `<blox:cellAlert>` or `<blox:cellStyle>` tag), you can substitute the closing `</blox:grid>` tag using the shorthand notation, closing the tag at the end of the attribute list that looks as follows:

```
width="650" />
```

When there are nested tags, the shorthand notation is not valid and a closing tag is required.

Examples

```
<blox:grid id="myGrid"
     height="400"
     width="500"
     bandingEnabled="true" />
<blox:grid id="anotherGrid"
     height="300"
     width="500"
     bandingEnabled="true">
     <blox:cellAlert index="1"
          condition="any"
          background="cyan" />
</blox:grid>
```

GridBlox Properties and Methods By Category

This section lists the properties unique to GridBlox and associated methods. For a list of properties and methods common to several Blox, see the "Common Blox Properties and Methods by Category" on page 29. The properties and methods supported by GridBlox are organized in the cross reference as follows:

For GridBlox client-side API, see *Developer's Guide*.

Grid Appearance

The following properties and methods affect how the grid appears on a page.

Properties	Methods
`bandingEnabled`	
	`isBandingEnabled()` `setBandingEnabled()`
`cellLink`	
	`getCellLink()` `setCellLink()`
	`listCellLinkIds()`

columnHeadersWrapped	isColumnHeadersWrapped() setColumnHeadersWrapped()
columnWidths	getColumnWidths() setColumnWidths()
	getDataBlox()
expandCollapseMode	isExpandCollapseMode() setExpandCollapseMode()
gridLinesVisible	isGridLinesVisible() setGridLinesVisible()
headingIconsVisible	isHeadingIconsisible() setHeadingIconsVisible()
informationWindowName	getInformationWindowName() setInformationWindowName()
menubarVisible	isMenubarVisible() setMenubarVisible()
missingValueString	getMissingValueString() setMissingValueString()
noAccessValueString	getNoAccessValueString() setNoAccessValueString()
relationalRowNumbersOn	isRelationalRowNumbersOn() setRelationalRowNumbersOn()
removeAction	getRemoveAction() setRemoveAction()
rightClickMenuEnabled	isRightClickMenuEnabled() setRightClickMenuEnabled()
rowHeadersWrapped	isRowHeadersWrapped() setRowHeadersWrapped()
rowHeadingsVisible	isRowHeadingsVisible() setRowHeadingsVisible()
rowHeadingWidths	getRowHeadingWidths() setRowHeadingWidths()
rowHeight	getRowHeight() setRowHeight()

rowIndentation	
	getRowIndentation() setRowIndentation()
showColumnDataGeneration	
	isShowColumnDataGeneration() setShowColumnDataGeneration()
showColumnHeaderGeneration	
	isShowColumnHeaderGeneration() setShowColumnHeaderGeneration()
showRowDataGeneration	
	isShowRowDataGeneration() setShowRowDataGeneration()
showRowHeaderGeneration	
	isShowRowHeaderGeneration() setShowRowHeaderGeneration()

Numeric Formatting

These properties define how numeric values appear in the grid data area. For details on available formats, see:

http://java.sun.com/j2se/1.4.2/docs/api/java/text/DecimalFormat.html

Properties	Methods
cellFormat	
	getCellFormat() setCellFormat()
defaultCellFormat	
	getDefaultCellFormats() setDefaultCellFormats()
formatMask	
	getFormatMask() setFormatMask()
formatName	
	getFormatName() setFormatName()
localeCode	
	getLocaleCode() setLocaleCode()
	listCellFormatIds()

Cell Alerts

The following table shows properties and methods associated with cell alerts.

Properties	Methods
cellAlert	
	getCellAlert() setCellAlert()
	listCellAlertIds()

Drill to Relational Detail

Drillthrough operations are supported for IBM DB2 OLAP Server, IBM DB2 OLAP Server Deployment Services, Hyperion Essbase, or Essbase Deployment Services data sources which have drillthrough reports set up through Essbase Integration Services (EIS). This feature is also support for Microsoft Analysis Services.

Properties	Methods
drillThroughEnabled	
	isDrillThroughEnabled() setDrillThroughEnabled()
drillThroughWindow	
	getDrillThroughWindow() setDrillThroughWindow()

Printing

Printing properties apply to the grid when it is rendered for delivery in print format.

Properties
defaultCellFormat
headingsEnabled

Grid UI for Writeback and Comments

The Writeback UI properties set conditions on the GridBlox data area that permit users to change data cell values. The Comments UI property (commentsEnabled) specifies whether 1) the menu items for adding and displaying comments are displayed in the Grid cell right-click menu and 2) whether to display the comment indicator on the upper right corner when cell comments are available. These properties are used with "GridBlox Properties and Associated Methods" on page 546.

Properties	Methods
cellEditor	
	getCellEditor() setCellEditor()
	listCellEditorIds()
commentsEnabled	
	IsCommentsEnabled() setCommentsEnabled()
editableCellStyle	
	getEditableCellStyles() setEditableCellStyles()
editedCellStyle	
	getEditedCellStyles() setEditedCellStyles()
writebackEnabled	
	isWritebackEnabled() setWritebackEnabled()
	getChangedCellList()

	getChangedCellValues()
	updateProperties()

Popped Out Properties

The following table lists the properties regarding displaying GridBlox in a separate, popped out browser window.

Chart Labels

Properties	Methods
enablePoppedOut	isEnablePoppedOut() setPoppedOut()
poppedOut	isPoppedOut() setPoppedOut()
poppedOutHeight	getPoppedOutHeight() setPoppedOutHeight()
poppedOutTitle	getPoppedOutTitle() setPoppedOutTitle()
poppedOutWidth	getPoppedOutWidth() setPoppedOutWidth()

server-side Event Filters and Listeners Methods

The following table lists the methods for capturing events for pre- and post-event processing.

Methods
addEventFilter()
addEventListener()
removeEventFilter()
removeEventListener()

GridBlox Properties and Associated Methods

This section describes the properties supported by GridBlox and the methods associated with those properties. The properties are listed alphabetically by property name. For a list of GridBlox methods with which no properties are associated, see "GridBlox Methods" on page 590. Common Blox properties available from GridBlox are listed but not described. For complete descriptions of common Blox properties, see "Properties and Associated Methods Common to Multiple Blox" on page 32.

id

This is a common Blox tag attribute. For a complete description, see "id" on page 39.

applyPropertiesAfterBookmark

This is a common Blox property. For a detailed description, see "applyPropertiesAfterBookmark" on page 32.

autosizeEnabled

Specifies whether columns should automatically resize to accommodate the largest data value.

Data Sources

All

Syntax

JSP Tag Attribute

```
autosizeEnabled="autosize"
```

Java Methods

```
boolean isAutosizeEnabled();
void    setAutosizeEnabled(boolean autosize);
```

where:

Argument	Default	Description
`autosize`	`true`	Specify `false` to render columns and rows based on `columnWidths`, `rowHeadingWidths` and `rowHeight` settings.

Examples

```
autosizeEnabled = "true"
isAutosizeEnabled();
setAutosizeEnabled(true);
```

bandingEnabled

Specifies whether to enable alternate background colors for grid rows.

Data Sources

All

Syntax

JSP Tag Attribute

```
bandingEnabled="enable"
```

Java Methods

```
boolean isBandingEnabled();
void    setBandingEnabled(boolean enabled);
```

where:

Argument	Default	Description
`enabled`	`true`	Specify `true` to enable cell banding; `false` to disable it.

Usage

The default is `true` when the application's default render mode is set to DHTML.

Examples

```
isBandingEnabled();
setBandingEnabled(true);
```

bloxEnabled

This is a common Blox property. For a complete description, see “bloxEnabled” on page 35.

bloxModel

This is a common Blox property. For a complete description, see “bloxModel” on page 38

bloxName

This is a common Blox property. For a complete description, see “bloxName” on page 35.

bookmarkFilter

This is a common Blox property. For a complete description, see “bookmarkFilter” on page 33.

cellAlert

Specifies a rule for highlighting values in numeric data cells.

Data Sources

All

Syntax

JSP Tag Attribute

```
<blox:cellAlert
      index="cellAlertNumber"
      apply="row|column|cell"
      background="background"
      condition="condition"
      description="description"
      enabled="enabled"
      font="font"
      foreground="foreground"
      format="formatmask"
      group="groupName"
      image="image"
      image_align="left|right|center"
      link="link"
      scope="scope"
      value="value1"
      value2="value2"
/>
```

Java Methods

```
String  getCellAlert(int id);
void setCellAlert(int id, String alertRule);
```

where:

Argument	Default	Description
`id`	null	For `setCellAlert()`, any positive integer representing the number of the cell alert to define. For `getCellAlert()`, any previously-defined alert number.
`alertRule`	empty string	A comma-delimited string of attribute settings (*name=value*). The entire string must be enclosed within double quotes. The table below lists the supported `alertRule` attributes.

`alertRule` Attributes and `cellAlert` Tag Attributes

The following table lists and describes the `alertRule` attributes as well as the tag attributes for `cellAlert`. If you do not specify a given attribute, its default value applies to the cell. For example, if the default cell background is white and you do not specify a background attribute, the cell background remains white.

Attribute	Required/ Optional	Description
`apply`	Optional	The area of the grid subject to highlighting. Possible values: • `ROW`: Highlight the entire row if any cell within the row meets the condition. The first row-applied alert takes precedence over other row-applied alerts. • `COLUMN`: Highlight the entire column if any cell within the column meets the condition. The first column-applied alert takes precedence over other column-applied alerts. • `CELL`: Highlight only the specific cells that meet the condition (the default). A cell-applied alert will override a row- or column-applied alert.
`background`	Optional	The cell's background color. Use a color name or hexadecimal value.
`condition`	Required	A condition to apply to the cell value. Cells that meet the condition activate the alert. Possible values: • `ANY`: Accept cells of any value. • `MISSING`: Accept only cells with missing values. • `NA`: Accept only cells where the value is not available. The following possible values work together with the `value` and `value2` attributes: • `LT` or `<` • `GT` or `>` • `EQ` or `=` • `LTEQ` or `<=` • `GTEQ` or `>=` • `BETWEEN` (*value*`,` *value2*): The value is greater than or equal to *value* and less than or equal to *value2*.
`description`	Optional	A description of the cell alert. This description appears as a tool tip when the user hovers the mouse pointer over the cell.
`enabled`	Optional	Specifies whether the cell alert is active. Set to `false` to disable a cell alert. The default is `true`.

Attribute	Required/ Optional	Description
font	Optional	The *font name:style:point* use for the cell's text. • *font name*: Acceptable font name values vary widely by browser and client machine. The following font names are generally accepted: Arial, Courier, Helvetica, TimesRoman, SansSerif, Serif, Monospace. • *style*: Valid font styles are: plain, italic, bold, and bolditalic • *point*: An integer for point size (usually 8-36). If any of the three attributes is not specified, the default or the currently inherited font value is applied. However, the colons separating the attributes should be included. For examples: font="Arial:bolditalic:12" font=":Bold:12"
foreground	Optional	The cell's text color. Use a color name or hexadecimal value.
format	Optional	Format mask to apply to cell data. For more information, see "defaultCellFormat" on page 568.
group	Optional	The name of a group of cell alerts to be treated as a traffic lighting group. Cell alerts with the same group name will show up as one set of traffic lights in the traffic lights creation and management user interface.
image	Optional	A custom image that points to the defined link. If you define a link but do not specify a custom image, the cell contents are shown as a link. However, if a custom image is specified but no link is defined, the image will still appear. The URL to the image can be either absolute or relative: • 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. Note that 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.
image_align	Optional	Sets the position of the custom image specified by the image attribute. Valid values are: • LEFT: Place image before the cell contents (default) • RIGHT: Place image after the cell contents

Attribute	Required/ Optional	Description

link	Optional	A hyperlink for HTML rendering or a call to a JavaScript method.
		
		When the link is to a URL, it always opens in a new window.
		
		The following entities in the link value are replaced by the indicated values when the link is generated. Entities must start with an ampersand (&) and end with a semicolon (;).
		• &Description; — the cell alert description
		• &Value; — the cell value
		• &ColHeader; — ampersand-separated dimension/member value pairs
		• &RowHeader; — ampersand-separated dimension/member value pairs
		• &ColIndex; — the display index (0-based) of the column in the grid
		• &RowIndex; — the display index (0-based) of the row in the grid
		
		In the following example, the link for the cell alert is set to:
		`link="decoderequest.jsp?row=&RowHeader;&column= &ColHeader;&value=&Value;&rowIndex=&RowIndex;&col Index=&ColIndex;"`
		
		If the cell with the value of 1234.55 on the third row of the Time dimension and the fourth column of the Product dimension is clicked on, the URL passed through will be:
		`decoderequest.jsp?row=Time=Q3&column=Product=Chocolate%20 Nuts&value=1234.55&rowIndex=2&colIndex=3`

Attribute	Required/ Optional	Description
scope	Optional	The cells to which the alert should be applied, specified as a series of dimension and member sets enclosed in braces. Use unique names to ensure the right member is found. In IBM DB2 OLAP Server or Hyperion Essbase, using display names will not work if DataBlox's useAliases is set to false (users can set this through the user interface). In MSAS, use unique names to ensure the correct member at the correct level is found. SCOPE applies to all axes of the result set, not just the row and column axes. Specify the scope as follows: scope={d0:m00[,m01,... m0n]} {d1:m10[,m11,... m1n]}...] where d0 denotes a dimension and m00 denotes a member within that dimension. For example, for IBM DB2 OLAP Server or Hyperion Essbase data sources: scope={Product:Coke} {Scenario: Actual, Budget} For Microsoft Analysis Services data sources, use unique names as follows: scope={[Product]: [Product].[Code]} {[Scenario]: [Scenario].[All Scenario].[Actual], [Scenario].[All Scenario].[Budget]} The following member search functions are available for specifying the level of members the cell alert should be applied to: • Leaf(): the leaf-level descendants of the specified member. Only one member can be specified in the function. Example: scope="{Market:leaf(East)}" • Child(): the children of the specified member. Only one member can be specified in the function. Example: scope="{Market:child(East)}" • Descendants(): all descendants of the specified member. Only one member can be specified in the function. Example: scope="{Market:descendants(East)}" • Gen(): all members of the specified generation. Example: scope="{Market:gen(2)}" • Not(): members to which the cell alert should not be applied. You can specify multiple members, separated by a comma. Example: scope="{Market:not(East, West)}" The function names are case-insensitive. You can combine the functions in the scope statement.
value	Optional	The value to compare when condition is LT, GT, EQ, LTEQ, GTEQ, or the lesser value when condition is BETWEEN.
value2	Optional	The greater value when condition is BETWEEN.
index	Optional	Note: This attribute is only valid as a cellAlert tag attribute. For setCellAlert Java method, use the id argument instead. The number of the cellAlert to define. If you do not specify this attribute, the next available cellAlert number is used. For instance, if cellAlerts 1-4 are already defined, cellAlert 5 is used.

Usage

Cell alert formatting applies only to numerical content. The number of the cell alert dictates the order in which it is evaluated. Each data cell value is evaluated against all defined alerts, starting with cellAlert1, to the highest defined alert number. The

first cell alert that matches the data cell's value and scope is the only alert applied to that cell. Be sure to define cell alerts in the appropriate order if there are overlaps.

Note the following:

- For editable cells, cell alert formatting takes precedence over cell editor formatting when a grid first appears. Once a cell is edited, cell editor color settings take precedence over those specified by cell alerts.
- Use a unique name (base name in IBM DB2 OLAP Server or Hyperion Essbase) or display name for the dimension and member name string specified in the Scope. This allows assemblers to differentiate between different members or dimensions with the same display names. In IBM DB2 OLAP Server or Hyperion Essbase, an assembler can specify a member, regardless of the alias table in use, by using the base name.

Examples

```
getCellAlert(4);
setCellAlert(2, "condition=GT, value=1000, scope={Market:Central},
foreground=green, background=white, align=right");
```

Note that in the above example using the `setCellAlert()` method, the entire alert rule is enclosed in double quotes. If a member name in the `scope` attribute contains a comma, then double quote the member name and escape the quotes (\"). Since member names may contain single quotes, when a quotation mark is needed inside the alert rule string, always use an escaped double quote.

The following are more examples using Blox tags:

- Divide a value by 1000 before displaying it:

```
<blox:cellAlert index="1"
                condition="any"
                format="#,###/1000;[red](#,###/1000)"
                background="#3333FF"
                scope="{Scenario: Budget}" />
```

- Use a symbol (in this case, the percent sign) as a literal character:

```
<blox:cellAlert index="3"
                condition="any"
                format="#'%';[red](#'%')"
                background="#9999FF"
                scope="{Scenario: Variance %}" />
```

- Multiply a value by 100 before it is displayed and use Times New Roman, boldface, size 14 for font:

```
<blox:cellAlert index="4"
                condition="any"
                format="#.##*100%;[red](#.##*100%)"
                background="#CCCCFF"
                scope="{Scenario: Variance %}"
                font="Times New Roman:bold:14" />
```

- The following example sets cell alert 2 on a grid displayed in a PresentBlox. The alert tests the Central member of the Market dimension for values greater than 1000. When a data value meets this criterion:
  - The foreground (font) color becomes green.
  - The background color becomes white.
  - Values are aligned on the right.

```
PresentBlox.getGridBlox().setCellAlert(2, "condition=GT, value=1000,
scope={Market:Central}, foreground=green, background=white, align=right");
```

- The following example highlights all numerical content with a cyan background:

```
<blox:cellAlert index="1"
                condition="any"
                background="cyan" />
```

This will apply a cyan background to all cells that contain numerical data. If a cell contains no data, this background color will not be applied.

- A series of related cell alerts paints increasing cell amounts in different colors:

```
<blox:cellAlert index="1"
                 condition="Between" value="1000" value2="3000"
                 format="00.00"
                 foreground="Orange"
                 scope="{Market:East,West,South,Central}" />

<blox:cellAlert index="2"
                 condition="Between" value="3001" value2="5000"
                 scope="{Year:Qtr1,Qtr2}"
                 format="00.00"
                 foreground="Blue" />

<blox:cellAlert index="3"
                 condition="GT"
                 value="5000"
                 format="00.00"
                 foreground="Magenta" />

<blox:cellAlert index="4"
                condition="LT"
                value="1000"
                format="(00.00)"
                 foreground="Red" />
```

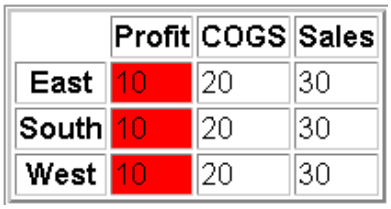
and results in the following grid:

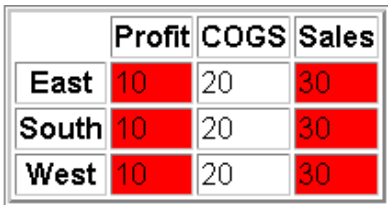
Year	Product	East	West	South	Central	Market
Qtr1	Colas	2747.00	1042.00	1051.00	2208.00	7048.00
	Root Beer	(562.00)	2325.00	1465.00	2369.00	6721.00
	Cream Soda	(591.00)	2363.00	(561.00)	2414.00	5929.00
	Fruit Soda	1480.00	1407.00		2118.00	5005.00
	Diet Drinks	(555.00)	2025.00	1146.00	3291.00	7017.00
	Product	5380.00	7137.00	3077.00	9109.00	24703.00
Qtr2	Colas	3352.00	(849.00)	1198.00	2473.00	7872.00
	Root Beer	(610.00)	2423.00	1540.00	2457.00	7030.00
	Cream Soda	(922.00)	2739.00	(529.00)	2579.00	6769.00
	Fruit Soda	1615.00	1504.00		2317.00	5436.00
	Diet Drinks	(652.00)	1975.00	1289.00	3420.00	7336.00
	Product	6499.00	7515.00	3267.00	9826.00	27107.00

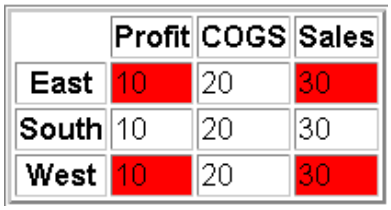
Scope Examples

The examples in the following table assume `background=red` and `condition=any`, and show the result using different scopes.

Note: The `scope` attribute applies to all axes of the result set, not just the row and column axes. For example, if the grid is filtered on the profit member of the accounts dimension (that is, the accounts dimension is placed on the page filter with the profit member selected), the grid appears as shown in the first example below.

scope Example	Result
The following scope applies only to the profit member of the accounts dimension: `scope="{Accounts: Profit}"`	

The following scope applies to the profit and sales members of the accounts dimension:
`scope="{Accounts: Profit, Sales}"`
Note that this means profit OR sales. You can also specify the scope as follows:
`scope="{Accounts: not(COGS)}"`
	

The following scope applies where the profit and sales members of the accounts dimension intersect with the east and west members of the market dimension. This is an AND operation.
`scope="{Accounts: Profit, Sales}, {Market: East, West}"`
or
`scope="{Accounts: not(COGS)}, {Market: not(South)}"`
	

	Profit	COGS	Sales
East	10	20	30
South	10	20	30
West	10	20	30

	Profit	COGS	Sales
East	10	20	30
South	10	20	30
West	10	20	30

	Profit	COGS	Sales
East	10	20	30
South	10	20	30
West	10	20	30

See Also

"cellEditor" on page 555, "cellFormat" on page 558, "cellLink" on page 562, "clearCellAlerts()" on page 591, "isAlertEnabled() setAlertEnabled()" on page 593, "listCellAlertIds()" on page 594, "Cell Alerts" on page 544

cellEditor

Specifies a rule for defining and highlighting an editable area of data cells.

Data Sources

All

Syntax

JSP Tag Attribute

```
<blox:cellEditor
     index="cellEditorNumber"
     scope="scope" >
</blox:cellEditor>
```

Java Methods

```
String  getCellEditor(int id);
void setCellEditor(int id, String editorRule);
```

where:

Argument	Default	Description
`id`	null	For `setCellEditor()`, any positive integer representing the number of the cell editor to define. For `getCellEditor()`, any previously-defined editor number.

Argument	Default	Description
editorRule	empty string	A comma-delimited string of attribute settings (*name=value*). The entire string must be enclosed within quotation marks. The table below lists the supported editorRule attributes.

EditorRule Attributes and cellEditor Tag Attributes

The following table lists and describes the editorRule attributes as well as the cellEditor tag attributes.

Attribute	Required?	Description

scope	Required	The cells to which the editor should be applied, specified as a series of dimension and member sets enclosed in braces. Use unique names to ensure the right member is found. In IBM DB2 OLAP Server or Hyperion Essbase, using display names will not work if DataBlox's useAliases is set to false (users can set this through the user interface). In MSAS, use unique names to ensure the correct member at the correct level is found.
		
		SCOPE applies to all axes of the result set, not just the row and column axes.
		
		Specify the scope as follows:
		`scope={d0:m00[,m01,... m0n]} {d1:m10[,m11,... m1n]}...]`
		
		where d0 denotes a dimension and m00 denotes a member within that dimension. For example, for IBM DB2 OLAP Server or Hyperion Essbase data sources:
		`scope={Product:Coke} {Scenario: Actual, Budget}`
		
		For Microsoft Analysis Services data sources, use unique names as follows:
		`scope={[Product]: [Product].[Code]} {[Scenario]: [Scenario].[All Scenario].[Actual], [Scenario].[All Scenario].[Budget]}`
		
		The following member search functions are available for specifying the level of members the editor should be applied to:
		• Leaf(): the leaf-level descendants of the specified member. Only one member can be specified in the function. Example: scope="{Market:leaf(East)}"
		• Child(): the children of the specified member. Only one member can be specified in the function. Example: scope="{Market:child(East)}"
		• Descendants(): all descendants of the specified member. Only one member can be specified in the function. Example: scope="{Market:descendants(East)}"
		• Gen(): all members of the specified generation. Example: scope="{Market:gen(2)}"
		• Not(): members to which the cell editor should not be applied. You can specify multiple members, separated by a comma. Example: scope="{Market:not(East, West)}"
		
		The function names are case-insensitive. You can combine the functions in the scope statement. See "Scope Examples" on page 554 for cellAlert. The same syntax applies to cellEditor.

index	Optional	Note: This attribute is only valid as a cellEditor tag attribute. For the setCellEditor Java method, use the id argument instead. The number of the cell editor to define. If you do not specify this attribute, the next available cell editor number is used. For example, if cell editors 1-4 are already defined, cell editor 5 is used.

Usage

Use the scope attribute to define areas of editable cells in the grid.

To activate cell editors you must set writebackEnabled to true.

You can make non-numeric cells to be editable on the grid user interface. However, non-numeric values are written back as missing values. Unlike IBM DB2 OLAP Server or Hyperion Essbase, which only allows writeback of numeric values, Microsoft Analysis Services allows writeback of any data type. Care should be taken when specifying the scope so non-numeric cells are not overwritten with the "#MISSING" string. With relational data sources, DB2 Alphablox does not perform data writeback. You need to programmatically get the list of changed cells and their new values and write back the values using JDBC. The benefit of this approach is that you can write back non-numeric data. For an example, see the RDB Writeback from Grid example in the Application Studio (the Examples link under the Assembly tab.

Cell alert formatting takes precedence over cell editor formatting when a grid first appears. Once a cell is edited, cell editor color settings take precedence over those specified by cell alerts.

Examples

```
getCellEditor(5);
setCellEditor(3, "scope={Market: East}");
setCellEditor(3, "scope={leaf(Market)}");
```

The following example sets cell editor 2 on a grid displayed in a PresentBlox. Any values within the Market dimension except those for the Central member are editable.

```
myPresent.getGridBlox().setCellEditor(2, "Scope={Market:not(Central)}");
```

See Also

"clearCellEditors()" on page 591, "cellAlert" on page 548, "cellFormat" on page 558, "cellLink" on page 562, "listCellEditorIds()" on page 594

cellFormat

Specifies the format for data values in the grid's cells.

Data Sources

All

Syntax

JSP Tag Attribute

```
<blox:cellFormat
      index="cellFormatNumber"
      background="background"
      font="font"
      foreground="foreground"
      format="formatmask"
      group="group"
      scope="scope" >
</blox:cellFormat>
```

Java Methods

```
String  getCellFormat(int id);
void setCellFormat(int id, String formatRule);
```

where:

Argument	Default	Description
`id`	null	For `setCellFormat()`, any positive integer representing the number of the cell format to define. For `getCellFormat()`, any previously-defined cell format number.
`formatRule`	empty string	A comma-delimited string of attribute settings (`name=value`). The entire string must be enclosed within quotation marks. The table below lists the supported `formatRule` attributes.

`FormatRule` Attributes and `cellFormat` Tag Attributes

The following table lists and describes the `formatRule` attributes as well as the `cellFormat` tag attributes.

Attribute	Required/ Optional	Description
`background`	Optional	The cell's background color. Use a color name or hexadecimal value.

`font`	Optional	The `font name:style:point` use for the cell's text.
		• *font name*: Acceptable font name values vary widely by browser and client machine. The following font names are generally accepted: Arial, Courier, Helvetica, TimesRoman, SansSerif, Serif, Monospace.
		• *style*: Valid font styles are: `plain`, `italic`, `bold`, and `bolditalic`
		• *point*: An integer for point size (usually 8-36).
		
		If any of the three attributes is not specified, the default or the currently inherited font value is applied. However, the colons separating the attributes should be included. For examples:
		`font="Arial:bolditalic:12"`
		`font=":Bold:12"`

`foreground`	Optional	The cell's text color. Use a color name or hexadecimal value.
`format`	Required	Format mask to apply to cell data. For more information, see "defaultCellFormat" on page 568.
`group`	Optional	A name space to group cell formats of the same name as a set.

Attribute	Required/ Optional	Description

scope	Required	The cells to which the format should be applied, specified as a series of dimension and member sets enclosed in braces. Use unique names to ensure the right member is found. In IBM DB2 OLAP Server or Hyperion Essbase, using display names will not work if DataBlox's useAliases is set to false (users can set this through the user interface). In MSAS, use unique names to ensure the correct member at the correct level is found.
		
		SCOPE applies to all axes of the result set, not just the row and column axes.
		
		Specify the scope as follows:
		`scope="{d0:m00[,m01,... m0n]} {d1:m10[,m11,... m1n]}...]"`
		
		where d0 denotes a dimension and m00 denotes a member within that dimension. For example, for IBM DB2 OLAP Server or Hyperion Essbase data sources:
		`scope={Product:Coke} {Scenario: Actual, Budget}`
		
		For Microsoft Analysis Services data sources, use unique names as follows:
		`scope={[Product]: [Product].[Code]} {[Scenario]: [Scenario].[All Scenario].[Actual], [Scenario].[All Scenario].[Budget]}`
		
		The following member search functions are available for specifying the level of members the format should be applied to:
		• Leaf(): the leaf-level descendants of the specified member. Only one member can be specified in the function. Example: scope="{Market:leaf(East)}"
		• Child(): the children of the specified member. Only one member can be specified in the function. Example: scope="{Market:child(East)}"
		• Descendants(): all descendants of the specified member. Only one member can be specified in the function. Example: scope="{Market:descendants(East)}"
		• Gen(): all members of the specified generation. Example: scope="{Market:gen(2)}"
		• Not(): members to which the cell format should not be applied. You can specify multiple members, separated by a comma. Example: scope="{Market:not(East, West)}"
		
		The function names are case-insensitive. You can combine the functions in the scope statement. See "Scope Examples" on page 554 for cellAlert. The same syntax applies to cellFormat.

Attribute	Required/ Optional	Description
index	Optional	Note: This attribute is only valid as a cellFormat tag attribute. For the setCellFormat Java method, use the id argument instead. The number of the cell format to define. If you do not specify this attribute, the next available cell format number is used. For example, if cell formats 1-4 are already defined, cell format 5 is used.

Usage

The cellFormat property can be used in conjunction with cell alerts.

Note the following about cell formatting:

- Do not use the backslash escape character (\) in the format string.
- To display a symbol such as the percent sign (%), use single quotes. If the symbol to display is a double quote, then precede it with a backslash escape character (\).
- The cell format number (*N*) dictates the order in which the format mask is evaluated. Each data cell value is evaluated against all defined masks in order starting from cellFormat1. If a later cellFormat overlaps with an earlier one, the last one is applied. This means the cell format with largest id (Java method) or index (JSP tag) wins. Be sure to define cell format masks in the correct order if there are overlaps.
- A unique name (base name in IBM DB2 OLAP Server or Hyperion Essbase) or display name can be used for the dimension and member name string specified in the scope. This allows assemblers to differentiate between different members or dimensions with the same display names. In IBM DB2 OLAP Server or Hyperion Essbase, an assembler can specify a member, regardless of the alias table in use, by using the base name.
- For IBM DB2 OLAP Server or Hyperion Essbase data sources, use defaultCellFormat or cellFormat to control the formatting of data values instead of the {DECIMAL} report script command.

Examples

```
getCellFormat(7);
setCellFormat(4, "format=#,##0.00, scope={Accounts:COGS}");
setCellFormat(5, "format=##.##'%', scope={Accounts:COGS, Total}");
setCellFormat(6, "format='$'#,##0.00, scope={Product:Coke} {Scenario:
Actual, Budget}");
setCellFormat(8, "format='$'#,##0.00, scope={Market:gen(2)}");
```

Snippets

- For all cell values in the Profit member of the Accounts dimension and the TV and Video members of the Product dimension, apply a format with two decimal places and a comma separating the hundreds and thousands positions. Show the cell value in red if it is 999.99 or less:

```
<blox:cellFormat
                  index="1"
                  format="#,###.##; [red]###.##"
                   scope="{Accounts:Profit}{Product:TV, Video}" />
```

- For all cell values in the Accounts dimension except the COGS member, apply the thousands format:

```
<blox:cellFormat
                  index="2"
                  format="#,###K"
                   scope="{Accounts:not(COGS)}" />
```

- For all cell values in the Total member of the Accounts dimension, apply a format that includes a dollar sign and rounds to the whole dollar with a specific background color and font style for the cells:

```
<blox:cellFormat
                  index="4"
                  format="$#,##0"
                  scope="{Accounts:Total}"
                  font="Arial:Bold:20"
                   background="#CCCCFF" />
```

- Format all cell values in the COGS and Total members of the Accounts dimension as two-decimal percentages, and if the percentage is less then 1%, show the number as, for example, 0.55 % rather than .55%:

```
<blox:cellFormat
                  index="5"
                  format="0.##'%'"
                   scope="{Accounts:COGS, Total}" />
```

See Also

"defaultCellFormat" on page 568, "cellAlert" on page 548, "cellEditor" on page 555, "cellLink" on page 562, "listCellFormatIds()" on page 594

cellLink

Specifies a rule for defining cells that contain a link.

Data Sources

All

Syntax

JSP Tag Attribute

```
<blox:cellLink
      index="cellLinkNumber"
      description="description"
      image="image"
      image_align="left|right|center"
      link="link"
      scope="scope" >
</blox:cellLink>
```

Java Methods

```
String  getCellLink(int id);
boolean setCellLink(int id, String linkRule);
```

where:

Argument	Default	Description
id	null	For setCellLink(), any positive integer representing the number of the cell link to define. For getCellLink(), any previously-defined cell link number.
linkRule	empty string	A comma-delimited string of attribute settings (*name=value*). The entire string must be enclosed within quotation marks. The table below lists the supported linkRule attributes.

LinkRule Attributes and cellLink Tag Attributes

This following table lists and describes the linkRule attributes as well as the tag attributes for cellLink. If you do not specify a given property, its default value applies to the cell.

Attribute	Required/ Optional	Description
description	Optional	A description of the cell link.
image	Optional	A custom image that points to the defined link. If you define a link but do not specify a custom image, the cell contents are shown as a link. However, if a custom image is specified but no link is defined, the image will still appear. The URL to the image can be either absolute or relative: • 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. Note that 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.
image_align	Optional	Sets the position of the custom image specified by the image attribute. Valid values are: • LEFT: Place image before the cell contents (default) • RIGHT: Place image after the cell contents

Attribute	Required/ Optional	Description
`link`	Required	A hyperlink for HTML rendering or a call to a JavaScript method. When the link is to a URL, it always opens in a new window. **Note:** The URL can be an absolute or relative URL: • 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. Note that 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. The following entities in the link value are replaced by the indicated values when the link is generated. Entities must start with an ampersand (&) and end with a semicolon (;). • `&Description;` — the cell alert description • `&Value;` — the cell value • `&ColHeader;` — ampersand-separated dimension/member value pairs • `&RowHeader;` —ampersand-separated dimension/member value pairs • `&ColIndex;` — the display index (0-based) of the column in the grid • `&RowIndex;` — the display index (0-based) of the row in the grid For an example of how the replacement variables work, see the Examples section for this property.

Attribute	Required/ Optional	Description
scope	Optional	The cells to which the link should be applied, specified as a series of dimension and member sets enclosed in braces. Use unique names to ensure the right member is found. In IBM DB2 OLAP Server or Hyperion Essbase, using display names will not work if DataBlox's useAliases is set to false (users can set this through the user interface). In MSAS, use unique names to ensure the correct member at the correct level is found. SCOPE applies to all axes of the result set, not just the row and column axes. Specify the scope as follows: scope={d0:m00[,m01,... m0n]} {d1:m10[,m11,... m1n]}...] where d0 denotes a dimension and m00 denotes a member within that dimension. For example, for IBM DB2 OLAP Server or Hyperion Essbase data sources: scope={Product:Coke} {Scenario: Actual, Budget} For Microsoft Analysis Services data sources, use unique names as follows: scope={[Product]: [Product].[Code]} {[Scenario]: [Scenario].[All Scenario].[Actual], [Scenario].[All Scenario].[Budget]} The following member search functions are available for specifying the level of members the link should be applied to: • Leaf(): the leaf-level descendants of the specified member. Only one member can be specified in the function. Example: scope="{Market:leaf(East)}" • Child(): the children of the specified member. Only one member can be specified in the function. Example: scope="{Market:child(East)}" • Descendants(): all descendants of the specified member. Only one member can be specified in the function. Example: scope="{Market:descendants(East)}" • Gen(): all members of the specified generation. Example: scope="{Market:gen(2)}" • Not(): members to which the cell link should not be applied. You can specify multiple members, separated by a comma. Example: scope="{Market:not(East, West)}" The function names are case-insensitive. You can combine the functions in the scope statement. See "Scope Examples" on page 554 for cellAlert. The same syntax applies to cellLink.

Attribute	Required/ Optional	Description
index	Optional	Note: This attribute is only valid as a cellLink tag attribute. When using the setCellLink Java method, use the id argument instead. The number of the cell link to define. If you do not specify this attribute, the next available cell link number is used. For example, if cell link 1-4 are already defined, cell link 5 is used.

Usage

The cellLink property allows you to specify the conditions under which cells will point to a defined hyperlink for HTML rendering or a call to a JavaScript method. The number of the cell link dictates the order in which it is evaluated, starting with cellLink1. The first defined cell link that matches the cell's condition and scope is the only link applied to that cell. Be sure to consider possible overlaps when defining cell links.

Note the following:

- Links defined using cellLink do appear in editable cells defined using cellEditor.
- Links defined using cellAlert take precedence over links defined with cellLink. On a given cell, if both a cellAlert containing a link and a cellLink are defined and the conditions for both parameters are true, the cellAlert link is used. If the condition of the cellAlert is not true, the cellLink is used.

Examples

The following example adds a cell link to all cells in {Market:Central} regardless of the cell values. The cell link indicator is displayed to the left of the cell contents. When users click on the links, the page "www.ibm.com" is displayed in a new browser window.

```
setCellLink(3, "scope={Market:Central},
description=Cells with the DB2 Alphablox link,
link=http://www.ibm.com, image=myIcon.gif, image_align=left");
```

In the following example, the link for the cell alert is set to:

```
link="decoderequest.jsp?row=&RowHeader;&column=&ColHeader;&value=&Value;
&rowIndex=&RowIndex;&colIndex=&ColIndex;"
```

If a user clicks on the cell for Q3, John Bob in the following Grid:

Time	Hank	Jack	Mary Lou	John Bob
Q1		*(i)***115551.471**	14025.051	82578.896
Q2	13135.487	*(i)***117395.421**	34878.844	39445.495
Q3	*(i)***191617.066**	93620.337	51401.572	*(i)***111110.831**
Q4	24777.37	84440.596	#No Access	44903.466

the URL passed through will be:

```
decoderequest.jsp?row=Time=Q3&column=Customer=John%20Bob&value=111110.831&
rowIndex=2&colIndex=3
```

See Also

"cellAlert" on page 548, "cellEditor" on page 555, "cellFormat" on page 558, "listCellLinkIds()" on page 595

columnHeadersWrapped

Specifies whether multi-word headings in grid column headers should be wrapped onto more than one line.

Data Sources

All

Syntax

JSP Tag Attribute

```
columnHeadersWrapped="wrapped"
```

Java Methods

```
boolean isColumnHeadersWrapped();       // returns boolean
void setColumnHeadersWrapped(boolean wrapped);
```

where:

Argument	Default	Description
wrapped	false	Specify true to enable wrapping of column headers.

Usage

When this property is set to `false` (the default), the column width is sized to fit the entire column header text without wrapping. When this property is set to `true`, the column header text will wrap to reduce column width. The width of the column will be automatically determined to ensure the longest word in the header and the longest data in the data cell will fit.

See Also

"rowHeadersWrapped" on page 584

columnWidths

Specifies the widths of the columns in the grid.

Data Sources

All

Syntax

JSP Tag Attribute

```
columnWidths="widths"
```

Java Methods

```
String getColumnWidths();   //throws ServerBloxException
void setColumnWidths(String widths);
          // throws InvalidBloxPropertyValueException,
                     ServerBloxException
```

where:

Argument	Default	Description
widths	null	A comma-delimited string of integers that defines column widths in pixels.

Usage

The autosizeEnabled property needs to be false for this property to be used. The browser automatically determines the column widths to fit the longest data value. If you explicitly set the column widths, the value will be ignored if the value to be displayed in a column exceeds the defined width.

See Also

"autosizeEnabled" on page 547

commentsEnabled

Specifies whether 1) the menu items for adding and displaying comments are displayed in the Grid's right-click menu and 2) whether to display the comment indicator on the upper right corner when cell comments are available.

Data Sources

Multidimensional

Syntax

JSP Tag Attribute

```
commentsEnabled = "boolean"
```

Java Methods

```
boolean isCommentsEnabled();
void setCommentsEnabled(boolean enabled);
```

where:

Argument	Default	Description
enabled	false	Whether the Comments menu item and its sub menu items Add Comments and Display Comments are available to users in the Grid cell's right-click menu. When set to false, these choices will not be displayed and the comment indicator (a red triangle) on the cell's upper right corner will not display when there are comments associated with the cell.

See Also

Chapter 9, "CommentsBlox Reference," on page 279.

defaultCellFormat

Specifies the default format mask for all data values in the grid.

Data Sources

All

Syntax

JSP Tag Attribute

```
defaultCellFormat="mask"
```

Java Methods

```
String getDefaultCellFormat();
void setDefaultCellFormat(String mask);
```

where:

Argument	Default	Description
mask	empty string	String that defines cell formatting attributes. For more information on the format string, see the examples below.

Usage

The formatting specified by `defaultCellFormat` is applied when no other style format exists (either through the cellFormat property or through the user interface).

Note the following about the `defaultCellFormat` property:

- Do not use the backslash escape character (\) in the format string.
- To display a symbol such as the percent sign (%), use single quotes. If the symbol to display is a double quote, then precede it with a backslash escape character (\).
- In some virtual machines (Sun 1.1.6, for example), the number mask (i.e., `#,###.00`) must be the same for the positive and negative masks. If they are not, the negative mask does not work properly.
- For IBM DB2 OLAP Server or Hyperion Essbase data sources, use `defaultCellFormat` or cellFormat to control the formatting of data values instead of the `{DECIMAL}` report script command.

Examples

- Red negatives: #,###.00;[red]-#,###.00

  ```
  defaultCellFormat="#,###.00;[red]-#,###.00"
  ```

- Parenthesis around red negatives: #,###.00;[red](#,###.00)

  ```
  defaultCellFormat="#,###.00;[red](#,###.00)"
  ```

- Millions: #,###M

  ```
  defaultCellFormat="#,###M"
  ```

- Thousands: #,###K

  ```
  defaultCellFormat="#,###K"
  ```

- Percent with 2 decimal places: ###.00'%'

  ```
  defaultCellFormat="###.00'%'"
  ```

- Show integers padded with zeros to 6 digits: 000000

  ```
  defaultCellFormat="000000"
  ```

- Show two places of decimals (regardless of the precision of the underlying value: #,###.00

  ```
  defaultCellFormat="#,###.00"
  ```

See Also

"cellFormat" on page 558, "formatMask" on page 575

drillThroughEnabled

Specifies if drillthrough operation is enabled on the GridBlox user interface.

Data Sources

IBM DB2 OLAP Server, Hyperion Essbase, Microsoft Analysis Services

Syntax

JSP Tag Attribute

```
drillThroughEnabled="drillThroughEnabled"
```

Java Methods

```
boolean isDrillThroughEnabled();
                // throws ServerBloxException
void setDrillThroughEnabled(boolean drillThroughEnabled);
   // throws InvalidBloxPropertyValueException, ServerBloxException
```

where:

Argument	Default	Description
drillThroughEnabled	false	If set to true , the "Drill through" option is displayed on the client's GridBlox right-click menu, allowing users to drill to relational detail.

Usage

For IBM DB2 OLAP Server, IBM DB2 OLAP Server Deployment Services, Hyperion Essbase, or Essbase Deployment Services, this is for data sources which have drillthrough reports set up through IBM DB2 OLAP Server Integration Services or Essbase Integration Services.

When drillThroughEnabled is set to true on a GridBlox, by default DB2 Alphablox sends the coordinates of the cell where the drillthrough operation is initiated to a RDBResultSetDataBlox and renders the relational detail using a ReportBlox. The report is displayed in a separate, resizable browser window. If the user right-clicks on a different cell and selects Drill through from the right-click menu, another browser window will pop up displaying the relational detail. This allows users to compare detailed data for different cells.

If the cell where the drillthrough operation is triggered is at the lowest level of its hierarchy, only one row set is returned. Otherwise, all of the row sets that make up the source data of that cell are returned. For MSAS, the maximum number of rows that can be returned is determined by the Maximum DrillThrough Rows setting specified in the DB2 Alphablox Home Page's Administration tab, Data Sources link. For IBM DB2 OLAP Server or Hyperion Essbase, this is set in EIS by the Essbase administrator.

To define your own window properties for displaying relational details or to bring up your custom JSP, use the drillThroughWindow tag. For details on how this works, see the Drillthrough Support for Microsoft Analysis Services section in the *Developer's Guide*. A live example is available in the Retrieving Data section in Blox Sampler.

See Also

"drillThroughWindow" on page 570. For detail on RDBResultSetDataBlox and ReportBlox, see the *Relational Reporting Developer's Guide*.

drillThroughWindow

Specifies the properties of the popped up browser window when a drillthrough operation is triggered on the GridBlox user interface.

Data Sources

IBM DB2 OLAP Server, Hyperion Essbase, Microsoft Analysis Services

Syntax

JSP Tag Attribute

```
drillThroughWindow="drillThroughWindowProperties"
```

or

```
<blox:drillThroughWindow
    url=""
    name=""
    height=""
    width=""
    resizable=""
    locationbarVisible=""
    menubarVisible=""
    scrollbarsVisible=""
    statusbarVisible=""
    toolbarVisible=""
>
</blox:drillThroughWindow>
```

Java Methods

```
String getDrillThroughWindow();
     // throws ServerBloxException
void setDrillThroughWindow(String drillThroughWindowProperties);
   // throws InvalidBloxPropertyValueException, ServerBloxException
```

where:

Argument	Description
`drillThroughWindowProperties`	A string containing a comma-delimited name-value pair that specifies the window properties. Valid names in the string are the following tag attributes for the `drillThroughWindow` tag: • `url`: a String containing the URL of the JSP page to load into the popped-up window • `name`: a String containing the name of the popped-up window • `height`: the height (in pixels) of the popped-up window • `width`: the width (in pixels) of the popped-up window • `resizable`: `true` or `false`; whether the popped-up window is resizable. The default is `true`. • `locationbarVisible`: `true` or `false`; whether the location bar should be visible in the popped-up window. The default is `true`. • `menubarVisible`: `true` or `false`; whether the menubar should be visible in the popped-up window. The default is `true`. • `scrollbarsVisible`: `true` or `false`; whether the scroll bars should be visible in the popped-up window. The default is `true`.

- `statusbarVisible`: `true` or `false`; whether the status bar should be visible in the popped-up window. The default is `true`.
- `toolbarVisible`: `true` or `false`; whether the toolbar (the browser's toolbar) should be visible in the popped up window. The default is `true`.

Usage

For IBM DB2 OLAP Server, IBM DB2 OLAP Server Deployment Services, Hyperion Essbase, or Essbase Deployment Services, this is for data sources which have drillthrough reports set up through IBM DB2 OLAP Server Integration Services or Essbase Integration Services.

When `drillThroughEnabled` is set to `true` on a GridBlox, by default DB2 Alphablox sends the coordinates of the cell where the drillthrough operation is initiated to a RDBResultSetDataBlox and renders the relational detail using a ReportBlox. The report will be displayed in a popped-up browser window. This popped up browser window, by default, is resizable, with its toolbar, scroll bars, menubar, status bar, and location bar visible.

If you want to specify your own window properties for the popped-up browser window, specify a comma delimited name/value pair string representing the url, name, and/or features of the drillthrough window. The window properties are similar to those available to JavaScript's window object.

The URL you specify can be of one of the following formats:

- 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. Note that 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.

Examples

```
drillThroughWindow =
"url=myDrillThroughPage.jsp,name=myDrillThroughWindowName,height=600,
width=800,statusbarVisible=false, locationbarVisible=false"

setDrillThroughWindow("url=myDrillThroughPage.jsp,
name=myDrillThroughWindowName,height=600,width=800,
statusbarVisible=false, locationbarVisible=false");
```

See Also

"drillThroughEnabled" on page 569

editableCellStyle

Specifies the foreground and background colors of editable cells.

Data Sources

All

Syntax

JSP Tag Attribute

```
editableCellStyle="style"
```

or

```
<blox:editableCellStyle
    background=""
    font=""
    foreground="" >
</blox:editableCellStyle>
```

Java Methods

```
String getEditableCellStyle();
boolean setEditableCellStyle(String style);
```

where:

Argument	Default	Description
`style`	`background=white, foreground=blue`	A comma-delimited string specifying: `foreground`: the cell's text color `background`: the cell's background color `font`: the *font name:style:point* to use Use the color's name or hexadecimal value.

Usage

For font, you can specify the font name, style to use, and the point size using the following syntax:

font name:style:point

- *font name*: Acceptable font name values vary widely by browser and client machine. The following font names are generally accepted: Arial, Courier, Helvetica, TimesRoman, SansSerif, Serif, Monospace.
- *style*: Valid font styles are: `plain`, `italic`, `bold`, and `bolditalic`
- *point*: An integer for point size (usually 8-36).

If any of the three attributes is not specified, the default or the currently inherited font value is applied. However, the colons separating the attributes should be included. The following examples show how to specify the font using the JSP tags:

```
font="Arial:bolditalic:12"
font=":Bold:12"
```

Examples

```
getEditableCellStyle();
setEditableCellStyle("background=red, foreground=green, font=Arial:bold:12");
```

See Also

"editedCellStyle" on page 573, "cellEditor" on page 555

editedCellStyle

Specifies the foreground and background colors of cells that have been edited.

Data Sources

All

Syntax

JSP Tag Attribute

```
editedCellStyle="style"
```

or

```
<blox:editedCellStyle
    background=""
    font=""
    foreground="">
</blox:editedCellStyle>
```

Java Methods

```
String  getEditedCellStyle();
boolean setEditedCellStyle(String style);
```

where:

Argument	Default	Description
`style`	`background=white, foreground=blue`	A comma-delimited string specifying: `foreground`: the cell's text color `background`: the cell's background color `font`: the `font name:style:point` to use Use the color's name or hexadecimal value.

Usage

This property specifies the colors for editable cells after the user has changed a value. Specifying a different color for changed cells provides visual cues for a user who is editing many cells.

For font, you can specify the font name, style to use, and the point size using the following syntax:

font name:style:point

- *font name*: Acceptable font name values vary widely by browser and client machine. The following font names are generally accepted: Arial, Courier, Helvetica, TimesRoman, SansSerif, Serif, Monospace.
- *style*: Valid font styles are: `plain`, `italic`, `bold`, and `bolditalic`
- *point*: An integer for point size (usually 8-36).

If any of the three attributes is not specified, the default or the currently inherited font value is applied. However, the colons separating the attributes should be included. The following examples show how to specify the font using the JSP tags:

```
font="Arial:bolditalic:12"
font=":Bold:12"
```

Examples

```
getEditedCellStyle();
setEditedCellStyle("background=gray, foreground=orange,
font=Helvetica:plain:12");
```

See Also

"editableCellStyle" on page 572, "cellEditor" on page 555

enablePoppedOut

This is a common Blox property. If the GridBlox is nested within a PresentBlox:

- If the `poppedOut` property and its related properties have been specified in the PresentBlox, the settings in the PresentBlox are used.
- If the `poppedOut` property and its related properties have not been specified in the PresentBlox, the popped out settings in the nested GridBlox are applied to the PresentBlox.

For a complete description, see "enablePoppedOut" on page 313.

expandCollapseMode

Specifies whether the grid should display the expand and collapse (plus and minus) signs on members.

Data Sources

All

Syntax

JSP Tag Attribute

```
expandCollapseMode="expandCollapseMode"
```

Java Methods

```
boolean isExpandCollapseMode();
void setExpandCollapseMode(boolean expandCollapseMode);
```

where:

Argument	Default	Description
`expandCollapseMode`	`false`	Set to `true` to enable expand and collapse; `false` to disable it.

Usage

When `expandCollapseMode` is set to `true`, plus and minus signs are displayed to indicate a drill up or drill down operation in the user interface of GridBlox. Note that if you want parent members to come first before their children, you should set DataBlox `parentFirst` property to `true` rather than do so through the query. This is to ensure the expand/collapse mode can search through the result set correctly to determine the base members and shared members.

Examples

```
getExpandCollapseMode();
setExpandCollapseMode(true);
```

formatMask

Specifies a predefined format mask for cells when using the format mask user interface.

Data Sources

All

Syntax

JSP Tag Attribute

```
<blox:formatMask
       index="maskNumber"
       mask="mask"
/>
```

Java Methods

```
String getFormatMask(int index);
void setFormatMask(int index, String mask);
```

where:

Argument	Default	Description
index	null	The index number of the mask to define or retrieve. Must be an integer between 1 and 15.
mask	empty string	String that defines formatting attributes.

Usage

Unlike defaultCellFormat or cellFormat, this property has no effect on the grid itself, it only effects what appears in the Apply Format Mask dialog.

The following table lists the predefined mask values, and their associated format names, for each mask. The format names and masks may be different for language versions other than English. You can create your own number masks, beginning with 12, in addition to the predefined ones.

Format Mask number	Mask	Format Name
formatMask1		No mask
formatMask2	#,##0.00;[red]-#,##0.00	Negative red
formatMask3	#,##0.00;[red](#,##0.00)	Negative red parenthesis
formatMask4	#,##0.00;(#,##0.00)	Parenthesis
formatMask5	#,###K	Thousands
formatMask6	#,###M	Millions
formatMask7	##0.00'%'	Percentage
formatMask8	$,##0	Dollars
formatMask9	#,##0.00;(#,##0.00)	Euros
formatMask10	#,##0.00	2 decimal places
formatMask11	0	Integer

Note the following:

- Do not use the backslash escape character (\) in the value string.
- The `setFormatMask()` method returns `false` if the index property is out of range (1 to 15).
- You can use a slash (/) character to divide the one value by another ($#,###/1000, for example).

Examples

```
getFormatMask(7);
setFormatMask(3, "#,##0.00;[red]-#,##0.00");
```

This property is used in conjunction with the formatName1-15 property. For example, the following two lines would allow the user to select a format name called “Negative red” with an associated number mask of #,##0.00;[red]-#,##0.00 from the format mask user interface.

```
<blox:formatMask
               index="2"
                mask="#,##0.00;[red]-#,##0.00" />
<blox:formatName
               index="2"
                name="Negative red" />
```

See Also

“formatName” on page 577

formatName

Specifies a predefined format name for cells when using the format mask user interface.

Data Sources

All

Syntax

JSP Tag Attribute

```
<blox:formatName
       index="formatNumber"
       name="name"
/>
```

Java Methods

```
String getFormatName(int index);
void setFormatName(int index, String name);
```

where:

Argument	Default	Description
index	null	The index number of the format name to define or retrieve. Must be an integer between 1 and 15.
name	empty string	String defining the name of the specified format mask

Usage

Every predefined formatMask property is assigned a predefined formatName. The property name is one of formatName1 through formatName15.

The following table lists the format name properties, their predefined names, and their associated format masks. The format names and number mask syntax may be different for language versions other than English.

Format Name property	Format Name	Format Mask
formatName1	No mask	
formatName2	Negative red	#,##0.00;[red]-#,##0.00
formatName3	Negative red parenthesis	#,##0.00;[red](#,##0.00)
formatName4	Parenthesis	#,##0.00;(#,##0.00)
formatName5	Thousands	#,###K
formatName6	Millions	#,###M

Format Name property	Format Name	Format Mask
formatName7	Percentage	##0.00'%'
formatName8	Dollars	$,##0
formatName9	Euros	#,##0.00;(#,##0.00)
formatName10	2 decimal places	#,##0.00
formatName11	Integer	0

Examples

```
getFormatName(9);
setFormatName(12, "Format description");
```

The following two tags allow the user to select a format name called "Negative red" with an associated number mask of #,##0.00;[red]-#,##0.00 from the format mask user interface.

```
<blox:formatMask
               index="2"
                mask="#,##0.00;[red]-#,##0.00" />
<blox:formatName
               index="2"
                name="Negative red" />
```

See Also

"formatMask" on page 575

gridLinesVisible

Specifies whether lines appear between each cell in the grid.

Data Sources

All

Syntax

JSP Tag Attributes

```
gridLinesVisible="enabled"
```

Java Methods

```
boolean isGridLinesVisible();
void setGridLinesVisible(boolean visible);
```

where:

Argument	Default	Description
visible	true	Specify true to display grid lines; false to hide them.

Examples

```
getGridLinesVisible();
setGridLinesVisible(false);
```

headingIconsVisible

Specifies whether the clickable up/down arrow icons appear on row or column headings.

Data Sources
Multidimensional

Syntax
JSP Tag Attribute

```
headingIconsVisible="visible"
```

Java Methods

```
boolean isHeadingIconsVisible();
void setHeadingIconsVisible(boolean visible);
```

where:

Argument	Default	Description
visible	true	Specify true to display icons; false to hide them.

Usage
Clicking on one of these icons activates the collapse/expand or drill up/down features.

Examples

```
isHeadingIconsVisible();
setHeadingIconsVisible(false);
```

headingsEnabled
Specifies whether row and column headings appear when the grid is printed.

Data Sources
All

Syntax
JSP Tag Attribute

```
headingsEnabled="enable"
```

Java Methods

```
boolean isHeadingsEnabled();
void setHeadingsEnabled(boolean enable);
```

where:

Argument	Default	Description
enable	true	Specify true to display headings; false to hide them.

Examples

```
isHeadingsEnabled();
setHeadingsEnabled(false);
```

height
This is a common Blox property. For a complete description, see “height” on page 38.

helpTargetFrame

This is a common Blox property. For a complete description, see "helpTargetFrame" on page 39.

htmlGridScrolling

Specifies whether to display scroll bars on the grid.

Data Sources

All

Syntax

JSP Tag Attribute

```
htmlGridScrolling="scroll"
```

Java Methods

```
boolean isHtmlGridScrolling();
void setHtmlGridScrolling(boolean scroll);
```

where:

Argument	Default	Description
scroll	true	Specify true to display scroll bars when necessary; false not to display them.

Usage

Setting this property to true causes the scroll bars to appear only when needed. If the display area can accommodate all the requested data or if the value is false, no scroll bars appear.

htmlShowFullTable

Specifies if all rows and columns in the grid to appear (ignoring the defined Blox area and the setting of the htmlGridScrolling property). Scrollbars are not part of the display. If the contents of the grid require it, the data may extend beyond the viewable area on the screen. In this case, HTML page scroll bars enable the user to scroll to and view off-screen data. The default is false, which causes the display to stay within the Blox bounds and not display the full table.

Data Sources

All

Syntax

JSP Tag Attribute

```
htmlShowFullTable="show"
```

Java Methods

```
boolean isHtmlShowFullTable();
void setHtmlShowFullTable(boolean show);
```

where:

Argument	Default	Description
show	false	Specify true to display all rows and columns on the grid; false to use scroll bars if they do not fit in the space.

informationWindowName

Specifies the name of the window used for displaying HTML pages defined in the Header Links for a particular application.

Data Sources

All

Syntax

JSP Tag Attribute

```
informationWindowName="name"
```

Java Methods

```
String getInformationWindowName();
boolean setInformationWindowName(String name);
```

where:

Argument	Default	Description
name	"Information"	String representing the window name.

Usage

By defining a window name using this property, all header link URLs will be opened in the defined window, rather than opening new windows for each URL.

localeCode

This is a common Blox property. For a complete description, see “localeCode” on page 40.

maximumUndoSteps

This is a common Blox property. For a complete description, see “maximumUndoSteps” on page 41.

menubarVisible

This is a common Blox property. For a complete description, see “menubarVisible” on page 42.

missingValueString

Specifies a string to display in a cell for which there is no data in the database.

Data Sources

All

Syntax

JSP Tag Attribute

```
missingValueString="value"
```

Java Methods

```
String getMissingValueString();
void setMissingValueString(String value);
```

where:

Argument	Default	Description
value	empty string	Any string

Usage

When accessing relational data sources, the message appears when a cell has a null value.

Examples

```
getMissingValueString();
setMissingValueString("Data is missing");
```

See Also

"noAccessValueString" on page 582

noAccessValueString

Specifies the string to display in a grid cell for which the user has not been granted data access.

Data Sources

Multidimensional

Syntax

JSP Tag Attribute

```
noAccessValueString="value"
```

Java Methods

```
String getNoAccessValueString();
void setNoAccessValueString(String value);
```

where:

Argument	Default	Description
value	#NoAccess	Any string

Examples

```
getNoAccessValueString();
setNoAccessValueString("Access denied");
```

See Also

missingValueString

noDataMessage

This is a common Blox property. For a complete description, see "noDataMessage" on page 42.

poppedOut

This is a property inherited from ContainerBlox. If the GridBlox is nested within a PresentBlox:

- If the `poppedOut` property and its related properties have been specified in the PresentBlox, the settings in the PresentBlox are used.
- If the `poppedOut` property and its related properties have not been specified in the PresentBlox, the popped out settings in the nested GridBlox are applied to the PresentBlox.

For a complete description, see "poppedOut" on page 314.

poppedOutHeight

This is a property inherited from ContainerBlox. For a complete description, see "poppedOutHeight" on page 315.

poppedOutTitle

This is a property inherited from ContainerBlox. For a complete description, see "poppedOutTitle" on page 315.

poppedOutWidth

This is a property inherited from ContainerBlox. For a complete description, see "poppedOutWidth" on page 316.

relationalRowNumbersOn

Specifies whether to display the row numbers from a relational data source.

Data Sources

Relational

Syntax

JSP Tag Attribute

```
relationalRowNumbersOn="enable"
```

Java Methods

```
boolean isRelationalRowNumbersOn();
void setRelationalRowNumbersOn(boolean enable);
```

where:

Argument	Default	Description
`enable`	`false`	Specify `true` to display row numbers; `false` to hide them.

Examples

```
getRelationalRowNumbersOn();
setRelationalRowNumbersOn(true);
```

See Also

"rowHeadingsVisible" on page 584

removeAction

This is a common Blox property. For a complete description, see “removeAction” on page 44.

render

This is a common Blox property. For a complete description, see “render” on page 45.

rightClickMenuEnabled

This is a common Blox property. For a complete description, see “rightClickMenuEnabled” on page 46.

rowHeadersWrapped

Specifies whether multi-word headings in grid row headers should be wrapped onto more than one line.

Data Sources

All

Syntax

JSP Tag Attribute

```
rowHeadersWrapped="wrapped"
```

Java Methods

```
boolean isRowHeadersWrapped();       // returns boolean
void setRowHeadersWrapped(boolean wrapped);
```

where:

Argument	Default	Description
`wrapped`	`false`	Specify `true` to enable wrapping of row headers.

Usage

When this property is set to `false` (the default), the row width is sized to fit the entire row header text without wrapping. When this property is set to `true`, the row header text will wrap to reduce the width of row header column.

See Also

“columnHeadersWrapped” on page 567

rowHeadingsVisible

Specifies whether the row headings to the left of the data values appear on the grid.

Data Sources

Multidimensional

Syntax

JSP Tag Attribute

```
rowHeadingsVisible="visible"
```

Java Methods

```
boolean isRowHeadingsVisible();
void setRowHeadingsVisible(boolean visible);
```

where:

Argument	Default	Description
visible	true	Specify true to display row headings; false to hide them.

Examples

```
getRowHeadingsVisible():
setRowHeadingsVisible(false);
```

See Also

"relationalRowNumbersOn" on page 583

rowHeadingWidths

Specifies the widths of the row headings on the grid.

Data Sources

All

Syntax

JSP Tag Attribute

```
rowHeadingWidths="widths"
```

Java Methods

```
String getRowHeadingWidths();   // throws ServerBloxException
void setRowHeadingWidths(String widths);
 // throws InvalidBloxPropertyValueException, ServerBloxException
```

where:

Argument	Default	Description
widths	none	A comma-separated list of integers, each representing the row headers in pixels.

Usage

The autosizeEnabled property needs to be false for this property to be used. If the width for a row heading is not specified, the width will be automatically calculated to fit the entire heading. If the heading is longer than the width specified, the grid may not display properly.

See Also

"autosizeEnabled" on page 547

rowHeight

Specifies the height (in pixels) for each row.

Data Sources

All

Syntax

JSP Tag Attribute

```
rowHeight="height"
```

Java Methods

```
int getRowHeight();
void setRowHeight(int height);
```

where:

Argument	Default	Description
height	-1	An integer representing the row height in pixels

Usage

The autosizeEnabled property needs to be set to false. The default (-1) sets the row height to an appropriate value for the selected font.

Examples

```
getRowHeight();
setRowHeight(15);
```

See Also

"autosizeEnabled" on page 547

rowIndentation

Specifies whether (and how) to indent row headings.

Data Sources

Multidimensional

Syntax

JSP Tag Attribute

```
rowIndentation="strType"
```

Java Methods

```
String getRowIndentation();
void setRowIndentation(String strType);
```

where:

Argument	Default	Description
strType	parentLeft	Possible values (case-sensitive): parentRight: The lowest-generation child appears farthest to the left, with each parent indented slightly to the right. parentLeft: The lowest-generation child appears farthest to the right, with each parent slightly to the left. none: No indentation.

Usage

Indenting row headings helps to indicate the dimension hierarchy. Returned string for getRowIndentation() are parentRight, parentLeft, or none.

Examples

```
getRowIndentation():
setRowIndentation("none");
```

See Also

"rowHeadingsVisible" on page 584

showColumnDataGeneration

Enables the use of generation styles for columns.

Data Sources

Multidimensional

Syntax

JSP Tag Attribute

```
showColumnDataGeneration="show"
```

Java Methods

```
boolean isShowColumnDataGeneration();
void setShowColumnDataGeneration(boolean show);
```

where:

Argument	Default	Description
show	false	Specify true to use generation styles; false not to use them.

Usage

The setShowColumnDataGeneration() and setShowRowDataGeneration() methods must be set to true to apply generation styles to the data cells.

The styles are all set from the theme currently in use. Therefore, you should set the styles in the theme to control row data generation styles (csClmnDtGnrtn0, csClmnDtGnrtn1,... csClmnDtGnrtn*N* classes). The stylesheet for the supported themes are in <alphablox>/repository/theme/{*themeName*}. Also, any column styles will override row styles in cells where the rows and columns intersect.

Examples

```
getShowColumnDataGeneration();
setShowColumnDataGeneration(true);
```

See Also

"showRowDataGeneration" on page 588

showColumnHeaderGeneration

Enables the use of generation styles for column headers.

Data Sources

Multidimensional

Syntax

JSP Tag Attribute

```
showColumnHeaderGeneration="show"
```

Java Methods

```
boolean isShowColumnHeaderGeneration();
void setShowColumnHeaderGeneration(boolean show);
```

where:

Argument	Default	Description
show	false	Specify true to use generation styles; false not to use them.

Usage

When the value is set to true, a predefined style for each data generation is applied to the header text. To customize the style, modify the csClmnHdrGnrtn0, csClmnHdrGnrtn1,... csClmnHdrGnrtn*N* classes in the underlying theme. The stylesheet for the supported themes are in <alphablox>/repository/theme/{*themeName*}.

See Also

"showRowHeaderGeneration" on page 589

showRowDataGeneration

Enables the use of generation styles for rows.

Data Sources

Multidimensional

Syntax

JSP Tag Attribute

```
showRowDataGeneration="show"
```

Java Methods

```
boolean isShowRowDataGeneration();
void setShowRowDataGeneration(boolean show);
```

where:

Argument	Default	Description
show	false	Specify true to use generation styles; false not to use them.

Usage

The setShowRowDataGeneration() and setShowColumnDataGeneration() methods must be set to true to apply generation styles to the data cells.

The styles are all set from the theme currently in use. Therefore, you should set the styles in the underlying theme to control row data generation styles (csRwDtGnrtn0, csRwDtGnrtn1,... csRwDtGnrtn*N* style classes). The stylesheet for the supported themes are in <alphablox>/repository/theme/{*themeName*}. Also, any column styles will override row styles in cells where the rows and columns intersect.

Examples

```
getShowRowDataGeneration();
setShowRowDataGeneration(true);
```

See Also

"showColumnDataGeneration" on page 587

showRowHeaderGeneration

Enables the use of generation styles for row headers.

Data Sources

Multidimensional

Syntax

JSP Tag Attribute

```
showRowHeaderGeneration=show
```

Java

```
boolean isShowRowHeaderGeneration();
void setShowRowHeaderGeneration(boolean show);
```

where:

Argument	Default	Description
show	false	Specify true to use generation styles; false not to use them.

Usage

When the value is set to true, a predefined style for each data generation is applied to the header text. To customize the style, in the DHTML client, modify the csRwHdrGnrtn0, csRwHdrGnrtn1,... csRwHdrGnrtn*N* classes in the underlying theme. The stylesheet for the supported themes are in <alphablox>/repository/theme/{*themeName*}.

See Also

"showColumnHeaderGeneration" on page 587

toolbarVisible

Specifies if the toolbar is visible.

Data Sources

All

Syntax

JSP Tag Attribute

```
toolbarVisible="visible"
```

where:

Argument	Default	Description
visible	true	true: the toolbar is visible; false: the toolbar is not visible.

Usage

By default, the toolbar is visible in a standalone GridBlox. If a nested <blox:toolbar> tag is added, its setting overwrites the value of this attribute. For example, the following code will result in a visible toolbar.

```
<blox:grid id="myGrid" toolbarVisible="false" ....>
   <blox:toolbar visible="true" />
   <blox:data bloxRef="myDataBlox"/>
</blox:grid>
```

Tip: `toolbarVisible` is only a tag attribute, not a property.

visible

This is a common Blox property. For a complete description, see "visible" on page 46.

width

This is a common Blox property. For a complete description, see "width" on page 47.

writebackEnabled

Permits users to edit cells in the grid.

Data Sources

All

Syntax

JSP Tag Attribute

```
writebackEnabled="enabled"
```

Java Methods

```
boolean isWritebackEnabled();
void setWritebackEnabled(boolean enable);
```

where:

Argument	Default	Description
`enable`	`false`	Specify `true` to enable writeback; `false` to disable it.

Usage

This method must be specified on GridBlox for the associated grid writeback properties and methods to take effect.

Examples

```
getWritebackEnabled();
setWritebackEnabled(true);
```

See Also

"updateProperties()" on page 596, "cellEditor" on page 555

GridBlox Methods

This section describes GridBlox methods that are not associated with a specific property. For the syntax and descriptions of GridBlox methods that have a property associated with them, see "GridBlox Properties and Associated Methods" on page 546. For client-side API common to Blox, see "Client-Side APIs" on page 31.

addEventFilter()

This is a common Blox method that for capturing a server-side event (such as saving and loading bookmarks) and perform custom actions *after* the operation is complete on the server. For details, see "addEventListener()" on page 49.

addEventListener()

This is a common Blox method that allows you to capture a server-side event (such as saving and loading bookmarks) and perform custom actions *after* the operation is complete on the server. For details, see “addEventListener()” on page 49.

call()

This is a common client-side Blox method. For a complete description, see “call()” on page 51.

clearCellAlerts()

Removes all defined cell alerts.

Data Sources

All

Syntax

Java Method

```
void clearCellAlerts();
```

Usage

You define cell alerts using the cellAlert property. You can define an unlimited number of cell alerts. To remove a specific cell alert, use the `cellAlert` property to set it to an empty string. To remove all cell alerts in one step, use `clearCellAlerts()`.

Examples

```
clearCellAlerts();
```

See Also

“cellAlert” on page 548, “Cell Alerts” on page 544

clearCellEditors()

Removes all defined cell editors.

Data Sources

All

Syntax

Java Method

```
void clearCellEditors();
```

Usage

You define cell editors using the cellEditor property. You can define an unlimited number of cell editors. To remove a specific cell editor, use the cellEditor property to set it to an empty string. To remove all cell editors in one step, use `clearCellEditors()`.

Examples

```
clearCellEditors();
```

See Also

“cellEditor” on page 555, “writebackEnabled” on page 590, “Grid UI for Writeback and Comments” on page 545

clearCellFormats()

Removes all defined cell formats.

Data Sources
All

Syntax
Java Method

```
void clearCellFormats();
```

Usage
You define cell formats using the cellFormat property. You can define an unlimited number of cell formats. To remove a specific cell format, use the cellFormat property to set it to an empty string. To remove all cell formats in one step, use `clearCellFormats()`.

Examples

```
clearCellFormats();
```

See Also
"cellFormat" on page 558

flushProperties()

This is a common client-side Blox method. For a complete description, see "flushProperties()" on page 52.

getChangedCellList()

Returns a String of edited cells.

Data Sources
All

Syntax
Java Method

```
String getChangedCellList();
```

Usage
Use this method and the getChangedCellValues() method as arguments to the DataBlox `writeback()` method. The result is that the String of edited cells and the corresponding String of cell values is available for updating the underlying data source.

Examples

```
getChangedCellList();
```

The following example shows typical usage:

```
gridBlox.getDataBlox().writeback(gridBlox.getChangedCellList(),
gridBlox.getChangedCellValues(),"" );
```

See Also
"getChangedCellValues()" on page 593; the Inputting and Modifying Data section in the *Developer's Guide* and the corresponding examples in Blox Sampler (DHTML).

getChangedCellValues()

Returns a String of edited cell values.

Data Sources

All

Syntax

Java Method

```
String getChangedCellValues();
```

Usage

Use this method and the getChangedCellList() method as arguments to the DataBlox `writeback` method.

Examples

```
getChangedCellValues();
```

The following example shows typical usage:

```
gridBlox.getDataBlox().writeback(gridBlox.getChangedCellList(),
gridBlox.getChangedCellValues(),"" );
```

See Also

"getChangedCellList()" on page 592; the Inputting and Modifying Data section in the *Developer's Guide* and the corresponding examples in Blox Sampler.

getDataBlox()

This is a common Blox method. For a complete description, see "setDataBlox()" on page 60.

isAlertEnabled()
setAlertEnabled()

Specifies whether a cell alert defined using `cellAlert`*N* is enabled or disabled.

Data Sources

All

Syntax

Java Methods

```
boolean isAlertEnabled(int ID);
void    setAlertEnabled(int ID, boolean enable);
```

where:

Argument	Default	Description
`ID`	null	A positive integer representing the number of the cell alert to enable or disable.
`enable`	`true`	Specify `true` to enable the cell alert; `false` to disable it.

Usage
Use `set/getAlertEnabled` to temporarily disable a cell alert without having to delete it. The `ID` for the method is a positive integer corresponding to the `index` of the cell alert you want to control. For example, to disable a cellAlert defined with `index="4"`:

```
myGridBlox.setAlertEnabled(4,true);
```

The `isAlertEnabled()` method returns a boolean value indicating whether the cell alert specified by `ID` is enabled or disabled.

To specify whether a cell alert is enabled using the Blox Tag Library, use the `enabled` attribute of the `cellAlert` tag.

See Also
"cellAlert" on page 548, “Cell Alerts” on page 544

listCellAlertIds()
Returns a list of IDs of all the cell alerts defined as an array of integers.

Data Sources
All

Syntax
Java Method

```
int[] listCellAlertIds();    //throws ServerBloxException
```

Usage
To get the associated cell alert rule for a specific cell alert ID, use the `getCellAlert()` method. To identify if the cell alert for a specific ID is enabled, use the `isAlertEnabled()` method.

listCellEditorIds()
Returns a list of IDs of all the cell editors defined as an array of integers.

Data Sources
All

Syntax
Java Method

```
int[] listCellEditorIds();    //throws ServerBloxException
```

Usage
To get the associated cell editor rule for a specific cell editor ID, use the `getCellEditor()` method.

listCellFormatIds()
Returns a list of IDs of all the cell format masks defined as an array of integers.

Data Sources
All

Syntax
Java Method

```
int[] listCellFormatIds();    //throws ServerBloxException
```

Usage

To get the associated cell format rule for a specific cell format ID, use the `getCellFormat()` method.

listCellLinkIds()

Returns a list of IDs of all of the cell links defined as an array of integers.

Data Sources

All

Syntax

Java Method

```
int[] listCellLinkIds();    //throws ServerBloxException
```

Usage

To get the associated cell link rule for a specific cell link ID, use the `getCellLink()` method.

loadBookmark()

This is a common Blox method. For a complete description, see "loadBookmark()" on page 54.

removeEventFilter()

This is a common Blox method that allows you to remove an event filter object added using addEventFilter() for capturing a server-side event (such as saving and loading a bookmark) *before* the event is processed on the server. For details, see "removeEventFilter()" on page 55.

removeEventListener()

This is a common Blox method that allows you to remove an event listener object created using addEventListener() for capturing a server-side event (such as saving and loading a bookmark) *after* that operation is complete on the server. For details, see "removeEventListener()" on page 56.

saveBookmark()

This is a common Blox method. For a complete description, see "saveBookmark()" on page 57.

saveBookmarkHidden()

This is a common Blox method. For a complete description, see "saveBookmarkHidden()" on page 58.

setAlertEnabled

For a description of this method, see "isAlertEnabled() setAlertEnabled()" on page 593.

setDataBusy()

This is a common client-side Blox method. For a complete description, see "setDataBusy()" on page 60.

setDataBlox()

This is a common Blox method. For a complete description, see “setDataBlox()” on page 60.

updateProperties()

This is a common client-side Blox method. For a complete description, see “updateProperties()” on page 61.

Chapter 16. JDBCConnection Bean Reference

This chapter describes the JDBCConnection bean, which is a Java Bean that allows you to construct JDBC connection strings from DB2 Alphablox JDBC data sources.

JDBCConnection Bean Overview

The JDBCConnection bean is a Java bean that allows you to get information about an DB2 Alphablox relational data source. Through the JDBCConnection bean you can get the JDBC URL connection string and perform JDBC calls without creating a Blox.

Additionally, you can use this bean to override properties of a relational (JDBC) data source defined in DB2 Alphablox.

The JDBCConnection bean is a class in the `com.alphablox.blox.data.rdb` package, and you must use the following JSP import statement at the beginning of any JSP file that uses any of the APIs in this bean:

```
<%@ page import="com.alphablox.blox.data.rdb.*" %>
```

JDBCConnection Bean JSP useBean Examples

The following is a sample JSP file that uses the JDBCConnection bean to print out the JDBC URL connection string.

```
<%@ page import="com.alphablox.blox.data.rdb.*" %>
<%@ page import="java.sql.*" %>
<%@ page import="java.io.*" %>

<html>
<head>
<title>JDBC Connection Bean Example</title>
</head>

<body>

<%
   String ds = (String)request.getParameter( "ds" ) ;
%>

<form name=form method=get>
Enter data source name: 
<input name="ds" value="<%= ds == null ? "" : ds %>"><br />
<input type=submit value="Go"><br />
</form>

<%-- Create the Bean --%>
<jsp:useBean id="jbean"
   class="com.alphablox.blox.data.rdb.JDBCConnection"
   scope="session" />
```

```
<%-- Put in try statement to catch errors --%>
<% try { %>

<%--Test if there is a data source --%>
<% if ( ds != null ) { %>

<%
jbean.setDataSourceName( ds );
%>

<%-- Use the Alphablox bean to get the connection JDBC string --%>
<%= "URL = " + jbean.getURL() %><br />
Properties = <%= jbean.getConnectionProperties( ) %><br />
<%
Connection connection = jbean.createConnection( );
%>
Connection = <%= connection  %><br />
<br />

<%-- If no data source, prompt for one --%>
<% } else { %>
<br />
<b>Please enter a relational data source name!</b>
<br />
<% } %>
<%-- Catch the exception --%>
<% } catch ( Exception e ) {
   out.write( "<br />An error has occured: <b>"
      + e.getMessage() + "</b>" ); } %>
</body>
</html>
```

JDBCConnection Bean Properties and Methods By Category

The following properties and methods are available on the JDBCConnection bean. Note that:

- These properties and the associated get and set methods are local to the JDBCConnection bean and override any properties set in the DB2 Alphablox data source definition.
- You need to import the following package in your JSP in order to use this bean:

  ```
  <%@ page import="com.alphablox.blox.data.rdb.*" %>
  ```

Properties	Methods
catalog	
	getCatalog() setCatalog()
dataSourceName	
	getDataSourceName() setDataSourceName()
password	
	getPassword() setPassord()
schema	
	getSchema() setSchema()
userName	
	getUserName() setUserName()

```
closeConnection()
createConnection()


getConnection()
getConnectionProperties()


getURL()
```

JDBCConnection Bean Properties and Associated Methods

This section describes the properties supported by the JDBCConnection bean and the methods associated with those properties. The properties are listed alphabetically by property name. For a list of JDBCConnection bean methods with which no properties are associated, see "JDBCConnection Bean Methods" on page 601.

catalog

Specifies the catalog to be used by the JDBC connection, overriding the setting in the DB2 Alphablox data source definition.

Data Sources

Relational

Syntax

Java Methods

```
String getCatalog();
void setCatalog(String catalog);
```

where:

Argument	Default	Description
`catalog`	`null`	Specifies the catalog.

dataSourceName

Specifies the name of the DB2 Alphablox data source definition.

Data Sources

Relational

Syntax

Java Methods

```
String getDataSourceName();
void setDataSourceName(String dataSourceName);
```

where:

Argument	Default	Description
`dataSourceName`	`null`	Specifies the data source name for an DB2 Alphablox data source.

password

Specifies the password used by the JDBC connection, overriding the setting in the DB2 Alphablox data source definition.

Data Sources

Relational

Syntax

Java Methods

```
String getPassword();
void setPassword(String password);
```

where:

Argument	Default	Description
password	null	Specifies the password defined in the JDBC data source.

schema

Specifies the schema used by the JDBC connection, overriding the setting in the DB2 Alphablox data source definition.

Data Sources

Relational

Syntax

Java Methods

```
String getSchema();
void setSchema(String schema);
```

where:

Argument	Default	Description
schema	null	Specifies the schema set by the JDBCConnection Bean.

userName

Specifies the user name used by the JDBC connection, overriding the setting in the DB2 Alphablox data source definition.

Data Sources

Relational

Syntax

Java Methods

```
String getUserName();
void setUserName(String userName);
```

where:

Argument	Default	Description
userName	null	Name of a user defined in the JDBC data source.

JDBCConnection Bean Methods

This section describes JDBCConnection bean methods that are not associated with a specific property. For the syntax and descriptions of JDBCConnection bean methods that have a property associated with them, see "JDBCConnection Bean Properties and Associated Methods" on page 599.

closeConnection()

Closes the JDBC connection.

Data Sources

Relational

Syntax

Java Method

```
void closeConnection();    //throws java.sql.SQLException
```

createConnection()

Returns a new JDBC connection.

Data Sources

Relational

Syntax

Java Method

```
java.sql.Connection createConnection();
```

Usage

This is a convenience method that returns a new JDBC connection. The caller is responsible for closing the connection. The connections are not associated with JDBC connections used by the DB2 Alphablox data manager and the bean does not track these connections.

Examples

See "JDBCConnection Bean JSP useBean Examples" on page 597.

getConnection()

Gets the JDBC Connection object.

Data Sources

Relational

Syntax

Java Method

```
java.sql.Connection getConnection();
      // throws com.alphablox.util.DataException
```

getConnectionProperties()

Returns the JDBC Connection properties.

Data Sources

Relational

Syntax

Java Method

```
java.util.Properties getConnectionProperties();
```

Examples

See "JDBCConnection Bean JSP useBean Examples" on page 597.

getURL()

Returns the JDBC connection URL string.

Data Sources

Relational

Syntax

Java Method

```
String getURL();
```

Chapter 17. MemberFilterBlox Reference

This chapter contains reference material for MemberFilterBlox properties, methods and objects. For general reference information about Blox, see Chapter 3, "General Blox Reference Information," on page 17. For information on how to use this reference, see Chapter 1, "Using This Reference," on page 1.

MemberFilterBlox Overview

MemberFilterBlox allows you to present the Member Filter dialog for users to select members. Member Filter is built into the Blox user interface, available to users when they:

- select Member Filter... from a GridBlox's right-click menu,
- select Member Filter... from the drop down lists in the Data Layout panel, or
- select More... from the Page panel.

With MemberFilterBlox, you can specify a DataBlox to use and then put a Member Filter dialog on the page to allow your users to select members from all available dimensions or just the dimensions you specified. This dimension selection drop list is populated based on the data query for the underlying DataBlox.

If the underlying DataBlox is used in a PresentBlox on the same page, the PresentBlox will automatically reflect the selections made.

By default, the `dimensionSelectionEnabled` property is set to `true`, making all available dimensions as a result of the data query appear in the list. These dimensions are listed in alphabetical order. Unless specified otherwise, the first dimension in the list is the initial selected dimension and it will appear in the Dimension Hierarchy panel on the left. To specify the initial selected dimension, use the selectedDimension property. To limit the dimensions you want to appear in the drop list, you can specify the list of dimensions using the selectableDimensions property.

MemberFilterBlox JSP Custom Tag Syntax

The Alphablox Tag Libraries provides custom tags to use in a JSP page for creating each Blox. This section describes how to create the custom tag to create a MemberFilterBlox. For a copy and paste version of the tag with all the attributes, see "MemberFilterBlox JSP Custom Tag" on page 888.

Syntax

```
<blox:memberFilter
    [attribute="value"] >
    <blox:data bloxRef="" />
</blox:memberFilter>
```

where:

attribute is one of the attributes listed in the attribute table.

value is a valid value for the attribute.

and where the attributes are one of the following:

Attribute
id
applyButtonEnabled
bloxEnabled
bloxName
dimensionSelectionEnabled
height
selectableDimensions
selectedDimension
visible
width

Usage

Each custom tag can have one or more attributes, separated by one or more space or new line characters. Any extra space or new line characters are ignored. For readability, attributes can each go on a separate line with the same indenting.

You can substitute the closing `</blox:memberFilter>` tag with a closing slash (/) after the last attribute in the tag but before the closing greater than character. For example, if the last attribute is `width`, the end of the tag looks as follows:

```
selectedDimension="All Products" />
```

Examples

```
<blox:data id="myDataBlox"
    dataSourceName="QCC-Essbase"
    query="!" />

<blox:memberFilter id="myMemberFilter">
    <blox:data bloxRef="myDataBlox" />
</blox:memberFilter>
```

MemberFilterBlox Examples

This section provides examples that demonstrate how MemberFilterBlox can be used as a utility to filter members for all available dimensions in a PresentBlox, to filter members for only the specified dimensions, or to filter only one dimension.

Example 1: Filtering Members for All Available Dimensions

This example adds a MemberFilterBlox on the same page as the PresentBlox using the same DataBlox.

1. A MemberFilterBlox is added to the page using the `<blox:memberFilter>` tag.
2. The dimension selection drop list is enabled. Since no `selectableDimensions` are specified, all dimensions available from the DataBlox will appear in the drop list.
3. The initial selected dimension in the drop list is set to All Time Periods.
4. Use the `bloxRef` tag attribute to specify the underlying DataBlox.
5. The same DataBlox is used in a PresentBlox. The selections the users make in MemberFilterBlox will be automatically reflected in the PresentBlox.

```
      <%@ taglib uri="bloxtld" prefix="blox"%>

      <blox:data id="myDataBlox"
          dataSourceName="QCC-Essbase"
          useAliases="true"
          query="<ROW (\"All Products\") <ICHILD \"All Products\"
                 <COLUMN (\"All Time Periods\" Measures Scenario)
                 <CHILD \"All Time Periods\" !"
          selectableSlicerDimensions="All Locations" />

      <html>
      <head>
          <blox:header />
      </head>
      <body>
(1)   <blox:memberFilter id="memberFilterBlox"
(2)       dimensionSelectionEnabled = "true"
(3)       selectedDimension="All Time Periods">
(4)       <blox:data bloxRef="myDataBlox" />
      </blox:memberFilter>
      <br>
      <blox:present id="myPresentBlox" width="600" height="400">
(5)       <blox:data bloxRef="myDataBlox" />
      </blox:present>
      </body>
      </html>
```

Example 2: Filtering Members for Specified Dimensions Only

This example adds a MemberFilterBlox that only has the All Products and All Time Periods dimensions on the dimension selection drop list using the `selectableDimensions` tag attribute.

```
<%@ taglib uri="bloxtld" prefix="blox"%>

<blox:data id="myDataBlox"
   dataSourceName="QCC-Essbase"
   useAliases="true"
   query="<ROW (\"All Products\") <ICHILD \"All Products\"
      <COLUMN (\"All Time Periods\" Measures Scenario)
      <CHILD \"All Time Periods\" !" />
...

<blox:memberFilter id="memberFilterBlox"
   dimensionSelectionEnabled="true"
   selectableDimensions="All Products, All Time Periods">
   <blox:data bloxRef="myDataBlox" />
</blox:memberFilter>
...
```

Example 3: Filtering Members for One Dimension Only

This examples adds a MemberFilterBlox that does not have a dimension selection drop list (dimensionSelectionEnabled = "false"). With selectedDimension set to All Products, All Products will show up in the left Dimension Hierarchy panel. All the users can do is to select members from this dimension.

```
<%@ taglib uri="bloxtld" prefix="blox"%>

<blox:data id="myDataBlox"
   dataSourceName="QCC-Essbase"
   useAliases="true"
   query="<ROW (\"All Products\") <ICHILD \"All Products\"
      <COLUMN (\"All Time Periods\" Measures Scenario)
      <CHILD \"All Time Periods\" !" />
...

<blox:memberFilter id="memberFilterLocked"
   dimensionSelectionEnabled="false"
   selectedDimension="All Products">
   <blox:data bloxRef="myDataBlox" />
</blox:memberFilter>
...
```

MemberFilterBlox Properties and Methods Cross-Reference Table

The following table list unique MemberFilterBlox properties and methods. For lists of properties and methods common to several Blox, see "Common Blox Properties and Methods by Category" on page 29.

Properties	Methods
applyButtonEnabled	isApplyButtonEnabled() setApplyButtonEnabled()
dimensionSelectionEnabled	isDimensionSelectionEnabled() setDimensionSelectionEnabled()
selectableDimensions	getSelectableDimensions() setSelectableDimensions()
selectedDimension	getSelectedDimension() setSelectedDimension()
	getMemberFilterBloxModel()

MemberFilterBlox Properties and Associated Methods

This section describes the properties supported by MemberFilterBlox and the methods associated with those properties. The properties are listed alphabetically by property name. For a list of MemberFilterBlox methods with which no properties are associated, see "MemberFilterBlox Methods" on page 610. Common Blox properties available from DataBlox are listed but not described. For complete descriptions of common Blox properties, see "Properties and Associated Methods Common to Multiple Blox" on page 32.

id

This is a common Blox tag attribute. For a complete description, see “id” on page 39.

applyButtonEnabled

Shows the Apply button in the Member Filter user interface.

Data Sources

Multidimensional

Syntax

JSP Tag Attribute

```
applyButtonEnabled = "applyButtonEnabled"
```

Java Method

```
boolean isApplyButtonEnabled();
void setApplyButtonEnabled(boolean applyButtonEnabled);
```

where:

Argument	Default	Description
`applyButtonEnabled`	`true`	`true`: shows the Apply button; `false`: hides the Apply button.

Usage

Sometimes you may not need the Apply button in the Member Filter because the DataBlox referenced in the MemberFilterBlox is not used in a user interface Blox on the same page, and the MemberFilterBlox is only used to let users specify the members of interest in order to construct a separate query or perform some calculation. To hide the Apply button, set this property to `false`.

bloxEnabled

This is a common Blox property. For a complete description, see “bloxEnabled” on page 35.

bloxModel

This is a common Blox property. For a complete description, see “bloxModel” on page 38

bloxName

This is a common Blox property. For a complete description, see “bloxName” on page 35.

dimensionSelectionEnabled

Specifies whether the dimension selection drop list should be displayed.

Data Sources

Multidimensional

Syntax

JSP Tag Attribute

```
dimensionSelectionEnabled = "dimensionSelectionEnabled"
```

Java Method

```
boolean isDimensionSelectionEnabled();
void setDimensionSelectionEnabled(boolean dimensionSelectionEnabled);
```

where:

Argument	Default	Description
dimensionSelectionEnabled	true	true: display the dimension selection drop list; false: hide the dimension selection drop list.

Usage

When dimensionSelectionEnabled is set to true, all available dimensions as a result of the data query appear in the drop list. The dimensions are listed in alphabetical order. By default, the first dimension in the list is the initial selected dimension and appears in the Dimension Hierarchy panel on the left. To specify a different dimension as the initial selection, use selectedDimension. To limit the dimensions in the drop list, use selectableDimensions.

See Also

"selectedDimension" on page 609, "selectableDimensions" on page 608

height

This is a common Blox property. For a complete description, see "height" on page 38.

Member Filter has a default width and height for best layout. Do not specify the width or height unless you really need a specific size. If the size you specify is too small (less than 325 pixels in height and 600 pixels in width), the size will be set to the 325 X 600.

selectableDimensions

Specifies the dimensions to appear in the dimension selection drop list.

Data Sources

Multidimensional

Syntax

JSP Tag Attribute

```
selectableDimensions = "selectableDimensions"
```

Java Method

```
String getSelectableDimensions();
void setSelectableDimensions(String selectableDimensions);
```

where:

Argument	Default	Description
selectableDimensions	All available dimensions in the result set	A comma-separated list of dimensions.

Usage

When dimensionSelectionEnabled is set to `true` (the default), all available dimensions in the data result set appear in the dimension selection drop list unless specified otherwise in selectableDimensions. The dimensions you specified always appear in alphabetical order in the drop list regardless of the order you specified them. The initial selected dimension (the dimension that appears in the Dimension Hierarchy panel) is the first one on the list unless specified otherwise using selectedDimension.

Examples

```
<blox:memberFilter id="myMemberFilter"
   dimensionSelectionEnabled = "true"
   selectableDimensions = "Year, Scenario, Products">
   <blox:data bloxRef = "myDataBlox" />
</blox:memberFilter>
```

See Also

"dimensionSelectionEnabled" on page 607, "selectedDimension" on page 609

selectedDimension

Specifies the initial selected dimension.

Data Sources

Multidimensional

Syntax

JSP Tag Attribute

```
seletedDimension = "selectedDimension"
```

Java Method

```
String getSelectedDimension();
void setSelectedDimension(String selectedDimension);
```

where:

Argument	Default	Description
`selectedDimension`	The first dimension among the available dimensions from the data query	The name of the dimension to be the initial selected dimension.

Usage

Since only members from one dimension can be displayed in the Dimension Hierarchy panel, you should specify the initial selected dimension. If this is not specified, the first dimension in alphabetical order will be the selected dimension.

visible

This is a common Blox property. For a complete description, see "visible" on page 46.

width

This is a common Blox property. For a complete description, see "width" on page 47

Member Filter has a default width and height for best layout. Do not specify the width or height unless you really need a specific size. If the size you specify is too small (less than 325 pixels in height and 600 pixels in width), the size will be set to the 325 X 600.

MemberFilterBlox Methods

This section describes MemberFilterBlox methods that are not associated with a specific property. For the syntax and descriptions of MemberFilterBlox methods that have a property associated with them, see “MemberFilterBlox Methods” on page 610. For client-side API common to Blox, see “Client-Side APIs” on page 31.

call()

This is a common client-side Blox method. For a complete description, see “call()” on page 51.

flushProperties()

This is a common client-side Blox method. For a complete description, see “flushProperties()” on page 52.

getDataBlox()

This is a common Blox method. For a complete description, see “setDataBlox()” on page 60.

getMemberFilterBloxModel()

A convenience method to return a typed UI model for the MemberFilterBlox. Invoking this method causes the server to create the DHTML framework for the Blox.

Data Sources

Multidimensional

Syntax

Java Method

```
MemberFilterBloxModel getMemberFilterBloxModel();
   // throws ServerBloxException
```

See Also

See the `com.alphablox.blox.uimodel` package in the Javadoc for the MemberFilterBloxModel API.

setDataBusy()

This is a common client-side Blox method. For a complete description, see "setDataBusy()" on page 60.

setDataBlox()

This is a common Blox method. For a complete description, see “setDataBlox()” on page 60.

updateProperties()

This is a common client-side Blox method. For a complete description, see “updateProperties()” on page 61.

Chapter 18. PageBlox Reference

This chapter contains reference material for PageBlox properties, methods and objects. For general reference information about Blox, see Chapter 3, "General Blox Reference Information," on page 17. For information on how to use this reference, see Chapter 1, "Using This Reference," on page 1.

PageBlox Overview

PageBlox enables users to filter data that appears in the grid or chart. Each dimension in the current result set that resides on the Page axis appears as a drop list in the Page Filter. When the user selects a dimension member from the drop list, the member is used to filter the data appearing in the grid or chart.

PageBlox JSP Custom Tag Syntax

The Alphablox Tag Libraries provides custom tags to use in a JSP page for creating each Blox. This section describes how to create the custom tag to create a PageBlox. For a copy and paste version of the tag with all the attributes, see "PageBlox JSP Custom Tag" on page 888.

Syntax

```
<blox:page
    [attribute="value"] >
</blox:page>
```

where:

attribute is one of the attributes listed in the attribute table.

value is a valid value for the attribute.

and where the attributes are one of the following:

Attribute
id
applyPropertiesAfterBookmark
bloxEnabled
bloxName
bookmarkFilter
fixedChoiceLists
height
helpTargetFrame

Attribute
labelPlacement
localeCode
maximumUndoSteps
moreChoicesEnabled
moreChoicesEnabledDefault
noDataMessage
render
visible
width

Usage

Each custom tag can have one or more attributes, separated by one or more space or new line characters. Any extra space or new line characters are ignored. For readability, attributes can each go on a separate line with the same indenting.

You can substitute the closing </blox:page> tag with a closing slash (/) after the last attribute in the tag but before the closing greater than character. For example, if the last attribute is width, the end of the tag looks as follows:

```
width="650" />
```

Examples

```
<blox:page
     fixedChoiceLists="Year:Qtr1,Qtr2;Market:East"
     >
</blox:page>
```

PageBlox Properties and Methods by Category

The following tables list unique PageBlox properties. The tables also list methods for which there are no corresponding properties. For lists of properties and methods common to several Blox, see "Common Blox Properties and Methods by Category" on page 29.

The properties and methods supported by PageBlox are organized in the cross reference as follows:

- "Choice Lists" on page 612
- "Panel Type and Appearance" on page 613

Choice Lists

The following table shows the properties and methods associated with the choice lists for PageBlox.

Properties	Methods
fixedChoiceLists	
	getFixedChoiceLists() setFixedChoiceLists()
moreChoicesEnabled	
	isMoreChoicesEnabled() setMoreChoicesEnabled()

moreChoicesEnabledDefault	isMoreChoicesEnabledDefault() setMoreChoicesEnabledDefault()

Panel Type and Appearance

The following table shows the property and associated methods for setting the page panel type and for settings that affect the PageBlox appearance.

Property	Methods
labelPlacement	getLabelPlacement() setLabelPlacement()

PageBlox Properties and Associated Methods

This section describes the properties supported by PageBlox and the methods associated with those properties. The properties are listed alphabetically by property name. For a list of PageBlox methods with which no properties are associated, see "PageBlox Methods" on page 617. Common Blox properties available from DataBlox are listed but not described. For complete descriptions of common Blox properties, see "Properties and Associated Methods Common to Multiple Blox" on page 32.

id

This is a common Blox property. For a complete description, see "id" on page 39.

applyPropertiesAfterBookmark

This is a common Blox property. For a complete description, see "applyPropertiesAfterBookmark" on page 32.

bloxEnabled

This is a common Blox property. For a complete description, see "bloxEnabled" on page 35.

bloxModel

This is a common Blox property. For a complete description, see "bloxModel" on page 38

bloxName

This is a common Blox property. For a complete description, see "bloxName" on page 35.

bookmarkFilter

This is a common Blox property. For a complete description, see "bookmarkFilter" on page 33.

fixedChoiceLists

Places the named dimensions and members on the drop list so that the user can access them.

Data Sources

Multidimensional

Syntax

JSP Tag Attribute

```
fixedChoiceLists="dimensionMemberList"
```

Java Methods

```
String getFixedChoiceLists(String dimensionName);
void setFixedChoiceLists(String dimensionMemberList);
```

where:

Argument	Default	Description
`dimensionMemberList`	empty string	Pairs of comma-delimited string of member names of the specified dimension, separated by semicolons: *dimension1:member1,member2; dimension2:member1,member2* There should be no spaces in between unless the dimension or member names contain one.
`dimensionName`	empty string	A valid dimension name.
`members`	empty string	A comma separated list of members.

Usage

The default permits the user to access all dimensions and members.

Any dimension(s) specified in the `fixedChoiceLists` property must also appear in the DataBlox selectableSlicerDimensions property.

The query for the initial display must include only one of the members for each dimension specified in the `fixedChoiseLists` property, otherwise the root level member of the dimension initially appears in the fixed choice list. The member(s) specified in the query will be the default member(s) in the fixed choice list(s). For example, if you have a fixed choice list on the `Year` dimension with members `Q1` and `Q2` and neither `Q1` or `Q2` is specified in the query, then the fixed choice list shows `Year`, `Q1`, and `Q2` initially. After selecting `Q1` or `Q2` in the page filter, `Year` is then removed from the list. If two or more members are specified in the query, the fixed choice list will not appear in the PageBlox.

If the PageBlox is nested within another Blox (for example, a PresentBlox) and a user moves a member from the PageBlox to a row or column axis (for example), then the fixed choice list does not apply to that row and column axis; it only applies to the PageBlox. If you do not want this to occur, use a standalone PageBlox instead of a nested PageBlox.

A unique name (base name in IBM DB2 OLAP Server or Hyperion Essbase) or display name can be used for the dimension and member names specified in the property's value. This allows assemblers to differentiate between different members or dimensions with the same display names.

In IBM DB2 OLAP Server or Hyperion Essbase, you can specify a member, regardless of the alias table in use, by using the base name.

Examples

```
setFixedChoiceLists("Year:Qtr1,Qtr2;Market:East");
getFixedChoiceLists(); //returns "Year":"Qtr1","Qtr2";"Market":"East";
```

See Also

"moreChoicesEnabled" on page 616, "moreChoicesEnabledDefault" on page 616, "selectableSlicerDimensions" on page 376

height

This is a common Blox property. For a complete description, see "height" on page 38.

helpTargetFrame

This is a common Blox property. For a complete description, see "helpTargetFrame" on page 39.

labelPlacement

Sets the placement of the PageBlox label relative to the PageBlox drop down list. Valid `labelPlacement` values are `left`, `top`, and `none`. The default value is `left`.

Data Sources

Multidimensional

Syntax

JSP Tag Attribute

```
labelPlacement="placement"
```

Java Methods

```
String getLabelPlacement();
void setLabelPlacement(String placement);
```

where:

Argument	Default	Description
`placement`	`left`	A String indicating where the PageBlox label is placed relative to the drop down list. Valid values are `left`, `top`, and `none`.

Examples

```
myPageBlox.setLabelPlacement("Top");
```

localeCode

This is a common Blox property. For a complete description, see "localeCode" on page 40.

maximumUndoSteps

This is a common Blox property. For a complete description, see "maximumUndoSteps" on page 41.

moreChoicesEnabled

Specifies whether the "More..." option in the PageBlox drop list for the named dimensions should be available to permit the user to view more choices using the Member Filter.

Data Sources

Multidimensional

Syntax

JSP Tag Attribute

```
moreChoicesEnabled="choices"
```

Java Methods

```
String getMoreChoicesEnabled();
void setMoreChoicesEnabled(String choices);
```

where:

Argument	Default	Description
`choices`	empty string	A string of *dimension:enabled* pairs separated by a semicolon (";"). The list should not contain any spaces, unless it is a space in the dimension name. • *dimension*: dimension name • *enabled*: `true` or `false`

Usage

By default, the **More...** option is available for all dimensions placed on the page axis. This property allows you to hide the **More...** option for the named dimensions so the user cannot access Member Filter for more choices.

Examples

The following lines make the More... option available in the PageBlox drop lists for the Product dimension but not for the Market dimension. Note that there is no space in between unless the dimension name contains one.

```
setMoreChoicesEnabled("Product:true;Market:false");
```

See Also

"fixedChoiceLists" on page 613, "moreChoicesEnabledDefault" on page 616

moreChoicesEnabledDefault

Specifies the default for whether the "More..." option in the PageBlox drop list for the named dimensions should be available to permit the user to view more choices using the Member Filter.

Data Sources

Multidimensional

Syntax

JSP Tag Attribute

```
moreChoicesEnabledDefault="boolean"
```

Java Methods

```
boolean isMoreChoicesEnabledDefault();
void setMoreChoicesEnabledDefault(boolean enabledDefault);
```

where:

Argument	Default	Description
enabledDefault	true	Valid values are true and false.

Examples

```
isMoreChoicesEnabledDefault();
setMoreChoicesEnabledDefault(false);
```

See Also

"fixedChoiceLists" on page 613, "moreChoicesEnabled" on page 616

noDataMessage

This is a common Blox property. For a complete description, see "noDataMessage" on page 42.

render

This is a common Blox property. For a complete description, see "render" on page 45.

visible

This is a common Blox property. For a complete description, see "visible" on page 46.

width

This is a common Blox property. For a complete description, see "width" on page 47.

PageBlox Methods

This section describes PageBlox methods that are not associated with a specific property. For the syntax and descriptions of PageBlox methods that have a property associated with them, see "PageBlox Methods" on page 617. For client-side API common to Blox, see "Client-Side APIs" on page 31.

addEventFilter()

This is a common Blox method that for capturing a server-side event (such as saving and loading bookmarks) and perform custom actions *after* the operation is complete on the server. For details, see "addEventListener()" on page 49.

addEventListener()

This is a common Blox method that allows you to capture a server-side event (such as saving and loading bookmarks) and perform custom actions *after* the operation is complete on the server. For details, see "addEventListener()" on page 49.

call()

This is a common client-side Blox method. For a complete description, see "call()" on page 51.

flushProperties()

This is a common client-side Blox method. For a complete description, see “flushProperties()” on page 52.

getDataBlox()

This is a common Blox method. For a complete description, see “setDataBlox()” on page 60.

loadBookmark()

This is a common Blox method. For a complete description, see “loadBookmark()” on page 54.

removeEventFilter()

This is a common Blox method that allows you to remove an event filter object added using addEventFilter() for capturing a server-side event *before* the event is processed on the server. For details, see “removeEventFilter()” on page 55.

removeEventListener()

This is a common Blox method that allows you to remove an event listener object created using addEventListener() for capturing a server-side event *after* that operation is complete on the server. For details, see “removeEventListener()” on page 56.

saveBookmark()

This is a common Blox method. For a complete description, see “saveBookmark()” on page 57.

saveBookmarkHidden()

This is a common Blox method. For a complete description, see “saveBookmarkHidden()” on page 58.

setDataBusy()

This is a common client-side Blox method. For a complete description, see "setDataBusy()" on page 60.

setDataBlox()

This is a common Blox method. For a complete description, see “setDataBlox()” on page 60.

updateProperties()

This is a common client-side Blox method. For a complete description, see “updateProperties()” on page 61.

Chapter 19. PresentBlox Reference

This chapter contains reference material for the PresentBlox. For general reference information about Blox, see Chapter 3, "General Blox Reference Information," on page 17. For information on how to use this reference, see Chapter 1, "Using This Reference," on page 1.

PresentBlox Overview

PresentBlox provides a graphical user interface that can nest ChartBlox, GridBlox, PageBlox, ToolbarBlox, and DataLayoutBlox within a single presentation. Application assemblers use PresentBlox properties to tailor how these Blox appear.

PresentBlox makes extensive use of Blox qualifiers, as described in "Nested Blox" on page 6. For information on each nested Blox, refer to one of the following pages:

- "ChartBlox Overview" on page 193
- "DataLayoutBlox Overview" on page 457
- "GridBlox Overview" on page 537
- "PageBlox Overview" on page 611
- "ToolbarBlox Overview" on page 691

PresentBlox combines several Blox in one, providing users with simultaneous chart and grid views of the same data in the same window real estate.

Note: The user can click on a the Grid, Chart, Page Filter and Data Layout Panel buttons in the toolbar to show and hide these components in the PresentBlox. The user can also move the slider bar between the grid and chart to change the amount of space devoted to each view.

PresentBlox JSP Custom Tag Syntax

The Alphablox Tag Libraries provides custom tags to use in a JSP page for creating each Blox. This section describes how to create the custom tag to create a PresentBlox. For a copy and paste version of the tag with all the attributes, see "PresentBlox JSP Custom Tag" on page 889.

Syntax

```
<blox:present
    [attribute="value"] >
</blox:present>
```

where:

attribute is one of the attributes listed in the attribute table.

value is a valid value for the attribute.

and where the attributes are one of the following:

Attribute
id
applyPropertiesAfterBookmark
bloxEnabled
bloxName
chartAvailable
chartFirst
dataLayoutAvailable
dividerLocation
enablePoppedOut
gridAvailable
height
helpTargetFrame
localeCode
maximumUndoSteps
menubarVisible
noDataMessage
pageAvailable
poppedOut
poppedOutHeight
poppedOutTitle
poppedOutWidth
render
splitPane
splitPaneOrientation
toolbarVisible
visible
width

Usage

Each custom tag can have one or more attributes, separated by one or more space or new line characters. Any extra space or new line characters are ignored. For readability, attributes can each go on a separate line with the same indenting.

You can substitute the closing `</blox:present>` tag with a closing slash (`/`) after the last attribute in the tag but before the closing greater than character. For example, if the last attribute is `width`, the end of the tag looks as follows:

```
width="650" />
```

Examples

```
<blox:present
   id="myPresent1"
   width="650"
   height="600"
   >
   <blox:data
      dataSourceName="TBC"
      query="<SYM <ROW(Product) <ICHILD Product <COLUMN(Year,
         Scenario) Qtr1 Qtr2 <CHILD Scenario Sales !"
   />
</blox:present>
```

PresentBlox Properties and Methods by Category

The following tables list the PresentBlox properties and methods organized by categories of functionality. For lists of properties and methods common to several Blox, see "Common Blox Properties and Methods by Category" on page 29. The properties and methods supported by PresentBlox are organized in the cross reference as follows:

Data Area

The following properties and methods affect the data area of a PresentBlox.

Properties	Methods
dividerLocation	
	getDividerLocation() setDividerLocation()
splitPane	
	getSplitPane() setSplitPane()
splitPaneOrientation	
	getSplitPaneOrientation() setSplitPaneOrientation()

Chart Appearance

The following properties and methods affect the appearance of a ChartBlox.

Properties	Methods
chartAvailable	
	isChartAvailable() setChartAvailable()
chartFirst	
	isChartFirst() setChartFirst()

Data Layout Appearance

The following properties and methods affect the appearance of the DataLayoutBlox on a PresentBlox.

Properties	Methods
dataLayoutAvailable	getDataLayoutAvailable() setDataLayoutAvailable()

Grid Appearance

The following properties and methods affect the appearance of a GridBlox on a PresentBlox.

Properties	Methods
gridAvailable	getGridAvailable() setGridAvailable()

Page Appearance

The following properties and methods affect the appearance of a PageBlox on a PresentBlox.

Property	Methods
pageAvailable	isPageAvailable() setPageAvailable()

Menubar Appearance

The following properties and methods affect the appearance of a menubar on a PresentBlox.

Property	Methods
menubarVisible	isMenubarVisible() setMenubarVisible()

Toolbar Appearance (Tag Attribute)

The following tag attribute affects the appearance of the toolbar on a PresentBlox.

Property	Methods
toolbarVisible	No method. This is not a property.

Popped Out Properties

The following table lists the properties regarding displaying PresentBlox in a separate, popped out browser window.

Chart Labels

Properties	Methods
enablePoppedOut	isEnablePoppedOut() setPoppedOut()

poppedOut	isPoppedOut() setPoppedOut()
poppedOutHeight	getPoppedOutHeight() setPoppedOutHeight()
poppedOutTitle	getPoppedOutTitle() setPoppedOutTitle()
poppedOutWidth	getPoppedOutWidth() setPoppedOutWidth()

Server-Side Event Filters and Listeners

The following table lists the methods for capturing events for pre- and post-event processing.

Methods
addEventFilter()
addEventListener()
removeEventFilter()
removeEventListener()

PresentBlox Properties and Associated Methods

This section describes the properties supported by PresentBlox and the methods associated with those properties. The properties are listed alphabetically by property name. For a list of PresentBlox methods with which no properties are associated, see "PresentBlox Methods" on page 630. Common Blox properties available from DataBlox are listed but not described. For complete descriptions of common Blox properties, see "Properties and Associated Methods Common to Multiple Blox" on page 32.

id

This is a common Blox tag attribute. For a complete description, see "id" on page 39.

applyPropertiesAfterBookmark

This is a common Blox property. For a complete description, see "applyPropertiesAfterBookmark" on page 32.

bloxEnabled

This is a common Blox property. For a complete description, see "bloxEnabled" on page 35.

bloxModel

This is a common Blox property. For a complete description, see "bloxModel" on page 38

bloxName

This is a common Blox property. For a complete description, see “bloxName” on page 35.

chartAvailable

Specifies whether the chart is available to the user in PresentBlox.

Data Sources

All

Syntax

JSP Tag Attribute

```
chartAvailable="available"
```

Java Method

```
boolean isChartAvailable();
void setChartAvailable(boolean available);
```

where:

Argument	Default	Possible Values
available	true	Specify true to make the chart available; false to make it unavailable.

Usage

The default is true. If set to false, the user cannot cause the chart to appear at all. To suppress the appearance of the chart until the user invokes it, use the dividerLocation property.

Examples

```
isChartAvailable();
isChartAvailable();
```

See Also

“chartFirst” on page 624, “dividerLocation” on page 626, “getChartBlox()” on page 631

chartFirst

Sets whether the chart appears before the grid when both appear in the PresentBlox display area.

Data Sources

All

Syntax

JSP Tag Attribute

```
chartFirst="first"
```

Java Methods

```
boolean isChartFirst();
void setChartFirst(boolean first);
```

where:

Argument	Default	Possible Values
first	false	Specify true to make the chart appear first; false to make it appear after the grid.

Usage
Depending on the value specified for the splitPaneOrientation property, “before” means “to the left of” or “above” the grid.

Examples
```
isChartFirst();
setChartFirst(true);
```

See Also
“chartAvailable” on page 624, “chartFirst” on page 624, “dividerLocation” on page 626, “getChartBlox()” on page 631

dataLayoutAvailable
Specifies whether the data layout panel is available in PresentBlox.

Data Sources
All

Syntax
JSP Tag Attribute

```
dataLayoutAvailable="available"
```

Java Methods

```
boolean isDataLayoutAvailable();
void setDataLayoutAvailable(boolean available);
```

where:

Argument	Default	Possible Values
available	true	Specify true to make the data layout panel available; false to make it unavailable.

Usage
Note the following about the dataLayoutAvailable property:
- With this property set to false, the user cannot invoke the data layout panel. The Layout button will not appear on the toolbar.
- With this property set to true and the visible property set to false, the data layout panel appears only when the user clicks the toolbar layout button.

Examples
```
setDataLayoutAvailable(false);
isDataLayoutAvailable();
```

See Also
“getDataLayoutBlox()” on page 631

dividerLocation

Specifies where the available area should be divided into panes for displaying the chart and the grid.

Data Sources

All

Syntax

JSP Tag Attribute

```
dividerLocation="location"
```

Java Methods

```
double getDividerLocation();
void setDividerLocation(double location);
```

where:

Argument	Default	Possible Values
location	.5	A numeric value from 0 to 1 (of type double).

Usage

A value of 1 means that only the display on the left (or top, depending on the value of the splitPaneOrientation property) should appear. A value of 0 means that only the display on the right (or bottom) should appear. A value of .5 indicates that the area should be divided equally between the two displays.

Examples

```
setDividerLocation(.7);
```

See Also

"chartAvailable" on page 624, "gridAvailable" on page 626, "splitPaneOrientation" on page 629.

enablePoppedOut

This is a property inherited from ContainerBlox. For a complete description, see "enablePoppedOut" on page 313.

gridAvailable

Specifies whether the grid is available in PresentBlox.

Data Sources

All

Syntax

JSP Tag Attribute

```
gridAvailable="available"
```

Java Methods

```
boolean isGridAvailable();
void setGridAvailable(boolean available);
```

where:

Argument	Default	Possible Values
`available`	`true`	Specify `true` to make the grid available; `false` to make it unavailable.

Examples

```
setGridAvailable(false);
isGridAvailable();
```

See Also

"dividerLocation" on page 626, "getGridBlox()" on page 632

height

This is a common Blox property. For a complete description, see "height" on page 38.

helpTargetFrame

This is a common Blox property. For a complete description, see "helpTargetFrame" on page 39.

localeCode

This is a common Blox property. For a complete description, see "localeCode" on page 40.

maximumUndoSteps

This is a common Blox property. For a complete description, see "maximumUndoSteps" on page 41.

menubarVisible

This is a common Blox property. For a complete description, see "menubarVisible" on page 42.

noDataMessage

This is a common Blox property. For a complete description, see "noDataMessage" on page 42.

pageAvailable

Specifies whether the page panel is available in PresentBlox.

Data Sources

All

Syntax

JSP Tag Attribute

```
pageAvailable="available"
```

Java Methods

```
boolean isPageAvailable();
void setPageAvailable(boolean available);
```

Usage

The default is `true`. When assembling an application where PresentBlox will be delivered only in non-Java format, set this value to `false`.

When this value is set to `false`, page filters do not appear when a user drags items onto the Page axis on the Data Layout panel.

Examples

```
setPageAvailable();
isPageAvailable();
```

See Also

"getPageBlox()" on page 632

poppedOut

This is a property inherited from ContainerBlox. For a complete description, see "poppedOut" on page 314.

poppedOutHeight

This is a property inherited from ContainerBlox. For a complete description, see "poppedOutHeight" on page 315.

poppedOutTitle

This is a property inherited from ContainerBlox. For a complete description, see "poppedOutTitle" on page 315.

poppedOutWidth

This is a property inherited from ContainerBlox. For a complete description, see "poppedOutWidth" on page 316.

render

This is a common Blox property. For a complete description, see "render" on page 45.

splitPane

Specifies whether the data display area is split into panes once PresentBlox is instantiated.

Data Sources

All

Syntax

JSP Tag Attribute

```
splitPane="split"
```

Java Methods

```
boolean isSplitPane();
void setSplitPane(boolean split);
```

where:

Argument	Default	Possible Values
split	true	Specify true to split into panes; false to render a single data display area that chart and grid share.

Usage
The user toggles between the two data presentations using the Grid and Chart toolbar buttons.

See Also
"dividerLocation" on page 626, "splitPaneOrientation" on page 629

splitPaneOrientation
Specifies how to split the available area into panes for the chart and the grid.

Data Sources
All

Syntax
JSP Tag Attribute

```
splitPaneOrientation="orientation"
```

Java Methods

```
String getSplitPaneOrientation();
void setSplitPaneOrientation(String orientation);
```

where:

Argument	Default	Possible Values
orientation	vertical	Specify vertical to display Blox side-by-side (similar to portrait), or horizontal to display one Blox above the other (similar to landscape).

Usage
The value of the chartFirst property determines whether ChartBlox or GridBlox appears "first" (on the top or at the left).

Examples
```
getSplitPaneOrientation();
setSplitPaneOrientation("horizontal");
```

See Also
"chartFirst" on page 624, "splitPane" on page 628

toolbarVisible
Specifies if the toolbar is visible.

Data Sources
All

Syntax

JSP Tag Attribute

```
toolbarVisible="visible"
```

where:

Argument	Default	Description
visible	true	true: the toolbar is visible; false: the toolbar is not visible.

Usage

By default, the toolbar is visible in a PresentBlox. If a nested <blox:toolbar> tag is added, its setting overwrites the value of this attribute. For example, the following code will result in a visible toolbar.

```
<blox:present id="myPresent" toolbarVisible="false" ....>
   <blox:toolbar visible="true" />
   <blox:data bloxRef="myDataBlox"/>
</blox:chart>
```

Tip: toolbarVisible is only a tag attribute, not a property.

visible

This is a common Blox property. For a complete description, see “visible” on page 46.

width

This is a common Blox property. For a complete description, see “width” on page 47.

PresentBlox Methods

This section describes PresentBlox methods that are not associated with a specific property. For the syntax and descriptions of PresentBlox methods that have a property associated with them, see “PresentBlox Properties and Associated Methods” on page 623. For client-side API common to Blox, see “Client-Side APIs” on page 31.

addEventFilter()

This is a common Blox method that for capturing a server-side event (such as saving and loading bookmarks) and perform custom actions *after* the operation is complete on the server. For details, see “addEventListener()” on page 49.

addEventListener()

This is a common Blox method that allows you to capture a server-side event (such as saving and loading bookmarks) and perform custom actions *after* the operation is complete on the server. For details, see “addEventListener()” on page 49.

call()

This is a common client-side Blox method. For a complete description, see “call()” on page 51.

flushProperties()

This is a common client-side Blox method. For a complete description, see “flushProperties()” on page 52.

getApplicationName()

This is a common Blox method. For a complete description, see “getApplicationName()” on page 52.

getChartBlox()

Returns an interface to ChartBlox.

Data Sources

All

Syntax

Java Method

```
ChartBlox getChartBlox();
        throws ChartBloxUnavailableException
               ServerBloxException
```

Usage

All of the ChartBlox methods are available through this method.

Examples

```
getChartBlox();
```

See Also

“chartAvailable” on page 624, “chartFirst” on page 624, “getDataLayoutBlox()” on page 631, “getGridBlox()” on page 632, “getPageBlox()” on page 632, “ChartBlox Methods” on page 267

getDataBlox()

This is a common Blox method. For a complete description, see “setDataBlox()” on page 60.

getDataLayoutBlox()

Returns an interface to DataLayoutBlox.

Data Sources

All

Syntax

Java Method

```
DataLayoutBlox getDataLayoutBlox();
```

Usage

All of the methods on DataLayoutBlox are available through this method.

Examples

```
getDataLayoutBlox();
```

See Also

“dataLayoutAvailable” on page 625, “getChartBlox()” on page 631, “getGridBlox()” on page 632, “getPageBlox()” on page 632, “DataLayoutBlox Methods” on page 461

getGridBlox()

Returns an interface to GridBlox.

Data Sources

All

Syntax

Java Method

```
GridBlox getGridBlox();
```

Usage

All of the GridBlox methods are available through this method.

Examples

```
getGridBlox();
```

See Also

"getChartBlox()" on page 631, "getDataLayoutBlox()" on page 631, "getPageBlox()" on page 632, "gridAvailable" on page 626, "GridBlox Methods" on page 590

getPageBlox()

Returns an interface to PageBlox.

Data Sources

Multidimensional

Syntax

Java Method

```
PageBlox getPageBlox();
```

Usage

All of the PageBlox methods are available through this method.

Examples

```
getPageBlox();
```

See Also

"getChartBlox()" on page 631, "getDataLayoutBlox()" on page 631, "getGridBlox()" on page 632, "pageAvailable" on page 627, "PageBlox Methods" on page 617

getProperty()

This is a common Blox method. For a complete description, see "getProperty()" on page 53.

init()

This is a common Blox method. For a complete description, see "init()" on page 54.

loadBookmark()

This is a common Blox method. For a complete description, see "loadBookmark()" on page 54.

removeEventFilter()

This is a common Blox method that allows you to remove an event filter object added using addEventFilter() for capturing a server-side event *before* the event is processed on the server. For details, see “removeEventFilter()” on page 55.

removeEventListener()

This is a common Blox method that allows you to remove an event listener object created using addEventListener() for capturing a server-side event *after* that operation is complete on the server. For details, see “removeEventListener()” on page 56.

render()

This is a common Blox method. For a complete description, see “render()” on page 56.

renderHtmlHeader()

This is a common Blox method. For a complete description, see “renderHtmlHeader()” on page 57.

saveBookmark()

This is a common Blox method. For a complete description, see “saveBookmark()” on page 57.

saveBookmarkHidden()

This is a common Blox method. For a complete description, see “saveBookmarkHidden()” on page 58.

setDataBusy()

This is a common client-side Blox method. For a complete description, see “setDataBusy()” on page 60.

setDataBlox()

This is a common Blox method. For a complete description, see “setDataBlox()” on page 60.

setProperty()

This is a common Blox method. For a complete description, see “setProperty()” on page 61.

updateProperties()

This is a common client-side Blox method. For a complete description, see “updateProperties()” on page 61.

Chapter 20. RepositoryBlox Reference

This chapter contains reference material for the RepositoryBlox. For general reference information about Blox, see Chapter 3, "General Blox Reference Information," on page 17. For information on how to use this reference, see Chapter 1, "Using This Reference," on page 1.

RepositoryBlox Overview

RepositoryBlox provides a means for developers to save and retrieve application properties and various objects stored in the DB2 Alphablox Repository. This capability is key to building a personalized application. Methods on RepositoryBlox fall into three categories:

- those for saving and maintaining multiple application states
- those providing access to application, user, and group properties
- those for saving and accessing objects stored in the DB2 Alphablox Repository

If multiple application states reside in the Repository, users can select the desired instance from the Applications page of the DB2 Alphablox Home Page.

Besides methods for saving and retrieving user, application, application state, and group properties, RepositoryBlox also provides methods for saving and retrieving Java objects of different types. These types are expressed as constants such as `TYPE_BINARY`, `TYPE_TEXT`, `TYPE_CONTAINER` (subfolders in the directory), `TYPE_HASHTABLE` (an array of objects), and `TYPE_XMLDOCUMENT`. This provides great flexibility and capability in the kind of data you can save and retrieve utilizing the DB2 Alphablox Repository.

Note: Group names are converted to all lowercase letters when stored in the repository to enhance performance.

RepositoryBlox has no graphical user interface. To invoke server-side RepositoryBlox methods, you can use the DHTML Client API, described in "Client-Side API Overview" on page 63.

RepositoryBlox JSP Custom Tag Syntax

The Alphablox Tag Libraries provides custom tags to use in a JSP page for creating each Blox. This section describes how to create the custom tag to create a RepositoryBlox. For a copy and paste version of the tag with all the attributes, see "RepositoryBlox JSP Custom Tag" on page 890.

Syntax

```
<blox:repository
    [attribute="value"] >
</blox:repository>
```

where:

attribute is one of the attributes listed in the attribute table.

value is a valid value for the attribute.

and where the attributes are one of the following:

Attribute
id
bloxName
render

Usage

Each custom tag can have one or more attributes, separated by one or more space or new line characters. Any extra space or new line characters are ignored. For readability, attributes can each go on a separate line with the same indenting. You can substitute the closing </blox:repository> tag using the shorthand notation, closing the tag at the end of the attribute list that looks as follows:

```
id="myRepositoryBlox" />
```

Examples

```
<blox:repository id="myRepository" />
```

RepositoryBlox Properties and Methods by Category

The following tables list the unique RepositoryBlox methods. The RepositoryBlox properties (and their corresponding methods) are all common to several Blox. For a listing of common properties and methods, see "Common Blox Properties and Methods by Category" on page 29. The RepositoryBlox cross reference is organized into the following tables:

- Applications and Application State
- Groups
- Users
- Themes
- General Objects
- Session Management
- HTML Fragment Conversion

Applications and Application State

The following table lists the RepositoryBlox methods relating to applications.

Java Method
deleteApplicationState()
renameApplicationState()
restoreApplicationState()
getAllApplications() getApplicationProperty() setApplicationProperty() getApplicationPropertyMap() getApplicationStateNameAndDescription()
getAllApplications()
getDataSourceNames()
getInstanceProperty() setInstanceProperty() getInstancePropertyMap()

Groups

The following table lists RepositoryBlox methods relating to groups. Keep in mind that group names are converted to all lowercase letters when stored in the repository.

Java Methods

getGroupNames()

getUsersCurrentGroup()

getGroupProperty()setGroupProperty()getGroupPropertyMap()

Users

The following table lists RepositoryBlox methods relating to users.

Java Methods

getUserNames()

getUserProperty()
setUserProperty()
getUserPropertyMap()

Themes

The following table lists the RepositoryBlox method relating to themes.

Java Method
getThemes()

General Objects

The following table lists RepositoryBlox methods relating to saving and accessing objects.

Java Methods

delete()

rename()

`exists()`

`save()`

`list()`

`search()`

`load()`

Session Management

The following table lists the RepositoryBlox methods relating to session management.

Java Methods

`killSession()`

`logout()`

HTML Fragment Conversion

The following table lists the RepositoryBlox method for converting HTML fragments in Alphablox 3 to Alphablox 4 or Alphablox 5.

Java Method
`readFragment()`

RepositoryBlox Properties and Associated Methods

This section describes the properties supported by RepositoryBlox and the methods associated with those properties. The properties are listed alphabetically by property name. All of the RepositoryBlox properties are common to multiple Blox and they are listed but not described in this section. For complete descriptions of common Blox properties, see "Properties and Associated Methods Common to Multiple Blox" on page 32. For a list of RepositoryBlox methods with which no properties are associated, see "RepositoryBlox Methods" on page 638.

id

This is a common Blox tag attribute. For a complete description, see "id" on page 39.

bloxName

This is a common Blox tag attribute. For a complete description, see "bloxName" on page 35.

render

This is a common Blox property. For a complete description, see "render" on page 45.

RepositoryBlox Methods

This section describes RepositoryBlox methods that are not associated with a specific property. For the syntax and descriptions of RepositoryBlox methods that have a property associated with them, see "RepositoryBlox Properties and Associated Methods" on page 638. For client-side API common to Blox, see "Client-Side APIs" on page 31.

delete()

Deletes the specified object from the repository.

Data Sources

All

Syntax

Java Method

```
public void delete(int visibility, String owner, String name,
                   int type);
     throws ServerBloxMissingResourceException,
            InvalidRepositoryTypeException,
            RepositoryIOException,
            RepositorySecurityException,
            ServerBloxException
```

where:

Argument	Default	Description
visibility	none	Visibility of the application state to be removed: VISIBILITY_PRIVATE, VISIBILITY_APPLICATION, or VISIBILITY_GROUP.
owner	none	The name of the user who owns the object.
name	none	The name of the object in the repository.
type	none	One of the following constants indicating the type of object: • TYPE_BINARY: a binary type repository object • TYPE_TEXT: a text type repository object • TYPE_HASHTABLE: a property map type repository object (an array of objects) • TYPE_XMLDOCUMENT: an XML type repository object • TYPE_CONTAINER : a subdirectory

Usage

This method deletes the specified object with the specified path from the repository. When an object is saved, the specified owner and name of the object are appended to the repository path to the object. To add additional directories to the path, you can add "/" to the name. The same rule applies when you delete, load, list, rename or test the existence of an object.

To delete all directories under your specified directory path, set the type to TYPE_CONTAINER. To delete all files of the specified type in your specified directory path, set the name to an empty string "".

Examples

```
<blox:repository id="myRepoBlox" />
<% int visibility = myRepoBlox.VISIBILITY_APPLICATION;
   String owner = "admin";
   String name = "sales/westdata";
   int type = myRepoBlox.TYPE_CONTAINER;
   myRepoBlox.delete(visibility,owner,name,type);
%>
```

See Also

"exists()" on page 640, "list()" on page 648, "load()" on page 649, "rename()" on page 652, "save()" on page 654, "search()" on page 656

deleteApplicationState()

Deletes a saved application state from the DB2 Alphablox repository.

Data Sources

All

Syntax

Java Method

```
void deleteApplicationState(String name, int visibility,
                                    int scope);
            throws RepositoryIOException,
                   ServerBloxMissingResourceException,
                   InvalidRepositoryVisibilityException,
                   InvalidRepositoryScopeException,
                    ServerBloxException
```

where:

Argument	Default	Description
name	none	Name of the saved application state to be removed.
visibility	none	Visibility of the application state to be removed: VISIBILITY_APPLICATION, VISIBILITY_GROUP, or VISIBILITY_PRIVATE.
scope	none	Scope of the application state to be removed: SCOPE_APPLICATION or SCOPE_SINGLE_BLOX (i.e. bookmark scope).

Examples

Given a RepositoryBlox named myRepo:

```
myRepo.deleteApplicationState("SomeAppState",
          myRepo.VISIBILITY_APPLICATION, myRepo.SCOPE_APPLICTION)
```

exists()

Tests to see if an object exists in the DB2 Alphablox repository.

Data Sources

All

Syntax

Java Method

```
boolean exists(int visibility, String owner, String name, int type);
            throws ServerBloxMissingResourceException,
                   InvalidRepositoryTypeException,
                   RepositoryIOException,
                   RepositorySecurityException,
                    ServerBloxException
```

where:

Argument	Default	Description
visibility	none	Visibility of the application state to be removed: VISIBILITY_APPLICATION, VISIBILITY_GROUP, or VISIBILITY_PRIVATE.
owner	none	The name of the user who owns the object.
name	none	The name of the object in the repository.
type	none	One of the following constants indicating the type of object: • TYPE_BINARY: a binary type repository object • TYPE_TEXT: a text type repository object • TYPE_HASHTABLE: a property map type repository object (an array of objects) • TYPE_XMLDOCUMENT: an XML type repository object • TYPE_CONTAINER: a subdirectory

Usage

This method tests if the object with the specified visibility, owner, name, or type exists in the repository. Returns true if the object exists. When an object is saved, the specified owner and name of the object are appended to the repository path to the object. To add additional directories to the path, you can add "/" to the name. The same rule applies when you delete, load, list, rename or test the existence of an object. To test if a folder exists, set the type to TYPE_CONTAINER.

Examples

```
<blox:repository id="myRepoBlox" />
<% int visibility = myRepoBlox.VISIBILITY_APPLICATION;
   String owner = "admin";
   String name = "sales/westdata";
   int type = myRepoBlox.TYPE_TEXT;

   if (myRepoBlox.exists(visibility,owner,name,type))
     out.write("The object exists!");
   else
     out.write("The object does not exist!");
%>
```

See Also

"delete()" on page 639, "list()" on page 648, "load()" on page 649, "rename()" on page 652, "save()" on page 654, "search()" on page 656

getAllApplications()

Returns an array of containing the names of all of the applications that exist in DB2 Alphablox.

Data Sources

All

Syntax

Java Method

```
String[] getAllApplications();
            throws ServerBloxMissingResourceException,
                    ServerBloxException
String[] getAllApplications(String user);
            throws ServerBloxMissingResourceException,
                    ServerBloxException
```

where:

Argument	Default	Description
user	All users	When this argument is used, the method returns all the applications that are visible to the specified user.

getApplicationProperty()

Returns the value of the specified application property, or an empty String if no value has been set for the property.

Data Sources

All

Syntax

Java Method

```
String getApplicationProperty(String name);
            throws ServerBloxMissingResourceException,
                   ServerBloxException
```

where:

Argument	Default	Description
name	none	Name of an application property.

The following table shows the system-defined application properties. These application properties are defined in the DB2 Alphablox application definition. You can also add your own custom user properties, which can also be retrieved through the `getApplicationProperty` method.

Application Property	
ContextName	The application context name.
AutoSaveEnabled	The setting for autosave.
URL	The home page file specified in the application definition.
DisplayName	The name displayed on the Applications tab.
Description	The description in the application definition.
Support3xPageHandling	The setting for Alphablox 3 compatibility.

See Also

"getApplicationPropertyMap()" on page 642, "setApplicationProperty()" on page 657

getApplicationPropertyMap()

Returns all properties for the current application as a Java hash table.

Data Sources

All

Syntax

Java Method

```
java.util.Hashtable getApplicationPropertyMap()
            throws ServerBloxMissingResourceException,
                   ServerBloxException
```

Examples

```
getApplicationPropertyMap();
```

See Also

"getApplicationProperty()" on page 642, "setApplicationProperty()" on page 657

getApplicationStateNameAndDescription()

Returns a two-dimensional array of Strings containing the names and states for the specified visibility and scope.

Data Sources

All

Syntax

Java Method

```
String[][] getApplicationStateNameAndDescription(int visibility,
                                int scope)
               throws RepositoryIOException,
                      ServerBloxMissingResourceException,
                      InvalidRepositoryVisibilityException,
                      InvalidRepositoryScopeException,
                      ServerBloxException
```

where:

Argument	Default	Description
visibility	none	Visibility of the application state to retrieve: VISIBILITY_APPLICATION, VISIBILITY_GROUP, or VISIBILITY_PRIVATE
scope	none	The scope of the saved states to retrieve: SCOPE_APPLICATION or SCOPE_SINGLE_BLOX (i.e. bookmark scope).

getDataSourceNames()

Returns a list of all the valid data source names as a String array. Each element of the array is the name of a data source.

Data Sources

All

Syntax

Java Method

```
String[] getDataSourceNames();
          throws RepositorySecurityException,
                 ServerBloxException
```

getGroupNames()

Returns a list of all valid group names as a String array.

Data Sources

All

Syntax

Java Method

```
String[] getGroupNames();
       throws RepositorySecurityException,
              ServerBloxException
```

Usage

You must be logged in as a user who is assigned the `AlphabloxAdministrator` role to use this method. Otherwise, it will throw a repository security exception.

See Also

"getGroupProperty()" on page 644, "getUserNames()" on page 646, "getUserProperty()" on page 646

getGroupProperty()

Returns the value of the named property for this user group.

Data Sources

All

Syntax

Java Method

```
String getGroupProperty(String name);
          throws ServerBloxMissingResourceException,
                 ServerBloxException
```

where:

Argument	Default	Description
`name`	none	Name of the property.

Usage

If a user's access to the current application is not via a group, the `getGroupProperty()` method returns an empty String.

See Also

"getGroupPropertyMap()" on page 644, "setGroupProperty()" on page 658

getGroupPropertyMap()

Returns all properties for the current group as a Java hash table.

Data Sources

All

Syntax

Java Method

```
java.util.Hashtable getGroupPropertyMap();
             throws ServerBloxMissingResourceException,
                    ServerBloxException
```

If a user's access to the current application is not via a group, the method returns null.

Examples

```
getGroupPropertyMap();
```

See Also

"getGroupProperty()" on page 644, "setGroupProperty()" on page 658

getInstanceProperty()

Returns the value of the named property for this instance of the application for the current user.

Data Sources

All

Syntax

Java Method

```
String getInstanceProperty(String name);
             throws ServerBloxMissingResourceException,
                     ServerBloxException
```

where:

Argument	Default	Description
name	none	Name of the property.

Examples

```
getInstanceProperty("datasource");
```

See Also

"getInstancePropertyMap()" on page 645, "setInstanceProperty()" on page 658

getInstancePropertyMap()

Returns all of the application instance properties as a Java hash table.

Data Sources

All

Syntax

Java Method

```
java.util.Hashtable getInstancePropertyMap();
             throws ServerBloxMissingResourceException,
                     ServerBloxException
```

These properties are saved when the application is saved. Instance properties determine the behavior of the application for the current user only.

Examples

```
getInstancePropertyMap();
```

See Also

"getInstanceProperty()" on page 645

getServerProperty()

Returns the value for the specified server property.

Data Sources
All

Syntax
Java Method

```
String getServerProperty(String name);
```

where:

Argument	Default	Description
name	none	Name of the server property.

Usage
Examples of server properties are SMTPServer, ServletPrefix, and ServerCharacterEncoding. The property names correspond to the properties in the server.properties file:

```
<alphablox_dir>/repository/servers/<instance_name>/server.properties
```

getThemes()
Returns an array of theme names of the themes defined in the repository.

Data Sources
All

Syntax
Java Method

```
String[] getThemes();
```

getUserNames()
Returns a list of all valid users as a String array.

Data Sources
All

Syntax
Java Method

```
String[] getUserNames();
            throws RepositorySecurityException,
                    ServerBloxException
```

Usage
You must be logged in as a user assigned to the AlphabloxAdministrator role to use this method. Otherwise, it will throw a repository security exception.

See Also
"getGroupNames()" on page 643, "getGroupProperty()" on page 644, "getUserProperty()" on page 646

getUserProperty()
Returns the value of the named property for this user.

Data Sources
All

Syntax

Java Method

```
String getUserProperty(String name);
            throws ServerBloxMissingResourceException,
                   ServerBloxException
```

where:

Argument	Default	Description
`name`	none	Name of the property for which this method returns the value.

Usage

If the current user is guest, the `getUserProperty` method returns an empty `String`.

The following table shows the system-defined user properties. These user properties are defined in the profile for each user. You can also add your own custom user properties, which can also be retrieved through the `getUserProperty` method.

User Property	
`Description`	A textual description of the user.
`EMailAddress`	The user's email address.
`Name`	The login name defined for the user.
`ProperName`	The alternate name (proper name) defined for the user.
`CanEditProfile`	Allows the user to edit her user profile.

See Also

"getUserPropertyMap()" on page 647, "setUserProperty()" on page 659

getUserPropertyMap()

Returns all properties for the current user as a Java hash table.

Data Sources

All

Syntax

Java Method

```
java.util.Hashtable getUserPropertyMap();
             throws ServerBloxMissingResourceException,
                    ServerBloxException
```

Usage

If the current user is a guest, this method returns null.

Examples

```
getUserPropertyMap();
```

See Also

"getUserProperty()" on page 646, "setUserProperty()" on page 659

getUsersCurrentGroup()

Returns the name of the group through which the current user is accessing an application.

Data Sources

All

Syntax

Java Method

```
String getUsersCurrentGroup();
            throws ServerBloxMissingResourceException,
                    ServerBloxException
```

Usage

The `getUsersCurrentGroup` method returns an empty `String` in the following situations:

- The user is in a role, and the user has "better" access than the role provides.
- The application does not support roles.

Examples

```
getUsersCurrentGroup();
```

init()

This is a common Blox method. For a complete description, see "init()" on page 54.

killSession()

Kills the user's current session on the client, which causes all related server peers to be discarded.

Data Sources

All

Syntax

Java Method

```
void killSession();
    // throws RepositorySecurityException, ServerBloxException
```

Usage

This method kills the user's current session. The user can reload the page to be re-authenticated. If the cookie on the browser has not expired, the same authorization header is passed and the user is automatically re-authenticated.

To ensure that same authorization header is ignored so different users can log in and out without having to close and reopen a browser window, use logout().

Examples

```
killSession();
```

See Also

"logout()" on page 651.

list()

Lists the specified object from the DB2 Alphablox repository.

Data Sources
All

Syntax
Java Method

```
String[] list((int visibility, String owner, String name, int type);
               throws ServerBloxMissingResourceException,
                       InvalidRepositoryTypeException,
                       RepositoryIOException,
                       RepositorySecurityException,
                        ServerBloxException
```

where:

Argument	Default	Description
visibility	none	Visibility of the application state to be removed: VISIBILITY_APPLICATION, VISIBILITY_GROUP, or VISIBILITY_PRIVATE.
owner	none	The name of the user who owns the object.
name	none	The name of the object in the repository.
type	none	One of the following constants indicating the type of object: • TYPE_BINARY: a binary type repository object • TYPE_TEXT: a text type repository object • TYPE_HASHTABLE: a property map type repository object (an array of objects) • TYPE_XMLDOCUMENT: an XML type repository object • TYPE_CONTAINER: a subdirectory

Usage
This method lists all objects of the specified visibility, owner, name, or type in the repository. When an object is saved, the specified owner and name of the object are appended to the repository path to the object. To add additional directories to the path, you can add "/" to the name. The same rule applies when you delete, load, list, rename or test the existence of an object.

To list all directories under your specified directory path, set the type to TYPE_CONTAINER. To list all files of the specified type and visibility in your specified directory path, set owner and name to an empty string "".

Examples

```
<blox:repository id="myRepoBlox" />
<% int visibility = myRepoBlox.VISIBILITY_APPLICATION;
   String owner = "";
   String name = "";
   int type = myReporBlox.TYPE_TEXT
   String[] objects = myRepoBlox.list(visibility,owner,name,type);
%>
```

See Also
"delete()" on page 639, "list()" on page 648, "load()" on page 649, "rename()" on page 652, "save()" on page 654, "search()" on page 656

load()

Loads the specified object from the DB2 Alphablox repository.

Data Sources

All

Syntax

Java Method

```
Object load((int visibility, String owner, String name, int type);
               throws ServerBloxMissingResourceException,
                      InvalidRepositoryTypeException,
                      RepositoryIOException,
                      RepositorySecurityException,
                       ServerBloxException
```

where:

Argument	Default	Description
visibility	none	Visibility of the application state to be removed: VISIBILITY_APPLICATION, VISIBILITY_GROUP, or VISIBILITY_PRIVATE.
owner	none	The name of the user who owns the object.
name	none	The name of the object in the repository.
type	none	One of the following constants indicating the type of object: • TYPE_BINARY: a binary type repository object • TYPE_TEXT: a text type repository object • TYPE_HASHTABLE: a property map type repository object (an array of objects) • TYPE_XMLDOCUMENT: an XML type repository object

Usage

This method loads the object with the specified visibility, owner, name, and type from the repository for further processing. When an object is saved, the specified owner and name of the object are appended to the repository path to the object. To add additional directories to the path, you can add "/" to the name. The same rule applies when you delete, load, list, rename or test the existence of an object.

Examples

The following example demonstrates how an object of TYPE_TEXT is saved and later retrieved. It also demonstrates how to save the text object as an array of bytes to use the appropriate character encoding of the server.

```
<blox:repository id="myRepoBlox" />
<%
int vis = myRepoBlox.VISIBILITY_APPLICATION;
String owner = "admin";
String name = "sales/westdata";
int type = myRepoBlox.TYPE_TEXT;

// Assuming a string is to be saved in the repository. Here we are
// converting the string into bytes according to the default
// character encoding.
String text = "Some data to be stored as text in the repository"
myRepoBlox.save(vis,owner,name,text.getBytes(),myRepoBlox.TYPE_TEXT);

// To load the saved text object
byte[] bytes;
bytes=(byte[])myRepoBlox.load(vis,owner,name,myRepoBlox.TYPE_TEXT);

// We can now convert the byte array back into a string for further
```

```
// processing
String retrievedText = new String(bytes);
...
%>
```

See Also

"delete()" on page 639, "list()" on page 648, "list()" on page 648, "rename()" on page 652, "save()" on page 654, "search()" on page 656

logout()

Kills the DB2 Alphablox session and expires the cookie.

Data Sources

All

Syntax

Java Method

```
void logout(javax.servlet.http.HttpServletRequest request,
            javax.servlet.http.HttpServletResponse response);
            // throws ServerBloxException
```

where:

Argument	Description
request	The HTTP request object.
response	The HTTP response object.

Usage

This method calls killSession() first and then sends an expired cookie to force re-authentication. If users reloads the page, since the header information has expired, the browser will prompt the users to reenter their username and password in order to be authenticated. This is useful in cases where you want to build a login page to allow different users to log in and out without having to close and reopen a browser window.

Note that the expired cookie sent by this method is for DB2 Alphablox sessions only. If the user, after logging out, goes to a different page that does not contain any Blox component and therefore no DB2 Alphablox session is created, he will not be authenticated.

Note: This method is for Tomcat only.

Examples

```
<%@ taglib uri="bloxtld" prefix="blox" %>
...
<blox:repository id="repositoryBlox" render="none" />
<% String userName= repositoryBlox.getUserProperty("name");
   repositoryBlox.logout(request, response);
%>
<h1> User <%=userName%> has logged out successfully. </h1>
```

See Also

"killSession()" on page 648

readFragment()

Returns the contents of the specified repository fragment as a String.

Data Sources
All

Syntax
Java Method

```
String readFragment(String fragmentName);
```

where:

Argument	Default	Description
fragmentName	none	The name of the fragment to retrieve.

Examples

```
<blox:repository id="myRepositoryBlox" />
<%= myRepositoryBlox.readFragment("myHTMLFragment") %>
```

Usage
This method is for converting HTML fragments in Alphablox 3 to DB2 Alphablox.

rename()
Renames an object in the DB2 Alphablox repository.

Data Sources
All

Syntax
Java Method

```
void rename(int visibility, String owner, String oldname,
                String newname, int type);
            throws ServerBloxMissingResourceException,
                   InvalidRepositoryTypeException,
                   RepositoryIOException,
                   RepositorySecurityException,
                    ServerBloxException
```

where:

Argument	Default	Description
visibility	none	Visibility of the application state to be removed: VISIBILITY_APPLICATION, VISIBILITY_GROUP, or VISIBILITY_PRIVATE.
owner	none	The name of the user who owns the object.
oldName	none	Name of saved object.
newName	none	New name of saved object.
type	none	One of the following constants indicating the type of object: • TYPE_BINARY: a binary type repository object • TYPE_TEXT: a text type repository object • TYPE_HASHTABLE: a property map type repository object (an array of objects) • TYPE_XMLDOCUMENT: an XML type repository object • TYPE_CONTAINER: a subdirectory

Usage

This method renames the specified object in the repository. When an object is saved, the specified `owner` and `name` of the object are appended to the repository path to the object. To add additional directories to the path, you can add "/" to the `name`. The same rule applies when you delete, load, list, rename or test the existence of an object. To rename a directory, set the `type` to `TYPE_CONTAINER`.

Examples

```
<blox:repository id="myRepoBlox" />
<% int visibility = myRepoBlox.VISIBILITY_APPLICATION;
   String owner = "admin";
   String oldname = "westdata";
   String newname = "west_sales";
   int type = myRepoBlox.TYPE_CONTAINER;
   myRepoBlox.rename(visibility,owner,oldname,newname,type);
%>
```

See Also

"delete()" on page 639, "list()" on page 648, "list()" on page 648, "load()" on page 649, "save()" on page 654, "search()" on page 656

renameApplicationState()

Renames a saved application state.

Data Sources

All

Syntax

Java Method

```
void renameApplicationState(String oldName, String newName,
                    int visibility, int scope, String description);
               throws RepositoryIOException,
                       ServerBloxMissingResourceException,
                       InvalidRepositoryVisibilityException,
                       InvalidRepositoryScopeException,
                        ServerBloxException
```

where:

Argument	Default	Description
`oldName`	none	Name of saved application state to be changed.
`newName`	none	New name of saved application state.
`visibility`	none	Visibility of the application state to be renamed: `VISIBILITY_APPLICATION`, `VISIBILITY_GROUP`, or `VISIBILITY_PRIVATE`
`scope`	none	The scope of the saved application: `SCOPE_APPLICATION`, `SCOPE_SINGLE_BLOX` (for example, bookmark scope).
`description`	none	A textual description of the saved state.

Usage

If the application state is renamed to an already existing state, the existing state is overwritten with the renamed state.

restoreApplicationState()

Restores either the named application state, or the application state most recently saved or restored by the current user.

Data Sources
All

Syntax
Java Method

```
void restoreApplicationState(String name, int visibility,
                             int scope);
                 throws RepositoryIOException,
                        ServerBloxMissingResourceException,
                        InvalidRepositoryVisibilityException,
                        InvalidRepositoryScopeException,
                         ServerBloxException
```

where:

Argument	Default	Description
name	none	Name of the saved application state to be removed.
visibility	none	Visibility of the application state to be restored: VISIBILITY_APPLICATION, VISIBILITY_GROUP, or VISIBILITY_PRIVATE.
scope	none	The scope of the saved application: SCOPE_APPLICATION, SCOPE_SINGLE_BLOX (for example, bookmark scope).

Examples
The following code snippet uses a URL parameter to retrieve a stored application state. It uses the request Object to get the URL parameter value, then uses the RepositoryBlox restoreApplicationState() method to set the application state for the requested page.

```
<blox:repository id="myRepoBlox" />
<%
  String savedState=request.getParameter("savedstate");

if (savedState != null)
  myRepoBlox.restoreApplicationState(savedState,
     myRepoBlox.VISIBILITY_PRIVATE, myRepoBlox.SCOPE_APPLICATION);
%>
```

save()
Saves an object to the DB2 Alphablox repository.

Data Sources
All

Syntax
Java Method

```
void save(int visibility, String owner, String name, Object object,
                        int type);
                throws ServerBloxMissingResourceException,
                       InvalidRepositoryTypeException,
                       RepositoryIOException,
                       RepositorySecurityException,
                        ServerBloxException
```

where:

Argument	Default	Description
visibility	none	Visibility of the application state to be saved: VISIBILITY_APPLICATION, VISIBILITY_GROUP, or VISIBILITY_PRIVATE.
owner	none	The name of the user who owns the object.
name	none	The name of the object in the repository.
object	none	The Object to save to the repository.
type	none	One of the following constants indicating the type of object: • TYPE_BINARY: a binary type repository object • TYPE_TEXT: a text type repository object • TYPE_HASHTABLE: a property map type repository object (an array of objects) • TYPE_XMLDOCUMENT: an XML type repository object

Usage

This method saves the specified object with the specified visibility, owner, name, and type into the repository. When an object is saved, the owner and name of the object specified are appended to the repository path to the object. To add additional directories to the path, you can add "/" to the name. The same rule applies when you delete, load, list, rename or test the existence of an object.

You can save and load an object as one of the four supported types: TYPE_BINARY, TYPE_TEXT, TYPE_HASHTABLE, and TYPE_XMLDOCUMENT. This gives you greater flexibility in the kind of data you can store in and retrieve from the repository.

Examples

The following example demonstrates how an object of TYPE_TEXT is saved and later retrieved. It also demonstrates how to save the text object as an array of bytes to use the appropriate character encoding of the server.

```
<blox:repository id="myRepoBlox" />
<%
int vis = myRepoBlox.VISIBILITY_APPLICATION;
String owner = "admin";
String name = "sales/westdata";
int type = myRepoBlox.TYPE_TEXT;

// Assuming a string is to be saved in the repository. Here we are
// converting the string into bytes according to the default
// character encoding.
String text = "Some data to be stored as text in the repository"
myRepoBlox.save(vis,owner,name,text.getBytes(),myRepoBlox.TYPE_TEXT);

// To load the saved text object
byte[] bytes;
bytes=(byte[])myRepoBlox.load(vis,owner,name,myRepoBlox.TYPE_TEXT);

// We can now convert the byte array back into a string for further
// processing
String retrievedText = new String(bytes);
...
%>
```

See Also

"delete()" on page 639, "list()" on page 648, "list()" on page 648, "load()" on page 649, "rename()" on page 652, "search()" on page 656

saveApplicationState()

Saves the state of the current application with the specified arguments.

Data Sources

All

Syntax

Java Methods

```
void saveApplicationState(String name, int visibility, int scope,
                          String description, boolean hideFromUser);
                throws RepositoryIOException,
                       ServerBloxMissingResourceException,
                       InvalidRepositoryVisibilityException,
                       InvalidRepositoryScopeException,
                        ServerBloxException
```

where:

Argument	Default	Description
name	none	Name of the saved application state to be saved.
visibility	none	Visibility of the application state to be saved: VISIBILITY_APPLICATION, VISIBILITY_GROUP, or VISIBILITY_PRIVATE.
scope	none	The scope of the saved application: SCOPE_APPLICATION, SCOPE_SINGLE_BLOX (for example, bookmark scope).
description	none	A textual description of the saved state.
hideFromUser	none	A boolean argument. When set to true, this state does not show up in the persistence user interface.

search()

Searches the DB2 Alphablox repository for objects meeting the search criteria specified, and returns the names of the repository objects as a String array.

Data Sources

All

Syntax

Java Methods

```
String[] search(int visibility, String owner, String path,
                String name, int type, int depth);
         throws ServerBloxMissingResourceException,
                InvalidRepositoryTypeException,
                RepositoryIOException,
                RepositorySecurityException,
                 ServerBloxException
```

where:

Argument	Default	Description
visibility	none	Visibility of the objects to search for: VISIBILITY_APPLICATION, VISIBILITY_GROUP, or VISIBILITY_PRIVATE.
path	none	A String to add to the end of the path when searching through the repository.
name	none	Name of the object to search for
type	none	One of the following constants indicating the type of object: • TYPE_BINARY: a binary type repository object • TYPE_TEXT: a text type repository object • TYPE_HASHTABLE: a property map type repository object (an array of objects) • TYPE_XMLDOCUMENT: an XML type repository object • TYPE_CONTAINER: a subdirectory
depth	none	The number of subdirectories from the specified path that should be searched.

Usage

This method searches the repository for objects of specified visibility, path, name, type, or number of subdirectories. When an object is saved, the specified owner and name of the object are appended to the repository path to the object. To add additional directories to the path, "/" can be added to the name.

Examples

The following example searches for all objects in the repository that are of type TYPE_TEXT, VISIBILITY_APPLICATION visibility, 2 levels of subdirectories below the specified subfolder/anotherfolder directory, regardless of their owners or names:

```
<blox:repository id="myRepoBlox" />
<% int vis = myRepoBlox.VISIBILITY_APPLICATION;
   String owner = "";
   String path = "subfolder/anotherfolder"
   String name = "";
   int type = myRepoBlox.TYPE_TEXT;

   String[] objects;
   objects=myRepoBlox.search(vis,owner,path,name,repoBlox.TYPE_TEXT,2);
%>
```

See Also

"delete()" on page 639, "list()" on page 648, "list()" on page 648, "load()" on page 649, "rename()" on page 652, "save()" on page 654

setApplicationProperty()

Sets the value of the named application property. Application properties can affect the behavior of an application for all users.

Data Sources

All

Syntax

Java Method

```
void setApplicationProperty(String name, String value);
            throws ServerBloxMissingResourceException,
                   ServerBloxException
```

where:

Argument	Default	Description
name	none	Name of property to set.
value	none	Value to assign to specified property.

Examples

```
setApplicationProperty("datasource", "TBC");
```

See Also

"getApplicationProperty()" on page 642, "getApplicationPropertyMap()" on page 642

setGroupProperty()

Sets the value of the named property for this user group.

Data Sources

All

Syntax

Java Method

```
void setGroupProperty(String name, String value);
            throws ServerBloxMissingResourceException,
                   ServerBloxException
```

where:

Argument	Default	Description
name	none	Name of property to set.
value	none	Value to assign to specified property.

Usage

If a user's access to the current application is not via a group, the `setGroupProperty` method has no effect.

See Also

"getGroupProperty()" on page 644, "getGroupPropertyMap()" on page 644

setInstanceProperty()

Sets the value of the named property for this instance of the application for the current user.

Data Sources

All

Syntax

Java Method

```
void setInstanceProperty(String name, String value);
            throws ServerBloxMissingResourceException,
                   ServerBloxException
```

where:

Argument	Default	Description
`name`	none	Name of property to set.
`value`	none	Value to assign to specified property.

Examples

```
setInstanceProperty("dataSourceName", "TBC");
```

See Also

"getInstanceProperty()" on page 645, "getInstancePropertyMap()" on page 645

setUserProperty()

Sets the value of the named property for this user.

Data Sources

All

Syntax

Java Method

```
void setUserProperty(String name, String value);
            throws ServerBloxMissingResourceException,
                   ServerBloxException
```

where:

Argument	Default	Description
`name`	none	Name of property to set.
`value`	none	Value to assign to specified property.

See Also

"getUserProperty()" on page 646, "getUserPropertyMap()" on page 647.

Chapter 21. ResultSetBlox Reference

This chapter contains reference material for the ResultSetBlox. For general reference information about Blox, see Chapter 3, "General Blox Reference Information," on page 17. For information on how to use this reference, see Chapter 1, "Using This Reference," on page 1.

ResultSetBlox Overview

ResultSetBlox can be attached to a DataBlox to extend the normal functions associated with a JDBC data source. You can arbitrarily push a custom ResultSet into a DataBlox using ResultSetBlox. Or you can attach a method to the Blox to intercept queries in the associated DataBlox and to return arbitrary ResultSet objects to the DataBlox.

ResultSetBlox has `aresultSetHandler` property that lets you specify your result set handler class as follows:

```
<blox:data id="rsData"
   dataSourceName="canned"
   connectOnStartup="false"
/>
...
<blox:resultSet id="rset1"
   dataBlox="<%=rsData%>"
   resultSetHandler="<%=new TupleResultSet()%>"
/>
```

where `rsData` is a previously defined DataBlox, and TupleResultSet is your result set handler. This handler should implement the IResultSetHandler interface, which provides an `executeQuery()` method that returns a java.sql.ResultSet object, and a `fetchComplete()` method that terminates the connection when the data from the result set is fetched:

```
<%@ page import="com.alphablox.blox.*,
                 java.sql.*" %>
...

<%!
 public class TupleResultSet implements IResultSetHandler {
    ...
    // Store your custom query into the String myQuery

    Connection conn = null;
    public ResultSet executeQuery(String myQuery) throws Exception {
       //code here to get connected
    }

    public void fetchComplete() throws Exception {
       //close the connection
```

```
        conn.close();
        conn = null;
    }
%>
```

Note the following:

- The DataBlox in the above example does not initially connect to any data source (`connectionOnStartup = "false"`). This is a useful technique in cases where we do not want to get the initial result set, and we want users to make some selections in order for us to construct the query string dynamically and to issue the query using JDBC connection.
- You need to add the import statement for java.sql.*.
- Depending on the type of data you are getting from the result set, there is a minimum set of APIs you need to implement on java.sql.ResultSet. These APIs are listed in the next section.

For a complete live example, see the Blox Sampler's Retrieving Data section.

Minimum APIs to Implement on ResultSet

The following is a list of the minimum set of APIs you need to implement on java.sql.ResultSet, depending on the type of data you are getting from the result set.

- `next()` : void
- Getter methods to implement depending on the data types your result set returns:
  - `getInt(int)`: Integer
  - `getBoolean(int)`: Boolean
  - `getBigDecimal(int)`: BigDecimal
  - `getFloat(int)`: Float
  - `getDouble(int)`: Double
  - `getString(int)`: String
  - `getDate(int)`: Date
  - `getObject(int)`: Object
  - `getMetaData()`: java.sql.ResultSetMetaData

If the returned result set is of type java.sql.ResultSetMetaData, you should implement the following methods:

- `getColumnCount()` : int
- `getColumnType(int)` : int
- `getScale(int)` : int
- `getPrecision(int)` : int
- `getColumnName(int)` : String
- `getColumnLabel(int)`: String
- `getColumnTypeName(int)` : String
- `getColumnType(int)` : int

ResultSetBlox JSP Custom Tag Syntax

The Alphablox Tag Libraries provides custom tags to use in a JSP page for creating each Blox. This section describes how to create the custom tag to create a ResultSetBlox. For a copy and paste version of the tag with all the attributes, see "ResultSetBlox JSP Custom Tag" on page 890.

Syntax

```
<blox:resultSet
    [attribute="value"] >
</blox:resultSet>
```

where:

attribute is one of the attributes listed in the attribute table.

value is a valid value for the attribute.

and where the attributes are one of the following:

Attribute
id
bloxName
dataBlox
resultSetHandler

Usage

Each custom tag can have one or more attributes, separated by one or more space or new line characters. Any extra space or new line characters are ignored. For readability, attributes can each go on a separate line with the same indenting. You can substitute the closing `</blox:resultSet>` tag using the shorthand notation, closing the tag at the end of the attribute list that looks as follows:

```
id="myResultSet" />
```

Examples

```
<blox:resultSet id="myResultSet" />
```

ResultSetBlox Properties and Methods Cross Reference Table

The following tables list the unique ResultSetBlox properties and their corresponding methods. For a listing of common properties and methods, see "Common Blox Properties and Methods by Category" on page 29.

Property	Java Method
dataBlox	getDataBlox() setDataBlox()
resultSetHandler	getResultSetHandler() setResultSetHandler()
	detachDataBlox()
	loadResultSet()

ResultSetBlox Properties and Associated Methods

This section describes the properties supported by ResultSetBlox and the methods associated with those properties. The properties are listed alphabetically by property name. All of the ResultSetBlox properties are common to multiple Blox and they are listed but not described in this section. For complete descriptions of common Blox properties, see "Properties and Associated Methods Common to Multiple Blox" on page 32. For a list of ResultSetBlox methods with which no properties are associated, see "ResultSetBlox Methods" on page 665.

id

This is a common Blox tag attribute. For a complete description, see "id" on page 39.

bloxName

This is a common Blox tag attribute. For a complete description, see "bloxName" on page 35.

dataBlox

The DataBlox associated with this ResultSetBlox.

Data Sources

Relational

Syntax

JSP Tag Attribute

```
dataBlox="dataBlox"
```

Java Methods

```
DataBlox getDataBlox();
void setDataBlox(DataBlox dataBlox);
```

where:

Argument	Default	Description
dataBlox		The DataBlox this ResultSetBlox is attached to.

resultSetHandler

The handler that implements the `executeQuery()` method to take a query and return a result set.

Data Sources

Relational

Syntax

JSP Tag Attribute

```
resultSetHandler="handler"
```

Java Methods

```
IResultSetHandler getResultSetHandler();
void setResultSetHandler(IResultSetHandler handler);
```

where:

Argument	Default	Description
handler		The handler to implement in order to execute a query.

Usage
This handler needs to implement the methods in IResultSetHandler. See "IResultSetHandler Methods" on page 666 for the methods to implement.

ResultSetBlox Methods

This section describes ResultSetBlox methods that are not associated with a specific property. For the syntax and descriptions of ResultSetBlox methods that have a property associated with them, see "ResultSetBlox Properties and Associated Methods" on page 664.

detachDataBlox()

Removes the connection to and the association with a DataBlox by setting it to null.

Data Sources
Relational

Syntax
Java Method

```
public void detachDataBlox();
     // throws ServerBloxException
```

getProperty()

This is a common Blox method. For a complete description, see "getProperty()" on page 53.

init()

This is a common Blox method. For a complete description, see "init()" on page 54.

loadResultSet()

Sets the relational result set on the associated DataBlox.

Data Sources
Relational

Syntax
Java Method

```
void loadResultSet(java.sql.ResultSet resultSet);
```

where:

Argument	Default	Description
resultSet		A java.sql.ResultSet object.

setProperty()

This is a common Blox method. For a complete description, see “setProperty()” on page 61.

IResultSetHandler Methods

This section describes methods available in the IResultSetHandler interface. This interface lets you run any relational query using code as long as that code returns an implementation of java.sql.ResultSet.

executeQuery()

Fetches a relational result set.

Data Sources

Relational

Syntax

Java Method

```
java.sql.ResultSet executeQuery(java.lang.String query);
                              // throws java.lang.Exception
```

fetchComplete()

This method is called after all data from the result set returned from `executeQuery()` are fetched.

Data Sources

Relational

Syntax

Java Method

```
void fetchComplete();
     //throws java.lang.Exception
```

Chapter 22. StoredProceduresBlox Reference

This chapter contains reference material for StoredProceduresBlox and related objects in the `com.alphablox.blox.data.rdb.storedprocedure` package for using stored procedures.

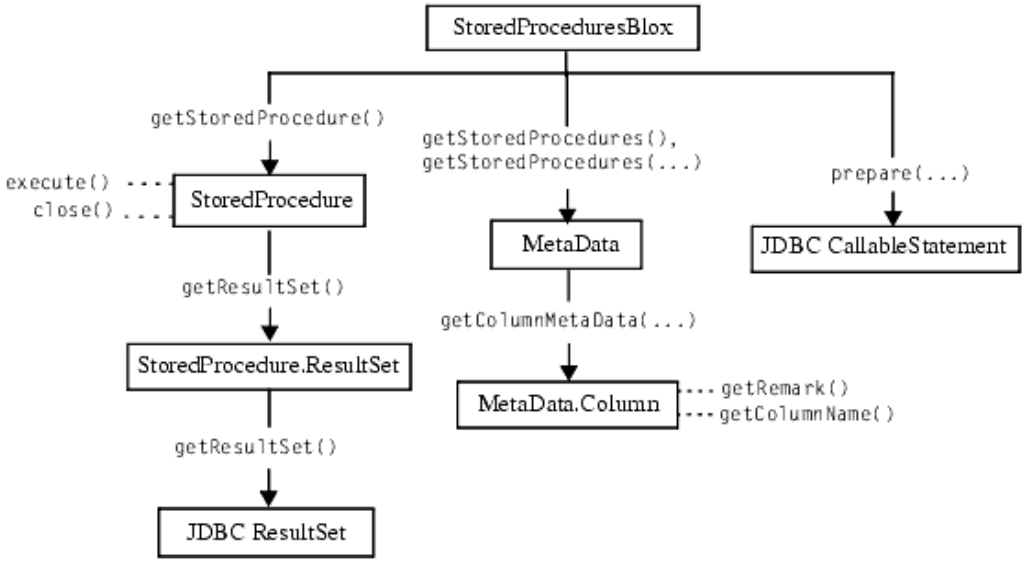
StoredProceduresBlox Overview

StoredProceduresBlox is the starting point for using relational database stored procedures. It allows you to create a connection to a database and prepare a stored procedure statement. Once the correct DB2 Alphablox data source and any other connection parameters are set, you can:

- use the `prepare(...)` method to return a JDBC CallableStatement object, which can be used to set up any stored procedure parameters necessary to execute the stored procedure
- use the `getStoredProcedure()` method to access the current StoredProcedure object; you can then execute the stored procedure, get to the ResultSet of the executed stored procedure, or access the JDBC ResultSet
- use the `getStoredProcedures()` or `getStoredProcedures(...)` methods to return one or more MetaData objects that give you access to the individual parameters

The StoredProcedure object and the MetaData object are separate classes in the `com.alphablox.blox.data.rdb.storedprocedure` package. By having separate objects for StoredProcedure and MetaData from StoredProceduresBlox, you can prepare a stored procedure once and then execute it multiple times. Even though stored procedure parameters can be altered between executions, you can enhance the performance by not preparing the stored procedures at every execution.

The following diagram shows the object hierarchy of stored procedure related objects.



Because the StoredProcedure and MetaData objects are in a separate package, you must use the following JSP import statement at the beginning of any JSP file to use any of the APIs in these objects:

```
<%@ page import="com.alphablox.blox.data.rdb.storedprocedure.*" %>
```

Note: JDBC Stored procedures are supported for IBM DB2 UDB, Sybase, Oracle, and Microsoft SQL Server databases.

Note the following when using the StoredProcedure object to execute a prepared stored procedure:

- If a DataBlox is used to display information from a stored procedure, the DataBlox must be separately connected to the same data source as StoredProceduresBlox.
- If a DataBlox is used to display information from a stored procedure and the stored procedure also has output parameters, the result set must first be used before getting the output parameters. This is a JDBC restriction.
- If the stored procedure has input and output parameters, you should use `StoredProceduresBlox.prepare(...)` to get the JDBC CallableStatement object. This object allows you to get and set input and output parameters on the stored procedure.
- Once the stored procedure has been executed and any output parameters or result sets are used, you need to call the `StoredProceduresBlox.disconnect()` to disconnect and free up any resources. If you want to keep the connection to the data base open, call `StoredProceduresBlox.close()` to free up any resources used.
- If a DataException is thrown, extra information might be available as a SQLException by looking at `DataException.getNestedException()`.

Once the stored procedure is executed, it returns a StoredProcedure.ResultSet object, which gives you access to the JDBC ResultSet object. If you need to use the JDBC ResultSet object directly, use the `ResultSet.getResultSet()` method to get to this object.

It is recommended that you also import the java.sql package when working with stored procedures, so your JSP files should import two packages:

```
<%@ page import="com.alphablox.blox.data.rdb.storedprocedure.*" %>
<%@ page import="java.sql.*" %>
```

StoredProceduresBlox JSP Custom Tag Syntax

The Alphablox Tag Libraries provides custom tags to use in a JSP page for creating each Blox. This section describes how to create the custom tag to create a StoredProceduresBlox.

Syntax

```
<blox:storedProcedures
    [attribute="value"] >
</blox:storedProcedures>
```

where:

attribute is one of the attributes listed in the attribute table.

value is a valid value for the attribute.

and where the attributes are one of the following:

Attribute
id
bloxName

Usage

Each custom tag can have one or more attributes, separated by one or more space or new line characters. Any extra space or new line characters are ignored. For readability, attributes can each go on a separate line with the same indenting.

You can substitute the closing `</blox:storedProcedures>` tag using the shorthand notation: closing the tag at the end of the attribute list that looks as follows:

```
id="myBlox" />
```

Examples

```
<blox:storedProcedures
     id="namedStoredProceduresBlox" />
```

StoredProceduresBlox Examples

This section includes six examples that demonstrates the use of StoredProceduresBlox and its associated objects. For more examples, see the Javadoc.

- "Example 1: Connecting to the data source without a DataBlox" on page 670
- "Example 2: Using the StoredProceduresBlox to connect the data source for use with DataBlox" on page 670
- "Example 3: Getting a list of stored procedures whose name matches a specified pattern" on page 670
- "Example 4: Getting a list of all parameters for each stored procedure" on page 671
- "Example 5: Executing a stored procedure that has one input parameter and two output parameters" on page 672

• "Example 6: Setting a stored procedure result set to a DataBlox" on page 672

Example 1: Connecting to the data source without a DataBlox

This example demonstrates how to connect to the data source without a DataBlox as you may only want to get the parameters or run an INSERT SQL stored procedure that does not require a DataBlox.

```
<%@ page import="com.alphablox.blox.StoredProceduresBlox" %>
<%@ page import="com.alphablox.blox.data.rdb.storedprocedure.*" %>
<%@ page import="java.sql.*" %>
<%@ taglib uri="bloxtld" prefix="blox" %>

<blox:storedProcedures id="mySP"/>
<%
    mySP.setDataSourceName("sales");
    mySP.connect();
%>
```

Example 2: Using the StoredProceduresBlox to connect the data source for use with DataBlox

This example demonstrates how the DataBlox used to display information from a stored procedure needs to be separately connected to the same data source as StoredProceduresBlox.

```
<%@ page import="com.alphablox.blox.StoredProceduresBlox" %>
<%@ page import="com.alphablox.blox.data.rdb.storedprocedure.*" %>
<%@ page import="java.sql.*" %>
<%@ taglib uri="bloxtld" prefix="blox"%>

<blox:storedProcedures id="mySP"/>

<blox:data id="myDataBlox" visible="false"/>

<%
    myDataBlox.setDataSourceName("sales-sql");
    myDataBlox.connect();
    mySP.setDataSourceName("sales-sql");
    mySP.connect();
%>
```

Example 3: Getting a list of stored procedures whose name matches a specified pattern

This example demonstrates how to use the `getStoredProcedures(...)` method to get a list of stored procedures whose name starts with "procedure". This method returns an array of MetaData objects. The MetaData object contains information on the parameters for each stored procedure.

```
<%@ page import="com.alphablox.blox.StoredProceduresBlox" %>
<%@ page import="com.alphablox.blox.data.rdb.storedprocedure.*" %>
<%@ page import="java.sql.*" %>
<%@ taglib uri="bloxtld" prefix="blox"%>

<blox:storedProcedures id="mySP"/>
<%
    mySP.setDataSourceName("sales-sql");
    mySP.connect();
    MetaData procedures[] =
        mySP.getStoredProcedures("procedure%");
%>
<%
   if (procedures.length == 0) {
%> <strong>No procedures found.</strong> <%
} %>
```

Through the MetaData object, you can then access the individual parameter for a specified stored procedure.

Example 4: Getting a list of all parameters for each stored procedure

This example demonstrates how to use the MetaData object to get to each stored procedure and the parameters for each stored procedure. This example assumes you already have a MetaData object returns as shown in the previous example:

```
MetaData procedures[] =
        mySP.getStoredProcedures("procedure%");
```

We will now list each stored procedure and its catalog, schema, name, and remark information in a table:

```
<table border="1" >
<tr><th colspan="4">Stored Procedure Information</th></tr>
<tr><th>Catalog</th><th>Schema</th><th>Name</th><th>Remarks</th></tr>
<%
   for (int i = 0; i < procedures.length; i++) {
      String catalog = procedures[i].getCatalog();
      String schema = procedures[i].getSchema();
      String name = procedures[i].getName();
      String rem = procedures[i].getRemark();
      String type = null;
   %>
   <tr><td><%= catalog %></td>
       <td><%= schema %></td>
       <td><%= name %></td>
       <td><%= rem %></td></tr>
   <%
   }
%>
</table>
```

We can also get the detail of each parameter for each stored procedure:

```
//for each of the stored procedure, we will get the MetaData.Column //
object which contains the detail of the parameters
<%
for (int spCount = 0; spCount < procedures.length; spCount++) {
   String currProcedure = procedures[spCount].getName();
   MetaData.Column cMeta[] =    procedures[spCount].getColumnMetaData();%>

   //for the current stored procedure, we will get the list the
   //detail for each parameter in a table

   <table border="1">
   <tr><th colspan="7">Stored Procedure Params for
   <%=currProcedure %></th></tr>
   <tr><th>Catalog</th><th>Schema</th><th>Name</th><th>Column Name</
th><th>Type</th><th>Type Name</th><th>Remark</th></tr>

   //Iterate through the parameters in the current stored procedure
   <% for (int i = 0; i < cMeta.length; i++) {
       String catalog = cMeta[i].getCatalog();
       String schema = cMeta[i].getSchema();
       String name = cMeta[i].getName();
       String colName = cMeta[i].getColumnName();
       short type = cMeta[i].getType();
       String typeName = cMeta[i].getTypeName();
         String remark = cMeta[i].getRemark();
   %><tr><td><%= catalog %></td>
         <td><%= schema %></td>
         <td><%= name %></td>
```

```
        <td><%= colName %></td>
        <td><%= type %></td>
        <td><%= typeName %></td>
        <td><%= remark %></td></tr><%
   } %>
   </table>
<% }
%>
```

Example 5: Executing a stored procedure that has one input parameter and two output parameters

This example demonstrates how to the `prepare()` method to return a JDBC CallableStatement object that you can use to execute a stored procedure with input and output parameters.

```
<%@ page import="com.alphablox.blox.data.rdb.storedprocedure.*" %>
<%@ page import="com.alphablox.blox.data.rdb.*" %>
<%@ page import="com.alphablox.blox.StoredProceduresBlox" %>
<%@ page import="java.sql.*" %>
<%@ taglib uri="bloxtld" prefix="blox"%>

<blox:storedProcedures id="mySP"/>

<%
    mySP.setDataSourceName("storeSales");
    mySP.connect();

    // param 1 is an integer output, param 2 is a string input,
    // param 3 is a string output
    CallableStatement cstmt = mySP.prepare("{call a_procedure(?, ?, ?)}");
    cstmt.setString(2, "users/admin%");
    cstmt.registerOutParameter(1, Types.INTEGER);
    cstmt.registerOutParameter(3, Types.VARCHAR);
    mySP.execute();
    int out1 = cstmt.getInt(1);
    String out3 = cstmt.getString(3);
%>
...

<!-- Closes all resources associated with executing the stored procedure -->
<%
    mySP.close();
%>
...
<!--Disconnects from the data source -->
<%
    mySP.disconnect();
%>
```

Example 6: Setting a stored procedure result set to a DataBlox

This example demonstrates how to get a stored procedure result set to a DataBlox.

```
<%@ page import="com.alphablox.blox.data.rdb.*" %>
<%@ page import="com.alphablox.blox.data.rdb.storedprocedure.*" %>
<%@ page import="com.alphablox.blox.StoredProceduresBlox" %>
<%@ page import="java.sql.*" %>
<%@ taglib uri="bloxtld" prefix="blox"%>

<blox:storedProcedures id="mySP"/>

<blox:data id="myDataBlox" visible="false" />
<%
    myDataBlox.setDataSourceName("sales-sql");
```

```
    myDataBlox.connect();
    mySP.setDataSourceName("sales-sql");
    mySP.connect();
    mySP.prepare("{call a_procedure}");
    mySP.execute();
    mySP.loadResultSet(myDataBlox, 1);
%>
```

Properties and Methods Cross Reference

This section lists the unique properties and methods for all objects related to StoredProceduresBlox:

StoredProceduresBlox

The following table shows the unique StoredProceduresBlox properties and methods. For lists of properties and methods common to several Blox, see "Common Blox Properties and Methods by Category" on page 29.

Properties	Methods
catalog	
	getCatalog() setCatalog()
connection	
	getConnection()
dataSourceName	
	getDataSourceName() setDataSourceName()
password	
	getPassword() setPassword()
schema	
	getSchema() setSchema()
storedProcedures	
	getStoredProcedures()
userName	
	getUserName() setUserName()
	connect()
	close()
	disconnect()
	execute()
	loadResultSet()

	prepare()

The StoredProcedure Object

The following properties and methods are available on the StoredProcedure object. To access this object from StoredProceduresBlox, use the StoredProceduresBlox.getStoredProcdure(...) method.

Properties	Methods
callableStatement	
	getJDBCCallableStatement()
resultSet	
	getResultSet()
	close()
	execute()

The StoredProcedure class has an inner class object called ResultSet. The ResultSet object represents a result set from the execution of a stored procedure. Use ResultSet to set the results from a stored procedure to a DataBlox. See "StoredProceduresBlox Overview" on page 667 for details on the object hierarchy and how to access the ResultSet object.

The StoredProcedure.ResultSet Inner Class

The following methods are available on the ResultSet object. You can access this object from StoredProceduresBlox using the StoredProceduresBlox.getStoredProcdure(...).getResultSet() method.

Methods
getResultSet()
loadResultSet()
useResultSet()

The MetaData Object

The MetaData object contains information on a specific parameter in a stored procedure. You can access the MetaData object via the StoredProceduresBlox.getStoredProcedures(...).

Properties	Methods
catalog	
	getCatalog()
columnMetaData	
	getColumnMetaData()
name	
	getName()
remark	
	getRemark()
schema	
	getSchema()

type
getType()

The MetaData.Column Class

The following methods are available on the MetaData.Column object, which contains information on a single stored procedure parameter. To get to this object from StoredProceduresBlox, use `StoredProceduresBlox.getStoredProcedures(...).getColumnMetaData()`.

Methods
getCatalog()
getColumnName()
getDataType()
getLength()
getName()
getNullable()
getPrecision()
getRadix()
getRemark()
getScale()
getSchema()
getType()
getTypeName()

StoredProceduresBlox Properties and Associated Methods

This section describes the properties supported by StoredProceduresBlox and the methods associated with those properties. The properties are listed alphabetically by property name. For a list of StoredProceduresBlox methods with which no properties are associated, see "StoredProceduresBlox Methods" on page 679.

bloxName

This is a common Blox property. For a complete description, see "bloxName" on page 35.

catalog

The catalog name.

Data Sources

Relational

Syntax

Java Methods

```
String getCatalog();
void setCatalog(String catalog);
```

where:

Argument	Default	Description
`catalog`	`null`	The catalog name.

Usage
The `setCatalog()` method sets the data source's catalog with a different name before connecting.

connection
Gets the JDBC Connection object.

Data Sources
Relational

Syntax
Java Method

```
java.sql.Connection getConnection();
      //throws DataException
```

Usage
This `getConnection()` method can be helpful if any operation needs to be done on the Connection object used by the Stored Procedure, such as running many stored procedures in a single transaction. Throws `DataException` if an DB2 Alphablox specific database access error occurs.

dataSourceName
The name of the predefined DB2 Alphablox data source.

Data Sources
Relational

Syntax
Java Methods

```
String getDataSourceName();
void setDataSourceName(String dataSourceName);
```

where:

Argument	Default	Description
`dataSourceName`	`null`	The name of a predefined DB2 Alphablox data source to use.

password
The password to use.

Availability
Relational

Syntax
Java Methods

```
String getPassword();
void setPassword(String password);
```

where:

Argument	Default	Description
password	null	The password to use

Usage
The setPassword() method overrides the setting in the DB2 Alphablox data source definition.

Examples
See "Example 1: Connecting to the data source without a DataBlox" on page 670

schema
The name of the schema.

Data Sources
Relational

Syntax
Java Methods

```
String getSchema();
void setSchema(String schema);
```

where:

Argument	Default	Description
schema	null	The name of the schema

Usage
Use this setSchema() method to override the setting in the DB2 Alphabloxdata source definition before connecting to the data source.

storedProcedure
The current stored procedure object.

Data Sources
Relational

Syntax
Java Method

```
StoredProcedure getStoredProcedure();
```

Usage
Returns the current StoredProcedure object. A DataException will be thrown if errors occur when accessing the database or if the StoredProcedure object has not been instantiated by calling StoredProcedureBlox.prepare(...).

See Also
"StoredProcedure Object Properties and Associated Methods" on page 687

storedProcedures

A list of all stored procedures or stored procedures that match the specified pattern in all or specified catalogs and schemas.

Data Sources

Relational

Syntax

Java Methods

```
MetaData[] getStoredProcedures(); //returns an array of
                                  //StoredProcedures objects

MetaData[] getStoredProcedures(String pattern);
//returns a list of stored procedures in all catalogs and schemas
//that match the pattern

MetaData[] getStoredProcedures(String selectedCatalog,
                               String selectedSchema);
                     //returns a list of stored procedures
                     //in the specified catalog and schema

MetaData[] getStoredProcedures(String selectedCatalog,
                               String selectedSchema,
                                String pattern);
             //returns a list of stored procedures that match
             //the pattern in the specified catalog and schema
```

where:

Argument	Default	Description
pattern	null	The JDBC pattern used to match stored procedures
selectedCatalog	null	The catalog in which the stored procedures are
selectedSchema	null	The schema in which the stored procedures are

Usage

Returns an array of StoredProcedure objects. If there are no stored procedures found, the array will be of zero length and will never be null. Therefore, you do not need to check the result against null.

When matching stored procedures with a pattern, within the pattern String, "%" means match any substring of 0 or more characters, and "_" means match any one character. Only metadata entries matching the search pattern are returned. If a search pattern argument is set to null, that argument's criteria will be dropped from the search.

Examples

"Example 3: Getting a list of stored procedures whose name matches a specified pattern" on page 670

userName

The user name.

Data Sources

Relational

Syntax

Java Methods

```
String getUserName();
void setUserName(String userName);
```

where:

Argument	Default	Description
userName	null	The user name.

Usage

Use this setUsername() method to override the setting in the DB2 Alphablox data source definition before connecting to the data source.

StoredProceduresBlox Methods

This section describes StoredProceduresBlox methods that are not associated with a specific property. For the syntax and descriptions of StoredProceduresBlox methods that have a property associated with them, see "StoredProceduresBlox Properties and Associated Methods" on page 675.

connect()

Connects to the supplied RDB data source.

Data Sources

Relational

Syntax

Java Method

```
void connect();
```

Usage

Use the setDataSourceName(String *dataSource*) method before calling connect().

Examples

"Example 1: Connecting to the data source without a DataBlox" on page 670

See Also

"dataSourceName" on page 676

close()

Close all resources associated with executing a stored procedure. This frees up resources used without disconnecting to the database.

Data Sources

Relational

Syntax

Java Method

```
void close();   // throws java.sql.SQLException
```

Usage

If you want to disconnect from the data source, use StoredProceduresBlox.disconnect().

Examples
"Example 5: Executing a stored procedure that has one input parameter and two output parameters" on page 672

See Also
"disconnect()" on page 680

disconnect()

Disconnects from the RDB data source.

Data Sources
Relational

Syntax
Java Method

```
void disconnect();
```

Usage
If any statements or result sets are open, statements and result sets will also be closed.

Examples
"Example 5: Executing a stored procedure that has one input parameter and two output parameters" on page 672

execute()

Execute a stored procedure. The stored procedure executed must first be prepared with the `prepare(...)` method. Also, if the stored procedure has parameters, these must be setup. This method is a convenience method for `StoredProcedure.execute()`.

Data Sources
Relational

Syntax
Java Method

```
void execute();   // throws a DataException
```

Examples
See "Example 5: Executing a stored procedure that has one input parameter and two output parameters" on page 672

See Also
"execute()" on page 688

loadResultSet()

Load the stored procedure's JDBC ResultSet into a DataBlox. This method is a convenience method for `StoredProcedure.ResultSet.loadResultSet()`.

Data Sources
Relational

Syntax

Java Method

```
void loadResultSet(DataBlox dataBlox, int n)
```

where:

Argument	Default	Description
dataBlox	null	The already connected DataBlox to set the JDBC ResultSet to.
n	null	The 1-based number representing the result set to use

Usage

Once a result set is retrieved, a previous result set cannot be retrieved. For example, calling `loadResultSet(myDataBlox, 2)` then `loadResultSet(myDataBlox, 1)` will throw a DataException. Calling `loadResultSet(myDataBlox, 3)` then `loadResultSet(mydatablox, 3)` will also throw a DataException. This is a JDBC restriction.

Examples

"Example 6: Setting a stored procedure result set to a DataBlox" on page 672

See Also

"The StoredProcedure.ResultSet Inner Class" on page 674, "loadResultSet()" on page 689

prepare()

Prepares the stored procedure using either the default or supplied result set type and concurrency. This is a call to the JDBC `Connection.prepareCall(String sql)` method.

Data Sources

Relational

Syntax

Java Methods

```
java.sql.CallableStatement prepare(String sql);
            //use the default result set type and concurrency

java.sql.CallableStatement prepare(String sql,
                                   int resultSetType,
                                   int resultSetConcurrency);
            //use the supplied result set type and concurrency
```

where:

Argument	Default	Description
sql	null	The stored procedure to prepare using the JDBC stored procedure syntax. The exact syntax is driver-specific and depends on your data source.
resultSetType	null	A result set type
resultSetConcurrency	null	A result set concurrency

Examples
See "Example 5: Executing a stored procedure that has one input parameter and two output parameters" on page 672.

See Also
`JDBC Connection.prepareCall()` in Java 2 Platform Javadoc.

MetaData Object Properties and Associated Methods

This section describes properties supported by the MetaData object and associated methods. You must use the following JSP import statement at the beginning of any JSP file to use the API in this object:

```
<%@ page import="com.alphablox.blox.data.rdb.storedprocedure.*" %>
```

catalog

The stored procedure catalog name.

Data Sources
Relational

Syntax
Java Method

```
String getCatalog();
```

Examples
"Example 4: Getting a list of all parameters for each stored procedure" on page 671

columnMetaData

The parameter information for a particular stored procedure.

Data Sources
Relational

Syntax
Java Method

```
MetaData.Column[] getColumnMetaData();
   //returns an array of the MetaData.Column objects, one
   //for each parameter
```

See Also
"MetaData.Column Object Methods" on page 684

name

Gets the stored procedure name.

Data Sources
Relational

Syntax
Java Method

```
String getName();
```

Examples
"Example 4: Getting a list of all parameters for each stored procedure" on page 671

remark

The stored procedure remarks and/or comments.

Data Sources

Relational

Syntax

Java Method

```
String getRemark();   //returns String
```

Examples

"Example 4: Getting a list of all parameters for each stored procedure" on page 671

schema

The stored procedure schema name.

Data Sources

Relational

Syntax

Java Method

```
String getSchema();  //returns String
```

Examples

"Example 4: Getting a list of all parameters for each stored procedure" on page 671

type

The stored procedure type. The type is mapped to the following fields in java.sql.DatabaseMetaData: DatabaseMetaData.procedureResultUnknown, DatabaseMetaData.procedureNoResult, and DatabaseMetaData.procedureReturnsResult.

Data Sources

Relational

Syntax

Java Method

```
short getType();  //returns short
```

Examples

The following example assumes the `procedures[]` array contains a list of stored procedure objects. We find out the stored procedure type by comparing with each of the three fields in java.sql.DatabaseMetaData.

```
<%
for (int i = 0; i < procedures.length; i++) {
   String type = null;
   switch (procedures[i].getType()) {
      case DatabaseMetaData.procedureResultUnknown: type = "Unknown"; break;
      case DatabaseMetaData.procedureNoResult: type = "No result"; break;
      case DatabaseMetaData.procedureReturnsResult: type = "Returns result";
      break;
      default: type = "Could not determine type";
   } %>
   Stored Procedure Type is: <%= type %> <br/>
   <%
}
%>
```

See Also
SUN's Javadoc for java.sql.DatabaseMetaData.

MetaData.Column Object Methods

This section describes methods associated with the MetaData.Column object. The MetaData.Column object contains information on a single stored procedure parameter. You must use the following JSP import statement at the beginning of any JSP file to use the API in this object:

```
<%@ page import="com.alphablox.blox.data.rdb.storedprocedure.*" %>
```

getCatalog()

Gets the stored procedure catalog name.

Data Sources
Relational

Syntax
Java Method

```
String getCatalog();
```

Examples
"Example 4: Getting a list of all parameters for each stored procedure" on page 671

getColumnName()

Gets the parameter's column name.

Data Sources
Relational

Syntax
Java Method

```
String getColumnName();
```

Examples
"Example 4: Getting a list of all parameters for each stored procedure" on page 671

getDataType()

Gets the parameter's data type.

Data Sources
Relational

Syntax
Java Method

```
short getDataType();
```

See Also
SUN's Javadoc on java.sql.Types

getLength()

Gets the parameter's length, returning the length of the data in bytes.

Data Sources
Relational

Syntax
Java Method

```
int getLength();
```

getName()
Gets the stored procedure's name.

Data Sources
Relational

Syntax
Java Method

```
String getName();
```

Examples
"Example 4: Getting a list of all parameters for each stored procedure" on page 671

getNullable()
Identifies if the parameter is nullable; mapped to the following fields in java.sql.DatabaseMetaData: DatabaseMetaData.procedureNoNulls, DatabaseMetaData.procedureNullable, and DatabaseMetaData.procedureNullableUnknown.

Data Sources
Relational

Syntax
Java Method

```
short getNullable();
```

See Also
SUN's Javadoc for java.sql.DatabaseMetaData.

getPrecision()
Gets the parameter's precision, returning the total number of digits.

Data Sources
Relational

Syntax
Java Method

```
int getPrecision();
```

getRadix()
Gets the parameter's radix.

Data Sources
Relational

Syntax
Java Method
```
int getRadix();
```

getRemark()
Gets the parameter's remarks and/or comments.

Data Sources
Relational

Syntax
Java Method
```
String getRemark();
```

Examples
"Example 4: Getting a list of all parameters for each stored procedure" on page 671

getScale()
Gets the parameter's scale, returning the number of digits to the right of the decimal point.

Data Sources
Relational

Syntax
Java Method
```
short getScale();
```

getSchema()
Gets the stored procedure schema name.

Data Sources
Relational

Syntax
Java Method
```
String getSchema();
```

Examples
"Example 4: Getting a list of all parameters for each stored procedure" on page 671

getType()
Gets the parameter's type. The type is mapped to the following fields in java.sql.DatabaseMetaData: DatabaseMetaData.procedureColumnUnknown, DatabaseMetaData.procedureColumnIn, DatabaseMetaData.procedureColumnInOut, DatabaseMetaData.procedureColumnOut, DatabaseMetaData.procedureColumnReturn, and DatabaseMetaData.procedureColumnResult.

Data Sources
Relational

Syntax
Java Method

```
short getType();
```

Examples
"Example 4: Getting a list of all parameters for each stored procedure" on page 671

See Also
SUN's Javadoc for java.sql.DatabaseMetaData.

getTypeName()
Gets the parameter's type as a String.

Data Sources
Relational

Syntax
Java Method

```
String getTypeName();
```

Examples
"Example 4: Getting a list of all parameters for each stored procedure" on page 671

StoredProcedure Object Properties and Associated Methods
The StoredProcedure object has two properties. For methods not associated with any properties, see "StoredProcedure Object Methods" on page 688. You must use the following JSP import statement at the beginning of any JSP file to use the API in this object:

```
<%@ page import="com.alphablox.blox.data.rdb.storedprocedure.*" %>
```

callableStatement
The interface in java.sql used to execute SQL stored procedures. The CallableStatement object can be used to get and set stored procedure parameters.

Data Sources
Relational

Syntax
Java Method

```
java.sql CallableStatement getJDBCCallableStatement();
     // throws a DataException
```

Usage
The preferred method for getting the CallableStatement object is by using `StoredProceduresBlox.prepare()`. Do not close the CallableStatement using `CallableStatement.close()`. See the Java 2 Platform API Specification for more information on the CallableStatement object.

Examples
See "Example 5: Executing a stored procedure that has one input parameter and two output parameters" on page 672.

See Also
"prepare()" on page 681

resultSet

The ResultSet object representing the result set for the executed stored procedure.

Data Sources
Relational

Syntax
Java Method

```
StoredProcedure.ResultSet getResultSet();  //throws a DataException
```

Usage
See the Java 2 Platform API Specification for more information on the ResultSet object.

See Also
"StoredProcedure.ResultSet Inner Class Methods" on page 689

StoredProcedure Object Methods

This section describes StoredProcedure object methods that are not associated with a specific property. For the StoredProcedure object method that is associated with a property, see "StoredProcedure Object Properties and Associated Methods" on page 687. You must use the following JSP import statement at the beginning of any JSP file to use the API in this objects:

```
<%@ page import="com.alphablox.blox.data.rdb.storedprocedure.*" %>
```

close()

Closes the StoredProcedure and the corresponding JDBC CallableStatement.

Data Sources
Relational

Syntax
Java Method

```
void close();   // throws a java.sql.SQLException
```

Usage
The preferred method for closing a stored procedure is `StoredProceduresBlox.close()` convenience method.

See Also
"close()" on page 679

execute()

Executes a stored procedure. All input parameters must be set before calling `execute()` by using `getCallableStatement()` to get the JDBC CallableStatement object. The preferred method for executing stored procedures is `StoredProceduresBlox.execute()`.

Data Sources
Relational

Syntax

Java Method

```
StoredProcedure.ResultSet execute();
          // throws a DataException, java.sql.SQLException
```

Usage

Throws a `DataException` if an DB2 Alphablox specific database access error occurs; Throws an `SQLException` if a database access error occurs.

Examples

See "Example 5: Executing a stored procedure that has one input parameter and two output parameters" on page 672.

See Also

"callableStatement" on page 687, "execute()" on page 680

StoredProcedure.ResultSet Inner Class Methods

This section describes all methods for StoredProcedure.ResultSet inner class. You must use the following JSP import statement at the beginning of any JSP file to use the API in this object:

```
<%@ page import="com.alphablox.blox.data.rdb.storedprocedure.*" %>
```

getResultSet()

Gets the JDBC ResultSet object (an interface in the java.sql package). This can be used to iterate through the ResultSet directly via JDBC ResultSet methods.

Data Sources

Relational

Syntax

Java Method

```
java.sql.ResultSet getJDBCResultSet();   // throws a DataException
```

Usage

Do not close the ResultSet using `ResultSet.close()`. StoredProceduresBlox keeps track of which objects are opened or closed. It you close the objects manually, you may get an exception from StoredProceduresBlox when you use it later.

loadResultSet()

Loads the stored procedure's JDBC ResultSet into a DataBlox.

Data Sources

Relational

Syntax

Java Method

```
void loadResultSet(DataBlox dataBlox, int n);
    //throws either a ServerDataBlox exception or a DataException
```

where:

Argument	Default	Description
dataBlox	null	The already connected DataBlox to set the JDBC ResultSet to.
n	null	The 1-based number representing the result set to use.

Usage
Preferably, use the StoredProceduresBlox.loadResultSet() convenience method instead.

See Also
"loadResultSet()" on page 680

useResultSet()

Gets a result set from a stored procedure that produces multiple result sets. This should only be used on stored procedures that contain multiple result sets. If only one result set is produced by the stored procedure, use getResultSet() instead.

Data Sources
Relational

Syntax
Java Method

```
java.sql.ResultSet useResultSet(int n);  // throws a DataException
```

where:

Argument	Default	Description
n	null	The 1-based number representing the result set to get.

Usage
Once a result set is retrieved, a previous result set cannot be retrieved. For example, calling useResultSet(2) then useResultSet(1) will throw a DataException. Calling useResultSet(3) then useResultSet(3) will also throw a DataException. This is a JDBC restriction.

See Also
"getResultSet()" on page 689

Chapter 23. ToolbarBlox Reference

This chapter contains reference material for ToolbarBlox. For general reference information about Blox, see Chapter 3, "General Blox Reference Information," on page 17. For information on how to use this reference, see Chapter 1, "Using This Reference," on page 1.

ToolbarBlox Overview

ToolbarBlox presents a customized Blox toolbar. It is added in two ways:

- Using the nested `<blox:toolbar>` tag inside a PresentBlox, ChartBlox, or GridBlox.
- Setting the `toolbarVisible` tag attribute to `true` for `<blox:present>`, `<blox:chart>`, or `<blox:grid>` tag.

In the DHTML client, you cannot have a standalone ToolbarBlox. By default, the toolbar is available and visible in a PresentBlox, a standalone GridBlox, and a standalone ChartBlox.

Graphical User Interface

ToolbarBlox appears in the DHTML client with two toolbars: Standard toolbar and Navigation toolbar. By default, these two toolbars contain the following buttons:

- Pop out
- Copy
- Redo
- Undo
- Load Bookmark
- Export to PDF
- Export to Excel
- Help
- Data Navigation
- Sort
- Member Filter
- Grid
- Chart
- Page Filter
- Data Layout Panel

The buttons and the toolbars are fully customizable. The Blox UI Tag Library includes tags that allow you to add, edit, or remove a toolbar or a toolbar button. See Chapter 26, "Blox UI Tags Reference," on page 791 for details.

For instructions on using the ToolbarBlox user interface, see the online user help. You can access the user help by clicking the help button on the toolbar of the Blox user interface.

ToolbarBlox JSP Custom Tag Syntax

The Alphablox Tag Libraries provides custom tags to use in a JSP page for creating each Blox. This section describes how to create the custom tag to create a toolbar within a PresentBlox, GridBlox, or ChartBlox. For a copy and paste version of the tag with all the attributes, see "Miscellaneous Tags in blox.tld" on page 891.

Parameters

```
<blox:toolbar
    [attribute="value"] >
</blox:toolbar>
```

where:

attribute is one of the attributes listed in the attribute table.

value is a valid value for the attribute.

and where the attributes are one of the following:

Attribute
id
applyPropertiesAfterBookmark
bloxEnabled
bloxName
bookmarkFilter
helpTargetFrame
localeCode
removeAction
removeButton
rolloverEnabled
textVisible
toolTipsVisible
visible

Usage

Each custom tag can have one or more attributes, separated by one or more space or new line characters. Any extra space or new line characters are ignored. For readability, attributes can each go on a separate line with the same indenting.

You can substitute the closing `</blox:toolbar>` tag with a closing slash (/) after the last attribute in the tag but before the closing greater than character. For example, if the last attribute is `width`, the end of the tag looks as follows:

```
width="650" />
```

Examples

```
<blox:toolbar
   id="myToolbar1"
   toolTipsVisible="false">
</blox:toolbar>
```

ToolbarBlox Properties/Method by Category

The following tables list the unique ToolbarBlox properties and their corresponding methods, if any. The tables also list ToolbarBlox methods for which there are no corresponding properties. For lists of properties and methods common to several Blox, see "Common Blox Properties and Methods by Category" on page 29. The properties and methods supported by ToolbarBlox are organized in the cross reference as follows:

Appearance

The following table lists properties and methods relating to the appearance of a ToolbarBlox.

Properties	Methods
rolloverEnabled	
	isRolloverEnabled() setRolloverEnabled()
textVisible	
	isTextVisible() setTextVisible()
toolTipsVisible	
	isToolTipsVisible() setToolTipsVisible()

Contents

The following table lists the property and its methods relating to the contents of a ToolbarBlox.

Property	Methods
removeButton	
	getRemoveButton() setRemoveButton()

Event Filters and Listeners

The following table lists the methods for capturing events for pre- and post-event processing.

Methods
addEventFilter()
addEventListener()
removeEventFilter()
removeEventListener()

ToolbarBlox Properties and Associated Methods

This section describes the properties supported by ToolbarBlox and the methods associated with those properties. The properties are listed alphabetically by property name. For a list of ToolbarBlox methods with which no properties are associated, see “ToolbarBlox Methods” on page 697. Common Blox properties available from ToolbarBlox are listed but not described. For complete descriptions of common Blox properties, see “Properties and Associated Methods Common to Multiple Blox” on page 32.

id

This is a common Blox property. For a complete description, see “id” on page 39.

applyPropertiesAfterBookmark

This is a common Blox property. For a complete description, see “applyPropertiesAfterBookmark” on page 32.

bloxEnabled

This is a common Blox property. For a complete description, see “bloxEnabled” on page 35.

bloxModel

This is a common Blox property. For a complete description, see “bloxModel” on page 38

bloxName

This is a common Blox property. For a complete description, see “bloxName” on page 35.

bookmarkFilter

This is a common Blox property. For a complete description, see “bookmarkFilter” on page 33.

helpTargetFrame

This is a common Blox property. For a complete description, see “helpTargetFrame” on page 39.

localeCode

This is a common Blox property. For a complete description, see “localeCode” on page 40.

removeAction

This is a common Blox property. For a complete description, see “removeAction” on page 44.

removeButton

Identifies the buttons to remove from the ToolbarBlox (before it appears to the user).

Data Sources

All

Syntax

JSP Tag Attribute

```
removeButton = "removeButton"
```

Java Methods

```
String getRemoveButton();
boolean setRemoveButton(String removeButton);
```

where:

Argument	Default	Description
removeButton	"Save,Load"	Comma-delimited string of button names.

Usage

The value is a quoted, comma-delimited list of button names, such as “Save, Load”, which removes those buttons from the toolbar, thus removing user access to the Save/Load application state function. The possible values for the button names are Chart, Layout, Grid, Swap, Bookmark, Help, Load, Save.

Tip: The list you supply in `setRemoveButton()` method will overwrite the default. If you do not want the Save and Load buttons, make sure you include them in your list of buttons to remove. To remove buttons not listed here, use the Blox UI tags. See “Custom Toolbar Tags” on page 823.

Tip: The Save and Load buttons allow a user to save the state of the application as private or public in much the same way as the bookmark functionality. The difference is, when you have multiple presentation Blox that are not nested, the Save and Load buttons will save the state of all Blox on the page automatically and you do not need to specify which Blox you want to save the state, as is the case with the bookmark functionality. The saved application states are managed separately from bookmarks, so you may want to avoid confusion by offering only the bookmark or the save/load application state functionality.

Examples

```
getRemoveButton();
setRemoveButton("Chart,Save,Load");
```

rolloverEnabled

Specifies whether the color of toolbar buttons should change from grayscale to color when the mouse moves over the button.

Data Sources

All

Syntax

JSP Tag Attribute

```
rolloverEnabled = "boolean"
```

Java Methods

```
boolean isRolloverEnabled();
void setRolloverEnabled(boolean enable);
```

where:

Argument	Default	Description
enable	false	A boolean argument. A value of true indicates the toolbar button image changes on a mouse over to show a rollover effect; a value of false indicates the toolbar button image does not change.

Usage
Toolbar buttons appear with a rollover effect with mouse over if this property value is set to true. If you add a toolbar button using the <bloxui:toolbarButton> tag, you need to supply an image with the "_active" suffix for the mouse-over effect when this property is set to true. See "Custom Toolbar Tags" on page 823 for more information.

Examples
```
setRolloverEnabled(false);
```

textVisible
Specifies whether a text label should appear beneath the icon on a toolbar button.

Data Sources
All

Syntax
JSP Tag Attribute

```
textVisible = "boolean"
```

Java Methods

```
boolean isTextVisible();
void setTextVisible(boolean visible);
```

where:

Argument	Default	Description
visible	false	A boolean argument. A value of true indicates the text labels are visible, a value of false indicates they are not.

Usage
A text label appears beneath the icon on a toolbar button when the value is set to true.

Examples
```
setTextVisible(false);
```

toolTipsVisible
Specifies whether descriptive text should appear when the user holds the mouse over a toolbar button.

Data Sources
All

Syntax

JSP Tag Attribute

```
toolTipsVisible = "boolean"
```

Java Methods

```
boolean isTooltipsVisible();
void setTooltipsVisible(boolean visible);
```

where:

Argument	Default	Description
visible	true	A boolean value. A value of true indicates the tooltips display when you mouse over the toolbar, a value of false indicates the tooltips do not display.

Usage

Descriptive text appears when the user holds the mouse over a toolbar button when the value is set to true.

Examples

"rolloverEnabled" on page 695, "toolTipsVisible" on page 696

visible

This is a common Blox property. For a complete description, see "visible" on page 46.

ToolbarBlox Methods

This section describes ToolbarBlox methods that are not associated with a specific property. For the syntax and descriptions of ToolbarBlox methods that have a property associated with them, see "ToolbarBlox Properties and Associated Methods" on page 694. For client-side API common to Blox, see "Client-Side APIs" on page 31.

addEventFilter()

This is a common Blox method that for capturing a server-side event (such as saving and loading bookmarks) and perform custom actions *after* the operation is complete on the server. For details, see "addEventListener()" on page 49.

addEventListener()

This is a common Blox method that allows you to capture a server-side event (such as saving and loading bookmarks) and perform custom actions *after* the operation is complete on the server. For details, see "addEventListener()" on page 49.

call()

This is a common client-side Blox method. For a complete description, see "call()" on page 51.

flushProperties()

This is a common client-side Blox method. For a complete description, see "flushProperties()" on page 52.

loadBookmark()

This is a common Blox method. For a complete description, see “loadBookmark()” on page 54.

removeEventFilter()

This is a common Blox method that allows you to remove an event filter object added using addEventFilter() for capturing a server-side event *before* the event is processed on the server. For details, see “removeEventFilter()” on page 55.

removeEventListener()

This is a common Blox method that allows you to remove an event listener object created using addEventListener() for capturing a server-side event *after* that operation is complete on the server. For details, see “removeEventListener()” on page 56.

saveBookmark()

This is a common Blox method. For a complete description, see “saveBookmark()” on page 57.

saveBookmarkHidden()

This is a common Blox method. For a complete description, see “saveBookmarkHidden()” on page 58.

setDataBusy()

This is a common client-side Blox method. For a complete description, see “setDataBusy()” on page 60.

updateProperties()

This is a common client-side Blox method. For a complete description, see “updateProperties()” on page 61.

Chapter 24. Blox Form Tag Reference

There are many variations of FormBlox that allow you to add HTML form-like user interface in your JSP and link the form elements to server-side components or other form components on the page without page refreshes. The tags to add these FormBlox are available in the Blox Form Tag Library (`bloxform.tld`). This chapter contains reference material for tags in this library. For detailed API listing, see the com.alphablox.blox.form package in the Javadoc.

FormBlox Overview

FormBlox and business logic Blox (discussed in Chapter 25, "Business Logic Blox and TimeSchema DTD Reference," on page 735) are designed to solve two commonly encountered problems during analytical application development: the need for data-aware business logic and the need to maintain state. A series of specialized FormBlox let you generate time periods, data source, cube, dimension, and member selection lists simply by using the Blox Form Tag Library (`bloxform.tld`).

- These Blox let you create user interfaces similar to those standard HTML form elements such as radio buttons, check boxes, and edit fields.
- Unlike generic HTML form elements, FormBlox automatically maintain the state after page reloads during a session.
- FormBlox can automatically populate a selection list based on the data source, cube, dimension, or time schema you specify.

As a result, there is no need to write code to perform sophisticated time series calculation, find out the metadata in order to populate a user selection list, or to manage the state of the form elements.

FormBlox Variations

There are different variations of FormBlox, each designed to add a specific user interface that can be linked to other FormBlox or server-side components. The following tables lists all FormBlox and describes their purposes:

FormBlox	Description
FormBlox for Generic HTML Form Elements	
CheckBoxFormBlox	A FormBlox implementation of the HTML <input type="checkbox"...> tag. However, unlike an HTML form, which, when posted, does not send any value if a box is not checked, CheckBoxFormBlox always returns a value on a form post.
EditFormBlox	A FormBlox implementation of an HTML edit field (either <input type="text" ...> or <textarea ...>)
RadioButtonFormBlox	A FormBlox implementation of the HTML radio button set (<input type="radio" ... >).
SelectFormBlox	A FormBlox implementation of an HTML <select> element; supports both single and multiple select.
FormBlox for MetaData Selection List	
DataSourceSelectFormBlox	A FormBlox implementation of an HTML <select> list that displays available data sources; can be limited to multidimensional or relational data sources.
CubeSelectFormBlox	A FormBlox implementation of an HTML <select> list that displays cubes available in a given multidimensional data source.
DimensionSelectFormBlox	A FormBlox implementation of an HTML <select> list that displays dimensions available in a given multidimensional cube.
MemberSelectFormBlox	A specialized implementation of the HTML <select> element that displays members from a given multidimensional data source dimension. Given a DataBlox, a cube (if necessary), and a dimension, it displays a selection list containing members within the dimension.
FormBlox for TimeSchema-Related Selection List	
TimePeriodSelectFormBlox	A FormBlox implementation of the HTML <select> element that displays TimeSeries available in a time schema. A TimeSeries is a duration of time such as last two months, last quarter, last two quarters, month to current, quarter to current, current month, and current week.
TimeUnitSelectFormBlox	A specialized implementation of the HTML <select> element that displays period types from a given time schema. A time unit (PeriodType) is the unit used by a time schema, such as years, halves, quarters, months, weeks, and days.
FormBlox for Navigation Tree	
TreeFormBlox	A FormBlox encapsulation of the Blox UI tree control. A DHTML tree control is rendered to the page.

All FormBlox-related classes are under the com.alphablox.blox.form package. Their tags are available from the `bloxform.tld` tag library.

Common FormBlox Properties and Attributes

The FormBlox class is the base class for all FormBlox. As such, all FormBlox share some common properties, methods, tags, and behavior:

- They use the same event model FormEventListener. This is how all FormBlox events are handled.
- They use a form POST to post values (except TreeFormBlox).
- They all have the following tag attributes:

Common FormBlox Attribute	Description
`id`	The `id` of the object that is rendered on the page. It is also the `bloxName` unless the `bloxName` attribute is specified.
`bloxName`	The name of the object on the server (the peer).
`formElementName`	The name of the rendered page element and the parameter name used for a form POST.
`themeClass`	The name of a theme class.
`visible`	`true` if the object is rendered in place. The default is `true`.

FormBlox Events

The FormEventListener interface in com.alphablox.blox.form is the event handler for all FormBlox. The `addFormEventListener()` and `removeFormEventListener()` methods allow you to add/remove a FormEventListener to enable/disable event handling. When event handling, the `FormEventListener.valueChanged()` method is called whenever a FormBlox is changed (for example, a check box is checked/unchecked, a radio button is clicked, or a selection is made in a selection list).

The `getChangedProperty` and `setChangedProperty` tags support basic event handling and, in many cases, can make writing an event handler unnecessary. The cases where a `getChangedProperty` and `setChangedProperty` tags will suffice are those where a property on one object will always change a corresponding property on another Java bean.

The setChangedProperty Tag

FormBlox can set a property on any Java bean. The `<bloxform:setChangedProperty>` tag lets you specify which property should be changed to the new value selected when a FormBlox is changed. Consider a check box that allows a user to choose whether to enable alternate row colors on a grid. The checkbox is added using the CheckBoxFormBlox:

```
<bloxform:checkBox id="bandingCheckBox"
   checked="false"
   checkedValue="true"
   uncheckedValue="false">
   <bloxform:setChangedProperty
      targetRef="myGridBlox"
      targetProperty="bandingEnabled" />
</bloxform:checkBox> Enable alternate row banding
```

This check box is unchecked as it is rendered on the page. When users check the box, the checked value (`checkedValue`) will be set on the `bandingEnabled` property (`targetProperty`) of `myGridBlox` (`targetRef`).

The getChangedProperty Tag

The <bloxform:getChangedProperty> tag is nested in the tag for a FormBlox and establishes that a property on the FormBlox will change whenever a corresponding property on another FormBlox changes. For example, it lets you link several FormBlox so the selection from one FormBlox sets the selection available in another FormBlox. A common scenario is the so-called "cascading menus." In cascading menus, the selection of an option from the first menu dictates the options available in the next menu.

For example, in a report we may have a set of menus for users to select a city to see sales data. The cascading menus start from a zone menu. The selection of a zone dictates the areas available in the second menu. The selection of an area dictates the cities available in the subsequent menu. The following example creates the zone menu by getting generation 2 members of the All Locations dimension. The area menu is created by getting the selected member from the zone menu. The following is the code snippet:

```
<%--The zone menu: displaying all generation 2 members of
    the All Locations dimension. Note that for MSAS data sources
    the member name should be enclosed in square brackets (unique
    names). --%>
<bloxform:memberSelect id="zone"
   dataBloxRef="myData"
   dimensionName="All Locations"
   filterOperator="=="
   filterGeneration="2">
</bloxform:memberSelect>

<%--The area menu: displaying all generation 3 members of
    the selected member from the zone menu. --%>
<bloxform:memberSelect id="area"
   dataBloxRef="myData"
   dimensionName="All Locations"
   filterOperator="=="
   filterGeneration="3">
   <bloxform:getChangedProperty
      formBloxRef="zone"
      formProperty="selectedMembers"
      property="rootMembers" />
</bloxform:memberSelect>
```

The FormPropertyLink Object

The FormPropertyLink class in the com.alphablox.blox.form package is the object behind the `getChangedProperty` and `setChangedProperty` tags. It is used to link FormBlox together and transfer basic properties back and forth between them.

Whenever a property changes on a FormBlox, FormPropertyLink will set the new value on the target bean. Since it is using the normal Java bean introspection, the target can be any Java beans rather than just FormBlox. If necessary, one additional method on the bean can be called following the property change. This allows the link to handle cases such as a change to the `query` property on the DataBlox. In this case, in order to complete the change, it is necessary to call the DataBlox's `updateResultSet()` method following a change to the query property.

FormPropertyLink will automatically perform conversions of different data types when you use the tags to link two different properties:

- If the caller's argument and the callee's expected parameter are of the same type, then the argument is passed to the callee as is.

- If the caller's argument is an array and the callee's expected parameter is not, then the first element in the passed array will be passed to the callee.
- If the caller's argument is not an array and the callee is expecting an array, then the argument is converted into an array of length 1 to be passed to the callee.
- If the caller's argument is a String and the callee is expecting a boolean, then the string will be converted to a boolean. For example, the string `"true"` will be converted to a boolean `true` and passed to the callee.
- If the caller's argument is a non-primitive Java object and the callee is expecting a String, then the argument will be converted to a String using `toString()` before being passed to the callee.

Styling FormBlox

All FormBlox have a `themeClass` property and a `themeClass` tag attribute for you to specify a theme class for the component. The following TreeFormBlox uses a style class called `myMenuTree` to set the style for the menu item texts:

```
<!--some code ommitted here...>
<head>
   <blox:header/>
   <style>
   .myMenuTree { background-color: #FFFF80; }
   </style>
</head>
<body>
<bloxform:tree id="myMenu" rootVisible="false" themeClass="myMenuTree">
   <bloxform:folder> <%--root folder--%>
      <bloxform:folder label="Sales Analysis">
         <bloxform:item label="Sales Trend by Region"
            href="salesByRegion.jsp"
            target="mainFrame" />
         <bloxform:item label="Sales by Store"
            href="salesByStore.jsp"
            target="mainFrame" />
         <bloxform:item label="Units Sold by Product"
            href="unitsSoldByProduct.jsp"
            target="mainFrame" />
      </bloxform:folder>
   </bloxform:folder>
<!--more code omitted here-->...
```

You can also use the DB2 Alphablox theme classes defined in the `<themeName>_dhtml.css` file in `<alphablox_dir>/repository/theme/<themeName>`. This allows for a consistent look and feel through out your application. The following example shows a SelectFormBlox that uses a defined theme class called `csSlctBg`.

```
<!--some code omitted here-->...
<b>Select Chart Type:</b><br>
<bloxform:select id="ChartSelection" size="4"
   themeClass="csSlctBg">
   <bloxform:option label="Bar" value="Bar" selected="true"/>
   <bloxform:option label="Pie" value="Pie" />
   <bloxform:option label="Line" value="Line" />
   <bloxform:option label="3D Bar" value="3D Bar" />
   <bloxform:setChangedProperty
      targetRef="myChart"
      targetProperty="chartType" />
</bloxform:select>
<!--more code omitted here-->......
```

For details on how CSS themes are supported and used, see the Presenting Data chapter of the *Developer's Guide*. It also includes a listing of style classes supported in DB2 Alphablox themes.

FormBlox that Create a Selection List

Many FormBlox create a selection list. All the different variations of SelectFormBlox behave similarly as the first option in the list is the default selected option if this is a single selection list and no selected option is set explicitly. With the exception of DataSourceSelectFormBlox and TimePeriodSelectFormBlox, these Blox have a `multiple` tag attribute and a `size` tag attribute. When the selection list has a `size` greater than 1 or when multiple selections are allowed, at least one option needs to be set as the initial selection or an error may occur. Since most of these Blox are tied to a DataBlox and their instantiation involves a query to the data source, an initial selection should be set.

Note: With FormBlox that are tied to a DataBlox, every time a selection is made on the selection list, the `FormEventListener.valueChanged()` method is called and a query is issued. When working with large result set or complex queries, there could be delay or performance issues.

Blox Form Tag Library Reference by Category

To use the following FormBlox tags, include the following taglib directive in the beginning of your JSP files:

```
<%@ taglib uri="bloxformtld" prefix="bloxform"%>
```

For FormBlox methods, see the com.alphablox.blox.form package in the Javadoc.

The Blox Form Tag Library includes the following form tags:

FormBlox for Generic HTML Form Elements

FormBlox for Data-Related Selection List

FormBlox for TimeSchema-Related Selection List

TreeFormBlox

– "The Nested <bloxform:folder> Tag" on page 730
– "The Nested <bloxform:item> Tag" on page 731

Nested Tag for Connecting FormBlox and Specifying Actions

• "The <bloxform:getChangedProperty> Tag Reference" on page 732
• "The <bloxform:setChangedProperty> Tag Reference" on page 732

The following sections describe the properties, tags, and attributes for each of the FormBlox, with examples that demonstrate the usage and syntax.

CheckBoxFormBlox Reference

For each check box you add, you can specify whether this check box should be checked when it is rendered, and what value should be passed when the box is checked or unchecked. For better page layout, you may want to add each CheckBoxFormBlox inside a table cell in order to put text next to it. Note that as soon as users click the check box, the `valueChanged()` method on the FormEventListener is called and the new value is set immediately.

CheckBoxFormBlox Properties

When linking FormBlox using the `<bloxform:getChangedProperty>` and `<bloxform:setChangedProperty>` tags, you may need to specify the name of the property you want to get or change on the target FormBlox. This section lists all properties for CheckBoxFormBlox. For associated methods, see the FormBlox Javadoc under the com.alphablox.blox.form package.

Property	Type	Description
`checked`	`boolean`	`true` if the check box should be checked and the `checkedValue` should be set when the check box is rendered. The default is `false`.
`checkedValue`	`String`	The value to return when the check box is checked. This is the value to set on the specified property on the target object using the nested `<setChangedProperty>` tag.
`formElementName`	`String`	The name of the rendered page element and the parameter name used for a form POST.
`formValue`	`String`	The posted value of the component.
`formValues`	`String[]`	The posted values of the component.
`renderHook`	`FormBloxRenderHook`	Indicates if the Blox has been rendered; used to provide a custom renderer for a FormBlox.
`themeClass`	`String`	A String containing the theme class name(s) to set on this element. Separate class names with a space.
`uncheckedValue`	`String`	The value to return when the check box is unchecked. This is the value to set on the specified property on the target object using the nested `<setChangedProperty>` tag.

The <bloxform:checkBox> Tag

The following table lists all attributes for the `<bloxform:checkBox>` tag:

Attribute	Default	Description
`id`		The `id` of the object that is rendered on the page. It is also the `bloxName` unless the `bloxName` attribute is specified.

Attribute	Default	Description
bloxName	Defaults to id	The name of the object on the server (the peer).
checked	false	true if the check box should be checked and the checkedValue should be set when the check box is rendered. If this attribute is not specified, the check box is unchecked and the uncheckedValue will be set when the check box is rendered.
checkedValue	true	The value to return when the check box is checked. This is the value to set on the specified property on the target object using the nested <setChangedProperty> tag.
formElementName		The name of the rendered page element and the parameter name used for a form POST.
themeClass		The name(s) of theme class(es) to set on this element. If more than one classes are specified, separate the names with a space.
uncheckedValue	false	The value to return when the check box is unchecked. This is the value to set on the specified property on the target object using the nested <setChangedProperty> tag.
visible	true	true if the object is to be rendered in place.

A CheckBoxFormBlox Example

This example demonstrates how to allow users to turn on/off alternate row banding in a GridBlox.

- A GridBlox is added, but is not rendered (visible="false").
- A CheckBoxFormBlox is added inside a table cell, with the text to display next to it in another cell on the same table row.
- The checked attribute is set to true. Therefore, the check box, when rendered on the page, is checked and the checkedValue is set on myGridBlox's bandingEnabled property. Note that the checkedValue attribute is not specified, so the default value "true" is used.
- The nested <bloxform:setChangedProperty> tag is used to specify the target object and the property of the object to change.
- The GridBlox is rendered using the <blox:display> tag, and the GridBlox is displayed with the alternate row banding enabled.

```
<%@ taglib uri="bloxtld" prefix="blox" %>
<%@ taglib uri="bloxformtld" prefix="bloxform" %>

<blox:grid id="myGridBlox"
   visible="false"
   width="600"
   height="350">
   <blox:data
      dataSourceName="QCC-Essbase"
      query="<SYM <ROW (\"All Products\") <CHILD \"All Products\"
        <COL (\"All Time Periods\") <CHILD \"All Time Periods\"
         <PAGE(Measures) Sales !" />
</blox:grid>

<html>
<head>
   <blox:header />
</head>

<body>
<table>
<tr>
<td>
   <bloxform:checkBox id="bandingCheckBox"
      checked="true">
      <bloxform:setChangedProperty
         targetRef="myGridBlox"
```

```
        targetProperty="bandingEnabled" />
    </bloxform:checkBox>
</td>
<td>Enable Alternate Row Banding</td>
</tr>
</table>

<blox:display bloxRef="myGridBlox" />

</body>
</html>
```

CubeSelectFormBlox Reference

This FormBlox adds a selection list of cubes available in a given multidimensional data source.

CubeSelectFormBlox Properties

When linking FormBlox using the <bloxform:getChangedProperty> and <bloxform:setChangedProperty> tags, you may need to specify the name of the property you want to get or change on the target FormBlox. This section lists all properties for CubeSelectFormBlox. For associated methods, see the FormBlox Javadoc under the com.alphablox.blox.form package.

Property	Type	Description
dataBlox	DataBlox	The DataBlox from which to get the list of cubes.
formElementName	String	The name of the rendered page element and the parameter name used for a form POST.
formValue	String	The value to return when a selection is made. This is the value to set on the specified property on the target object using the nested <setChangedProperty> tag.
formValues	String[]	The values to return when selections are made. These are the values to set on the specified property on the target object using the nested <setChangedProperty> tag.
minimumWidth	String	The minimum width of the element in pixels.
multipleSelect	boolean	true if multiple selections are allowed. The default is false. Note that the tag attribute name is multiple.
renderHook	FormBloxRenderHook	Indicates if the Blox has been rendered; used to provide a custom renderer for a FormBlox.
selectedCube	Cube	The selected Cube object.
selectedCubeNames	String[]	The names of the selected cubes.
selectedCubes	Cube[]	The selected Cube objects.
size	int	The number of lines that are visible in the list. The default is 1. When the multipleSelect property is true, size has to be greater than 1, or the browser will be default to a size of 4.
themeClass	String	A String containing the theme class name(s) to set on this element. Separate class names with a space.

Note: Most FormBlox that create a selection list (CubeSelectFormBlox, DimensionSelectFormBlox, MemberSelectFormBlox, SelectFormBlox, and TimeUnitSelectFormBlox) have the same behavior: when the selection list

has a size of 1 (a drop down list), the first option is automatically set as the initial selection unless this selectedCube/Dimension/Member/Series attribute is explicitly specified. When the selection list has a size greater than 1 or when multiple selections are allowed, at least one option needs to be set as the initial selection or an error may occur.

The <bloxform:cubeSelect> Tag

The following table lists all attributes for the <bloxform:cubeSelect> tag:

Attribute	Default	Description
id		The id of the object that is rendered on the page. It is also the bloxName unless the bloxName attribute is specified.
bloxName	Defaults to id	The name of the object on the server (the peer).
dataBlox		A DataBlox. For example: dataBlox="<%=myDataBlox %>"
dataBloxRef		The name of a DataBlox already instantiated in the page.
formElementName		The name of the rendered page element and the parameter name used for a form POST.
minimumWidth	The width that fits the longest option in the list	The minimum width for this selection list in pixels. When a selection list contains no options, it appears as a very narrow list. You can use this attribute to make an empty selection list more appealing.
multiple	false	true if multiple selections are allowed.
selectedCube	The first Cube in the metadata	The Cube object initially selected in the list.
selectedCubeName		The name of the cube initially selected in the list.
size	1	The number of items that are visible in the list. When multiple is true, size has to be greater than 1, or the browser will be default to a size of 4.
themeClass		The name(s) of theme class(es) to set on this element. If more than one classes are specified, separate the names with a space.
visible	true	true if the object is to be rendered in place.

A CubeSelectFormBlox Example

The following example demonstrates how to populate a selection list with all cubes available in a given multidimensional data source.

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxformtld" prefix="bloxform"%>

<blox:data id="myDataBlox"
   useAliases="true"
   dataSourceName="Durico"
   connectOnStartup="false"
/>
...
<bloxform:cubeSelect id="cubes"
      dataBloxRef="myDataBlox"
      visible="true" />
...
```

DataSourceSelectFormBlox Reference

This FormBlox adds a selection list of data sources defined to DB2 Alphablox. You can specify the data source type (MDB, RDB, or ALL) or the specific data adapter such as IBM DB2 JDBC Driver, IBM DB2 OLAP Server, Hyperion Essbase Adapter, or Oracle Driver.

DataSourceSelectFormBlox Properties

When linking FormBlox using the <bloxform:getChangedProperty> and <bloxform:setChangedProperty> tags, you may need to specify the name of the property you want to get or change on the target FormBlox. This section lists all properties for DataSourceSelectFormBlox. For associated methods, see the FormBlox Javadoc under the com.alphablox.blox.form package.

Property	Type	Description
adapterNameFilter	String	The specific type of data sources, based on the adapter, to display in the list. The valid values are the following constants: • DB2Driver • DB2OLAPDeploymentServicesAdapter • DB2OLAPServerAdapter • EssbaseAdapter • EssbaseDeploymentServicesAdapter • JDBC_ODBCBridgeforMSVM • JDBC_ODBCBridgeforSunVM • MSOLAPAdapter • MSSQLDriver • OracleDriver • SybaseDriver • CannedDataAdapter
adminBlox	AdminBlox	An AdminBlox.
formElementName	String	The name of the rendered page element and the parameter name used for a form POST.
formValue	String	The value to return when a selection is made. This is the value to set on the specified property on the target object using the nested <setChangedProperty> tag.
formValues	String[]	The values to return when selections are made. These are the values to set on the specified property on the target object using the nested <setChangedProperty> tag.
minimumWidth	String	The minimum width of the element in pixels.
nullDataSourceLabel	String	The first label to display in the selection list which also indicates that a data source has not be selected. Typically, this label is set to instruct users to select a data source. See "A DataSourceSelectFormBlox Example" on page 710.
renderHook	FormBloxRenderHook	Indicates if the Blox has been rendered; used to provide a custom renderer for a FormBlox.
selectedDataSource	DataSource	The selected DataSource object (com.alphablox.blox.repository.DataSource)
selectedDataSourceName	String	The names of the selected data source
themeClass	String	A String containing the theme class name(s) to set on this element. Separate class names with a space.

Property	Type	Description
typeFilter	String	The type of data sources to display in the list. Valid values are MDB, RDB, or ALL. The default value is ALL.

The <bloxform:dataSourceSelect> Tag

The following table lists all attributes for the <bloxform:dataSourceSelect> tag:

Attribute	Default	Description
id		The id of the object that is rendered on the page. It is also the bloxName unless the bloxName attribute is specified.
bloxName	Defaults to id	The name of the object on the server (the peer).
adapter		The specific type of data sources, based on the adapter, to display in the list. Valid values are: • IBM DB2 JDBC Driver • IBM DB2 OLAP Server Deployment Services • IBM DB2 OLAP Server • Hyperion Essbase Adapter • Hyperion Essbase Deployment Services • Generic JDBC-ODBC Bridge for MS VM • Generic JDBC-ODBC Bridge for Sun VM • Microsoft SQL Server Driver • Sybase SQL Server Driver • OLEDB for OLAP • Oracle Driver • Canned Data Adapter
adminBloxRef		The AdminBlox already instantiated in the page.
formElementName		The name of the rendered page element and the parameter name used for a form POST.
minimumWidth	The width that fits the longest option in the list	The minimum width for this selection list in pixels. When a selection list contains no options, it appears as a very narrow list. You can use this attribute to make an empty selection list more appealing.
nullDataSourceLabel		If set, an extra menu item is added to the top and it becomes the default, indicating that a data source has not be selected unless something else is set (via selectedDataSourceName). Typically, this label is set to instruct users to select a data source. See "A DataSourceSelectFormBlox Example" on page 710.
selectedDataSourceName	The first one in the list	The name of the selected data source.
themeClass		The name(s) of theme class(es) to set on this element. If more than one classes are specified, separate the names with a space.
type	ALL	The type of data sources to display in the list. Valid values are MDB, RDB, or ALL.
visible	true	true if the object is to be rendered in place.

A DataSourceSelectFormBlox Example

The following example creates a drop down selection list containing all multidimensional data sources. The selection list appears with "Select a Data

Source" as the first item. For a complete example that shows how to connect the selection to a DataBlox, see the "Ad Hoc Analysis using DataSourceSelectFormBlox" example under the FormBlox section.

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxformtld" prefix="bloxform"%>
<html>
<head>
   <blox:header />
</head>
<body>
<bloxform:dataSourceSelect id="dataSourceName"
   type="MDB"
   nullDataSourceLabel="Select a Data Source">
</bloxform:dataSourceSelect>
</body>
</html>
```

DimensionSelectFormBlox Reference

This FormBlox adds a selection list of dimensions from the given cube in the given multidimensional data source.

DimensionSelectFormBlox Properties

When linking FormBlox using the `<bloxform:getChangedProperty>` and `<bloxform:setChangedProperty>` tags, you may need to specify the name of the property you want to get or change on the target FormBlox. This section lists all properties for DimensionSelectFormBlox. For associated methods, see the FormBlox Javadoc under the com.alphablox.blox.form package.

Property	Type	Description
`cube`	`Cube`	The Cube object to get the dimensions from.
`cubeName`	`String`	The name of the cube to get the dimensions from.
`dataBlox`	`DataBlox`	The DataBlox from which to get the metadata.
`formElementName`	`String`	The name of the rendered page element and the parameter name used for a form POST.
`formValue`	`String`	The value to return when a selection is made. This is the value to set on the specified property on the target object using the nested `<setChangedProperty>` tag.
`formValues`	`String[]`	The values to return when selections are made. These are the values to set on the specified property on the target object using the nested `<setChangedProperty>` tag.
`minimumWidth`	`String`	The minimum width of the element in pixels.
`multipleSelect`	`boolean`	`true` if multiple selections are allowed. The default is `false`. Note that the tag attribute name is `multiple`.
`renderHook`	`FormBloxRenderHook`	Indicates if the Blox has been rendered; used to provide a custom renderer for a FormBlox.
`selectedDimension`	`Dimension`	The selected Dimension object. Defaults to the first one in the list in a single selection list.
`selectedDimensions`	`Dimension[]`	The selected Dimension objects when multiple selections are supported. Defaults to null or an empty array in a multiple selection list.

Property	Type	Description
selectedUniqueName	String	The unique name of the selected dimension. Defaults to the first one in the list in a single selection list.
selectedUniqueNames	String[]	The unique names of the selected dimensions when multiple selections are supported. Defaults to null or an empty array in a multiple selection list.
size	int	The number of lines that are visible in the list. The default is 1. When the multipleSelect property is true, size has to be greater than 1, or the browser will be default to a size of 4.
themeClass	String	A String containing the theme class name(s) to set on this element. Separate class names with a space.

The <bloxform:dimensionSelect> Tag

The following table lists all attributes for the <bloxform:dimensionSelect> tag:

Attribute	Default	Description
id		The id of the object that is rendered on the page. It is also the bloxName unless the bloxName attribute is specified.
bloxName	Defaults to id	The name of the object on the server (the peer).
cube		The Cube object to get the dimensions from.
cubeName		The name of the cube to get the dimensions from.
dataBlox		The DataBlox from which to get the metadata.
dataBloxRef		The name of a DataBlox already instantiated in the page.
formElementName		The name of the rendered page element and the parameter name used for a form POST.
minimumWidth	The width that fits the longest option in the list	The minimum width for this selection list in pixels. When a selection list contains no options, it appears as a very narrow list. You can use this attribute to make an empty selection list more appealing.
multiple	false	true if multiple selections are allowed.
selectedDimension	Defaults to the first one in the list	The Dimension object initially selected in the list.
selectedDimensionName	Defaults to the first one in the list	The name of the dimension initially selected in the list. For MSAS data sources, the dimension name should be enclosed in square brackets ([ ]).
size	1	The number of items that are visible in the list. When multiple is true, size has to be greater than 1, or the browser will be default to a size of 4.
themeClass		The name(s) of theme class(es) to set on this element. If more than one classes are specified, separate the names with a space.
visible	true	true if the object is to be rendered in place.

Note: Most FormBlox that create a selection list—CubeSelectFormBlox, DimensionSelectFormBlox, MemberSelectFormBlox, SelectFormBlox, and TimeUnitSelectFormBlox—have the same behavior: when the selection list has a size of 1 (a drop down list), the first option is automatically set as the initial selection unless this selected*Cube/Dimension/Member/Series* attribute is explicitly specified. When the selection list has a size greater than 1 or

when multiple selections are allowed, at least one option needs to be set as the initial selection or an error may occur.

DimensionSelectFormBlox Examples

The following example creates a drop down list populated with all dimensions in the specified cube in the specified DataBlox.

```
<bloxform:dimensionSelect id="allDimensions"
   dataBloxRef="dataBlox"
   cubeName="Cube1"
   visible="true" />
```

The following example creates a drop down list populated with all cubes in the specified DataBlox. The dimensions to display in the dimension drop list is determined by the selection of a cube.

```
<table>
<tr>
<td width="100">Select a cube:</td>
<td width="140">Select a dimension:</td>
</tr>
<tr>
<td><bloxform:cubeSelect id="cubes"
      dataBloxRef="dataBlox"
      visible="true" /></td>
<td><bloxform:dimensionSelect id="dimensions"
       dataBloxRef="dataBlox"
       visible="true">
       <bloxform:getChangedProperty formBloxRef="cubes"
          formProperty="selectedCube"
          property="cube"/>
    </bloxform:dimensionSelect></td>
</tr>
</table>
```

EditFormBlox Reference

EditFormBlox adds either a `<text>` or `<textarea>` tag into the rendered page. If the `lines` attribute is not specified or is set to 1, a `<text>` tag is inserted. For better page layout, you may want to add a EditFormBlox inside a table cell in order to put text next to it.

When users click inside the text field (in order to enter information), the input focus is on the EditFormBlox. As soon as users click somewhere else on the page and the input focus is reset, the FormEventListener will call the `valueChanged()` method and the new value is set. Note that hitting the Enter key will not trigger a form POST.

EditFormBlox Properties

When linking FormBlox using the `<bloxform:getChangedProperty>` and `<bloxform:setChangedProperty>` tags, you may need to specify the name of the property you want to get or change on the target FormBlox. This section lists all properties for EditFormBlox. For associated methods, see the FormBlox Javadoc under the com.alphablox.blox.form package.

Property	Type	Description
`charactersPerLine`	int	The number of characters allowed per line in the text field. The default `charactersPerLine` is `20`.

Property	Type	Description
formElementName	String	The name of the rendered page element and the parameter name used for a form POST.
formValue	String	The value to return when a selection is made. This is the value to set on the specified property on the target object using the nested <setChangedProperty> tag.
formValues	String[]	The values to return when selections are made. These are the values to set on the specified property on the target object using the nested <setChangedProperty> tag.
lines	int	The number of lines that are rendered in the text field. The default is 1. If this attribute is set to more than 1, a text area is rendered.
maskInput	boolean	true if characters are to be masked (appear as asterisks).
maxCharacters	int	The number of characters allowed in the text field.
renderHook	FormBloxRenderHook	Indicates if the Blox has been rendered; used to provide a custom renderer for a FormBlox.
themeClass	String	A String containing the theme class name(s) to set on this element. Separate class names with a space.
value	String	The value entered in the text field.

The <bloxform:edit> Tag

The following table lists all attributes for the <bloxform:edit> tag:

Attribute	Default	Description
id		The id of the object that is rendered on the page. It is also the bloxName unless the bloxName attribute is specified.
bloxName	Defaults to id	The name of the object on the server (the peer).
charactersPerLine	20	The number of characters allowed per line in the text field.
formElementName		The name of the rendered page element and the parameter name used for a form POST.
lines	1	The number of lines that are rendered in the text field. If this attribute is set to more than 1, a text area is rendered.
maskInput	false	true if characters are to be masked (appear as asterisks).
maxCharacters		The number of characters allowed in the text field.
themeClass		The name(s) of theme class(es) to set on this element. If more than one classes are specified, separate the names with a space.
visible	true	true if the object is to be rendered in place.

An EditFormBlox Example

This example demonstrates how to use an EditFormBlox to allow users to specify the title of a ChartBlox.

- An EditFormBlox is added inside a table cell, with the text to display before it in another cell on the same table row.
- The maxCharacters and charactersPerLine attributes are both set to 30.
- The nested <bloxform:setChangedProperty> tag is used to specify the target object and the property of the object to change.
- As soon as the input focus is set to somewhere else on the page, the ChartBlox's title property is set to the value entered in the text field.

```
<%@ taglib uri="bloxtld" prefix="blox" %>
<%@ taglib uri="bloxformtld" prefix="bloxform" %>
<blox:chart id="myChartBlox"
   visible="false"
   width="500"
   height="500">
   <blox:data
      dataSourceName="QCC-Essbase"
      query="<SYM <ROW ('All Products') <CHILD 'All Products'
        <COL ('All Time Periods') <CHILD 'All Time Periods'
         <PAGE(Measures) Sales !" />
</blox:chart>

<html>
<head>
   <blox:header />
</head>

<body>
<table>
<tr>
<td>Title for this chart:</td>
<td>
   <bloxform:edit id="titleEdit"
      charactersPerLine="30"
      maxCharacters="30">
      <bloxform:setChangedPBroperty
         targetRef="myChartBlox"
         targetProperty="title" />
   </bloxform:edit>
</td>
</tr>
</table>
<font size="-1">(When you are done, click anywhere else on the page to set the
title.)</font><p>
<blox:display bloxRef="myChartBlox" />
</body>
</html>
```

MemberSelectFormBlox Reference

This FormBlox adds a selection list of members from a given dimension of a given cube (if necessary) of a given DataBlox. You can set the root members so only the root members and their descendants are displayed in the list. You can also filter the list by specifying whether only members equal to, less than, or greater than a specified generation should be displayed.

MemberSelectFormBlox Properties

When linking FormBlox using the `<bloxform:getChangedProperty>` and `<bloxform:setChangedProperty>` tags, you may need to specify the name of the property you want to get or change on the target FormBlox. This section lists all properties for MemberSelectFormBlox. For associated methods, see the FormBlox Javadoc under the com.alphablox.blox.form package.

Property	Type	Description
`dataBlox`	`DataBlox`	The DataBlox from which to get the metadata.
`dimension`	`Dimension`	The Dimension object whose members are to be listed.
`dimensionName`	`String`	The unique dimension name whose members are to be listed. For MSAS data sources, the member name should be enclosed in square brackets ([ ]).

Property	Type	Description
filterGeneration	int	The generation to be listed in the list. See the filterOperator property below. The default is 0.
filterOperator	String	The operator to apply to the filterGeneration. Valid values are ==, <, or >.
filterType	int	The operator to apply to the filterGeneration. Valid values are the following constants: • MemberSelectFormBlox.EQUALS • MemberSelectFormBlox.GREATERTHAN • MemberSelectFormBlox.LESSTHAN
formElementName	String	The name of the rendered page element and the parameter name used for a form POST.
formValue	String	The value to return when a selection is made. This is the value to set on the specified property on the target object using the nested <setChangedProperty> tag.
formValues	String[]	The values to return when selections are made. These are the values to set on the specified property on the target object using the nested <setChangedProperty> tag.
minimumWidth	String	The minimum width of the element in pixels.
multipleSelect	boolean	true if multiple selections are allowed. The default is false. Note that the tag attribute name is multiple.
renderHook	FormBloxRenderHook	Indicates if the Blox has been rendered; used to provide a custom renderer for a FormBlox.
rootMembers	Member[]	The root Members in this selection list. For MSAS data sources, the member names should be enclosed in square brackets ([ ]).
rootUniqueNames	String[]	The unique names of the root members.
selectedDisplayName	String	The display name of the selected member.
selectedDisplayNames	String[]	The display names of the selected members.
selectedMembers	Member[]	The selected Member objects when multiple selections are supported.
selectedUniqueName	String	The unique name of the selected member.
selectedUniqueNames	String[]	The unique names of the selected members when multiple selections are supported.
size	int	The number of items that are visible in the list.
themeClass	String	A String containing the theme class name(s) to set on this element. Separate class names with a space.

The <bloxform:memberSelect> Tag

The following table lists all attributes for the <bloxform:memberSelect> tag:

Attribute	Default	Description
id		The id of the object that is rendered on the page. It is also the bloxName unless the bloxName attribute is specified.
bloxName	Defaults to id	The name of the object on the server (the peer).
dataBlox		The DataBlox from which to get the metadata.
dataBloxRef		The name of a DataBlox already instantiated in the page.
dimension		The Dimension object whose members are to be listed.

Attribute	Default	Description
dimensionName		The unique dimension name whose members are to be listed. For MSAS data sources, the member name should be enclosed in square brackets ([ ]).
filterGeneration	0	The generation to be listed in the list. See the filterOperator attribute below.

filterOperator		The operator to apply to the filterGeneration. Valid values are ==, <, or >.
		
		The following example lists only members in generation 2:
		`filterGeneration="2"`
		`filterOperator="=="`

formElementName		The name of the rendered page element and the parameter name used for a form POST.
minimumWidth	The width that fits the longest option in the list	The minimum width for this selection list in pixels. When a selection list contains no options, it appears as a very narrow list. You can use this attribute to make an empty selection list more appealing.
multiple	false	true if the list supports multiple selections.
rootMemberName	Defaults to generation 1 roots	The unique name of the root member. For MSAS data sources, the member name should be enclosed in square brackets ([ ]).
rootMemberNames	Defaults to generation 1 roots	The unique names of the root members. For example, `rootMemberNames= "<%=new String[] {"[Location].[All Locations]","[Product].[Category]"} %>"`
rootMembers		The root members in this selection list. For example, `rootMembers="<%= mbrs%>"` where mbrs is a Member[] object.
selectedMember	Defaults to the first one in the list in single selection list.	The Member object selected.
selectedMemberName	Defaults to the first one in the list in single selection list	The unique name of the member selected. For MSAS data sources, the member name should be enclosed in square brackets ([ ]).
size	1	The number of items visible in the list. The list is rendered as a drop down list if this value is 1.
themeClass		The name(s) of theme class(es) to set on this element. If more than one classes are specified, separate the names with a space.
visible	true	true if the object is to be rendered in place.

Note: The rootMemberName, rootMemberNames, and selectedMemberName tag attributes all work with unique member names rather than display names. To use the tag attributes when you only have the display name, use the MDBMetaData.resolveMember(*memberName*, true) method to resolve the member name by display name first. See "resolveMember()" on page 439 for more information on how to resolve member names by display name.

Note: Most FormBlox that create a selection list—CubeSelectFormBlox, DimensionSelectFormBlox, MemberSelectFormBlox, SelectFormBlox, and TimeUnitSelectFormBlox—have the same behavior: when the selection list has a size of 1 (a drop down list), the first option is automatically set as the initial selection unless this *selectedCube/Dimension/Member/Series* attribute is explicitly specified. When the selection list has a size greater than 1 or when multiple selections are allowed, at least one option needs to be set as the initial selection or an error may occur.

A MemberSelectFormBlox Example

See "The getChangedProperty Tag" on page 702 and "MemberSecurityBlox Tags" on page 760.

RadioButtonFormBlox Reference

For each set of radio buttons you add, you can specify whether the buttons in this group should be aligned vertically or horizontally and whether a border should be drawn around the set of buttons. The buttons inside the group are mutually exclusive; selecting one deselects all others in the group. Note that as soon as users select a radio button, the `valueChanged()` method on the FormEventListener is called and the new value is set immediately.

RadioButtonFormBlox Properties

When linking FormBlox using the `<bloxform:getChangedProperty>` and `<bloxform:setChangedProperty>` tags, you may need to specify the name of the property you want to get or change on the target FormBlox. This section lists all properties for RadioButtonFormBlox. For associated methods, see the FormBlox Javadoc under the com.alphablox.blox.form package.

Property	Type	Description
`borderEnabled`	`boolean`	`true` if a border should be drawn around the set of radio buttons The default is `true`.
`buttons`	`String[]`	The values for all the buttons in the set.
`formElementName`	`String`	The name of the rendered page element and the parameter name used for a form POST.
`formValue`	`String`	The value of the selected radio button.
`formValues`	`String[]`	The value of the selected radio button. In the case of a RadioButtonFormBlox, if more than one value is passed in the array, the first one will be used.
`layout`	Layout	The Layout to apply: HorizontalLayout or VerticalLayout (in com.alphablox.blox.unimodel.core).
`renderHook`	`FormBloxRenderHook`	Indicates if the Blox has been rendered; used to provide a custom renderer for a FormBlox.
`selectedButton`	`String`	The value for the selected button.
`selectedObject`	`Object`	The selected user Object. If there is no user object associated, then this is the value of the selected button.
`themeClass`	`String`	A String containing the theme class name(s) to set on this element. Separate class names with a space.

The <bloxform:radioButton> Tag

The `<bloxform:radioButton>` tag is used to add a radio button set. To add individual radio buttons, use the `<bloxform:button>` tag. The following table lists all attributes for the `<bloxform:radioButton>` tag:

Attribute	Default	Description
`id`		The `id` of the object that is rendered on the page. It is also the `bloxName` unless the `bloxName` attribute is specified.
`bloxName`	Defaults to `id`	The name of the object on the server (the peer).
`align`	`horizontal`	`horizontal` or `vertical`. The default is `horizontal`.

Attribute	Default	Description
borderEnabled	true	true if a border should be drawn around the set of radio buttons The default is true.
formElementName		The name of the rendered page element and the parameter name used for a form POST.
themeClass		The name(s) of theme class(es) to set on this element. If more than one classes are specified, separate the names with a space.
visible	true	true if the object is to be rendered in place.

The Nested <bloxform:button> Tag

The <bloxform:button> tag is nested inside the <bloxform:radioButton> tag to add buttons.

Attribute	Description
label	The text rendered next to the radio button.
object	The Object value associated with the radio button. This attribute excludes the use of the value attribute.
selected	true if this item is set as the initial selection within the group of radio buttons.
value	The string value associated with the radio button. This attribute excludes the use of the object attribute.

To specify the target object and the property to set when the check box is checked or unchecked, use the nested <bloxform:setChangedProperty> tag. See "The <bloxform:setChangedProperty> Tag Reference" on page 732.

A RadioButtonFormBlox Example

This example demonstrates how to use a RadioButtonFormBlox to allow users to select from two reports. Since each report sets a different query on the DataBlox, selecting one report will deselect the other and reset the data query.

- A RadioButtonFormBlox is added with two buttons.
- The align attribute is set to vertical so the buttons are stacked vertically. The borderEnabled attribute is set to false.
- Notice that no data query is specified in the <blox:data> tag. Since the first radio button Sales By Product is set as the initial selection (selected="true"), its value is used to set the DataBlox's query when the GridBlox is rendered on the page.
- Since we need to set the property on an implicit DataBlox (nested within a GridBlox), we create a session variable ToggleData that we can refer to in the nested <bloxform:setChangedProperty> tag using the following scriptlet:

```
<%
  session.setAttribute("ToggleData",ToggleGridBlox.getDataBlox());
%>
```

- As soon as the user clicks a radio button, the ToggleData's query property is set to the value associated with that button. We then call the updateResultSet() method to update the result set:

```
<%@ taglib uri="bloxtld" prefix="blox" %>
<%@ taglib uri="bloxformtld" prefix="bloxform" %>
```

```
<blox:grid id="ToggleGridBlox"
   visible="false"
   width="450"
   height="250">
   <blox:data
      dataSourceName="QCC-Essbase"
      useAliases="true" />
</blox:grid>
<%
   session.setAttribute("ToggleData",ToggleGridBlox.getDataBlox());
%>
<html>
<head>
   <blox:header />
</head>
<body>
<b>Display Report:</b>
<bloxform:radioButton id="ReportSelection"
   borderEnabled="false"
   align="vertical" >
   <bloxform:button label="Sales By Product"
      value="<SYM <ROW('All Products') <CHILD 'All Products'
        <COLUMN('All Time Periods') <CHILD 'All Time Periods' Sales !"
      selected="true" />
   <bloxform:button label="Sales By Market"
      value="<SYM <PAGE(Measures) Sales <ROW(\"All Locations\") <CHILD
         \"All Locations\" <COLUMN(\"All Time Periods\") <CHILD
         \"All Time Periods\" !" />
   <bloxform:setChangedProperty
      targetRef="ToggleData"
      targetProperty="query"
      callAfterChange="updateResultSet" />
</bloxform:radioButton>

<blox:display bloxRef="ToggleGridBlox" />

</body>
</html>
```

SelectFormBlox Reference

SelectFormBlox lets you add an HTML `<select>` element on the page. Just like the HTML form `<select>` element, you can specify whether multiple selections are allowed and the number of options visible in the list. As soon as users make a selection, the `valueChanged()` method on the FormEventListener is called and the new value is set immediately.

SelectFormBlox Properties

When linking FormBlox using the `<bloxform:getChangedProperty>` and `<bloxform:setChangedProperty>` tags, you may need to specify the name of the property you want to get or change on the target FormBlox. This section lists all properties for SelectFormBlox. For associated methods, see the FormBlox Javadoc under the com.alphablox.blox.form package.

Property	Type	Description
`formElementName`	`String`	The name of the rendered page element and the parameter name used for a form POST.
`formValue`	`String`	The value to return when a selection is made. This is the value to set on the specified property on the target object using the nested `<setChangedProperty>` tag.

Property	Type	Description
formValues	String[]	The values to return when selections are made. These are the values to set on the specified property on the target object using the nested <setChangedProperty> tag.
items	String[]	An array of all labels in the selection list.
minimumWidth	String	The minimum width of the element in pixels.
multipleSelect	boolean	true if multiple selections are allowed. The default is false. When multiple is true, size has to be greater than 1. Note that the tag attribute name is multiple.
renderHook	FormBloxRenderHook	Indicates if the Blox has been rendered; used to provide a custom renderer for a FormBlox.
selectedIndexes	int[]	The indices of selected menu items. When the size of the list is 1 and if an empty array is passed, the first item will be selected by default.
selectedItem	String	The label of the selected item.
selectedItems	String[]	The labels of the selected items when multiple selections are supported.
selectedObject	Object	The user Object associated with the first selected item in the list.
selectedObjects	Objects[]	The user Objects associated with the selected items in the list.
size	int	The number of items that are visible in the list. The default is 1. When the multipleSelect property is true, size has to be greater than 1, or the browser will be default to a size of 4.
themeClass	String	A String containing the theme class name(s) to set on this element. Separate class names with a space.

The <bloxform:select>Tag

The <bloxform:select> tag is used to add a selection list. To add individual options in the list, use the <bloxform:option> tag. The following table lists all attributes for the <bloxform:select> tag:

Attribute	Default	Description
id		The id of the object that is rendered on the page. It is also the bloxName unless the bloxName attribute is specified.
bloxName	Defaults to id	The name of the object on the server (the peer).
formElementName		The name of the rendered page element and the parameter name used for a form POST.
minimumWidth	The width that fits the longest option in the list	The minimum width for this selection list in pixels. When a selection list contains no options, it appears as a very narrow list. You can use this attribute to make an empty selection list more appealing.
multiple	false	true if multiple selections are allowed.
size	1	The number of options that are visible in the list. When the value is 1, the list is rendered as a drop down list. If none of the options added using the <bloxform:option> tag has selected set to true, the first option on the list is set as the initial selection. When multiple is true, size has to be greater than 1, or the browser will be default to a size of 4.

Attribute	Default	Description
themeClass		The name(s) of theme class(es) to set on this element. If more than one classes are specified, separate the names with a space.
visible	true	true if the object is to be rendered in place.

The Nested <bloxform:option> Tag

This tag adds an option in the selection list. It has the following attributes:

Attribute	Default	Description
label		The text rendered as an option in the selection list.
object		The Object value associated with the option. This attribute excludes the use of the value attribute.
selected	The first one in the list	true if the item is set as the initial selection in the selection list.
value		The value associated with the option. This attribute excludes the use of the object attribute.

Note: Most FormBlox that create a selection list—CubeSelectFormBlox, DimensionSelectFormBlox, MemberSelectFormBlox, SelectFormBlox, and TimeUnitSelectFormBlox—have the same behavior: when the selection list has a size of 1 (a drop down list), the first option is automatically set as the initial selection unless this selected*Cube/Dimension/Member/Series* attribute is explicitly specified. When the selection list has a size greater than 1 or when multiple selections are allowed, at least one option needs to be set as the initial selection or an error may occur.

A SelectFormBlox Example

The following example demonstrates how to use a selection list to allow users to select a chart type.

- A selection list with four options are added to the page. This is a single selection list (multiple is false by default) with all four items visible in the list (size="4").
- The first option is set as the initial selection (selected="true").
- Since we need to set the property of a nested ChartBlox, we create a session variable myChart that can be later referenced in the <bloxform:setChangedProperty> tag:

```
<%
   session.setAttribute("myChart",myPresentBlox.getChartBlox());
%>
```

- Once a selection is made, the chartType property of myChart will be set to the value of the option selected.

The complete code is as follows:

```
<%@ taglib uri="bloxtld" prefix="blox" %>
<%@ taglib uri="bloxformtld" prefix="bloxform" %>
<blox:present id="myPresentBlox"
   visible="false"
   width="560"
   height="450">
   <blox:data
      dataSourceName="QCC-Essbase"
```

```
      query="<SYM <ROW('All Products') <CHILD 'All Products'
        <COLUMN('All Time Periods') <CHILD 'All Time Periods' Sales !"
      useAliases="true" />
</blox:present>

<%
   session.setAttribute("myChart",myPresentBlox.getChartBlox());
%>

<html>
<head>
   <blox:header/>
</head>
<body>
<b>Select Chart Type:</b><br>
<bloxform:select id="ChartSelection" size="4">
   <bloxform:option label="Bar" value="Bar" selected="true"/>
   <bloxform:option label="Pie" value="Pie" />
   <bloxform:option label="Line" value="Line" />
   <bloxform:option label="3D Bar" value="3D Bar" />
   <bloxform:setChangedProperty
      targetRef="myChart"
      targetProperty="chartType" />
</bloxform:select>
<blox:display bloxRef="myPresentBlox" />
</body>
</html>
```

TimePeriodSelectFormBlox Reference

TimePeriodSelectFormBlox creates a selection list displaying TimeSeries available in a TimeSchemaBlox. By default, the following TimeSeries entries are displayed:

- Last month
- Last two months
- Last three months
- Last six months
- Last twelve months
- Last quarter
- Last two quarters
- Last four quarters
- Last year
- Last two years
- Month to current
- Quarter to current
- Year to current
- Current month
- Current week

Note that if the time schema does not contain a given period type, then entries that depend on that period type are automatically removed from the defaults. It is also possible to add extra custom entries to the control programmatically. See "TimeSchemaBlox Tag" on page 769.

TimeSeries

The TimeSeries object, as its name suggests, represents a period of time that has the following properties:

- `baseInterval`: Basic period type, such as months, weeks, quarter, and years. It is used to determine the date range.
- `rollups`: Different types of time unit to include in roll-ups. For example, if the TimeSeries is last month, the rollup unit can be month, week, or day.
- `start`: The starting period; the offset from the current time period, with 0 being the current time period; -1, the previous period; -2, the previous 2 periods; 1, the next period, and so on.
- `count`: Number of periods to be included.
- `toDate`: Indicates if this TimeSeries represents a period to date (`TODATE`) or a sequence of periods (`SEQUENCE`). For example, `TODATE(Month)(Week)` indicates month-to-date with Week as the time unit in the rollup. `SEQUENCE(Month,-12,12)(Month,Quarter)` indicates last 12 months with Month and Quarter as the time units in the rollup.

The TimeSeries object is part of the com.alphablox.blox.logic package. Through the TimePeriodSelectFormBlox's nested `<bloxform:timeSeries>` tag, you can specify the time series to be included in the selection list.

TimePeriodSelectFormBlox Properties

When linking FormBlox using the `<bloxform:getChangedProperty>` and `<bloxform:setChangedProperty>` tags, you may need to specify the name of the property you want to get or change on the target FormBlox. This section lists all properties for TimePeriodSelectFormBlox. For associated methods, see the FormBlox Javadoc under the com.alphablox.blox.form package.

Property	Type	Description
`defaultSeriesVisible`	boolean	`true` if the default time series menu entries should be displayed. The default is `true`. See "TimePeriodSelectFormBlox Reference" on page 723 for a listing of all default entries.
`formElementName`	`String`	The name of the rendered page element and the parameter name used for a form POST.
`formValue`	`String`	The value to return when a selection is made. This is the value to set on the specified property on the target object using the nested `<setChangedProperty>` tag.
`formValues`	`String[]`	The values to return when selections are made. These are the values to set on the specified property on the target object using the nested `<setChangedProperty>` tag.
`minimumWidth`	`String`	The minimum width of the element in pixels.
`renderHook`	`FormBloxRenderHook`	Indicates if the Blox has been rendered; used to provide a custom renderer for a FormBlox.
`selectedSeries`	`TimeSeries`	The current selected TimeSeries.
`selectedSeriesName`	`String`	The label of the current selected TimeSeries.
`selectedSeriesString`	`String`	The current selected time series using the series string. See "The Nested <bloxform:timeSeries> Tag" on page 725 for more information on the time series expression string.
`themeClass`	`String`	A String containing the theme class name(s) to set on this element. Separate class names with a space.
`timeSchema`	`TimeSchemaBlox`	A TimeSchemaBlox already instantiated in the page.

Property	Type	Description
tuples	Member[][]	The tuples that correspond to the time series.

The <bloxform:timePeriodSelect> Tag

The <bloxform:timePeriodSelect> tag has the following attributes:

Attribute	Default	Description
id		The id of the object that is rendered on the page. It is also the bloxName unless the bloxName attribute is specified.
bloxName	Defaults to id	The name of the object on the server (the peer).
defaultSeriesVisible	true	true if the default time series menu entries should be displayed. See "TimePeriodSelectFormBlox Reference" on page 723 for a listing of all default entries.
formElementName		The name of the rendered page element and the parameter name used for a form POST.
minimumWidth	The width that fits the longest option in the list	The minimum width for this selection list in pixels. When a selection list contains no options, it appears as a very narrow list. You can use this attribute to make an empty selection list more appealing.
selectedSeries	The first one in the list	The current selected TimeSeries.
selectedSeriesString	The first one in the list	The current selected time series using the series string. See "The Nested <bloxform:timeSeries> Tag" on page 725 for more information on the time series expression string.
themeClass		The name(s) of theme class(es) to set on this element. If more than one classes are specified, separate the names with a space.
timeSchemaBloxRef		A TimeSchemaBlox already instantiated in the page.
visible	true	true if the object is to be rendered in place.

The Nested <bloxform:timeSeries> Tag

The <bloxform:timeSeries> tag is nested inside a <bloxform:timePeriodSelect> tag. It has the following attributes:

Attribute	Description
expression	The expression for constructing a TimeSeries. A TimeSeries can be either a SEQUENCE or a TODATE: • For SEQUENCE, specify the period type, the start, the count, and the time unit for rollup. • For TODATE, specify the period type and the time unit for rollup.

For example:

- expression="SEQUENCE(MONTH,-12,12)(MONTH)" indicates last 12 months (starts with twelve months ago and continues for 12 months), with Month as the time unit.
- expression="SEQUENCE(QUARTER,-1,1)(QUARTER)" indicates last quarter (startswith last month and continues for one month), with Quarter as the time unit.
- expression="SEQUENCE(MONTH,-1,1)(WEEK)" indicates last month, with Week as the time unit.

- expression="TODATE(MONTH)(WEEK)" indicates month-to-date with Week as the time unit in the rollup.
- expression="TODATE(QUARTER)(MONTH)" indicates quarter-to-date with Month as the time unit in the rollup.

Valid period types are: Year, Half Year, Quarter, Month, Week, and Day.

name — The displayed label for the TimeSeries.

A TimePeriodSelectFormBlox Example

The following example demonstrates the use of MDBQueryBlox and TimePeriodSelectFormBlox to allow users to select members for the row axis.

- A DataBlox is created first without a query.
- Tuples on the column axis are specified first using the <bloxlogic:tupleList> tag. This TupleList (id="histTuples") will be the TupleList for the column axis. It is defined outside of the <bloxlogic:mdbQuery> tag so it can have an id that we can set the property of this object when a selection of member on the column axis is made.
- A TimePeriodSelectFormBlox is added to show all the default time periods available, with the initial selected member set to the last six months with Month being the rollup unit (selectedSeriesString="SEQUENCE(MONTH,-6,6)(MONTH)").
- A MDBQueryBlox is added with the row and column axes of the query defined. For the row axis, we are displaying only Chocolate Blocks, Chocolate Nuts, and Specialties. For the column axis, histTuples is referenced.
- Once a selection is made by users, the listFromMetadataMembers property of histTuples is updated with the selected members of the TimePeriodSelectFormBlox. The changed() method is called to update the underlying DataBlox.

```
<%@ page import="java.util.Date"%>
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxformtld" prefix="bloxform"%>
<%@ taglib uri="bloxlogictld" prefix="bloxlogic"%>

<blox:data id="dataBlox" dataSourceName="QCC-MSAS"/>

<bloxlogic:timeSchema id="timeSchema"
   name="QCC-MSAS" dataBloxRef="dataBlox" />
<%
// Since QCC-MSAS only has data up to 2002, we are setting a fixed date
// for today so the example will run. In real applications, you do not need
// to set today's date.
   setToday(timeSchema);
%>
   <bloxlogic:tupleList id="histTuples">
      <bloxlogic:dimension list="<%=timeSchema.getDimensions()%>">
   </bloxlogic:dimension>
</bloxlogic:tupleList>

<bloxform:timePeriodSelect id="historySelector"
   timeSchemaBloxRef="timeSchema"
   selectedSeriesString="SEQUENCE(MONTH,-6,6)(MONTH)"
   visible="false">
   <bloxform:setChangedProperty formProperty="tuples"
      targetRef="histTuples"
      debugEnabled="true"
      targetProperty="listFromMetadataTuples"
      callAfterChange="changed"/>
</bloxform:timePeriodSelect>

<bloxlogic:mdbQuery id="query" dataBloxRef="dataBlox"
```

```
   cubeName="[QCC]">
   <bloxlogic:axis type="rows"
      queryFragment="{[Chocolate Blocks],[Chocolate Nuts],[Specialties]} ON
ROWS " />
   <bloxlogic:axis type="columns">
      <bloxlogic:tupleList tuplesRef="histTuples" />
   </bloxlogic:axis>
</bloxlogic:mdbQuery>

<html>
<head>
   <blox:header />
</head>
<body>

<b>Select a time period: </b>
<blox:display bloxRef="historySelector" />
<blox:grid id="myBlox" width="90%" height="75%"
   toolbarVisible="false" menubarVisible="false">
   <blox:data bloxRef="dataBlox" />
</blox:grid>
</body>
</hmtl>
<%!
   // Set today to a "fixed" date since the sample QCC-MSAS database
   // only has data up to 2002. In real applications, you do not need
   // to set today's date.
   public void setToday(com.alphablox.blox.logic.timeschema.TimeSchemaBlox
timeSchema) throws Exception {
      com.alphablox.blox.logic.timeschema.PeriodType small =
      com.alphablox.blox.logic.timeschema.PeriodType.getSmallest(timeSchema
.getPeriods());
      long end = timeSchema.last(small).getEndDate().getTime();
      timeSchema.setToday(new Date(end));
   }
%>
```

TimeUnitSelectFormBlox Reference

TimePeriodSelectFormBlox creates a selection list displaying time units available in a TimeSchemaBlox. By default, the following time unit entries are displayed:

- Year
- Quarter
- Month
- Week

TimeUnitSelectFormBlox Properties

When linking FormBlox using the <bloxform:getChangedProperty> and <bloxform:setChangedProperty> tags, you may need to specify the name of the property you want to get or change on the target FormBlox. This section lists all properties for TimeUnitSelectFormBlox. For associated methods, see the FormBlox Javadoc under the com.alphablox.blox.form package.

Property	Type	Description
formElementName	String	The name of the rendered page element and the parameter name used for a form POST.
formValue	String	The value to return when a selection is made. This is the value to set on the specified property on the target object using the nested <setChangedProperty> tag.

Property	Type	Description
formValues	String[]	The values to return when selections are made. These are the values to set on the specified property on the target object using the nested <setChangedProperty> tag.
items	String[]	An array of all labels in the selection list.
minimumWidth	String	The minimum width of the element in pixels.
multipleSelect	boolean	true if multiple selections are allowed. The default is false. Note that the tag attribute name is multiple.
renderHook	FormBloxRenderHook	Indicates if the Blox has been rendered; used to provide a custom renderer for a FormBlox.
selectedPeriodTypes	PeriodType	The current selected PeriodType.
size	int	The number of items that are visible in the list. When the multipleSelect property is true, size has to be greater than 1, or the browser will be default to a size of 4.
themeClass	String	A String containing the theme class name(s) to set on this element. Separate class names with a space.
timeSchema	TimeSchemaBlox	A TimeSchemaBlox already instantiated in the page.

The <bloxform:timeUnitSelect> Tag

The <bloxform:timeUnitSelect> tag has the following attributes:

Attribute	Default	Description
id		The id of the object that is rendered on the page. It is also the bloxName unless the bloxName attribute is specified.
bloxName	Defaults to id	The name of the object on the server (the peer).
formElementName		The name of the rendered page element and the parameter name used for a form POST.
minimumWidth	The width that fits the longest option in the list	The minimum width for this selection list in pixels. When a selection list contains no options, it appears as a very narrow list. You can use this attribute to make an empty selection list more appealing.
multiple	false	true if multiple selections are allowed.
selectedTimeUnit	The first one in the list	The current selected time unit using the PeriodType string. See "PeriodType" on page 739.
size	1	The number of items that are visible in the list. When multiple is true, size has to be greater than 1, or the browser will be default to a size of 4.
themeClass		The name(s) of theme class(es) to set on this element. If more than one classes are specified, separate the names with a space.
timeSchemaBloxRef		A TimeSchemaBlox already instantiated in the page.
visible	true	true if the object is to be rendered in place.

Note: Most FormBlox that create a selection list— CubeSelectFormBlox, DimensionSelectFormBlox, MemberSelectFormBlox, SelectFormBlox, and TimeUnitSelectFormBlox— have the same behavior: when the selection list has a size of 1 (a drop down list), the first option is automatically set as the initial selection unless this selected*Cube/Dimension/Member/Series* attribute is explicitly specified. When the selection list has a size greater than 1 or

when multiple selections are allowed, at least one option needs to be set as the initial selection or an error may occur.

TreeFormBlox Reference

The TreeFormBlox adds a navigation tree with folders and items based on the tree control in the Blox UI model. For each TreeFormBlox, you can specify whether the items in the tree are draggable, allowing users to move and reorder items, or whether the folder and item labels should be wrapped if the window or frame is too narrow.

Each TreeFormBlox requires one and only one root folder. Depending on your design, you may or may not want to display the root folder by setting the `rootVisible` attribute to `true` or `false`.

For each menu item, you can specify the value for the `href` attribute the same way as you would in an HTML `href` tag attribute. You can also specify the target window if a new page is to be loaded when the item is clicked. There can be links on both the folders and the items. Three tags are needed to create a tree: `<bloxform:tree>`, `<bloxform:folder>`, and `<bloxform:item>`.

TreeFormBlox Properties

This section lists all properties for TreeFormBlox. For associated methods and the inner classes (TreeFormBlox.Folder, TreeFormBlox.Item, TreeFormBlox.ItemDraggedEvent, and TreeFormBlox.ItemDraggedEventListener), see the FormBlox Javadoc under the com.alphablox.blox.form package.

Property	Type	Description
`draggingEnabled`	`boolean`	Specifies if items in the tree can be dragged to other folders or to reorder the items in a folder.
`folderStyle`	`Style`	The Style object used for each folder label in the tree.
`formElementName`	`String`	The name of the rendered page element and the parameter name used for a form POST.
`formValue`	`String`	The name of the rendered page element and the parameter name used for a form POST.
`formValues`	`String[]`	The names of the rendered page elements and the parameter names used for a form POST. If the case of a RadioButtonFormBlox, if more than one value is passed in the array, the first one will be used.
`itemPositioningEnabled`	boolean	Specifies whether to remain the position of items in a folder. When `draggingEnabled` is set to `true`, and `itemPositioningEnabled` is also set to `true`, users can only drag items into another folder, but cannot reorder the items in the folder.
`labelStyle`	Style	The Style object used for all item labels in the tree.
`renderHook`	`FormBloxRenderHook`	Indicates if the Blox has been rendered; used to provide a custom renderer for a FormBlox.
`root`	TreeFormBlox.Folder	The root folder of this tree.

Property	Type	Description
rootVisible	boolean	true to show the root folder. The default is true.
selected	String	The name of the selected Item.
selectedItem	TreeFormBlox.Item	The selected Item object.
textWrapped	boolean	true if the labels for all folders and items in this TreeFormBlox should be wrapped if the label is longer than the width of the browser window or frame. The default is true.
themeClass	String	A String containing the theme class name(s) to set on this element. Separate class names with a space.

The <bloxform:tree> Tag

This tag adds a TreeFormBlox on the page. This tag has the following attributes:

Attribute	Default	Description
id		The id of the object that is rendered on the page. It is also the bloxName unless the bloxName attribute is specified.
bloxName	Defaults to id	The name of the object on the server (the peer).
draggingEnabled		Specifies if items in the tree can be dragged to other folders or to reorder the items in a folder.
itemPositoningEnabled		Specifies whether to remain the position of items in a folder. When draggingEnabled is set to true, and itemPositioningEnabled is also set to true, users can only drag items into another folder, but cannot reorder the items in the folder.
rootVisible	true	true to show the root folder. When this attribute is true, the root folder is displayed: When this attribute is false, the root folder is not displayed, with its sub-folders displayed as the top-level folders.
textWrapped	true	true if the labels for all folders and items in this TreeFormBlox should be wrapped if the label is longer than the width of the browser window or frame.
themeClass		The name(s) of theme class(es) to set on this element. If more than one classes are specified, separate the names with a space.
visible	true	true if the object is to be rendered in place. The default is true.

To add a folder in the tree, use the nested <bloxform:folder> tag.

The Nested <bloxform:folder> Tag

Add this tag inside the <bloxform:tree> tag to add a folder. Note that each tree needs one and only root folder. Therefore, typically you should have at least two levels of folders in your tree. See "A TreeFormBlox Example" on page 731 for details.

To add items in the folder, use the nested <bloxform:item> tag. The <bloxform:folder> tag has the following attributes:

Attribute	Description

draggable	Specifies whether this folder should be draggable.
expanded	Specifies whether the folder should be expanded when it is rendered. The default is false.
href	The URI to load when the folder is clicked. This can be a link or a JavaScript function.
label	The text rendered next to the folder.
name	The name for the folder object.
object	The user object associated with this folder.
target	The target window or frame to load the URI specified in href. By default, the URI specified is loaded in the same window or frame.
tooltip	The text displayed when the mouse hovers over the folder.

The Nested <bloxform:item> Tag

This tag adds individual menu items in folders. It has the following attributes:

Attribute	Description
draggable	Specifies whether this item should be draggable.
href	The URI to load when the item is clicked. This can be a link or a JavaScript function.
label	The text rendered next to the item.
name	The name for the item object.
object	The user object associated with this item.
target	The target window or frame to load the URI specified in href. By default, the URI specified is loaded in the same window or frame.
tooltip	The text displayed when the mouse hovers over the item.

A TreeFormBlox Example

The following example creates a non-draggable menu tree with two folders. The root folder is not visible. It assumes the menu tree is in one frame, and when users click a menu item, a new page is loaded into a different target frame.

```
<%@ taglib uri="bloxtld" prefix="blox" %>
<%@ taglib uri="bloxformtld" prefix="bloxform" %>

<html>
<head>
    <blox:header/>
</head>
<body>
<bloxform:tree id="myMenu" rootVisible="false" >
   <bloxform:folder> <%--root folder--%>
      <bloxform:folder label="Sales Analysis">
         <bloxform:item label="Sales Trend by Region"
            href="salesByRegion.jsp"
            target="mainFrame" />
         <bloxform:item label="Sales by Store"
            href="salesByStore.jsp"
            target="mainFrame" />
```

```
         <bloxform:item label="Units Sold by Product"
            href="unitsSoldByProduct.jsp"
            target="mainFrame" />
      </bloxform:folder>

      <bloxform:folder label="Variance Analysis" expanded="false">
         <bloxform:item label="Sales Variance"
            href="varianceSales.jsp"
            target="mainFrame" />
         <%--Pop up an alert window as the report is not available.--%>
         <bloxform:item label="Ad-Hoc Variance Analysis"
            href="javascript:alert(\"Currently unavailable.\")" />
      </bloxform:folder>
   </bloxform:folder>
</bloxform:tree>
</body>
</html>
```

Note: You should use escaped double quotes inside the JavaScript call for the `href` attribute. Single quotes will cause JavaScript errors.

The <bloxform:getChangedProperty> Tag Reference

This tag is used to link FormBlox, allowing one FormBlox to get the selected property value of another FormBlox.

Attribute	Description
`debugEnabled`	When set to `true`, this turns on debug logging of property changes.

`formBlox`	The source FormBlox. This is the FormBlox where the property value is from. For example,
	`<bloxform:select id="mySelectFormBlox" ...>`
	`  <bloxform:getChangedProperty`
	`  formBlox="<%= anotherFormBloxn%>"`
	`  ..../>`
	`</bloxform:select>`
	
	Either this attribute or the `formBloxRef` attribute should be specified.

`formBloxRef`	The name of the source FormBlox already instantiated in the page. This is the FormBlox where the value is from. Either this attribute or the `formBlox` attribute should be specified.
`formProperty`	The name of the property on the source FormBlox whose change causes a notification.
`property`	The name of the property on the target FormBlox.

For examples of this tag, see "The getChangedProperty Tag" on page 702.

The <bloxform:setChangedProperty> Tag Reference

This tag is used to link FormBlox, allowing the selection in one FormBlox to set the selected property value of another FormBlox. It can set the properties on any Java bean since it is using the normal Java bean introspection. See the discussion of "The FormPropertyLink Object" on page 702.

Attribute	Description

`callAfterChange`	The method to call after the property is changed on the target. This method cannot have parameters. A common scenario is when a DataBlox property is changed, the `updateResultSet()` method needs to be called following the property change. Another example is when the property of a TupleList, a CrossJoin, or an Axis object is updated, its `changed()` method should be called in order to notify the query that the value has changed.
`debugEnabled`	When set to true, this turns on debug logging of property changes.
`formProperty`	The name of the property to propagate on changes.
`target`	The target object. It can be any Java bean. This is the target whose property will be changed. Either this attribute or the `targetRef` attribute should be set.
`targetRef`	The name of a bean already instantiated in the page. This is the target whose property will be changed. Either this attribute or the `target` attribute should be set.
`targetProperty`	The name of the property to change on the target bean.

For examples of this tag, see:

- "A CheckBoxFormBlox Example" on page 706
- "An EditFormBlox Example" on page 714
- "A RadioButtonFormBlox Example" on page 719
- "A SelectFormBlox Example" on page 722
- "A TimePeriodSelectFormBlox Example" on page 726
- "MemberSecurityBlox Tags" on page 760

Chapter 25. Business Logic Blox and TimeSchema DTD Reference

This chapter contains reference material for the three business logic Blox—TimeSchemaBlox, MDBQueryBlox, and MemberSecurityBlox—and their related objects. The Data Type Definition (DTD) for creating a TimeSchema XML is also described.

Blox Logic Tags Overview

DB2 Alphablox provides three business logic Blox to help you add commonly needed business logic in analytic applications— TimeSchemaBlox, MDBQueryBlox, and MemberSecurityBlox: These business logic Blox and FormBlox (discussed in Chapter 24, "Blox Form Tag Reference," on page 699) are designed to solve two commonly encountered problems during analytical application development: the need for data-aware business logic and the need to maintain state.

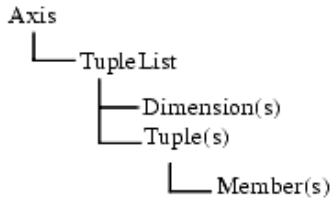
- All three Blox support IBM DB2 OLAP Server,
- Essbase, and Microsoft Analysis Services data sources.
- These Blox and their associated objects reside in the com.alphablox.blox.logic package.
- The tags for these business logic Blox are provided in the Blox Logic Tag Library. To use the Blox Logic tags, you need to include the following taglib import statement in your page:

```
<%@ taglib uri="bloxlogictld" prefix="bloxlogic" %>
```

MDBQueryBlox

MDBQueryBlox is an object representation of a multidimensional data query. It allows you to manipulate an MDB query without using the query language associated with the data source. Using the `<bloxlogic:mdbQuery>` tag or its API, you can manipulate parts of the query such as changing parts of the tuples of an axis. Once a change is made in MDBQueryBlox (by calling its `changed()` method), its source DataBlox is automatically updated and the data query re-executed.

An MDBQueryBlox has three axes: rows, columns and the slicer (or the "page" axis). Each represents a particular part of the query. Each axis is an Axis object composed of multiple tuples, and therefore a TupleList. Each TupleList is defined by dimensions and Tuples.



A Tuple contains a list of members that can be from one or multiple dimensions. The following example shows a GridBlox with one tuple on the row axis, consisting of member "All Products" (the root member) of the "All Products" dimension, and two tuples on the column axis, consisting of two members from the same dimension.

All Products	Qtr 1 01	Qtr 2 01
All Products	1770633.39	2826604.715

The following example shows a GridBlox with two tuples for the column axis. Each tuple is formed by members from two dimensions—All Time Periods and Scenario.

	Qtr 1 01	Qtr 2 01
All Products	Actual	Actual
All Products	1770633.39	2826604.715

An Axis object can also be composed of one or more CrossJoin objects. A CrossJoin produces a "cross product" of the tuples that it joins. For example, if `tuples1 = {"Jan", "Feb"}` and `tuples2 = {"Colas", "Root Beer"}`, then `CrossJoin.getTuples()` will return `{{"Jan", "Colas"}, {"Jan", "Root Beer"}, {"Feb", "Colas"},{"Feb", "Root Beer"}}`. The following example shows four tuples on the column axis as a result of a cross join of:

- "Qtr 1 01" and "Qtr 2 01" from the "All Time Periods" dimension
- "Actual" and "Forecast" from the "Scenario" dimension

	Qtr 1 01		Qtr 2 01	
All Products	Actual	Forecast	Actual	Forecast
All Products	1770633.39	1780642.53	2826604.715	2809041.365

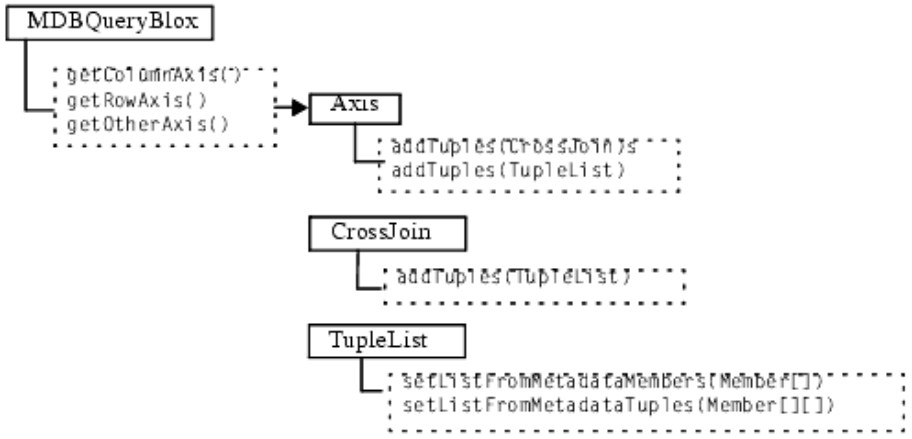
Tags for MDBQueryBlox generally have the following nested relationship:

```
<bloxlogic:mdbQuery>
   <bloxlogic:axis>
      <bloxlogic:tupleList>
```

Or

```
<bloxlogic:mdbQuery>
   <bloxlogic:axis>
      <bloxlogic:crossJoin>
         <bloxlogic:tupleList>
```

A TupleList represents a set of tuples that may be part of an Axis or a CrossJoin.



By setting the dimensions and members for the row, column, or page axis in MDBQueryBlox, you can make changes to the query in parts. For example, using MemberSelectFormBlox, you can create a member selection list that allows users to select members of interest to display on the row axis. The selected members can then be used to set the values of the TupleList's `listFromMetaDataMembers` or `ListFromMetadataTuples` property. This updates the DataBlox referenced once the `TupleList.changed()` method is called. For an example, see "An MDBQueryBlox Example" on page 748.

Specifying TupleList for Each Axis

To define a data query using MDBQueryBlox, specify the type of the axis (`rows`, `columns`, or `pages`) and then the TupleList(s) that form the axis.

```
<%@ taglib uri="bloxlogictld" prefix="bloxlogic"%>

<bloxlogic:mdbQuery id="query" dataBloxRef="myDataBlox">
   <bloxlogic:axis type="rows">
      <bloxlogic:tupleList>

       ... Define the dimension(s) and tuple(s)

      </bloxlogic:tupleList>
   </bloxlogic:axis>

...Specify the list of tuples for rows and columns in the same way

</bloxlogic:mdbQuery>
```

Example 1: A Simple Query: This example demonstrates how to define a simple query with one tuple each on the row and column axes. Each tuple consists of the root member from a dimension. The rendered GridBlox looks as follows:

All Products	All Time Periods
All Products	14072286.895

The tags that specify this query is as follows:

```
<%@ taglib uri="bloxlogictld" prefix="bloxlogic"%>
<bloxlogic:mdbQuery id="myQuery" dataBloxRef="someDataBlox">
   <bloxlogic:axis type="rows">
      <bloxlogic:tupleList>
         <bloxlogic:dimension>All Products</bloxlogic:dimension>
         <bloxlogic:tuple>
            <bloxlogic:member>All Products</bloxlogic:member>
         </bloxlogic:tuple>
      </bloxlogic:tupleList>
   </bloxlogic:axis>

   <bloxlogic:axis type="columns">
      <bloxlogic:tupleList>
         <bloxlogic:dimension>All Time Periods</bloxlogic:dimension>
         <bloxlogic:tuple>
            <bloxlogic:member>All Time Periods</bloxlogic:member>
         </bloxlogic:tuple>
      </bloxlogic:tupleList>
   </bloxlogic:axis>
</bloxlogic:mdbQuery>
```

Example 2: Two Dimensions on an Axis: This example demonstrates how to define a query with a tuple formed by members from different dimensions. The following GridBlox has:

- Chocolate Blocks and Chocolate Nuts from the All Products dimension on the row axis
- A tuple formed by Qtr 1 01 from All Time Periods and Actual from Scenario on the column axis
- Another tuple formed by Qtr 2 01 from All Time Periods and Actual from Scenario on the column axis

	Qtr 1 01	Qtr 2 01
All Products	Actual	Actual
Chocolate Blocks	347784.61	428594.33
Chocolate Nuts	660425.345	1294959.57

The tags that specify the members on the row and column axes are as follows:

```
<%@ taglib uri="bloxlogictld" prefix="bloxlogic"%>

<bloxlogic:mdbQuery>
   <bloxlogic:axis type="rows">
      <bloxlogic:tupleList>
         <bloxlogic:dimension>All Products</bloxlogic:dimension>
         <bloxlogic:tuple>
            <bloxlogic:member>Chocolate Blocks</bloxlogic:member>
         </bloxlogic:tuple>
         <bloxlogic:tuple>
            <bloxlogic:member>Chocolate Nuts</bloxlogic:member>
         </bloxlogic:tuple>
      </bloxlogic:tupleList>
   </bloxlogic:axis>

   <bloxlogic:axis type="columns">
      <bloxlogic:tupleList>
         <bloxlogic:dimension>All Time Periods</bloxlogic:dimension>
         <bloxlogic:dimension>Scenario</bloxlogic:dimension>
         <bloxlogic:tuple>
            <bloxlogic:member>Qtr 1 01</bloxlogic:member>
            <bloxlogic:member>Actual</bloxlogic:member>
         </bloxlogic:tuple>
         <bloxlogic:tuple>
            <bloxlogic:member>Qtr 2 01</bloxlogic:member>
            <bloxlogic:member>Actual</bloxlogic:member>
```

```
        </bloxlogic:tuple>
      </bloxlogic:tupleList>
   </bloxlogic:axis>
</bloxlogic:mdbQuery>
```

MemberSecurityBlox

MemberSecurityBlox provides a list of members a user has access to on a given dimension. It constructs the list by performing a suppressNoAccess on the DataBlox based on the specified MemberSecurityFilter. To set a MemberSecurityFilter, specify the dimension and the member(s) in that dimension using the addMember() or setMember() method.

TimeSchemaBlox

TimeSchemaBlox builds a time table for a given data source based on your definition of a TimeSchema. Using the TimeSchema Data Type Definition (DTD), you can define how the Time dimension is structured by specifying:

- Name(s) of the time dimension(s)
- The generation levels for Year, Quarter, Month and Week
- Start date of the time period in the cube
- Whether Normal Calendar time/Weekly time should be applied
- If the length of a year is exceptional (such as 48-week year)

The XML file containing the definition of the TimeSchema should be named `timeschema.xml` and stored in your application's `WEB-INF/` directory. A `timeschema.dtd` file should also be stored in the same location. The Data Type Definition (DTD) used to define the TimeSchema XML is described in "TimeSchema XML DTD" on page 786.

Once the time schema is configured, TimeSchemaBlox and its related objects will take care of determining the set of members mapped to a given date or time period. The time schema can perform basic date arithmetic and has the ability to produce a sequence of members between dates. Through TimeSchemaBlox, the time schema is available to TimePeriodSelectFormBlox and TimeUnitSelectFormBlox to create a selection list for users to choose a desired time period and unit. Or you can find out information such as the names of the time dimension(s), the current month, quarter, year, or the previous two months, quarters, years, and more through the TimeSchemaBlox API. For TimePeriodSelectFormBlox and TimeUnitSelectFormBlox, see Chapter 24, "Blox Form Tag Reference," on page 699.

The TimeSchemaManager object in the com.alphablox.blox.logic.timeschema package is the global manager that provides access to the TimeSchema object. You can get to the TimeSchema using the TimeSchemaManager's `getTimeSchema()` method. More conveniently, the `<bloxlogic:timeSchema>` tag does the work for you.

PeriodType

PeriodType describes the period type for a TimeSeries. Valid period types are the following constants:

- `PeriodType.YEAR`
- `PeriodType.HALFYEAR`
- `PeriodType.QUARTER`
- `PeriodType.MONTH`

- `PeriodType.WEEK`
- `PeriodType.DAY`.

See the discussion of TimeSeries in the "TimePeriodSelectFormBlox Reference" on page 723 for more details.

TimeMember

TimeMember is an interface representing a slice of the TimeSchema. With TimeMember, you can find out when this slice of the time table begins, when it ends, what tuple is associated with the date, or what Member objects are associated with the slice.

TimeSeries

TimeSeries represents a series of periods with the following properties:

- `baseInterval`: Basic period type, such as `month`, `week`, `quarter`, and `year`. It is used to determine the date range.
- `rollups`: Different types of time unit to include in roll-ups.
- `start`: The starting period; the offset from the current time period, with 0 being the current time period; -1, the previous period; -2, the previous 2 periods; 1, the next period, and so on.
- `count`: Number of periods to be included.
- `toDate`: Indicates if this TimeSeries represents a period to date (`TODATE`) or a sequence of periods (`SEQUENCE`). For example, `TODATE(Month)(Week)` indicates month-to-date with Week as the time unit in the rollup. `SEQUENCE(Month,-12,12)(Month,Quarter)` indicates last 12 months with Month and Quarter as the time units in the rollup.

A time series can be expressed as a string such as `SEQUENCE(QUARTER, 0, 1)(WEEK)`, which means this is a sequence of quarters, starting from this quarter (0), for a count of 1 quarter, and the unit for roll-ups is Week. With a defined time schema, you can use the TimeSeries bean to construct a time series. The following example shows how a TimeSeries of last quarter with Month as the unit for roll-ups may be constructed:

```
<%
TimeSeries lastQuarter = TimeSeries.parseString("SEQUENCE(Quarter, -1, 1)
(MONTH)");
%>
```

You can also specify a time series when using the TimePeriodSelectFormBlox to create a selection list out of the specified periods. For more information, see "TimePeriodSelectFormBlox Reference" on page 723.

Business Logic Blox Properties and Methods Cross-References

This section contains properties and methods cross references for the following components:

MDBQueryBlox Properties and Methods

This section lists all properties and methods for MDBQueryBlox. For associated tag syntax, see "<bloxlogic:mdbQuery> Tag Attributes" on page 746.

Properties	Methods
	`changed()`
	`generateQuery()`
`columnAxis`	
	`getColumnAxis()` `setColumnAxis()`
`cubeName`	
	`getCubeName()` `setCubeName()`
`dataBlox`	
	`getDataBlox()` `setDataBlox()`
`otherAxis`	
	`getOtherAxis()` `setOtherAxis()`
`rowAxis`	
	`getRowAxis()` `setRowAxis()`

Axis Properties and Methods

This section lists all properties and methods for Axis. For associated tag syntax, see "Nested <bloxlogic:axis> Tag" on page 746.

Property	Methods
	`addTuples()`
	`changed()`
`dimensions`	
	`getDimensions()`
`mutable`	
	`isMutable()` `setMutable()`
`queryFragment`	
	`getQueryFragment()` `setQueryFragment()`
`tuples`	
	`getTuples()`
`type`	
	`getType()` `setType()`

	size()

CrossJoin Properties and Methods

This section lists the properties and methods for CrossJoin. For CrossJoin tag syntax, see "Nested <bloxlogic:crossJoin> Tag" on page 747.

Property	Methods
	addTuples()
	changed()
dimensions	
	getDimensions()
tuples	
	getTuples()

TupleList Properties and Methods

This section lists the properties and methods for TupleList. For associated tag syntax, see "<bloxlogic:tupleList> Tag" on page 747.

Property	Methods
	changed()
	clear()
dimensions	
	getDimensions() setDimensions()
list	
	setList()
listFromCrossJoin	
	setListFromCrossJoin()
listFromMetadataMembers	
	setListFromMetadataMembers()
listFromMetadataTuples	
	setListFromMetadataTuples()
	setListFromNames()
tuples	
	getTuples()
	size()

MemberSecurityBlox Properties and Methods

This section lists the properties and methods for MemberSecurityBlox. For associated tag syntax, see "<bloxlogic:memberSecurity>" on page 761.

Property	Methods

cubeName	
	getCubeName() setCubeName()
dataBlox	
	getDataBlox() setDataBlox()
dimensionName	
	getDimensionName() setDimensionName()
displayMemberNames	
	getDisplayMemberNames()
memberSecurityFilter	
	getMemberSecurityFilter() setMemberSecurityFilter()
members	
	getMembers()
rootUniqueNames	
	getRootUniqueNames() setRootUniqueNames()
uniqueMemberNames	
	getUniqueMemberNames()

MemberSecurityFilter Properties and Methods

This section lists the properties and methods for MemberSecurityFilter. For associated tag syntax, see "<bloxlogic:memberSecurityFilter>" on page 761.

Property	Methods
	addMember() clear()
dimensions	
	getDimensions()
	getMember() getMembers() setMember()

TimeSchemaBlox Properties and Methods

This section lists all properties and methods for TimeSchemaBlox. For associated tag syntax, see "TimeSchemaBlox Tag" on page 769.

Property	Methods
	addTimeSchemaEventListener() removeTimeSchemaEventListener()
	current()
	first()
	get()

cubeName	
	getCubeName()
	getDimension()
dimensions	
	getDimensions()
name	
	getName()
periods	
	getPeriods()
	getSequence()
	getTuples()
splitHierarchy	
	isSplitHierarchy()
timeSchemaAvailable	
	isTimeSchemaAvailable()
today	
	getToday() setToday()
	last()
	next()
	previous()
	range()

PeriodType Properties and Methods

This section lists all properties and methods for PeriodType.

Property	Methods
	checkIntervals()
	compareTo()
	equals()
	findPeriod()
	getLargest()
	getSmallest()
value	
	getValue()

	hashCode()
	parseString()
	remove()
	toString()

TimeMember Properties and Methods

This section lists all properties and methods for TimeMember.

Property	Methods
endDate	
	getEndDate()
member	
	getMember()
startDate	
	getStartDate()
tuple	
	getTuple()
	isContainedBy()

TimeSeries Properties and Methods

This section lists all properties and methods for TimeSeries.

Properties	Methods
	equals()
baseInterval	
	getBaseInterval() setBaseInterval()
count	
	getCount() setCount()
rollups	
	getRollups() setRollups()
	getSequence()
start	
	getStart() setStart()
toDate	
	isToDate() setToDate()
	parseString() toString()

MDBQueryBlox Tags

This section describes the tag syntax for MDBQueryBlox:

<bloxlogic:mdbQuery> Tag Attributes

The `<bloxlogic:mdbQuery>` tag has the following attributes:

Attribute	Description
`id`	The unique `id` of this MDBQueryBlox.
`dataBloxRef`	A DataBlox that is already instantiated in the page.
`cubeName`	the name of the cube in the given DataBlox.

General Tag Syntax

Tags related to MDBQueryBlox have the following nested relationship:

```
<bloxlogic:mdbQuery>
   <bloxlogic:axis>
      <bloxlogic:tupleList>
```

Or

```
<bloxlogic:mdbQuery>
   <bloxlogic:axis>
      <bloxlogic:crossjoin>
         <bloxlogic:tupleList>
```

The `<bloxlogic:tupleList>` tag can also stand alone outside the `<bloxlogic:mdbQuery>` tag. See "<bloxlogic:tupleList> Tag" on page 747 for more details.

Nested <bloxlogic:axis> Tag

This tag needs to be nested inside a `<bloxlogic:mdbQuery>` tag. It has the following attributes:

Attribute	Description
`mutable`	`true` if the axis will change in response to changes in its associated DataBlox. The default is `false`. If user interaction is allowed in your presentation Blox, you should set `mutable` to `true` on both your row and column axes so the changes caused by user's data navigation actions will be reflected correctly. Sometimes you may not need to set `mutable` to `true` if you do not allow data drilling (such as by removing the Toolbar and the menubar and setting the component's `clickable` attribute to

	false), and only want the presentation Blox to display data based on some predefined selections.
queryFragment	The query fragment that represents the axis.
type	The type of the axis. Valid values are rows, columns, or pages.

Nested <bloxlogic:crossJoin> Tag

This is a nested tag inside the <bloxlogic:axis> tag. It has no attribute. Inside the <bloxlogic:crossJoin> tag, specify the TupleLists that should be joined. See the example below.

An CrossJoin Example

The following example demonstrates how two TupleLists, one from [Time].[Calendar] and another from [Scenario].[All Scenario].[Actual], are joined on the column axis.

```
<bloxlogic:timeSchema id="timeSchema"
   name="QCC-MSAS" dataBloxRef="myDataBlox" />

<bloxlogic:mdbQuery>
   <bloxlogic:axis type="columns" mutable="true">
      <bloxlogic:crossJoin>

         <bloxlogic:tupleList>
            <bloxlogic:dimension>
               [Time.Calendar]
            </bloxlogic:dimension>
            <bloxlogic:tuple>
               <bloxlogic:member>
                  [Time.Calendar].[2000]
               </bloxlogic:member>
            </bloxlogic:tuple>
         </bloxlogic:tupleList>

         <bloxlogic:tupleList>
            <bloxlogic:dimension>
               [Scenario]
            </bloxlogic:dimension>
            <bloxlogic:tuple>
               <bloxlogic:member>
                  [Scenario].[All Scenario].[Actual]
               </bloxlogic:member>
            </bloxlogic:tuple>
         </bloxlogic:tupleList>

      </bloxlogic:crossJoin>
   </bloxlogic:axis>
</bloxlogic:mdbQuery>
```

<bloxlogic:tupleList> Tag

The <bloxlogic:tupleList> tag is a nested tag within the <bloxlogic:axis> tag. It can also stand alone without being nested. This allows you to specify the id of the TupleList to be referenced later such as in a <bloxform:setChangeProperty> tag.

Attribute	Description
id	The id of the object.
tuplesRef	A TupleList object already instantiated in the page.

The <bloxlogic:tupleList> tag has two nested tags:

- "Nested <bloxlogic:dimension> Tag" on page 748

- "Nested <bloxlogic:tuple> Tag" on page 748

Nested <bloxlogic:dimension> Tag

To specify the dimension in an axis, name the dimension within the <bloxlogic:dimension> tag. For example:

```
<bloxlogic:dimension>[Scenario]</bloxlogic:dimension>
```

To specify a list of dimensions, use the following attribute:

Attribute	Description

list	An array of dimension names.
	
	A useful scenario is when you need to dynamically get the list of dimensions, such as from the TimeSchema:
	`list="<%=myTimeSchema.getDimensions()%>"`
	
	This gives you a list of TimeSchema dimensions defined in the application's timeshema.xml file, assuming a TimeSchemaBlox with an id of myTimeSchema is already created.
	
	You can also construct the list as follows:
	`list="<%= new String[] {"dim1", "dime2"} %>"`

Nested <bloxlogic:tuple> Tag

You can have multiple <bloxlogic:tuple> tags inside a <bloxlogic:tupleList> tag. Each <bloxlogic:tuple> tag can have one or more nested <bloxlogic:member> tags. To specify a list of tuples, use the following attribute:

Attribute	Description
list	An array of string array or Member array.

Nested <bloxlogic:member> Tag

This tag has no attributes. Members available to the dimensions and tuple specified should be added between the opening and closing tag. For example:

```
<bloxlogic:tuple>
   <bloxlogic:member>
      [Locations].[All Locations]
   </bloxlogic:member>
   <bloxlogic:member>
      [Products].[All Products]
   </bloxlogic:member>
</bloxlogic:tuple>
```

An MDBQueryBlox Example

The following example demonstrates the use of MDBQueryBlox and MemberSelectFormBlox to allow users to select members for the row axis.

- A DataBlox is created first without a query.
- Tuples on the row are specified first using the <bloxlogic:tupleList> tag. This TupleList (id="rowTuples") will be the TupleList for the row axis. It is defined outside of the <bloxlogic:mdbQuery> tag so it can have an id that we can set the

property of this object when a selection of member on the row axis is made. This TupleList will get all members under [Locations].[All Locations]. Notice that unique member names are required.

- A MDBQueryBlox is added with the row and column axes of the query defined. For the column axis, we are displaying only Sales, COGS, Gross Margin in the Measures dimension. For the row axis, `rowTuples` is referenced.
- A MemberSelectFormBlox is added to show the members under [Locations], with the initial selected member set to [Locations].[All Locations]. Notice that this setting is the same as the setting in `rowTuples`.
- Once a selection is made by users, the `listFromMetadataMembers` property of `rowTuples` is update. The `changed()` method is called to update the underlying DataBlox.

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxformtld" prefix="bloxform"%>
<%@ taglib uri="bloxlogictld" prefix="bloxlogic"%>
<html>
<head>
   <blox:header />
</html>
<body>
<blox:data id="myDataBlox"
   dataSourceName="QCC-MSAS"/>

<bloxlogic:tupleList id="rowTuples">
   <bloxlogic:dimension>[Locations]</bloxlogic:dimension>
   <bloxlogic:tuple>
      <bloxlogic:member>
         [Locations].[All Locations]
      </bloxlogic:member>
   </bloxlogic:tuple>
</bloxlogic:tupleList>

<bloxlogic:mdbQuery id="myQuery" dataBloxRef="myDataBlox" cubeName="[QCC]">
   <bloxlogic:axis type="rows">
      <bloxlogic:tupleList tuplesRef="rowTuples" />
   </bloxlogic:axis>
   <bloxlogic:axis type="columns" mutable="true">
      <bloxlogic:tupleList>
         <bloxlogic:dimension>[Measures]</bloxlogic:dimension>
            <bloxlogic:tuple>
               <bloxlogic:member>[Measures].[Sales]</bloxlogic:member>
            </bloxlogic:tuple>

            <bloxlogic:tuple>
               <bloxlogic:member>[Measures].[COGS]</bloxlogic:member>
            </bloxlogic:tuple>

            <bloxlogic:tuple>
               <bloxlogic:member>[Measures].[Gross Margin %]
            </bloxlogic:member>
         </bloxlogic:tuple>
      </bloxlogic:tupleList>
   </bloxlogic:axis>
</bloxlogic:mdbQuery>

<bloxform:memberSelect id="locationSelector"
   dataBloxRef="myDataBlox"
   dimensionName="[Locations]"
   selectedMemberName="[Locations].[All Locations]"
   multiple="true" visible="false">
   <bloxform:setChangedProperty formProperty="selectedMembers"
      targetRef="rowTuples"
      targetProperty="listFromMetadataMembers"
      callAfterChange="changed"/>
```

```
</bloxform:memberSelect>

<b>Select Locations for Row Axis:</b>
<blox:display bloxRef="locationSelector" />
<blox:grid id="myGridBlox" width="100%" height="100%">
   <blox:data bloxRef="myDataBlox" />
</blox:grid>
</body>
</html>
```

MDBQueryBlox Methods

This section describes all methods for MDBQueryBlox.

changed()

Notifies the MDBQueryBlox that an axis (or a part thereof) has changed.

Data Sources

Multidimensional

Syntax

Java Method

```
void changed();
```

Usage

The query will be converted to the appropriate scripting language and set on the associated DataBlox.

generateQuery()

Generates a query corresponding to the data in the various axes.

Data Sources

Multidimensional

Syntax

Java Method

```
String generateQuery();
String generateQuery(IDataBlox dataBlox);
```

where:

Argument	Description
dataBlox	The DataBlox to connect to.

Usage

The query will be converted to the appropriate scripting language and the text query is returned.

getColumnAxis()

Gets the axis that contains columns.

Data Sources

Multidimensional

Syntax

Java Method

```
Axis getColumnAxis();
     //returns a com.alphablox.blox.logic.query.Axis object
```

See Also
"setColumnAxis()" on page 752

getCubeName()

Gets the value of the `CubeName` property.

Data Sources
Microsoft Analysis Services

Syntax
Java Method

```
String getCubeName();
```

Usage
This property is not used for IBM DB2 OLAP Server or Hyperion Essbase data sources.

See Also
"setCubeName()" on page 752

getDataBlox()

Gets the DataBlox that is used to connect to a data source.

Data Sources
Multidimensional

Syntax
Java Method

```
IDataBlox getDataBlox();
```

Usage
Returns null if a DataBlox is not set.

See Also
"setDataBlox()" on page 752

getOtherAxis()

Gets the axis that contains the slicers.

Data Sources
Multidimensional

Syntax
Java Method

```
Axis getOtherAxis();
     //returns a com.alphablox.blox.logic.query.Axis object
```

See Also
"setOtherAxis()" on page 753

getRowAxis()

Gets the axis that contains rows.

Data Sources

Multidimensional

Syntax

Java Method

```
Axis getRowAxis();
     //returns a com.alphablox.blox.logic.query.Axis object
```

See Also

"setRowAxis()" on page 753

setColumnAxis()

Sets the axis that contains columns.

Data Sources

Multidimensional

Syntax

Java Method

```
void setColumnAxis(Axis columnAxis);
     //returns a com.alphablox.blox.logic.query.Axis object
```

where:

Argument	Description
`columnAxis`	The Axis that contains columns.

See Also

"getColumnAxis()" on page 750

setCubeName()

Sets the name of the cube from which to retrieve data.

Data Sources

Microsoft Analysis Services

Syntax

Java Method

```
void setCubeName(String cubeName);
```

where:

Argument	Description
`cubeName`	The name of the cube to run queries against.

See Also

"getCubeName()" on page 751.

setDataBlox()

Sets the DataBlox to use to connect to the data source.

Data Sources
Multidimensional

Syntax
Java Method

```
void setDataBlox(IDataBlox dataBlox);
```

where:

Argument	Description
dataBlox	The DataBlox to use.

See Also
"getDataBlox()" on page 751.

setOtherAxis()
Sets the axis that contains the slicers.

Data Sources
Multidimensional

Syntax
Java Method

```
void setOtherAxis(Axis otherAxis);
     //returns a com.alphablox.blox.logic.query.Axis object
```

where:

Argument	Description
otherAxis	The Axis that contains slicers.

See Also
"getOtherAxis()" on page 751.

setRowAxis()
Sets the axis that contains rows.

Data Sources
Multidimensional

Syntax
Java Method

```
void setRowAxis(Axis rowAxis);
     //returns a com.alphablox.blox.logic.query.Axis object
```

where:

Argument	Description
rowAxis	The Axis that contains rows.

See Also
"getRowAxis()" on page 752.

Axis Methods

This section describes all methods for the Axis object.

addTuples()

Adds a set of tuples to this object.

Data Sources

Multidimensional

Syntax

Java Method

```
void addTuples(CrossJoin tuples);
void addTuples(TupleList tuples);
```

where:

Argument	Description
`tuples`	A list of tuples.

changed()

Notifies the MDBQueryBlox that this Axis has changed.

Data Sources

Multidimensional

Syntax

Java Method

```
void changed();
```

Usage

The query will be converted to the appropriate scripting language and set on the associated DataBlox.

getDimensions()

Gets the names of dimensions in the tuples on this axis.

Data Sources

Multidimensional

Syntax

Java Method

```
String[] getDimensions();
```

getQueryFragment()

Gets the query fragment, if any, that represents the axis.

Data Sources

Multidimensional

Syntax

Java Method

```
String getQueryFragment();
```

getTuples()

Returns a two-dimensional array of strings representing tuples in the axis.

Data Sources
Multidimensional

Syntax
Java Method

```
String[][] getTuples();
```

Usage
When the data source is Microsoft Analysis Services, it returns the unique member names.

getType()

Sets the `Type` property of the Axis.

Data Sources
Multidimensional

Syntax
Java Method

```
String getType();
```

Usage
Returns either `Axis.ROW`, `Axis.COLUMN`, or `Axis.PAGE`.

isMutable()

Identifies if the axis will change in response to changes in its associated DataBlox.

Data Sources
Multidimensional

Syntax
Java Method

```
boolean isMutable();
```

setMutable()

Sets the `Mutable` property of the axis.

Data Sources
Multidimensional

Syntax
Java Method

```
void setMutable(boolean mutable);
```

where:

Argument	Description
`mutable`	`true` to have the Axis changes in response to changes in the associated DataBlox.

setQueryFragment()

Sets a query fragment that represents the axis.

Data Sources
Multidimensional

Syntax
Java Method

```
void setQueryFragment(String queryFragment);
```

where:

Argument	Description
`queryFragment`	A string of the query fragment.

Usage
This property overrides any tuples associated with the axis.

setType()

Sets the `Type` property of the Axis.

Data Sources
Multidimensional

Syntax
Java Method

```
void setType(String type);
```

where:

Argument	Description
`type`	Valid values are the following constants:`Axis.ROW`, `Axis.COLUMN`, or `Axis.PAGE`.

size()

Returns the number of TupleList or CrossJoin objects in the axis.

Data Sources
Multidimensional

Syntax
Java Method

```
int size();   //returns an integer
```

CrossJoin Methods

This section describes all methods for the CrossJoin object.

addTuples()

Adds a set of tuples to this object.

Data Sources
Multidimensional

Syntax

Java Method

```
void addTuples(CrossJoin tuples);
void addTuples(TupleList tuples);
```

where:

Argument	Description
`tuples`	A list of tuples.

changed()

Notifies the MDBQueryBlox that this object has changed.

Data Sources

Multidimensional

Syntax

Java Method

```
void changed();
```

Usage

If there is a DataBlox associated with the parent of this object, a new query is generated and updated on the DataBlox.

getDimensions()

Gets the names of dimensions in the tuples on this object.

Data Sources

Multidimensional

Syntax

Java Method

```
String[] getDimensions();
```

getTuples()

Returns a two-dimensional array of strings representing the tuples contained in the cross join.

Data Sources

Multidimensional

Syntax

Java Method

```
String[][] getTuples();
```

Usage

When the data source is Microsoft Analysis Services, it returns the unique member names.

TupleList Methods

This section describes all methods for the TupleList object.

changed()

Notifies the MDBQueryBlox that this object has changed.

Data Sources

Multidimensional

Syntax

Java Method

```
void changed();
```

Usage

If there is a DataBlox associated with the parent of this object, a new query is generated and issued.

clear()

Clears all tuples.

Data Sources

Multidimensional

Syntax

Java Method

```
void clear();
```

getDimensions()

Gets the names of dimensions in the tuples.

Data Sources

Multidimensional

Syntax

Java Method

```
String[] getDimensions();
```

getTuples()

Returns a two-dimensional array of strings of member names.

Data Sources

Multidimensional

Syntax

Java Method

```
String[][] getTuples();
String[][] getTuples(String[] dimensions);
```

where:

Argument	Description
`dimensions`	An array of dimensions.

Usage

When the data source is Microsoft Analysis Services, it returns the unique member names.

setDimensions()

Sets all dimensions for the tuples.

Data Sources

Multidimensional

Syntax

Java Method

```
void setDimensions(String[] dims);
```

where:

Argument	Description
dims	An array of dimensions.

setList()

Sets all the tuples from another TupleList object.

Data Sources

Multidimensional

Syntax

Java Method

```
void setList(TupleList tuples);
```

where:

Argument	Description
tuples	An array of TupleList objects.

setListFromCrossJoin()

Sets all the tuples from a CrossJoin object.

Data Sources

Multidimensional

Syntax

Java Method

```
void setListFromCrossJoin(CrossJoin tuples);
```

where:

Argument	Description
tuples	A CrossJoin object.

setListFromMetadataMembers()

Sets all the tuples from an array of members.

Data Sources

Multidimensional

Syntax

Java Method

```
void setListFromMetadataMembers(Member[] tuples);
```

where:

Argument	Description
`tuples`	An array of Member objects.

setListFromMetadataTuples()

Sets all the tuples from a two dimensional array of Members.

Data Sources

Multidimensional

Syntax

Java Method

```
void setListFromMetadataTuples(Member[][] tuples)
```

where:

Argument	Description
`tuples`	A two-dimensional array of Member objects.

setListFromNames()

Sets all the tuples from a two dimensional array of member names and a corresponding array of dimensions.

Data Sources

Multidimensional

Syntax

Java Method

```
void setListFromNames(String[] dimensions,
                      String[][] tuples);
```

where:

Argument	Description
`dimensions`	A list of dimensions.
`tuples`	A two dimensional array of member names.

size()

Returns the number of tuples in the TupleList.

Data Sources

Multidimensional

Syntax

Java Method

```
int size();
```

MemberSecurityBlox Tags

This section describes the tag syntax for MemberSecurityBlox. For its methods, see "MemberSecurityBlox Methods" on page 763.

<bloxlogic:memberSecurity>

The <bloxlogic:memberSecurity> tag has the following attributes:

Attribute	Description
id	The unique id of this MemberSecurityBlox.
cubeName	The name of the cube to set suppressNoAccess on.
dataBlox	A DataBlox.
dataBloxRef	The name of a DataBlox already instantiated on the page.
dimensionName	The name of the dimension to set suppressNoAccess on.

<bloxlogic:memberSecurityFilter>

The <bloxlogic:memberSecurityFilter> tag has the following attributes:

Attribute	Description
dimensionName	The name of the dimension.
memberName	The name of a member.

A MemberSecurityBlox Example

The following example demonstrates the use of the tags and their nested relationship:

```
<bloxlogic:memberSecurity id="memberSecurity"
   dataBloxRef="dataBlox"
   dimensionName="Market">
   <bloxlogic:memberSecurityFilter
      dimensionName="Measures"
      memberName="Profit" />
   <bloxlogic:memberSecurityFilter
      dimensionName="Measures"
      memberName="Inventory" />
</bloxlogic:memberSecurity>
```

Assuming we want to provide a report on Profit and Inventory by Market, and we want:

- A selection list populated with members from the Market dimension to allow the user to select the markets of interest.
- Only members in the Market dimension for which the user can access the data on Profit and Inventory to be on the list.

To do so, we need to:

- Set the MemberSecurityBlox's dimensionName attribute to Market.
- Set the member security filters to Profit and Inventory in the Measures dimension.

The complete code is as follows:

```
<%@ page import="com.alphablox.blox.logic.MemberSecurityFilter"%>
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxformtld" prefix="bloxform"%>
<%@ taglib uri="bloxlogictld" prefix="bloxlogic"%>
```

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

<blox:data id="dataBlox" query="!"
   dataSourceName="essbaseFilter"/>

<bloxlogic:memberSecurity id="memberSecurity"
   dataBloxRef="dataBlox"
   dimensionName="Market">

   <bloxlogic:memberSecurityFilter
      dimensionName="Measures"
      memberName="Profit" />
   <bloxlogic:memberSecurityFilter
      dimensionName="Measures"
      memberName="Inventory" />

</bloxlogic:memberSecurity>

<bloxform:select id="members"
   visible="false"
   multiple="true"
   size="5" >
   <%
      members.setItems(memberSecurity.getDisplayMemberNames());
   %>
</bloxform:select>

<body>
   <blox:display bloxRef="members" />
</body>
</html>
```

If the database administrator has limited access for the logged in user as follows:

Market	Profit	Inventory
New York	#No Access	8723
Massachusetts	#No Access	3699
Florida	#No Access	5632
Connecticut	#No Access	4852
New Hampshire	#No Access	2838
East	#No Access	25744
California	#No Access	#No Access
Oregon	#No Access	#No Access
Washington	#No Access	#No Access
Utah	#No Access	#No Access
Nevada	#No Access	#No Access
West	29861	#No Access
Texas	#No Access	#No Access
Oklahoma	#No Access	#No Access
Louisiana	#No Access	#No Access
New Mexico	#No Access	#No Access
South	#No Access	#No Access
Illinois	#No Access	#No Access
Ohio	#No Access	#No Access
Wisconsin	#No Access	#No Access
Missouri	#No Access	#No Access
Iowa	#No Access	#No Access
Colorado	#No Access	#No Access
Central	#No Access	#No Access
Market	#No Access	#No Access

The following members will be returned as a result:

- New York
- Massachusetts

- Florida
- Connecticut
- New Hampshire
- East
- West

MemberSecurityBlox Methods

This section describes all methods for MemberSecurityBlox.

getCubeName()

Gets the name of the cube.

Data Sources

Microsoft Analylsis Services

Syntax

Java Method

```
String getCubeName();
```

Usage

This property is not used for IBM DB2 OLAP Server or Hyperion Essbase data sources.

getDataBlox()

Gets the DataBlox referenced in this MemberSecurityBlox.

Data Sources

Multidimensional

Syntax

Java Method

```
DataBlox getDataBlox();
```

getDimensionName()

Gets the name of the dimension.

Data Sources

Multidimensional

Syntax

Java Method

```
String getDimensionName();
```

getDisplayMemberNames()

Gets an array of display member names based on the given `cubeName`, `dimensionName`, and `MemberSecurityFilter`.

Data Sources

Multidimensional

Syntax
Java Method
```
String[] getDisplayMemberNames();
```

Usage
Returns an array of display member names as a string array.

getMemberSecurityFilter()
Gets the MemberSecurityFilter used by this Blox.

Data Sources
Multidimensional

Syntax
Java Method
```
MemberSecurityFilter getMemberSecurityFilter();
```

Usage
Returns null if no MemberSecurityFilter is set.

getMembers()

Gets an array of TupleMember objects based on the given `cubeName`, `dimensionName`, and `MemberSecurityFilter`.

Data Sources
Multidimensional

Syntax
Java Method
```
TupleMembers[] getMembers();
```

Usage
Returns an array of TupleMember objects. See "TupleMember" on page 331.

getRootUniqueNames()
Gets the unique names of root members.

Data Sources
Multidimensional

Syntax
Java Method
```
String[] getRootUniqueNames();
```

Usage
Returns an array of unique names of root members as strings.

getUniqueMemberNames()

Gets an array of unique member names based on the given `cubeName`, `dimensionName`, and `MemberSecurityFilter`.

Data Sources
Multidimensional

Syntax
Java Method

```
String[] getUniqueMemberNames();
```

Usage
Returns an array of unique member names as strings.

setCubeName()
Sets the name of the cube.

Data Sources
Multidimensional

Syntax
Java Method

```
void setCubeName(String cubeName);
```

where:

Argument	Description
cubeName	The name of the cube.

setDataBlox()
Sets the DataBlox from which to return data.

Data Sources
Multidimensional

Syntax
Java Method

```
void DataBlox(DataBlox dataBlox);
```

where:

Argument	Description
dataBlox	The DataBlox which is connected to a data source.

Usage
A DataBlox is required for MemberSecurityBlox to function. The given DataBlox is used for executing queries. The result set of the DataBlox changes after a call to getMembers(), getUniqueMemberNames(), or getDisplayMemberNames() method.

setDimensionName()
Sets the name of the dimension for which to get a list of members.

Data Sources
Multidimensional

Syntax
Java Method

```
void setDimensionName(String dimensionName);
```

where:

Argument	Description
dimensionName	Name of the dimension.

setMemberSecurityFilter()

Sets the MemberSecurityFilter to be used.

Data Sources

Multidimensional

Syntax

Java Method

```
void setMemberSecurityFilter(MemberSecurityFilter memberSecurityFilter);
```

where:

Argument	Description
memberSecurityFilter	The MemberSecurityFilter object.

See Also

See "MemberSecurityFilter Methods" on page 766.

setRootUniqueNames()

Sets the root members.

Data Sources

Multidimensional

Syntax

Java Method

```
void setRootUniqueNames(String[] rootUniqueNames);
```

where:

Argument	Description
rootUniqueNames	An array of the unique names of the root members.

Usage

This method takes unique member names. If a null value is passed, the default dimension root members will be used.

MemberSecurityFilter Methods

This section describes all methods for the MemberSecurityFilter object.

addMember()

Adds the given dimension and member to the filter.

Data Sources

Multidimensional

Syntax

Java Method

```
void addMember(String dimension, String member);
```

where:

Argument	Description
`dimension`	The unique name of a dimension.
`member`	The unique name of a member of the dimension.

Usage
The member should belong to the given dimension. Also, if the same member in the same dimension is already given, it is not added again. To replace a given member, use the `setMember()` method.

See Also
"setMember()" on page 768.

clear()
Clears the filter.

Data Sources
Multidimensional

Syntax
Java Method

```
void clear();
```

getDimensions()
Gets all the dimensions that are available in the filter.

Data Sources
Multidimensional

Syntax
Java Method

```
String[] getDimensions();
```

Usage
Returns the dimensions that were set using addMember() or setMember().

See Also
"addMember()" on page 766, "setMember()" on page 768.

getMember()
Gets the member name of the given dimension that was set in this MemberSecurityFilter.

Data Sources
Multidimensional

Syntax
Java Method

```
String getMember(String dimension);
```

where:

Argument	Description

`dimension`	Unique name of the dimension.

Usage
Returns the member that was set using addMember() or setMember(). If there are multiple members in the given dimension, it returns the first one. Returns null if the given dimension does not have any member set in the filter.

See Also
"addMember()" on page 766, "setMember()" on page 768

getMembers()
Gets the member names of the given dimension that was set in this MemberSecurityFilter.

Data Sources
Multidimensional

Syntax
Java Method

```
String[] getMembers(String dimension);
```

where:

Argument	Description
`dimension`	Unique name of the dimension.

Usage
Returns a String array containing the names of the members that were set using addMember() or setMember(). If there are multiple members in the given dimension, it returns the first one. Returns null if the given dimension does not have any member set in the filter.

See Also
"addMember()" on page 766, "setMember()" on page 768

setMember()
Sets the given dimension and member to the filter.

Data Sources
Multidimensional

Syntax
Java Method

```
void setMember(String dimension, String member);
```

where:

Argument	Description
`dimension`	The unique name of a dimension.
`member`	The unique name of a member of the dimension.

Usage
The member should belong to the given dimension. This method is the same as the `addMember()` method except this method replaces any already given members of the given dimension with the member specified.

See Also

"addMember()" on page 766.

TimeSchemaBlox Tag

This section describes the tag syntax for TimeSchemaBlox. For its methods, see "TimeSchemaBlox Methods" on page 769.

<bloxlogic:timeSchema>

The `<bloxlogic:timeSchema>` tag has the following attributes:

Attribute	Description
`id`	An unique identifier for this TimeSchemaBlox.
`dataBloxRef`	The name of a DataBlox already instantiated in the page.
`name`	The name of the TimeSchemaBlox. The name should match the name specified in the `timeschema.xml` file (the `name` attribute of the `<timeSchema>` element).
`today`	The current date. The date should be specified in the format of `"mm/dd/yyyy"`. For example: today = ″05/01/2003″

A TimeSchemaBlox Example

The `<bloxlogic:timeSchema>` tag creates a TimeSchemaBlox that can be referenced by a TimePeriodSelectFormBlox, a TimeUnitSelectFormBlox, or a MDBQueryBlox to create a time period selection list or to manipulate the data query. The following code snippet shows a TimeSchemaBlox used by a TimePeriodSelectFormBlox . By default, TimePeriodSelectFormBlox presents the users with a list of time periods to choose from. When a selection is made, the `histTuples' listFromMetadataTuples` property is changed accordingly as the `changed()` method is called. For a complete example, see "A TimePeriodSelectFormBlox Example" on page 726.

```
<blox:data id="dataBlox" dataSourceName="MSAS" />
<bloxlogic:timeSchema id="timeSchema" name="MSAS"
   dataBloxRef="dataBlox" />
<bloxlogic:tupleList id="histTuples">
   <bloxlogic:dimension list="<%=timeSchema.getDimensions()%>">
   </bloxlogic:dimension>
</bloxlogic:tupleList>
<bloxform:timePeriodSelect id="historySelector"
   timeSchemaBloxRef="timeSchema"
   selectedSeriesString="SEQUENCE(QUARTER,-1,1)(QUARTER)"
   visible="false">
   <bloxform:setChangedProperty formProperty="tuples"
      targetRef="histTuples"
      targetProperty="listFromMetadataTuples"
      callAfterChange="changed"/>
   </bloxform:timePeriodSelect>
```

TimeSchemaBlox Methods

This section describes all methods for TimeSchemaBlox.

addTimeSchemaEventListener()

Adds the listener as an object to be notified of changes in the TimeSchemaBlox.

Data Sources

Multidimensional

Syntax

Java Method

```
void addTimeSchemaEventListener(
 TimeSchemaBlox.TimeSchemaEventListener listener listener);
```

where:

Argument	Description
`listener`	A TimeSchemaBlox.TimeSchemaEventListener.

Usage

The event is fired whenever the associated DataBlox connects or disconnects.

current()

Gets the current member with a given period type.

Data Sources

Multidimensional

Syntax

Java Method

```
TimeMember current(PeriodType interval);
```

where:

Argument	Description
`interval`	The unit of time to return. Valid time units are expressed as constants. See "PeriodType" on page 739 for a list of the values.

Examples

The following example returns a TimeMember for today's date:

```
<% mytimeSchema.current(PeriodType.DAY); %>
```

See Also

"TimeMember Methods" on page 780

first()

Gets the first (earliest) member in the schema with a given PeriodType.

Data Sources

Multidimensional

Syntax

Java Method

```
TimeMember first(PeriodType type);
```

where:

Argument	Description
type	The unit of time to return. Valid time units are expressed as constants. See "PeriodType" on page 739 for a list of the values.

Examples

The following example returns a TimeMember for the first week in the time schema:

```
<% mytimeSchema.first(PeriodType.WEEK); %>
```

See Also

"TimeMember Methods" on page 780

get()

Gets a member with the given PeriodType for a date or offset from the date.

Data Sources

Multidimensional

Syntax

Java Method

```
TimeMember get(java.util.Date date, PeriodType interval);
TimeMember get(java.util.Date date,
               PeriodType interval,
                int offset);
```

where:

Argument	Description
date	A Java Date object.
interval	The unit of time to return. Valid time units are expressed as constants. See "PeriodType" on page 739 for a list of the values.
offset	The number of units away from the current period. A value of 0 indicates the current period. A negative value indicates a previous period. A positive value indicates a future period.

Examples

The following example asks for last month:

```
get(Calendar.getDate(), PeriodType.MOMTH, -1)
```

The following example asks for this quarter:

```
get(Calendar.getDate(), PeriodType.QUARTER, 0)
```

The following example asks for next week:

```
get(Calendar.getDate(), PeriodType.WEEK, 1)
```

getCubeName()

Gets the name of the cube to which this TimeSchemaBlox applies.

Data Sources

Microsoft Analysis Services

Syntax

Java Method

```
String getCubeName();
```

getDimension()

Gets the name of the dimension that contains the given PeriodType.

Data Sources

Multidimensional

Syntax

Java Method

```
String getDimension(PeriodType periodType);
```

where:

Argument	Description
periodType	A PeriodType. Valid time units are expressed as constants. See "PeriodType" on page 739 for a list of the values.

getDimensions()

Gets the names of all dimensions that are available in the TimeSchemaBlox.

Data Sources

Multidimensional

Syntax

Java Method

```
String[] getDimensions();
```

getName()

Gets the name of the time schema corresponds to the name in `timeschema.xml`.

Data Sources

Multidimensional

Syntax

Java Method

```
String getName();
```

getPeriods()

Gets all PeriodTypes that are available in this TimeSchemaBlox or on a given dimension.

Data Sources

Multidimensional

Syntax

Java Method

```
PeriodType[] getPeriods();
PeriodType[] getPeriods(String dimName);
```

where:

Argument	Description
dimName	The name of a dimension.

getSequence()

Returns the TimeMembers from the given range of members.

Data Sources

Multidimensional

Syntax

Java Method

```
TimeMember[] getSequence(TimeMember[] members,
                         PeriodType[] intervals);
```

where:

Argument	Description
members	A list of TimeMembers.
intervals	A PeriodType as time unit for rollups. Valid time units are expressed as constants. See "PeriodType" on page 739 for a list of the values.

Usage

This is used to get the sequence of members with any desired roll-ups.

Examples

The following example gets a two-dimensional array of Member objects for the last six months including the roll-ups.

```
<% TimeMember nextSixMo[] = myTimeSchema.next(PeriodType.MONTH, 6);
   TimeMember nextSixMoRollups[] =
     myTimeSchema.getSequence(nextSixMo, PeriodType.MONTH);
   Member[][] nextSixMoTuples = myTimeSchema.getTuples(nextSixMoRollups);
%>
```

getToday()

Returns the current date.

Data Sources

Multidimensional

Syntax

Java Method

```
java.util.Date getToday();
```

Usage

Normally this is the same as `Calendar.getDate()`.

getTuples()

Converts an array of TimeMember into a two-dimensional array of Member.

This method is intended to be used with the query object.

Data Sources
Multidimensional

Syntax
Java Method

```
Member[][] getTuples(TimeMember[] members);
```

where:

Argument	Description
`members`	An array of TimeMembers.

Usage
Returns a two-dimensional array of Member objects in the metadata. For more information on the Member metadata object, see "Member" on page 329. For an example, see "getSequence()" on page 773.

isSplitHierarchy()

Returns `true` if the TimeSchemaBlox has years in a different dimension from lower level time periods.

Data Sources
Multidimensional

Syntax
Java Method

```
boolean isSplitHierarchy();
```

isTimeSchemaAvailable()

Returns `true` if a valid time schema is defined for the associated data source.

Data Sources
Multidimensional

Syntax
Java Method

```
boolean isTimeSchemaAvailable;
```

See Also
"TimeSchema XML DTD" on page 786 for information on how to define a time schema.

last()

Gets the last member in the schema with a given PeriodType.

Data Sources
Multidimensional

Syntax
Java Method

```
TimeMember last(PeriodType type);
```

where:

Argument	Description
type	The unit of time to return. Valid time units are expressed as constants. See "PeriodType" on page 739 for a list of the values.

next()

Gets the next N members for a given PeriodType relative to a given date or the current date.

Data Sources

Multidimensional

Syntax

Java Method

```
TimeMember[] next(java.util.Date date,
                  PeriodType interval,
                   int count);
TimeMember[] next(PeriodType interval,
                   int count);
```

where:

Argument	Description
date	A Java Date object.
interval	The unit of time to return. Valid time units are expressed as constants. See "PeriodType" on page 739 for a list of the values.
count	The number of units after the given date to return.

Examples

The following example gets an array of the TimeMember objects for the next six months:

```
<% TimeMember nextSixMon[] = myTimeSchema.next(PeriodType.MONTH, 6); %>
```

previous()

Gets the previous N members for a given PeriodType relative to a given date or the current date.

Data Sources

Multidimensional

Syntax

Java Method

```
TimeMember[] previous(java.util.Date date,
                      PeriodType interval,
                       int count);
TimeMember[] previous(PeriodType interval,
                       int count);
```

where:

Argument	Description
date	A Java Date object.
interval	The unit of time to return. Valid time units are

	expressed as constants. See “PeriodType” on page 739 for a list of the values.
`count`	The number of units before the given date to return.

Examples

The following example gets an array of the TimeMember objects for the previous six months:

```
<% TimeMember preSixMon[] = myTimeSchema.previous(PeriodType.MONTH, 6); %>
```

range()

Gets the members at the requested interval between the start and end dates.

Data Sources

Multidimensional

Syntax

Java Method

```
TimeMember[] range(java.util.Date start,
                   java.util.Date end,
                   PeriodType interval);
```

where:

Argument	Description
`start`	A Java Date object.
`end`	A Java Date object.
`interval`	The unit of time to return. Valid time units are expressed as constants. See “PeriodType” on page 739 for a list of the values.

removeTimeSchemaEventListener()

Removes the listener as an object to be notified of when changes occur in the TimeSchemaBlox.

Data Sources

Multidimensional

Syntax

Java Method

```
void removeTimeSchemaEvenListener(
  TimeSchemaBlox.TimeSchemaEventListener listener);
```

where:

Argument	Description
`listener`	A TimeSchemaBlox.TimeSchemaEventListener.

setToday()

Sets the current date.

Data Sources

Multidimensional

Syntax

Java Method

```
void setToday(java.util.Date date);
```

where:

Argument	Description
`date`	A Java Date object.

PeriodType Methods

This section describes all methods for the PeriodType object.

checkIntervals()

An error checking method that throws an exception if duplicate period types are found in an array of PeriodType.

Data Sources

Multidimensional

Syntax

Java Method

```
void checkIntervals(PeriodType[] intervals);
```

where:

Argument	Description
`intervals`	An array of PeriodTypes.

compareTo()

Compares PeriodTypes. Larger period types win.

Data Sources

Multidimensional

Syntax

Java Method

```
int compareTo(java.lang.Object obj);
```

where:

Argument	Description
`obj`	A PeriodType object to be compared with.

Usage

Years should win against Quarters; Quarters against Months. Returns 0 if both period types are the same. Returns a positive value if this period type is larger than the given period type. Returns a negative value if this period type is smaller than given period type

equals()

Returns `true` when comparing to another PeriodType of the same type.

Data Sources
Multidimensional

Syntax
Java Method

```
boolean equals(java.lang.Object obj);
```

where:

Argument	Description
obj	A PeriodType object to be compared with.

Usage
Returns true if both the objects represent the same period type; falses otherwise

findPeriod()
Given an array of PeriodType, searches for the given single PeriodType contained in the array.

Data Sources
Multidimensional

Syntax
Java Method

```
boolean findPeriod(PeriodType[] types, PeriodType target);
```

where:

Argument	Description
types	An array of PeriodType objects.
target	The PeriodType to find.

getLargest()
Gets the largest PeriodType within an array of PeriodTypes.

Data Sources
Multidimensional

Syntax
Java Method

```
PeriodType getLargest(PeriodType[] intervals);
```

where:

Argument	Description
intervals	An array of PeriodType objects.

getSmallest()
Gets the smallest PeriodType within an array of PeriodTypes.

Data Sources
Multidimensional

Syntax
Java Method

```
PeriodType getSmallest(PeriodType[] intervals);
```

where:

Argument	Description
intervals	An array of PeriodType objects.

getValue()
Gets an integer value that corresponds to the PeriodTypes.

Data Sources
Multidimensional

Syntax
Java Method

```
int getValue();
```

Usage
Bigger period types have higher integer values.

hashCode()
Returns a hash code value for the object. This method is supported for the benefit of hash tables such as those provided by java.util.Hashtable.

parseString()
Returns the PeriodType that corresponds to a given string.

Data Sources
Multidimensional

Syntax
Java Method

```
PeriodType parseString(java.lang.String periodString);
```

where:

Argument	Description
periodString	The possible values are: Year, Half Year, Quarter, Month, Week, and Day.

remove()
Removes a given PeriodType from an array of PeriodTypes.

Data Sources
Multidimensional

Syntax
Java Method

```
PeriodType[] remove(PeriodType[] intervals,
                    PeriodType target);
```

where:

Argument	Description
intervals	An array of PeriodType objects.
target	The PeriodType to be removed.

toString()

Returns the string that describes a period type.

Data Sources

Multidimensional

Syntax

Java Method

```
String toString();
```

Usage

The possible returned strings are: Year, Half Year, Quarter, Month, Week, and Day.

TimeMember Methods

This section describes all methods for the TimeMember object.

getEndDate()

Gets the last date in the TimeMember's range.

Data Sources

Multidimensional

Syntax

Java Method

```
java.util.Date getEndDate();
```

getMember()

Returns the primary member associated with the TimeMember.

Data Sources

Multidimensional

Syntax

Java Method

```
Member getMember();
```

Usage

Returns a Member object. For more information on the Member metadata object, see "Member" on page 329.

getStartDate()

Gets the start date (the first date) in the TimeMember's range.

Data Sources

Multidimensional

Syntax
Java Method

```
java.util.Date getStartDate();
```

getTuple()

Returns an array of members associated with the TimeMember.

Data Sources
Multidimensional

Syntax
Java Method

```
Member[] getTuple();
```

Usage
Returns an array of Member objects. For more information on the Member metadata object, see "Member" on page 329.

isContainedBy()

Returns `true` if this TimeMember is contained by the given TimeMember

Data Sources
Multidimensional

Syntax
Java Method

```
boolean isContainedBy(TimeMember member);
```

where:

Argument	Description
`member`	The TimeMember to be compared with.

TimeSeries Methods

This section describes the methods for the TimeSeries object.

equals()

Returns true if this TimeSeries object is the same as the given TimeSeries object.

Data Sources
Multidimensional

Syntax
Java Method

```
boolean equals(java.lang.Object obj);
```

where:

Argument	Description
`obj`	A TimeSeries object to compare with the current object.

Usage
Returns `true` if the two TimeSeries objects are the same. Two TimeSeries objects are the same if all of the attributes of the TimeSeries objects are the same in both objects. It does not do a pointer comparison of the given objects.

getBaseInterval()
Gets the base period type used to determine the date range.

Data Sources
Multidimensional

Syntax
Java Method
```
PeriodType getBaseInterval();
```

Usage
Returns the base period type.

getCount()
Gets the number of intervals to include in the series.

Data Sources
Multidimensional

Syntax
Java Method
```
int getCount();
```

getRollups()
Gets the different types of time periods to include in the rollups.

Data Sources
Multidimensional

Syntax
Java Method
```
PeriodType[] getRollups();
```

Usage
Returns an array of rollup period types.

getSequence()
Given a TimeSchemaBlox, returns a set of members that represents this series.

Data Sources
Multidimensional

Syntax
Java Method
```
TimeMember[] getSequence(TimeSchemaBlox timeSchema);
```

where:

Argument	Description

`timeSchema`	A TimeSchemaBlox.

Usage
For a TimeSeries of "last 2 quarters," this method returns an array of TimeMember objects corresponding to "last 2 quarters." Each TimeMember is time-aware and has the corresponding tuple information.

getStart()
Gets the offset from the current time period

Data Sources
Multidimensional

Syntax
Java Method

```
int getStart();
```

Usage
0 indicates the current time period; -1, the previous period; 1, the next period, and so on.

isToDate()

Returns true if this TimeSeries represents `TODATE`.

Data Sources
Multidimensional

Syntax
Java Method

```
boolean isToDate();
```

Usage
Returns `false` if the TimeSeries represents `SEQUENCE`. A TimeSeries can represent either a period to date (`TODATE`) or a sequence of periods (`SEQUENCE`).

parseString()
Parses the given time series and returns a TimeSeries object.

Data Sources
Multidimensional

Syntax
Java Method

```
TimeSeries parseString(String string);
```

where:

Argument	Description
`string`	The string to convert to a TimeSeries object in either of the following two formats: • `TODATE(`*period_type*`)(`*rollup_units*`)`. For example, `TODATE(Month)(Week)` indicates a TimeSeries of Month-to-date, with Week as the unit for rollups. `TODATE(Quarter)(Month, Week)` indicates a

TimeSeries of Quarter-to-date, with Month and Week as units in the rollup.

- `SEQUENCE(`*period_type, offset, count)(rollup_units*`)`. For example, `SEQUENCE(Month, -12, 12)(Month)` indicates a TimeSeries starting from 12 months ago and continuing on for 12 months (that is, the last 12 months), with Month as the unit for rollups. `SEQUENCE(Month,-12,12)(Month,Quarter)` indicates last 12 months with Month and Quarter as the units in the rollup.

period_type: valid values are `Day`, `Week`, `Month`, `Quarter`, `Half Year`, and `Year`.

rollup_units: A comma-separated list of *period_type.*

offset: 0 indicates the current time period; -1, the previous period; 1, the next period, and so on

count: the number of intervals to include in the series.

Examples

The following example creates a TimeSeries object for last quarter with Month as the unit for rollups:

```
<%
TimeSeries lastQuarter = TimeSeries.parseString("SEQUENCE(Quarter, -1, 1)
(MONTH)");
%>
```

setBaseInterval()

Sets the basic period type.

Data Sources

Multidimensional

Syntax

Java Method

```
void setBaseInterval(PeriodType baseInterval);
```

where:

Argument	Description
`baseInterval`	The base period type. Valid values are `PeriodType.`DAY, `PeriodType.`WEEK, `PeriodType.`MONTH, `PeriodType.`QUARTER, `PeriodType.`HALFYEAR, and `PeriodType.`YEAR.

setCount()

Sets the number of intervals to include in the series.

Data Sources

Multidimensional

Syntax

Java Method

```
int setCount(int count);
```

where:

Argument	Description
`count`	The number of intervals.

setRollups()

Sets the time unit to include in the rollups.

Data Sources

Multidimensional

Syntax

Java Method

```
void setRollups(PeriodType[] rollups);
```

where:

Argument	Description
`rollups`	An array of PeriodType. x

Usage

If a rollup is included that is bigger than the duration of the sequence, then it will still be included. For example, if `YEAR` is in the rollups for a one month sequence, the current year will be included.

setStart()

Sets the offset from the current time period.

Data Sources

Multidimensional

Syntax

Java Method

```
void setStart(int start);
```

where:

Argument	Description
`start`	0 indicates the current time period; -1, the previous time period; -2, two time periods ago; 1, the next period, and so on.

setToDate()

Specifies if this TimeSeries represents `TODATE` or `SEQUENCE`.

Data Sources

Multidimensional

Syntax

Java Method

```
void setToDate(boolearn toDate);
```

where:

Argument	Description
toDate	true to use TODATE; false to use SEQUENCE.

toString()

Returns this TimeSeries as a string.

Data Sources

Multidimensional

Syntax

Java Method

```
String toString();
```

Usage

Returns a string in the format described in "parseString()" on page 783.

TimeSchema XML DTD

The definition of a TimeSchema should be stored in the timeschema.xml file in a web application's WEB-INF/ directory. This file is reloaded each time it changes. The best way to create your timeschema.xml file is to copy the one in the FastForward application and then make changes to it. The FastForward application directory is located at:

```
<alphablox_dir>/system/ApplicationStudio/FastForward/
```

Each timeschema.xml file can contain multiple TimeSchema, one for each of the data sources needed for your application. This section contains the following topics:

Structure of timeschema.xml

This file has the following general structure:

```
<timeSchemas>
   <timeSchema dataSource="QCC-MSAS" name="QCC-MSAS" type="Weekly1D"
cube="qcc">
      ...
   </timeSchema>

   <timeSchema dataSource="TBC" name="tbc" type="Normal1D">
      ...
   </timeSchema>
</timeSchemas>
```

- timeSchemas is the outmost element.
- Use the timeSchema element for each data source needed in your application.

The following are two examples to demonstrate the general structure.

An Sample TimeSchema for an IBM DB2 OLAP Server or Hyperion Essbase Data Source

The following is an example using an IBM DB2 OLAP Server or Hyperion Essbase data source:

```
<timeSchema dataSource="TBC" name="tbc" type="Normal1D">
   <calculation startDate="01/01/1998"/>
   <dimension name="Year" rootMember="Year">
      <level type="years" generation="1" match="Year"/>
      <level type="quarters" generation="2" match="Qtr{0}"/>
      <level type="months" generation="3" match="{MMM}"/>
   </dimension>
</timeSchema>
```

- This TimeSchema is associated with a data source called `TBC`.
- This TimesShema's name is `tbc`. This is the name used to look up a TimeSchema
- This TimeSchema is of type `"Normal1D"`. This `type` parameter indicates how the length of a year is calculated (`Normal` or `Weekly`), and whether the year members are in the same dimension (`1D`) as the rest of the calendar related members. In this case the year is the same as the "Normal" calendar year and the year members are in same dimension as the rest of the members.
- The `<calculation>` element in the entry specifies that the time table should start with January 1, 1998.
- The `<dimension>` element in the entry specifies that the members are located in the Year dimension, and that the root member is Year.
- Within the `<dimension>` element are three `<level>` elements:
  - The `"years"` level is found on generation 1 of the Year dimension and its members should match the pattern "Year"
  - The `"quarters"` level is found on generation 2 of the Year dimension and its members should match the pattern `"Qtr{0}"` (such as Qtr1 and Qtr2).
  - The `"months"` level is found on generation 3 of the Year dimension and its members should match the pattern `"{MMM}"` (localized three-character month abbreviation such as Jan, Feb, and Mar.)

An Sample TimeSchema for an Microsoft Analysis Services Data Source

The following is an example using a Microsoft Analysis Services data source. This entry is for the QCC-MSAS data source that ships with DB2 Alphablox.

```
<timeSchema dataSource="QCC-MSAS"
   name="QCC-MSAS"
   type="Weekly1D"
   cube="qcc">
   <calculation startDate="01/30/2000">
      <exceptionYear lengthWeeks="48">2000</exceptionYear>
   </calculation>
   <dimension name="[Time].[Fiscal]">
      <level type="years" generation="2" match="[Time].[Fiscal].[All
       Time Periods].[FY{0000}]"/>
      <level type="quarters" generation="3"
       match="[Time].[Fiscal].[All Time Periods].[FY{0000}].[Qtr {0}
       FY{00}]"/>
      <level type="months" generation="4"
           match="[Time].[Fiscal].[All Time Periods].[FY{0000}].
       [Qtr {0} FY{00}].[{MMM} FY{00}]"/>
      <level type="weeks" generation="5" match="[Time].[Fiscal].[All
```

```
      Time Periods].[FY{0000}].[Qtr {0} FY{00}].[{MMM} FY{00}].[{00}-
      {00}-{0000}]"/>
  </dimension>
</timeSchema>
```

- The cube attribute is necessary in this case since MSAS data sources can have multiple cubes.
- The type attribute is set to Weekly1D as this is a year with only 48 weeks. The <exceptionYear> element's lengthWeeks attribute is set to 48 and the time table should be built starting from January 30, 2000.
- The <level> element has a match attribute. Since the TimeSchema only does metadata lookups against unique member names in Microsoft Analysis Services data sources, the match attribute allows you to specify the pattern such as "[Time].[Fiscal].[All Time Periods].[FY{0000}]". The rest of the patterns just add on to this as each higher generation member name incorporates the name of the lower generation.

DTD Elements and Attributes

This section describes the elements and their attributes in the TimeSchema XML DTD.

<timeSchemas>

This is the outmost element. It has no attribute. Inside <timeSchemas> you can have multiple timeSchema elements, one for each data source.

<timeSchema>

The time schema for each data source needed in the application should be defined inside the timeSchema element. It has the following attributes:

Attribute	Required?	Description
cube	Yes for MSAS	The name of the cube. Required for Microsoft Analysis Services data sources.
dataSource	Yes	The name of the data source.
name	Yes	The name for this time schema. This is the name used to do the lookup by the timeSchema tag.
type	Yes	Four valid types: • Normal1D • Normal2D • Weekly1D • Weekly2D In a "normal" TimeSchema the year corresponds exactly with the standard (Gregorian) calendar year: it will have 365 days unless it is a leap year. In a weekly TimeSchema, the year corresponds to 52 weeks except where otherwise noted. This weekly calendar is a common fiscal planning calendar. 1D indicates "one-dimensional," meaning all the members are found in one dimension of an MDB cube. A 2D calendar type is "two-dimensional," with the year members kept in one dimension and the rest of the members in another dimension. This split of dimensions is common practice in IBM DB2 OLAP Server or Hyperion Essbase cube implementation.

Attribute	Required?	Description
useAliases	No	true to use aliases. For IBM DB2 OLAP Server or Hyperion Essbase only. The default is false. When useAliases is set to true, make sure the pattern specified in the match attribute (in the <level> element) uses the aliases.

calculation

This element has the following attribute:

Attribute	Description
startDate	The start date for calculating the time table in the format of mm/dd/yyyy. For example: startDate="01/30/2000"

<exceptionYear>

Most commonly, when using a week-based time schema, the exceptionYear element is used to denote a 53-week year. Since each year is more than 52 weeks, it is necessary to mix in a 53-week year every 5 years or so. It can also be used to shorten a year if data is missing.

Attribute	Required?	Description
lengthDays	No	The number of Days in a year.
lengthWeeks	No	The number of Weeks in a year.

<dimension>

There can be at most two <dimension> elements in a TimeSchema. If the TimeSchema is one-dimensional (type of Normal1D or Weekly1D), there should only be one <dimension> element.

Attribute	Required?	Description
name	Yes	The name of the dimension where the members to use in the TimeSchema reside.
rootMember	No	The root member name.

<level>

The <level> element is nested within the <dimension> element for specifying how members in the specified generation in the dimension should be matched for each of the level type. It has the following attributes:

Attribute	Required?	Description
generation	Yes	The generation in the dimension that represents the specified type. See the type attribute of this element.

Attribute	Required?	Description
match	Yes	The pattern of the unique names to match. Specify the pattern in curly braces, with the following three special characters. • 0: a digit from 0 to 9. • #: an optional digit from 0 to 9. • M: an alphabetic character ([A-Z][a-z]). For example: `match="[Time].[Fiscal].[All Time Periods].[FY{0000}].[Qtr {0} FY{00}].{[MMM} FY {00}]"` will match members such as [Time].[Fiscal].[All Time Periods].[FY2002].[Qtr 1 FY02].[Jan FY02]. **Note:** When useAliases is set to true (in the <timeSchema> element), make sure the pattern specified in the match attribute (in the <level> element) uses the aliases.
order	No	Valid values are ascending or descending. The default is ascending, meaning that members ascend in time through the outline. In ascending order the year 1990 comes before 1991 and the month Jan comes before Feb. Use descending if the members are reversed in the outline for some reason.
startMember	No	The member the TimeSchema should start with. This member must match the pattern.
stopMember	No	The member the TimeSchema should stop at. This member must match the pattern.
type	Yes	Valid values are: • years • quarters • months • weeks

Chapter 26. Blox UI Tags Reference

This chapter contains reference material for the Blox UI modifier tags in the `bloxui.tld` tag library. These tags allow you to perform powerful Blox user interface and data layout modification and customization in the DHTML client.

Blox UI Tags Overview

The Alphablox Tag Libraries provide custom tags to use in a JSP page for creating each Blox. It also includes a Blox UI Tag Library for modifying Blox UI and for adding custom analysis functionality all through the use of tags. These tags work directly with the DHTML user interface model and do not affect other clients.

With the `blox.tld` tag library, you can create and add a Blox to your page. With the `bloxui.tld` tag library, you can customize Blox appearance and behavior above and beyond the Blox properties you can set through the Blox tags. The Blox UI tags usually nest inside a presentation Blox tag as they customize those Blox appearances and behaviors.

Whenever possible, you should use Blox tags to set data properties, general user interface organization such as `chartFirst`, `menubarVisible`, and `splitPaneOrientation`, and general Blox features such as cell alerts and writeback that are available in all clients. Use the Blox UI tags only if you are using the DHTML client and if you need higher level of UI customization than is provided by Blox properties. These tags use styles that override the theme-based Cascading Stylesheet classes settings used in the DHTML client.

There are four types of Blox UI tags:

- Component Customization tags: These are tags for UI component customization, such as customizing menus and toolbars. All the common component names used by the Blox UI model are constants. You can find all the constants in the `ModelConstants` interface under the com.alphablox.blox.uimodel package in the Javadoc. Using the component customization tags, you can identify the components by their names and then specify the values of their attributes such as their position, visibility, or style.
- Custom Layout tags: These are tags that allow primarily customization of grid layout, such as applying a butterfly layout or adding spaces among data columns or rows.
- Analysis tags: These are tags that add data analysis features to in your application.

- Utility tags: These are convenience tags to facilitate processing of actions.

To use the Blox UI modifier tags, you need to include the following taglib import statement in your page:

```
<%@ taglib uri="bloxuitld" prefix="bloxui" %>
```

Blox UI Tag Library Cross References

The Blox UI Tag Library contains the following tags:

Component Customization Tags

- "CalculationEditor Tag" on page 792
- "Component Tag" on page 793
- "Custom Menu Tags" on page 815
- "Custom Toolbar Tags" on page 823

Custom Analysis Tags

- "The <bloxui:bottomN> Tag" on page 797
- "The <bloxui:customAnalysis> Tag" on page 800
- "The <bloxui:topN> Tag" on page 803

Custom Layout Tags

- "The <bloxui:butterflyLayout> Tag" on page 805
- "The <bloxui:compressLayout> Tag" on page 807
- "The <bloxui:customLayout> Tag" on page 808
- "The <bloxui:gridHighlight> Tag" on page 809
- "The <bloxui:gridSpacer> Tag" on page 811
- "The <bloxui:title> Tag" on page 814 (also applies to PresentBlox and ChartBlox)

Utility Tags

- "The <bloxui:actionFilter> Tag" on page 829
- "The <bloxui:gridFilter> Tag" on page 831
- "The <bloxui:clientLink> Tag" on page 833
- "The <bloxui:setProperty> Tag" on page 834

These tags and their attributes are described in the following sections.

CalculationEditor Tag

This tag adds the Calculation Editor option to the right-click menu, the Data menu in the menubar, and the Calculation Editor icon to the toolbar.

The Calculation Editor is a user interface that allows users to add new members by specifying the members involved in the calculation and the calculation expression. Various arithmetic and special calculation functions are available and users can specify where the calculated member should be positioned, what generation level it should be, and how missing values should be treated in the calculation. When users select the Calculation Editor option, the Calculation Editor pops up. To see a sample of the Calculation Editor, bring up the Query Builder from the Assembly tab in the DB2 Alphablox home page. The Query Builder has a Calculation Editor icon added to the toolbar.

The <bloxui:calculationEditor> tag should be nested within a presentation Blox tag as follows:

```
<blox:present id="myPresentUI">
   <blox:data bloxRef="myDataBlox" />
   <bloxui:calculationEditor />
</blox:present>
```

This tag has no attributes.

Component Tag

Component is the base class for all UI model visual components. This class provides default behaviors and properties which are common across all visual components. You can find all the constants representing the components in the ModelConstants interface under the com.alphablox.blox.uimodel package in the Javadoc. Names for the constants are all in uppercase. Their values, when specified in your Blox UI tag attributes, should all be in lowercase with the first letter of second and each subsequent word in uppercase. For your convenience, a list of all constants are available at "Model Constants and Their Values" on page 835.

The <bloxui:component> tag should be nested within a presentation Blox tag to modify the UI component of that Blox.

The complete <bloxui:component> tag is as follows:

```
<bloxui:component
   alignment=""
   bloxRef=""
   clickable=""
   disabled=""
   height=""
   name=""
   positionBefore=""
   style=""
   themeClass=""
   title=""
   tooltip=""
   valignment=""
   visible=""
   width="">

      <bloxui:clientLink
         features=""
         link=""
         target="" />

</bloxui:component>
```

The component tag can be used to customize a named Menu, MenuItem, Toolbar, and ToolbarButton since it is the base for all UI model visual components.

The <bloxui:component> Tag Attributes

Attribute	Required	Description
alignment	No	The horizontal alignment setting of the component. Valid values are left, center, and right.
bloxRef	No	References an existing Blox to apply the tag to when the tag is used outside of the Blox's tag. This allows dynamically setting a Blox's UI components.

Attribute	Required	Description
clickable	No	Set this to false to disable interaction. The component is displayed but is not clickable. The default is true. This attribute needs to be set on the outmost user interface Blox. All nested user interface Blox will inherit the attribute. You cannot set the clickable attribute on a nested user interface Blox.
disabled	No	Set this to true to disable the named component. The component becomes greyed out. The default is false.
height	No	The height of this component in pixels.

name	Yes	The name of the component. To customize the default component in the Blox user interface, specify the value for the UI component. This component can be a Menu, a MenuItem, a Toolbar, a ToolbarButton, a Button, or any component that extends from the Component class. These constants representing the UI component are all in the ModelConstants interface in the com.alphablox.blox.uimodel package.
		
		The following examples show two ways to identify the View menu in the menubar.
		`name="<%= ModelConstants.VIEW_MENU %>"`
		`name="viewMenu"`

style	No	The style to attach to the component. Overrides the default style or theme-based style for the component. For example, the following style sets the background color of the named component to black with no border. `style="background-color: black; border-style:none;"`
themeClass	No	Name of the Cascading Stylesheet classes.
title	Yes for custom menus	The displayed title for the component. Custom components added must have a title. The title cannot contain slashes ("/").
tooltip	No	Tooltip displayed with mouse-over.
valignment	No	The vertical alignment setting of the component. Valid values are top, center, and bottom. The default is center.
visible	No	The visibility of the component. When set to false, the component is not displayed. The default is true.
width	No	The width of this component in pixels.

Nested <bloxui:clientLink> Tag

This is a nested tag for multiple Blox UI tags. See "The <bloxui:clientLink> Tag" on page 833 for details.

Component Tag Examples

Example 1: Customizing a menu item

The following example demonstrates how existing MenuItems (helpHelp, helpAbout, toolsGridOptions, and chartMenu) are customized using the <bloxui:component> tag.

- The <bloxui:component> tag is nested inside a PresentBlox.
- The Help... menu item under the Help menu (name = "helpHelp") is removed by setting its visibility to false (visible="false").
- The About Alphablox menu item (name = "helpAbout") is modified to say "About This App..." (title="About This App...").

- The Grid Options... menu item (`name = "toolsGridOptions"`) is disabled from the Tools menu (`disabled="true"`).
- The Chart menu (`name = "chartMenu"`) is repositioned to before the Data menu (`positionBefore="dataMenu"`).

```
<%@ taglib uri="bloxtld" prefix="blox" %>
<%@ taglib uri="bloxuitld" prefix="bloxui" %>

<blox:data id="dataBlox" dataSourceName="TBC" useAliases="true"
   query="<SYM <ROW(Product) <ICHILD Product <COLUMN(Year, Scenario) Qtr1
Qtr2 <CHILD Scenario Sales !"/>
<html>
<head>
   <blox:header />
</head>
<body>
<blox:present id="myPresentBlox" width="700" height="500" >
   <blox:data bloxRef="dataBlox" />
   <bloxui:component name="helpHelp" visible="false" />
   <bloxui:component name="helpAbout" title="About This App..."
      tooltip="About this application" />
   <bloxui:component name="toolsGridOptions" disabled="true" />
   <bloxui:component name="chartMenu" positionBefore="dataMenu" />
</blox:present>
</body>
</html>
```

This customization task can also be done using the `<bloxui:menu>` and `<bloxui:menuItem>` tags. See "Custom Menu Tags" on page 815 for more information.

Example 2: Dynamically setting visibility of UI components using the `bloxRef` attribute

The following example demonstrates how to let a user turn on or off the Standard toolbar interactively.

- Two HTML buttons are created—Hide Toolbar and Show Toolbar.
- Upon initial load, the `choice` parameter is null.
- When the user clicks one of the buttons, set the value for the `action` parameter, and reload the file (`UITagBloxRef.jsp`) with the appropriate action appended to the URL.
- Use the `<bloxui:component>` tag to set the visibility of the Standard toolbar.

```
<%--UITagBloxRef.jsp --%>
<%@ taglib uri='bloxtld' prefix='blox'%>
<%@ taglib uri='bloxuitld' prefix='bloxui'%>

<%--Check the choice parameter. Upon initial load, the choice
    is null.
--%>

<%
   String choice = request.getParameter( "choice" );
   if ( choice != null ) {
      if ( "showToolbar".equals( choice ) ) {
%>
         <bloxui:component bloxRef="tagBloxRefBlox"
            name="standardToolbar"
            visible="true" />
<%
      }
      else if ( "hideToolbar".equals( choice ) ) {
%>
         <bloxui:component bloxRef="tagBloxRefBlox"
```

```
            name="standardToolbar"
            visible="false" />
<%
      }
   return;
   }
%>

<blox:data id="dataBlox" dataSourceName="qcc-essbase"
   useAliases="true" visible="false"
   query="<ROW (\"All Locations\", \"Measures\") \"Central\" \"East\"
      \"West\" \"All Locations\" \"Gross Margin\" <CHILD \"Ratios\"
      <ASYM <COLUMN (\"Scenario\", \"All Time Periods\") \"Actual\"
      \"Actual\" \"Forecast\" \"Forecast\" \"2000.Q3\" \"2000.Q4\"
      \"2001.Q1\" \"2001.Q2\" !" />

<html>
<head>
<blox:header />
</head>

<body>
<!--Add two buttons to allow users to hide/show the toolbar
    When the button is clicked, reload the page with the
    choice parameter specified.
-->
<input type=button value="Hide Toolbar"
onclick="window.location.href='UITagBloxRef.jsp?render=dhtml&choice=hideToo
lbar'">

<input type=button value="Show Toolbar"
onclick="window.location.href='UITagBloxRef.jsp?render=dhtml&choice=showToo
lbar'">

<hr>
<blox:present id="tagBloxRefBlox" width="700" height="500" visible="true">
   <blox:data bloxRef="dataBlox" />
</blox:present>
</body>
</html>
```

Example 3: Setting a PresentBlox Unclickable

The following example demonstrates how to make a user interface Blox non interactive using the `<bloxui:component` tag's `clickable` attribute.

- The `<bloxui:component>` tag is nested inside a PresentBlox with the component's `name` pointing to the name of the PresentBlox.
- You can only set the `clickable` attribute on the outmost UI Blox. In this example, the `clickable` attribute is set on the PresentBlox. You cannot set the `clickable` attribute on the nested GridBlox or ChartBlox inside the PresentBlox.

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxuitld" prefix="bloxui"%>
<blox:data id="dataBlox"
   dataSourceName="QCC-Essbase" useAliases="true"
   query="<SYM <ROW (\"All Products\") <CHILD \"All Products\"
      <COL (\"All Time Periods\") <CHILD \"All Time Periods\"
      <PAGE(Measures) Sales !" />
<html>
<head>
   <blox:header />
</head>
<body>
<blox:present id="notclickablePresentBlox"
   width="80%" height="70%" menubarVisible="false">
```

```
    <blox:toolbar visible="false" />
    <blox:data bloxRef="dataBlox" />
    <bloxui:component name="notclickablePresentBlox" clickable="false" />
  </blox:present>
  </body>
  </html>
```

Custom Analysis Tags

Custom analysis tags allow you to add custom analytical functionality to your grids and charts. These tags include:

- The <bloxui:bottomN> Tag
- The <bloxui:customAnalysis> Tag
- The <bloxui:eightyTwenty> Tag
- The <bloxui:percentOfTotal> Tag
- The <bloxui:topN> Tag

Once you add these tags in your presentation Blox tags (PresentBlox, GridBlox, or ChartBlox), these custom analytical functions show up in the right-click menu and the menubar's Data menu under the Advanced option. For example, the following tags add six advanced data analysis options:

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxuitld" prefix="bloxui"%>
...
<html>
<head>
   <blox:header />
</head>
<body>
<blox:present ...>
   <bloxui:topN number="10" showRank="true" />
   <bloxui:topN number="5" showRank="true" />
   <bloxui:topN prompt="true" showRank="true"  number="20"/>
   <bloxui:bottomN number="10" showRank="true" />
   <bloxui:bottomN number="5" showRank="true" />
   <bloxui:bottomN prompt="true"  showRank="true"  number="7"/>
</blox:present>
</body>
</html>
```

The six added options show up in the right-click menu and the menubar's Data menu.

Tip: Only one analysis operation can be in effect at a time. If the user chooses Top 10 first and then Bottom 5 or Percent of Total, each is an independent operation and does not retain the result from the previous operations.

The <bloxui:bottomN> Tag

The following shows all tag attributes for the `<bloxui:bottomN>` tag. This tag should be nested within the tag of a PresentBlox or GridBlox:

```
<bloxui:bottomN
   description=""
   hideOthers=""
   membersToAnalyze=""
   number=""
   preserveGrouping=""
```

```
   prompt=""
   showOtherSummary=""
   showRank=""
/>
```

where:

Attribute	Required	Default	Description
description	No	Bottom *N*; Bottom N...	Specifies the text for this menu item under the Advanced menu. The default is "Bottom *N*" where *N* is the value of the number attribute. If the prompt attribute is set to true, the default is "Bottom N...."
hideOthers	No	all	Specifies whether the remaining non-ranked members and remaining members not involved in the calculation should be hidden from view. Valid values are: all: the default; hiding non-ranked members and members not involved in the calculations from both the column and the row axes. none: keep all the non-ranked members and members not involved in the calculations from both the column and row axes. unranked: hide only the non-ranked members; keep the members not involved on the opposite axis in the calculations.
membersToAnalyze	No	leaf	Specifies whether to rank only the currently displayed members or all the leaf members. Valid values are: • displayed: ranks only the currently displayed members. • leaf: ranks all leaf members.
number	No	10	Specifies the number of members with the lowest values to show. If a number is not specified, the option "Bottom 10" will be displayed in the menu. When the prompt attribute is set to true, value for this attribute will become the default number shown in the pop-up dialog, prompting for the number of members to show.
preserveGrouping	No	true	Specifies whether to preserve grouping when ranking. When true, ranking is calculated per group when there are multiple dimensions on the axis. See "Example 1: Bottom 10 Analysis" on page 799.

Attribute	Required	Default	Description
prompt	No	false	Specifies whether to prompt for a number. When prompt is set to true, a dialog pops up, prompting the user to enter the number of members of the lowest values they want to see. To specify the default value for the prompt, use the number attribute.
showOtherSummary	No	false	Specifies whether to add an "Other" row/column as a summary for the remaining, un-ranked members.
showRank	No	true	Specifies whether to show the ranking in an added column/row. If false, the members are sorted according to the rank without the added column/row showing the ranking

bottomN Tag Examples

Example 1: Bottom 10 Analysis

The following code will add a Bottom 10 advanced analysis option to the right-click menu.

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxuitld" prefix="bloxui"%>
...
<blox:grid ...>
   <bloxui:bottomN
     number="10"
     showRank="true" />
</blox:grid>
```

When a user selects this option, a column (or row, depending on whether the user selects a row header or column header) called "[member name] Bottom 10" will be added to the grid.

Example 2: Bottom N Analysis with Prompt

By default, unranked members on both the column and row axes are hidden unless hideOthers is set to none or unranked. To allow users to specify ranking options such as the number of members they want to rank and whether to rank only the currently displayed members or leaf members only, set the prompt attribute to true:

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxuitld" prefix="bloxui"%>
<blox:grid ...>
   <bloxui:bottomN
     prompt="true"
     number="7"
     showRank="true" />
</blox:grid>
```

When users choose this option, a dialog pops up with the value set in number being the default number of members to display.

Example 3: Bottom 5 And Other

The following code will add a Bottom 5 and Other menu option to the right-click menu:

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxuitld" prefix="bloxui"%>
...
<blox:grid ...>
   <bloxui:bottomN
     description="Bottom 5 and Other"
     number="5"
     hideOthers="all"
     showOtherSummary="true"
     showRank="true" />
</blox:grid>
```

When a user selects this option, a column (or row, depending on whether the user selects a row header or column header) called "[member name] Bottom 5" will be added to the grid, with an added "Other" member on the opposite axis that provides summary value for the remaining non-ranked members. Note that `hideOthers` is set to `all` (the default) so the unranked members and members not involved in the calculations from both the column and row axes are not displayed.

If `hideOthers` is set to `unranked`, then only unranked members are hidden. Members on the opposite axis that are not involved in the calculation remain in the grid:

The <bloxui:customAnalysis> Tag

The following shows all tag attributes for the `<bloxui:customAnalysis>` tag. This tag should be nested within the tag of a PresentBlox or GridBlox:

```
<bloxui:customAnalysis
   analysis="" />
```

where:

Attribute	Required	Description

`analysis`	Yes	Specifies the custom analysis object. For example:
		`<bloxui:customAnalysis`
		`analysis="<%= new TopN() %>" />`
		
		The custom analysis object (`TopN` in the above example) should implement `AbstractAnalysis` in the com.alphablox.blox.uimodel.tags.analysis package.

The <bloxui:eightyTwenty> Tag

The following shows all tag attributes for the `<bloxui:eightyTwenty>` tag. This tag should be nested within the tag of a PresentBlox and GridBlox.

```
<bloxui:eightyTwenty
   description=""
   hideOthers=""
   membersToAnalyze=""
   number=""
   preserveGrouping=""
   prompt="" />
```

where:

Attribute	Required	Default	Description
description	No	80/20 Analysis	Specifies the text for this menu item under the Advanced menu. If the prompt attribute is set to true, the default is "80/20 Analysis"
hideOthers	No	all	Specifies whether the remaining non-ranked members and remaining members not involved in the calculation should be hidden from view. Valid values are: all: the default; hiding non-ranked members and members not involved in the calculations from both the column and the row axes. none: keep all the non-ranked members and members not involved in the calculations from both the column and row axes. unranked: hide only the non-ranked members; keep the members not involved on the opposite axis in the calculations.
membersToAnalyze	No	leaf	Specifies whether to include only the currently displayed members or all the leaf members in the calculation. Valid values are: • displayed: includes only the currently displayed members. • leaf: includes all leaf members.
number	No	80	Specifies the top percentage of data to show. When the prompt attribute is set to true, the default value 80 is shown in the pop-up dialog unless the number attribute is set differently.
preserveGrouping	No	true	Specifies whether to perform the calculation per group or regardless of groups. When true, calculation is performed per group when there are multiple dimensions on the axis.
prompt	No	false	Specifies whether to prompt for a number. When prompt is set to true, a dialog pops up, prompting the user to set the options.

The <bloxui:percentOfTotal> Tag

The <bloxui:percentOfTotal> tag calculates the total of all the members and the percentage of each member and display them. It needs to be added within the tag of a PresentBlox or GridBlox. It has the following attributes:

```
<bloxui:percentOfTotal
   description=""
   hideOthers=""
   membersToAnalyze=""
```

```
   number=""
   preserveGrouping=""
   prompt=""
/>
```

where:

Attribute	Required	Default	Description
description	No	Percent of Total	Specifies the text for this menu item under the Advanced menu. If the prompt attribute is set to true, the default is "Percent of Total..."
hideOthers	No	all	Specifies whether the remaining non-ranked members and remaining members not involved in the calculation should be hidden from view. Valid values are: all: the default; hiding non-ranked members and members not involved in the calculations from both the column and the row axes. none: keep all the non-ranked members and members not involved in the calculations from both the column and row axes. unranked: hide only the non-ranked members; keep the members not involved on the opposite axis in the calculations.
membersToAnalyze	No	leaf	Specifies whether to include only the currently displayed members or all the leaf members in the calculation. Valid values are: • displayed: includes only the currently displayed members. • leaf: includes all leaf members.
number	No	100	Specifies the percentage of data to show. When the prompt attribute is set to true, the default value 100 is shown in the pop-up dialog unless the number attribute is set differently.
preserveGrouping	No	true	Specifies whether to perform the calculation per group or regardless of groups. When true, calculation is performed per group when there are multiple dimensions on the axis.
prompt	No	false	Specifies whether to prompt for a number. When prompt is set to true, a dialog pops up, prompting the user to set the options.

percentOfTotal Tag Example

The following example adds a "Percent of Total" option to the right-click menu and the menubar's Data menu.

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxuitld" prefix="bloxui"%>
...
<blox:grid ...>
   <bloxui:percentOfTotal/>
</blox:grid>
```

The <bloxui:topN> Tag

This `<bloxui:topN>` tag should be nested within the tag of a PresentBlox or GridBlox. It has the following tag attributes:

```
<bloxui:topN
   description=""
   hideOthers=""
   membersToAnalyze=""
   number=""
   preserveGrouping=""
   prompt=""
   showRank=""
   showOtherSummary=""
/>
```

where:

Attribute	Required	Default	Description
`description`	No	Top *N;* Top N...	Specifies the text for this menu item under the Advanced menu. The default is "Top *N*" where *N* is the value of the `number` attribute. If the `prompt` attribute is set to `true`, the default is "Top N...."
`hideOthers`	No	`all`	Specifies whether the remaining non-ranked members and remaining members not involved in the calculation should be hidden from view. Valid values are: `all`: the default; hiding non-ranked members and members not involved in the calculations from both the column and the row axes. `none`: keep all the non-ranked members and members not involved in the calculations from both the column and row axes. `unranked`: hide only the non-ranked members; keep the members not involved on the opposite axis in the calculations.
`membersToAnalyze`	No	`leaf`	Specifies whether to rank only the currently displayed members or all the leaf members. Valid values are: • `displayed`: ranks only the currenlty displayed members. • `leaf`: ranks all leaf members.

Attribute	Required	Default	Description
number	No	10	Specifies the number of members with the highest values to show. If a number is not specified, the option "Top 10" will be displayed in the menu. When the prompt attribute is set to true, value for this attribute will become the default number shown in the pop-up dialog, prompting for the number of members to show.
preserveGrouping	No	true	Specifies whether to rank the members per group or regardless of the group. When true, calculation is performed per group when there are multiple dimensions on the axis.
prompt	No	false	Specifies whether to prompt for a number. When prompt is set to true, a dialog pops up, prompting the user to enter the number of members of the highest values they want to see. To specify the default value for the prompt, use the number attribute.
showOtherSummary	No	false	Specifies whether to add an "Other" row/column as a summary for the remaining, unranked members.
showRank	No	true	Specifies whether to show the ranking in an added column/row. If false, the members are sorted according to the rank without the added column/row showing the ranking.

topN Tag Example

The following code will add a Top 10 and Other advanced analysis option to the right-click menu.

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxuitld" prefix="bloxui"%>
...
<blox:grid ...>
   <bloxui:topN
     description="Top 10 and Other"
     number="10"
     hideOthers="all"
     showOtherSummary="true"
     showRank="true" />
</blox:grid>
```

When a user selects this option, a column (or row, depending on whether the user selects a row header or column header) called "[member name] Top 10" will be added to the grid, with an added "Other" member on the opposite axis that provides summary value for the remaining non-ranked members. Note that hideOthers is set to all (the default) so:

- The unranked members are not displayed in addition to the added "Other" member, which may result in confusion.

- Members not involved in the calculation (such as Budget and Variance in the Scenario dimension) are hidden.

If you only want to hide the unranked member and keeping Budget and Variance in the Scenario dimension in the above example, then set `hideOthers` to `unranked`.

Custom Layout Tags

The custom layout tags work primarily on the GridBlox user interface in the DHTML client. These tags allow you to customize the layout of your grid by placing row or column headers in the middle or center of the grid, highlighting certain rows or columns, adding blank rows or columns, and more. The layout tags include the following:

- "The <bloxui:butterflyLayout> Tag" on page 805
- "The <bloxui:compressLayout> Tag" on page 807
- "The <bloxui:customLayout> Tag" on page 808
- "The <bloxui:gridHighlight> Tag" on page 809
- "The <bloxui:gridSpacer> Tag" on page 811
- "The <bloxui:title> Tag" on page 814 (also applies to PresentBlox and ChartBlox)

Except for the `<bloxui:title>` tag, these tags should be nested within a `<blox:present>` or a standalone `<blox:grid>` tag.

The <bloxui:butterflyLayout> Tag

This tag lets you position the row header column in the specified location in the grid. It has the following tag attributes. This tag should be nested within the tag of a `<blox:present>` or `<blox:grid>` tag.

```
<bloxui:butterflyLayout
   addSeparatorColumns=""
   applyLayout=""
   description=""
   position=""
   scope=""
   separatorWidth=""
   showOnLayoutMenu="" />
```

where:

Attribute	Required	Description
`addSeparatorColumns`	No	When set to `true`, this adds an empty column between the header column and the data columns on its two sides. The default is `false`.
`applyLayout`	No	`true` — apply this layout when the page is loaded; `false` — do not apply this layout when the page is loaded. The default is `true`. Sometimes you may not want a layout to be applied until the user chooses to. In this case, set the `showOnLayoutMenu` attribute to `true` and turn on menubar in your presentation Blox.
`description`	No	The display layout name when `showOnLayoutMenu` is set to `true`. The default value is "`Butterfly`."
`position`	No	Specifies whether the header column should be added before or after the scope specified. Valid values are `before` and `after`. The default is `before`.

Attribute	Required	Description
showOnLayoutMenu	No	When set to true, this adds a Format menu to the menubar (if it does not exist already), with the Butterfly menu option under the Layout submenu. The default is false.

scope	Yes	Defines the members relative to which the header column should be displayed. The following example has the header column positioned before Forecast:
		`scope="Forecast"`
		`postion="before"`
		
		To place the header column before a tuple, separate scope members with semicolons and enclose the entire scope in curly brackets as shown in the following example:
		`scope="{Forecast; Qtr 1 01}"`
		`position="before"`
		
		Specification of the tuple preserves the relative position of the header column to the tuple. In the example above where the tuple {Forecast; Qtr 1 01} is specified, when users drill down on Qtr 1 01, the tuples {Forecast; Jan 01}, {Forecast; Feb 01}, and {Forecast; Mar 01} will be displayed to the left of the row column header while {Forecast; Qtr 1 01} is displayed to the right of the header.

separatorWidth	No	Sets the width of the separator columns in pixels. The default is 10 pixels.
showOnLayoutMenu	No	When set to true, this adds a Format menu to the menubar (if it does not exist already), with the Butterfly menu option under the Layout submenu. The default is false.

You may want to limit the data navigation functions allowed in a grid displayed in a butterfly layout since the layout may become irrelevant if the data is changed. For example:

- If users choose to hide the tuple specified as the scope, the layout cannot be applied. Instead of a butterfly layout, users will see a normal grid with row header columns on the left.
- The <bloxui:butterflyLayout> tag only supports column-based layout (vertical butterfly), with row headers in the middle and data to the left and right. The format may be lost if users choose to pivot or swap axes (no horizontal butterfly).
- The row header column is always displayed based on the relative position to the scope you specified. If you specify the header column to appear before the tuple {Forecast; Qtr 1 01}, when users drill down on Qtr 1 01, the tuples {Forecast; Jan 01}, {Forecast; Feb 01}, and {Forecast; Mar 01} will appear to the left of the row column header while {Forecast; Qtr 1 01} is displayed to the right of the header.

Since this tag works on the grid UI to display in a specific layout, it may be confusing to users if the layout disappears after some interaction with the grid. You can discourage data navigation by turning off toolbar and menubar. You can also disable data navigation functions using the <bloxui:menu> or

<bloxui:component> tag, or remove certain actions by setting the common Blox property removeAction. Or you can catch the events and display a message box informing the users that the actions are not supported. See the *Developer's Guide* for additional information and an example.

butterflyLayout Tag Example

The following example applies the butterfly layout to the grid with the header column positioned before the member Forecast. This also adds a Butterfly layout menu option to the menubar's Format menu.

```
<%@ taglib uri="bloxtld" prefix="blox" %>
<%@ taglib uri="bloxuitld" prefix="bloxui" %>
...
<blox:present id="myPresent" width="600" height="500" >
   ...
   <bloxui:butterflyLayout scope="Forecast"
      showOnLayoutMenu="true"/>
...
</blox:present>
```

The <bloxui:compressLayout> Tag

This tag lets you compress the column and row headers into one level when there are multiple dimensions on the column or row axes. It has the following tag attributes. This tag should be nested within the tag of a <blox:present> or <blox:grid> tag.

```
<bloxui:compressLayout
   applyLayout=""
   compressColumns=""
   compressRows=""
   description=""
   memberSeparator=""
   showOnLayoutMenu="" />
```

where:

Attribute	Required	Description
applyLayout	No	true — apply this layout when the page is loaded; false — do not apply this layout when the page is loaded. The default is true. Sometimes you may not want a layout to be applied until the user chooses to. In this case, set the showOnLayoutMenu attribute to true and turn on menubar in your presentation Blox.
compressColumns	No	true — to compress the column headers into one level. If you have multiple dimensions on the column axis, the headers can be compressed into one level. The default for compressColumns is false. When this attribute is set to true, the default members separator is a vertical bar (" | "). **Note:** If this or the compressRows attributes are not specified, the tag will not do anything.

Attribute	Required	Description
compressRows	No	true — to compress the column headers into one level. If you have multiple dimensions on the row axis, the headers will be compressed into one level. The default for compressRows is false. When this attribute is set to true, the default members separator is a vertical bar (" | "). **Note:** If this or the compressColumns attributes are not specified, the tag will not do anything.
description	No	The display layout name when showOnLayoutMenu is set to true. The default value is "Compressed Headers."
memberSeparator	No	The text to use to separate the members. The default value is a vertical bar (" | ") with a space before and after.
showOnLayoutMenu	No	When set to true, this adds a Format menu to the menubar (if it does not exist already), with an added menu option under the Layout submenu, labeled after the value of the description attribute. The default is false.

compressLayout Tag Example

The following example compresses the row and column headers in the grid using " / " as the member separator. This also adds a Compressed Layout menu option to the menubar's Format menu.

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxuitld" prefix="bloxui"%>

<html>
<head>
   <blox:header />
</head>
...
<body>
<blox:present id="myPresent" width="600" height="500" >
   ...
   <bloxui:compressLayout
       compressRows="true"
       compressColumns="true"
       showOnLayoutMenu="true"
       memberSeparator = " / " />
...
</blox:present>
...
</body>
</html>
```

Note: When column or row headers are compressed, all the model components are copied into a single grid header cell. For example, when Actual and Qtr 3 01 are compressed using " / " as the separator, the three Static components are placed into a single cell.

The <bloxui:customLayout> Tag

This tag lets you custom grid layouts. It has the following tag attributes. This tag should be nested within the tag of a <blox:present> or <blox:grid> tag.

```
<bloxui:customLayout
   applyLayout=""
   layout=""
   showOnLayoutMenu=""/>
```

where:

Attribute	Required	Description
applyLayout	No	true — apply this layout when the page is loaded; false — do not apply this layout when the page is loaded. The default is true. Sometimes you may not want a layout to be applied until the user chooses to. In this case, set the showOnLayoutMenu attribute to true and turn on menubar in your presentation Blox.
layout	Yes	Specifies the layout object. For example: `<bloxui:customLayout layout="<%= new ButterflyLayout() %>" />` The name of the class with the custom layout. Can be a custom class that implements the AbstractLayout class in the com.alphablox.blox.uimodel.tags.layout package, or an existing class in the package.
showOnLayoutMenu	No	When set to true, this adds a Format menu to the menubar (if it does not exist already), with an added menu option under the Layout submenu, labeled after the value of the layout attribute. The default is false.

The <bloxui:gridHighlight> Tag

This tag lets you highlight a member or members by specifying the scope and the style to use. It has the following tag attributes. This tag should be nested within the tag of a <blox:present> or <blox:grid> tag.

```
<bloxui:gridHighlight
   applyLayout=""
   description=""
   includeData=""
   includeHeaders=""
   scope=""
   selection=""
   showOnLayoutMenu=""
   style="" />
```

where:

Attribute	Required	Description
applyLayout	No	true — apply this layout when the page is loaded; false — do not apply this layout when the page is loaded. The default is true. Sometimes you may not want a layout to be applied until the user chooses to. In this case, set the showOnLayoutMenu attribute to true and turn on menubar in your presentation Blox.
description	No	The display layout name when showOnLayoutMenu is set to true.

Attribute	Required	Description
includeData	No	true — include data cells in the specified scope; false — exclude data cells in the specified scope; the default is true.
includeHeaders	No	true — include header cells in the specified scope; false — exclude header cells in the specified scope; the default is true.
scope	Yes. Otherwise, specify the selection, or the tag will not do anything.	Defines the members to be highlighted. Separate the members in the scope using semicolons, enclosed with curly brackets. The following example has the gross margin for the West region to be highlighted: scope="{West;Gross Margin}"
selection	Yes. Otherwise, specify the scope, or the tag will not do anything.	Specifies either rowHeaders or columnHeaders to be highlighted. This allows you to customize the style for all row headers or column headers.
showOnLayoutMenu	No	When set to true, this adds a Format menu to the menubar (if it does not exist already), with an added menu option under the Layout submenu, labeled after the value of the description attribute. The default is false.
style	Yes. Otherwise the theme-based style is applied as usual and the tag will have no effect.	The style to attach to the grid highlight. Overrides the default style or theme-based style for the component. For example, the following style sets the background color of the highlighted cells to black with no border. style="background-color: black; border-style:none;"

gridHighlight Tag Example

The following example highlights the column headers in the grid with black text on yellow background color when the page is loaded. Another layout to highlight gross margin for the West region is not applied upon page load (applyLayout="false"). Users can choose to apply this layout from the Format menu in the menubar.

```
<%@ taglib uri="bloxtld" prefix="blox" %>
<%@ taglib uri="bloxuitld" prefix="bloxui" %>
...
<html>
<head>
   <blox:header />
</head>
<body>
...
<blox:present id="myPresent" width="600" height="500" >
   <blox:data dataSourceName="myData" query="<your query here>"... />
   <bloxui:gridHighlight
      description="Highlight Column Headers"
      selection="columnHeaders"
      style="color: black; background-color: yellow;"
      showOnLayoutMenu="true"/>
   <bloxui:gridHighlight
      description="Highlight West Gross Margin"
      scope="{west;gross margin}"
      style="font-weight:bold; color: teal; background-color: #FFFF99;"
      showOnLayoutMenu="true"
```

```
      applyLayout="false"/>
...
</blox:present>
</body>
</html>
```

The <bloxui:gridSpacer> Tag

This tag lets you add space between columns or rows. It has the following attributes. This tag should be nested within the tag of a <blox:present> or <blox:grid> tag.

```
<bloxui:gridSpacer
   applyLayout=""
   axis=""
   description=""
   height=""
   locked=""
   position=""
   scope=""
   showOnLayoutMenu=""
   style=""
   width="" />
```

where:

Attribute	Required	Description
applyLayout	No	true — apply this layout when the page is loaded; false — do not apply this layout when the page is loaded. The default is true. Sometimes you may not want a layout to be applied until the user chooses to. In this case, set the showOnLayoutMenu attribute to true and turn on menubar in your presentation Blox.
axis	Yes	Specifies column or row to add a spacer.
description	No	The display layout name when showOnLayoutMenu is set to true.
height	No	Number of pixels for the height of the spacer when a spacer is added to the row axis (axis="row"). The default is 10 pixels.
locked	No	When set to true, this locks the spacer location on the screen so it does not scroll outside of the viewing area. Since the space added is actually an empty row/column, if locked is set to false, this empty row/column will scroll as normal data rows/columns. This setting only applies when the spacer is added with a position of before or after. The default is false.

Attribute	Required	Description
position	Yes	Specifies the position for the spacer. Valid values are: • after: adds a spacer after the named member. Use the scope attribute to specify the member. • before: adds a spacer before the named member. Use the scope attribute to specify the member. • bottom: adds a spacer to the bottom of the grid. Attribute axis should be set to row. • interlace: adds a spacer between columns or rows. • left: adds a spacer to the left of the grid. Attribute axis should be set to column. • memberchange: adds a spacer when member in the specified dimension (through the scope attribute) changes. • right: adds a spacer to the right of the named axis. Attribute axis should be set to column. • top: adds a spacer to the top of the grid. Attribute axis should be set to row. • A number: adds a spacer at the Nth column or row. For example, the following code adds a spacer as the third rows (0 being the top): `position="2" axis="row"` A value of 0 is the same as adding a spacer to the top or left of the grid. The axis attribute needs to be set for this to work. When position is set to memberchange, a scope has to be specified.

scope	Yes	Defines the scope containing the dimension to use when position is set to memberchange. The following example specifies to add a spacer whenever the member in Forecast is changed:
		`scope="Forecast"`
		`position="memberchange"`

showOnLayoutMenu	No	When set to true, this adds a Format menu to the menubar (if it does not exist already), with an added menu option under the Layout submenu, labeled after the value of the description attribute. The default is false.
style	No	The style to attach to the spacer. Overrides the default style or theme-based style for the component. For example, the following style sets the background color of the spacer to black with no border. `style="background-color: black; border-style:none;"` If no style is specified, the theme-based style is used.
width	No	Number of pixels for the width of the spacer when a spacer is added to the column axis (axis="column"). The default is 10 pixels.

gridSpacer Tag Example

The following example adds six spacers to the grid—top border, bottom border, left border, right border, column separator, and Location separator.

- The top and bottom borders are added with the `axis` attribute set to `row` and the `position` attribute set to `top` and `bottom`.
- The left and right borders are added with the `axis` attribute set to `column` and the `position` attribute set to `left` and `right`.
- The column separators are added with the `axis` attribute set to `column` and the `position` set to `interlace`, so there is a spacer between every two columns.
- The Location separator is added to create a grouping effect by adding a spacer when the member in the All Locations dimension changes. This is done by setting the `axis` to `row`, the `position` to `memberchange`, and the `scope` to `All Locations`.

```
<%@ taglib uri="bloxtld" prefix="blox" %>
<%@ taglib uri="bloxuitld" prefix="bloxui" %>

<blox:data id="gridSpacerData"
   dataSourceName="qcc-essbase" useAliases="true" visible="false"
   query="<ROW ('All Locations', 'Measures')  'Central'  'East'  'West'
'All Locations' 'Gross Margin' <CHILD 'Ratios' <ASYM <COLUMN ('Scenario',
'All Time Periods')     'Actual'  'Actual'  'Forecast'  'Forecast'
'2000.Q3'  '2000.Q4'  '2001.Q1'  '2001.Q2'!" />

<html>
<head>
   <blox:header />
</head>
<body>
<blox:grid id="gridSpacer" width="80%" height="500" visible="true">
   <blox:data bloxRef="gridSpacerData" />
   <bloxui:toolbar name="standardToolbar" visible="false" />

   <bloxui:gridSpacer
      axis="column"
      position="right"
      width="5"
      style="background-color: red;"
      description="Right Border"
      showOnLayoutMenu="true" />

   <bloxui:gridSpacer
      axis="column"
      position="left"
      width="5"
      style="background-color: red;"
      description="Left Border"
      showOnLayoutMenu="true" />

   <bloxui:gridSpacer
      axis="row"
      position="top"
      height="5"
      style="background-color: red;"
      description="Top Border"
      showOnLayoutMenu="true" />

   <bloxui:gridSpacer
      axis="row"
      position="bottom"
      height="5"
      style="background-color: red;"
      description="Bottom Border"
      showOnLayoutMenu="true" />

   <bloxui:gridSpacer
      axis="column"
      position="interlace"
```

```
      width="2"
      style="background-color: yellow;"
      description="Column Separators"
      showOnLayoutMenu="true" />

   <bloxui:gridSpacer
      axis="row"
      position="memberchange" scope="All Locations"
      description="Location Separators"
      height="5"
      showOnLayoutMenu="true" />
</blox:grid>
</body>
</html>
```

The <bloxui:title> Tag

The <bloxui:title> tag allows you to add a title to the top of a presentation Blox (PresentBlox, GridBlox, and ChartBlox). The benefit of using this tag to add a title as opposed to using general HTML tags is better integration into the Blox user interface. Because the title becomes part of the presentation Blox, it automatically inherits the style applied to the Blox and wraps as the Blox is re-sized in the browser.

The <bloxui:title> tag should be added inside a presentation Blox. It contains the following tag attributes:

```
<bloxui:title
   title=""
   style=""
   alignment="" />
```

where:

Attribute	Description
title	The title to display.
style	The style to apply to the title. For example, `style="font-family: Arial; font-weight: bold; font-size: 14pt; color: black; background-color: #FFFFCC;"`
alignment	Alignment for the title. Valid values are left, center, and right.

The style defined for the title only applies to the title text rather than the entire rendered table cell. If you want to specify a background color for the title, you should also make sure that the background color of the presentation Blox use the same color.

To set the background color for presentation Blox, use the <bloxui:component> tag. This is demonstrated in the following example.

title Tag Example

The following example shows how to set the title of a GridBlox:

- The <bloxui:title> tag is nested inside a GridBlox.
- The background color, text color, font size and font style is set using the style attribute.

- The GridBlox's background color is set to the same background color as that of the title using the <bloxui:component> tag, with the component name set to the name of the GridBlox.

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxuitld" prefix="bloxui"%>

<blox:data id="myDataTest"
   dataSourceName="qcc-essbase"
   useAliases="true" visible="false"
   query="<ROW ('All Locations', 'Measures') 'Central' 'East'
      'West' 'All Locations' 'Gross Margin' <CHILD 'Ratios'
       <ASYM <COLUMN ('Scenario', 'All Time Periods') 'Actual'
      'Actual'  'Forecast' 'Forecast' '2000.Q3' '2000.Q4' '2001.Q1'
      '2001.Q2'!" />

<html>
<head>
   <blox:header />
</head>

<blox:grid id="myGridTest"
   width="80%"
   height="80%"
   visible="true"
   menubarVisible="false"
   bandingEnabled="true"
   gridLinesVisible="false">
   <blox:data bloxRef="myDataTest" />
   <bloxui:component name="navigationToolbar" visible="false"/>
   <bloxui:component name="standardToolbar" visible="false"/>

   <bloxui:component name="myGridTest"
      style="background-color: #FFFFCC; border-style:none;" />

   <bloxui:title title="Sales and Gross Margin By Location - FY'02"
      style="font-family: Arial; font-weight: bold;
         font-size: 14pt; color: black; background-color: #FFFFCC;"
      alignment="center" />

</blox:grid>
</body>
</html>
```

Custom Menu Tags

The UI tags for customizing the menubar allow you to add, remove, or disable menus and menu items to the default menubar in a PresentBlox, GridBlox, or ChartBlox. Its tags need to be nested inside the tags for these presentation Blox. By default, the menubar is on in PresentBlox or standalone GridBlox /ChartBlox. (`menubarVisible="true"`).

This section describes the general concepts related to the `Menubar`, `Menu`, and `MenuItem` components and provides tag reference for these components.

Menubar, Menu, and MenuItem

Each of the menus in the menubar is a Menu component that you can remove, disable, or reposition. Each menu has menu items. Each menu item can also be removed, disabled, or repositioned. In addition, you can add your custom menus and menu items to the menubar, or customize the operations associated with the menu items.

To specify actions associated with the menu items, you can use the <bloxui:clientLink> tag to load a URL or call a JavaScript function. You can also invoke server-side code through the <bloxui:actionFilter> tag. See "The <bloxui:actionFilter> Tag" on page 829 for more information.

Menu Tags Listing

```
<!--Nested within a PresentBlox, GridBlox, or ChartBlox -->

<bloxui:menu
   name=""
   bloxRef=""
   disabled=""
   positionBefore=""
   resourceName=""
   title=""
   tooltip=""
   visible=""
   >
   <bloxui:menuItem
      name=""
      checkable=""
      checked=""
      disabled=""
      imageURL=""
      positionBefore=""
      separator=""
      themeBasedImage=""
      title=""
      tooltip=""
      visible=""
      >
      <bloxui:clientLink
         features=""
         link=""
         target="" />
   </bloxui:menuItem>
</bloxui:menu>
```

<bloxui:menu> Tag Attributes

Attribute	Required	Description
name	Yes	The name of the menu. If a Menu with the specified name is found, then the tag acts on the component. Otherwise, a new Menu is created. To customize the default menus in the menubar, specify one of the following valid values: • bookmarkMenu • chartMenu • dataMenu • editMenu • fileMenu • helpMenu • toolsMenu • viewMenu Or you can specify the value using the constants. The following examples show two ways to specify the Tools menu in the menubar. `name="<%= ModelConstants.TOOLS_MENU %>" name="toolsMenu"`
bloxRef	No	References an existing Blox to apply the tag to when the tag is used outside of the Blox's tag. See "Example on bloxRef Attribute" on page 795.
disabled	No	Set this to true to disable a menu. The menu is displayed in the menubar but is greyed out.
positionBefore	No	The position before which the menu should be displayed. If this is not specified, newly added menu is appended to the end of the menubar. To position the menu before the DB2 logo, set the value of value of this attribute to logo: `positionBefore="logo"`
resourceName	No	Loads the named resource file into the component. This allows you to have a menu tag that creates a new menu from a menu XML file. See Chapter 27, "XML Resource Files Reference," on page 839 for more information.
title	Yes for custom menus	The displayed title for the menu. Custom menus added to the menubar must have a title. The title cannot contain slashes ("/").
tooltip	No	Tooltip displayed with mouse-over.
visible	No	The visibility of the menu. When set to false, the menu is not displayed in the menubar. The default is true.

Nested <bloxui:menuItem> Tag Attributes

Attribute	Required	Description

name	Yes	The name of the menu item. Specify your custom menu item names to add a custom menu item. If a MenuItem with the specified name is found, then the tag acts on the component. Otherwise, a new MenuItem is created. To customize a built-in menu item, see "Built-in Menu and Menu Item Names" on page 819 for valid values.
		
		Or you can specify the value using the constants. The following examples show two ways to specify the Expand All menu item under the Data menu:
		`name="<%= ModelConstants.DATA_EXPAND_ALL %>"`
		`name="dataExpandAll"`

checkable	No	`true` to make the menu item checkable. When the menu item is selected, a check mark appears before the menu item. **Note:** If you want to set `checkable` or `checked` attributes on built-in menu items, you will need to add your custom event handler and controller, or the settings will have no effect.
checked	No	`true` to have a check mark appear before the menu item.
disabled	No	Set this to `true` to disable the menu item. The menu item is displayed in the menu, but is greyed out.
imageURL	No	The URL of the image to use. If `themeBasedImage` is set to `true`, your custom images need to reside in the directory where the theme's images are stored in the DB2 Alphablox repository. Typically, this directory is located at: `<alphablox_dir>/repository/theme/<themeName>/i/` If `themeBasedImage` is set to `false`, specify the URL of the image. The URL can be: • An absolute URL. The string should begin with "`http://`". • A relative URL: – Starting the string with a slash (/) indicates that the URL is relative to the server root. Note that 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.
positionBefore	No	The position where the named menu item should be placed. If this is not specified, newly added menu item is appended to the end of the menu.
separator	No	Set to `true` to add a separator line.

Attribute	Required	Description
themeBasedImage	No	Set to true to use theme-based images. Images need to reside in the directory where theme's images are stored in the repository. Typically, this directory is located at: <alphablox_dir>/repository/theme/<themeName>/i/ Set to false to use images that do not reside in the theme's image directory. Use the imageURL attribute to specify the URL of the image file.
title	Yes for custom menu items	The displayed title of this menu item. Custom menu items added must have a title. The title cannot contain slashes ("/").
tooltip	No	Tooltip displayed with mouse-over.
visible	No	The visibility of the menu item. When set to false, the menu item is not displayed in the menu. The default is true.

Nested <bloxui:clientLink> Tag Attribute

This is a nested tag for multiple Blox UI tags. See "The <bloxui:clientLink> Tag" on page 833 for details.

Built-in Menu and Menu Item Names

All the common component names used by the Blox UI model are constants. You can find all the constants in the ModelConstants interface under the com.alphablox.blox.uimodel package in the Javadoc. Names for the constants are all in uppercase. Their values, when specified in your Blox UI tag attributes, should all be in lowercase with no underscores ("_"), with the first letter of second and each subsequent word in uppercase. The following table for the built-in menu and menu item names is provided for your convenience. For a complete list of model constants, see "Model Constants and Their Values" on page 835.

Menu	Menu Item	Menu Constants
fileMenu		FILE_MENU
	fileOpen	FILE_OPEN
	fileSaveAs	FILE_SAVE_AS
	fileExportToExcel	FILE_EXPORT_TO_EXCEL
	fileExportToPDF	FILE_EXPORT_TO_PDF
editMenu		EDIT_MENU
	editUndo	EDIT_UNDO
	editRedo	EDIT_REDO
	editFind	EDIT_FIND
	editHistory	EDIT_HISTORY
	editCopy	EDIT_COPY
	editDelete	EDIT_DELETE
	editSelectAll	EDIT_SELECTALL
viewMenu		VIEW_MENU
	viewChart	VIEW_CHART

Menu	Menu Item	Menu Constants
	viewGrid	VIEW_GRID
	viewPageFilter	VIEW_PAGE_FILTER
	viewDataLayout	VIEW_DATA_LAYOUT
	viewToolbarMenu	VIEW_TOOLBAR_MENU
	viewToolbarCustomize	VIEW_TOOLBAR_CCUSTOMIZE
	viewPoppedOut	VIEW_POPPED_OUT
dataMenu		DATA_MENU
	dataSortAscending	DATA_SORT_ASCENDING
	dataSortDescending	DATA_SORT_DESCENDING
	dataDrillUp	DATA_DRILL_UP
	dataDrillDown	DATA_DRILL_DOWN
	dataExpandAll	DATA_EXPAND_ALL
	dataPivot	DATA_PIVOT
	dataShowOnly	DATA_SHOW_ONLY
	dataRemoveOnly	DATA_REMOVE_ONLY
	dataKeepOnly	DATA_KEEP_ONLY
	dataHide	DATA_HIDE
	dataUnhideAll	DATA_UNHIDE_ALL
	dataSwapAxes	DATA_SWAP_AXES
	dataOptions	DATA_OPTIONS
	dataNavigateButton	DATA_NAVIGATION_BUTTON
	dataAdvancedMenu	DATA_ADVANCED_MENU
	dataAdvancedDrillThrough	DATA_ADVANCED_DRILL_THROUGH
	dataAdvancedFormatMask	DATA_ADVANCED_FORMAT_MASK
	dataAdvancedMergedHeaders	DATA_ADVANCED_MERGED_HEADERS
	dataAdvancedSetHiddenColumns	DATA_ADVANCED_SET_HIDDEN_COLUMNS
	dataAdvancedSetHiddenMembers	DATA_ADVANCED_SET_HIDDEN_MEMBERS
	dataAdvancedSetHiddenMenu	DATA_ADVANCED_SET_HIDDEN_MENU
	dataAdvancedSetHiddenRows	DATA_ADVANCED_SET_HIDDEN_ROWS
	dataAdvancedShowBottomLevel	DATA_ADVANCED_SHOW_BOTTOM_LEVEL
	dataAdvancedShowSiblings	DATA_ADVANCED_SHOW_SIBLILING
	dataAdvancedTrafficLights	DATA_ADVANCED_TRAFFIC_LIGHTS
	dataCalculationEditor	DATA_CALCULATION_EDITOR
	dataCommentsMenu	DATA_COMMENTS_MENU
	dataCommentsAddComment	DATA_COMMENTS_ADD_COMMENT
	dataCommentsDisplayComments	DATA_COMMENTS_DISPLAY_COMMENTS
	dataMemberFilter	DATA_MEMBER_FILTER
chartMenu		CHART_MENU
	chartTypesMenu	CHART_TYPES_MENU
	chartTypesLine	CHART_TYPES_LINE
	chartTypesBar	CHART_TYPES_BAR

Menu	Menu Item	Menu Constants
	chartTypes3DBar	CHART_TYPES_3DBAR
	chartTypes3DPie	CHART_TYPES_3DPIE
	chartTypesMore	CHART_TYPES_MORE
	chartAxisPlacement	CHART_AXIS_PLACEMENT
	chartComboTypes	CHART_COMBO_TYPES
	chartDataValues	CHART_DATA_VALUES
	chartAllData	CHART_ALL_DATA
	chartSelectedData	CHART_SELECTED_DATA
	chartOptions	CHART_OPTIONS
toolsMenu		TOOLS_MENU
	toolsGridOptions	TOOLS_GRID_OPTIONS
	toolsPresentOptions	TOOLS_PRESENT_OPTIONS
	toolsManageMenu	TOOLS_MANAGE_MENU
	toolsManageTrafficLights	TOOLS_MANAGE_TRAFFIC_LIGHTS
helpMenu	helpHelp	HELP_HELP
	helpAbout	HELP_ABOUT

Menu Tag Examples

Example 1: Removing a menu item

This example demonstrates how to remove a menu item from the menubar by setting the visibility of the menu item to `false`. In this example, the Edit menu item and the Grid Options... submenu under Tools are removed.

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxuitld" prefix="bloxui"%>
...
<html>
<head>
   <blox:header />
</head>
<body>
...
<blox:present menubarVisible="true" ...>
...
   <bloxui:menu name="editMenu" visible="false" />
   <bloxui:menu name="toolsMenu" >
      <bloxui:menuItem name="toolsGridOptions" visible="false" />
   </bloxui:menu>
...
</blox:present>
...
</body>
</html>
```

Example 2: Disabling a menu item

This example demonstrates how to disable a menu item from the menubar by setting the `disabled` attribute of the menu item to `true`. In this example, the Grid Options... submenu under Tools is disabled.

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxuitld" prefix="bloxui"%>
...
<html>
<head>
   <blox:header />
</head>
<body>
...
<blox:present menubarVisible="true"...>
...
   <bloxui:menu name="toolsMenu" >
      <bloxui:menuItem name="toolsGridOptions" disabled="true" />
   </bloxui:menu>
...
</blox:present>
</body>
</html>
```

Example 3: Creating a menu item

This example creates a menu item called “Quick Links” with three options. The second option has submenus.

- When the first option “Today’s Stock Quotes” is selected, a page located on another server is loaded in a separate browser window.
- A separator is added between option 1 and option 2.
- When either of the two submenus in the second option “Reports...” are selected, a page on the same server is loaded in a separate browser window.
- When the third option “Calendar” is selected, a JavaScript function is called.

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxuitld" prefix="bloxui"%>

<html>
<head>
   <blox:header />
</head>
<blox:present menubarVisible="true"...>
...
   <bloxui:menu name="myMenu" title="Quick Links" visible="true">
      <bloxui:menuItem name="option1" title="Today's Stock Quotes">
         <bloxui:clientLink link="http://myserver/quotes.jsp"
            target="mywindow" />
      </bloxui:menuItem>

      <bloxui:menuItem separator="true" />

      <bloxui:menu name="option2" title="Reports...">
         <bloxui:menuItem name="submenu1" title="YTD Sales- East">
            <bloxui:clientLink link="east.jsp"
               target="mywindow" />
         </bloxui:menuItem>

         <bloxui:menuItem name="submenu2" title="Google">
            <bloxui:clientLink link="central.jsp"
               target="myotherwindow" />
         </bloxui:menuItem>
      </bloxui:menu>

      <bloxui:menuItem name="option3" title="Calendar">
         <bloxui:clientLink link="javascript:getCalendar();" />
      </bloxui:menuItem>
```

```
      </bloxui:menu>

   </blox:present>

   </body>
   </hmtl>
```

Custom Toolbar Tags

The UI tags for customizing the toolbar allow you to add, remove, or disable menus and menu items to the default toolbar in a PresentBlox, GridBlox, or ChartBlox. Its tags need to be nested inside the tags for these user interface Blox. Whenever possible, however, you should use ToolbarBlox's tag attributes to set its property values. For example, you can use the `removeButton` tag attribute to remove buttons. Use the `<bloxui:toolbar>` and `<bloxui:toolbarButton>` tags only if you are using the DHTML client and if you need higher level of toolbar customization than is provided by Blox properties. As are all the Blox UI tags, these tags use styles that override the theme-based Cascading Stylesheet class settings used in the DHTML client.

This section describes the general concepts related to the `Toolbar` and `ToolbarButton` components and provides tag reference for these components.

Toobar and ToolbarButton

There are two default toolbars in a PresentBlox: Standard and Navigation. Each of the toolbars is a Toolbar component that you can remove, disable, or reposition. Each toolbar contains toolbar buttons. Each toolbar button can also be removed, disabled, or repositioned. In addition, you can add your custom toolbar and toolbar buttons, or customize the operations associated with the toolbar buttons.

When you add a custom toolbar, the menubar's View -> Toolbar menu option will automatically include your custom toolbar in the list.

Toolbar Tags Listing

```
<bloxui:toolbar
   disabled=""
   bloxRef=""
   name=""
   positionBefore=""
   resourceName=""
   title=""
   tooltip=""
   visible="">
   <bloxui:toolbarButton
      checkable=""
      checked=""
      disable=""
      imageURL=""
      name=""
      positionBefore=""
      separator=""
```

```
      themeBasedImage=""
      title=""
      tooltip=""
      visible="" >
      <bloxui:clientLink
         features=""
         link=""
         target="" />
   </bloxui:toolbarButton>
</bloxui:toolbar>
```

The <bloxui:toolbar> Tag Attributes

Attribute	Required	Description
name	Yes	The name of the toolbar. Specify your custom toolbar name to add a custom toolbar. If a Toolbar with the specified name is found, then the tag acts on the component. Otherwise, a new Toolbar is created. To customize the two out-of-the-box toolbars when a presentation Blox is created with its toolbar turned on, specify one of the following • Using the constants: – NAVIGATION_TOOLBAR – STANDARD_TOOLBAR • Using valid values: – navigationToolbar – standardToolbar The following examples show two ways to specify the Standard toolbar: `name="<%= ModelConstants.STANDARD_TOOLBAR %>" name="standardToolbar"`
bloxRef	No	References an existing Blox to apply the tag to when the tag is used outside of the Blox's tag. See "Example on bloxRef Attribute" on page 795.
disabled	No	Set this to true to disable a toolbar. The toolbar is displayed but clicking the buttons have no effect.
positionBefore	No	The position before which the toolbar should be displayed. If this is not specified, newly added toolbar is appended to the end (after the Navigation Toolbar). For example, to position your custom toolbar before the Navigation Toolbar: `positionBefore="navigationToolbar"`
resourceName	No	Loads the named resource file into the component. This allows you to have a toolbar tag that creates a new toolbar from a toolbar XML file. See Chapter 27, "XML Resource Files Reference," on page 839 for more information.
title	Yes for custom toolbar	The displayed title for the toolbar. Custom toolbars added must have a title. This title is displayed under the button icon image when the ToolbarBlox's textVisible property is set to true (default is false). It is also automatically added to the menubar's View -> Toolbar menu option in the same nesting presentation Blox (PresentBlox, GridBlox, or ChartBlox) when the Blox's menubarVisible property is set to true. The title cannot contain slashes ("/").

Attribute	Required	Description
tooltip	No	Tooltip displayed with mouse over.
visible	No	The visibility of the toolbar. When set to false, the entire toolbar is not displayed. The default is true.

The <bloxui:toolbarButton> Tag

Attribute	Required	Description

name	Yes	The name of the toolbar button. Specify your custom toolbar button names to add your custom toolbar button. If a ToolbarButton with the specified name is found, then the tag acts on the component. Otherwise, a new ToolbarButton is created. To customize a built-in toolbar button, see "Built-in Toolbar and ToolbarButton Names" on page 827 for valid values.
		
		The following examples show two ways to specify the Copy button in the Standard toolbar:
		`name="<%= ModelConstants.EDIT_COPY %>"`
		`name="editCopy"`

checkable	No	true to make the toolbar button sticky. That is, the button stays depressed until you click another button. **Note:** If you want to set checkable or checked attributes on built-in toolbar buttons, you will need to add your custom event handler and controller, or the settings will have no effect.
checked	No	Set this to true to have the toolbar button appear to be depressed.
disabled	No	Set this to true to disable the button. The button icon is displayed in the toolbar, but does not respond to mouseOver or onClick events. For custom buttons you want to disable, an image of the same name with a suffix of "_disabled" should be supplied. See the imageURL attribute for more information.

Attribute	Required	Description
imageURL	No	The URL of the image to use. If themeBasedImage is set to true, your custom images need to reside in the directory where the theme's images are stored in the DB2 Alphablox repository. Typically, this directory is located at: <alphablox_dir>/repository/theme/ <themeName>/i/ If themeBasedImage is set to false, specify the URL of the image. The URL can be: • An absolute URL. The string should begin with "http://". • A relative URL: – Starting the string with a slash (/) indicates that the URL is relative to the server root. Note that 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. **Tip:** For each icon, if the ToolbarBlox's rolloverEnabled property is set to true (the default), you should supply two images, one for non-active mode and the other for active mode. When the toolbar button is selected (active mode), DB2 Alphablox automatically looks for an image of the same name but with a "_active" suffix. If this image file does not exist, the browser will display a missing icon. For any disabled button (button that is displayed but does not respond to mouseOver or onClick events), you should also supply an image with the same name with a "_disabled" suffix.
positionBefore	No	The position where the named toolbar button should be placed. If this is not specified, newly added toolbar button is appended to the end of the toolbar.
separator	No	Set to true to add a separator bar.
themeBasedImage	No	Set to true to use theme-based images. Images need to reside in the directory where theme's images are stored in the repository. Typically, this directory is located at: <alphablox_dir>/repository/theme/ <themeName>/i/ Set to false to use images that do not reside in the theme's image directory. Use the imageURL attribute to specify the URL of the image file.
title	Yes for custom menu items	The displayed title of this toolbar button. Custom toolbar button added must have a title. The title cannot contain slashes ("/").
tooltip	No	Tooltip displayed with mouse over.
visible	No	The visibility of the toolbar button. When set to false, the toolbar button is not displayed in the menu. The default is true.

Built-in Toolbar and ToolbarButton Names

All the common component names used by the Blox UI model are constants. You can find all the constants in the `ModelConstants` interface under the com.alphablox.blox.uimodel package in the Javadoc. Names for the constants are all in uppercase. Their values, when specified in your Blox UI tag attributes, should all be in lowercase with the first letter of second and each subsequent word in uppercase. The following table for the built-in toolbars and toolbar button names is provided for your convenience. These names should only be used with the `<bloxui:toolbar>` and `<bloxui:toolbarButton>` tags and do not apply to ToolbarBlox's `removeButton` property. For a complete list of model constants, see "Model Constants and Their Values" on page 835.

Toolbar

Buttons in standardToolbar	Buttons in navigationToolbar
fileOpen	dataNavigateButton
fileSaveAs	dataSortAscending
editCopy	dataSortDescending
standardToolbarSeparator1(the separator after the editCopy button)	dataMemberFilter
viewPoppedOut	navigationToolbarViewSeparator
editRedoButton	viewGrid
editUndoButton	viewChart
fileExportToPDF	viewPageFilter
fileExportToExcel	viewDataLayout
standardToolbarSeparator3	
helpHelp	

The Undo, Redo, and data navigation buttons (which contains various navigation options such as Drill Down, Drill Up, Pivot, and Show Only) are `DropDownToolbarButton` components, not `ToolbarButton` components. However, they can still be removed using the `<bloxui:toolbarButton>` tag:

```
 <bloxui:toolbar name="navigationToolbar" visible="true">
   <bloxui:toolbarButton
      name="<%=ModelConstants.DATA_NAVIGATE_BUTTON%>" visible="false"/>
</bloxui:toolbar>
```

Alternatively, you can also remove these DropDownToolbarButtons using the generic `<bloxui:component>` tag:

```
<bloxui:toolbar name="navigationToolbar" visible="true">
   <bloxui:component name="<%=ModelConstants.DATA_NAVIGATE_BUTTON%>"
      visible="false"/>
</bloxui:toolbar>
```

Note: The setting of `maximumUndoSteps`, a common Blox property controls the availability of the Undo and Redo buttons. If `maximumUndoSteps` is set to 0, then the Undo and Redo buttons will be removed. If `maximumUndoSteps` is not 0, these buttons will show. See "maximumUndoSteps" on page 41

Toolbar Tags Examples

Example 1: Removing a toolbar buttons

This example demonstrates how to remove a toolbar button from the Navigation toolbar by setting the visibility of the toolbar button to `false`.

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxuitld" prefix="bloxui"%>

<html>
<head>
   <blox:header />
</head>

<blox:present ....>
...
<bloxui:toolbar name="navigationToolbar" >
   <bloxui:toolbarButton name="viewGrid" visible="false" />
   <bloxui:toolbarButton name="viewChart" visible="false" />
   <bloxui:toolbarButton name="viewPageFilter" visible="false" />
   <bloxui:toolbarButton name="viewDataLayout" visible="false" />
</bloxui:toolbar>
...
</blox:present>
</body>
</html>
```

Example 2: Adding a custom toolbar

This example creates a Toolbar called "myToolbar" (`name="myToolbar"`) with a display name of "My Toolbar" (`title="My Toolbar"`).

This example demonstrates:

- the use of the `positionBefore` attribute to specify the toolbar position.
- the use of absolute and relative URLs to specify the image URLs.
- the use of the `<bloxui:clientLink>` nested tag to specify a URL when the toolbar button is clicked (see "The <bloxui:clientLink> Tag" on page 833 for more information).

The menubar will automatically reflect this new toolbar in its View -> Toolbar... menu option.

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxuitld" prefix="bloxui"%>

<html>
<head>
   <blox:header />
</head>

<body>
<blox:present id="myPresentBlox" width="700" height="500" >
   <blox:data dataSourceName="TBC" useAliases="true"
      query="<SYM <ROW(Product) <ICHILD Product <COLUMN(Year,
         Scenario) Qtr1 Qtr2 <CHILD Scenario Sales !" />
   <bloxui:toolbar name="myToolbar" title="My Toolbar"
      visible="true" positionBefore="navigationToolbar">

      <bloxui:toolbarButton name="option1" title="mail"
         themeBasedImage="false"
         imageURL="http://myserver/myApp/email.gif"
         tooltip="Check email alerts">
         <bloxui:clientLink link="emailAlerts.jsp"
            target="mywindow"
            features="toolbar=no,status=no" />
      </bloxui:toolbarButton>
```

```
      <bloxui:toolbarButton name="option2" title="Stocks"
         themeBasedImage="false" imageURL="../money.gif"
         tooltip="Today's Stocks">
         <bloxui:clientLink link="http://www.my.com/app/file.jsp"
            target="mywindow" />
      </bloxui:toolbarButton>

      <bloxui:toolbarButton name="option3" title="KPI"
         themeBasedImage="false" imageURL="/myApp/lookup.gif"
         tooltip="Show KPI" >
         <bloxui:clientLink link="javascript:myLookupFunction()"
            target="mywindow" />
      </bloxui:toolbarButton>

      <bloxui:toolbarButton separator="true"
         positionBefore="option1" />
   </bloxui:toolbar>

</blox:present>
</body>
</html>
```

Utility Tags

The Blox UI Tag Library contains a set of utility tags. These tags allow you to specify actions to take when a ClickEvent is triggered in a UI component, set properties on classes referenced by the `<bloxui:customLayout>` and `<bloxui:customAnalysis>` tags, or to invoke server-side code to customize GridBlox layout. They include the following tags:

The <bloxui:actionFilter> Tag

The `<bloxui:actionFilter>` tag allows you to invoke server-side code from a Blox UI component when it is clicked using the Blox UI Tag Library.

```
<bloxui:actionFilter
   componentName=""
   filter="" />
```

where:

Attribute	Description
`componentName`	The name of the component this action filter should be attached to. When this named component is clicked, a ClickEvent is triggered and the action specified in the `actionFilter` method of the named class is performed.
`filter`	Specifies the filter object. For example: `<bloxui:actionFilter filter="<%= new MyActionFilterClass() %>" componentName="dataPivot" />` where `MyActionFilterClass` implements `IActionFilter` and is defined within the JSP page.

The IActionFilter interface in the com.alphablox.blox.uimodel.tags package must be implemented by all action filters added to Blox using the <bloxui:actionFilter> tag. This interface has one method with the following signature:

```
void actionFilter(DataViewBlox blox, Component component);
    //throws java.lang.Exception
```

where:

Argument	Description
blox	The action filter's Blox
component	The component that generates the ClickEvent

This method is called each time the component this action filter is attached to generate a ClickEvent. You can implement this method and add actions to take when the associated component is clicked.

In order to implement this method, make sure at least the following packages are imported:

```
<%@ page import="com.alphablox.blox.uimodel.*,
                 com.alphablox.blox.uimodel.tags.IActionFilter,
                 com.alphablox.blox.DataViewBlox,
                  com.alphablox.blox.uimodel.core.Component" %>
```

You may also need to import other packages. Use a Java Integrated Development Environment (IDE) to help you identify which packages to import.

Utility Tags Example

The following example demonstrates the use of the <bloxui:actionFilter> tag and how to implement the IActionFilter interface and to extend its actionFilter method to take some action when a component is clicked. In this case, when the custom menu item is clicked, a MessageBox is displayed with some message.

```
<%@ page import="com.alphablox.blox.uimodel.*,
                 com.alphablox.blox.uimodel.tags.IActionFilter,
                 com.alphablox.blox.DataViewBlox,
                 com.alphablox.blox.uimodel.core.Component,
                 com.alphablox.blox.uimodel.core.MessageBox"%>

<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxuitld" prefix="bloxui"%>

<html>
<head>
   <blox:header />
</head>

<blox:present ....>

   <bloxui:menu name="toolsMenu" >
      <bloxui:menuItem name="myToolMenuItem" title="Get Message" />
   </bloxui:menu>

   <bloxui:actionFilter
      filter="<%= new MyActionFilterClass() %>"
      componentName="myToolMenuItem" />
...
</blox:present>
...
<%!
```

```
public static class MyActionFilterClass implements IActionFilter
{
   public void actionFilter( DataViewBlox blox, Component component )
throws Exception {
      MessageBox.message( component, "Get Message", "The myToolMenuItem has
been clicked!" );
   }
}
%>
```

The <bloxui:gridFilter> Tag

The <bloxui:gridFilter> tag allows you to invoke server-side code and customize GridBlox layout using the Blox UI Tag Library. It has the following tag attributes:

```
<bloxui:gridFilter
   filter="" />
```

where:

Attribute	Description

filter	Specifies the filter object. For example:
	`<bloxui:gridFilter`
	`filter="<%= new MyGridFilterClass() %>"`
	`componentName="dataPivot" />`
	
	where MyGridFilterClass implements IGridFilter and is defined within the JSP page.

You can use the grid filters to customize and rebuild a grid before it is loaded. The IGridFilter interface in the com.alphablox.blox.uimodel.tags package must be implemented by all grid filters added to Blox using the <bloxui:gridFilter> tag. The grid is rebuilt after each data navigation command is processed. This interface has two methods with the following signature:

```
void gridFilter(DataViewBlox blox, GridBrixModel grid);
                  // throws java.lang.Exception
void cellFilter(DataViewBlox blox, GridCell cell);
                  // throws java.lang.Exception
```

where:

Argument	Description
blox	The grid filter's Blox
grid	The grid which has been rebuilt
cell	The grid cell which has been rebuilt

Since this filter may be one of many filters that are applied to the grid after a rebuild, the filter should not make assumptions regarding the layout of the grid.

In order to implement this method, make sure at least the following packages are imported:

```
<%@ page import="com.alphablox.blox.uimodel.*,
                 com.alphablox.blox.uimodel.tags.IActionFilter,
                 com.alphablox.blox.uimodel.tags.IGridFilter,
                 com.alphablox.blox.DataViewBlox,
                 com.alphablox.blox.uimodel.GridBrixModel,
                 com.alphablox.blox.uimodel.GridBrixCellModel,
```

```
                com.alphablox.blox.uimodel.core.grid.GridRow,
                com.alphablox.blox.uimodel.core.grid.GridCell,
                com.alphablox.blox.uimodel.core.Component" %>
```

You may also need to import other packages. Use a Java Integrated Development Environment (IDE) to help you identify which packages to import.

gridFilter Tag Example

The following example demonstrates the use of the `<bloxui:gridFilter>` tag and how to implement the `IGridFilter` interface and to extend its `gridFilter` method to rebuild a grid. In this case,

- We move
  - The row headers are moved to the end of the grid.
  - The column headers are moved to the bottom of the grid.
- A Button component is added to the end of each column.
- Once the grid is rebuilt, pop up a MessageBox notifying that the grid has changed.
- The grid is then displayed.

See the Blox Sampler for a similar example.

```
<%@ page import="com.alphablox.blox.uimodel.tags.IGridFilter,
                com.alphablox.blox.DataViewBlox,
                com.alphablox.blox.uimodel.GridBrixModel,
                com.alphablox.blox.uimodel.core.grid.GridRow,
                com.alphablox.blox.uimodel.core.grid.GridCell,
                com.alphablox.blox.uimodel.core.Button,
                com.alphablox.blox.uimodel.ModelConstants,
                com.alphablox.blox.uimodel.core.grid.GridColumn,
                com.alphablox.blox.uimodel.core.MessageBox,
                com.alphablox.blox.uimodel.GridBrixCellModel"%>
<%@ taglib uri="bloxtld" prefix="blox" %>
<%@ taglib uri="bloxuitld" prefix="bloxui" %>

<blox:data id="dataBlox" dataSourceName="qcc-essbase"
   useAliases="true" visible="false"
   query="<ROW ('All Locations') 'Central' 'East' 'West' 'All Locations'
<ASYM <COLUMN ('Scenario', 'All Time Periods') 'Actual' 'Actual' 'Forecast'
'Forecast' '2000.Q3' '2000.Q4'  '2001.Q1'  '2001.Q2'!" />

<html>
<head>
   <blox:header />
</head>

<body>
<blox:grid id="testGridMoveFilter" width="80%" height="500"
   bandingEnabled="true" menubarVisible="true">
   <blox:toolbar visible="true" />
   <bloxui:gridFilter
      filter="<%= new MyGridFilterClass() %>" />
   <blox:data bloxRef="dataBlox" />
</blox:grid>

</body>
</html>

<%!
public static class MyGridFilterClass implements IGridFilter
{
   public void gridFilter( DataViewBlox blox, GridBrixModel grid ) throws
Exception {
```

```
      // Move row headers to the end of the grid
      if ( grid.getColumnCount() > 1 )
         while ( grid.getColumn( 0 ).isHeader() )
            grid.moveColumn( 0, grid.getColumnCount() );

      // Move column headers to the end of the grid
      if ( grid.getRowCount() > 1 )
         while ( grid.getRow( 0 ).isHeader() )
            grid.moveRow( 0, grid.getRowCount() );

      GridRow row = new GridRow( );

      // Add a button to the end of each column. For this example,
      // these buttons do not do anything.
      for ( int i=0; i < grid.getColumnCount(); i++ ) {
         GridCell cell = new GridCell( "myCell" + (i+1)  );
         cell.add( new Button( cell.getName(), cell.getName() ) );
         cell.setClickable( false );
         row.add( cell );
      }

      grid.addRow( row );

      row = new GridRow();
      row.setHeight( 4 );
      row.setThemeClass( ModelConstants.THEME_STYLE_ROW_DATA_GENERATION +
"3" );
      grid.insertRow( 4, row );

      GridColumn column = new GridColumn();
      column.setWidth( 4 );
      column.setThemeClass( ModelConstants.THEME_STYLE_ROW_DATA_GENERATION
+ "3" );
      grid.insertColumn( 4, column );
      MessageBox.message( grid, "Change", "The grid has changed" );
   }
   public void cellFilter( DataViewBlox blox, GridCell cell ) throws
Exception {
   }
}
%>
```

Note that this example only serves to demonstrate how to customize and rebuild the grid layout with the <bloxui:gridFilter> tag. This is an advanced technique and may impact how the grid scrolls. In addition, the buttons added to the grid do not have an associated action defined when they are clicked.

The <bloxui:clientLink> Tag

The <bloxui:clientLink> tag allows you to specify an URL to load into the existing or a different browser window once a component is clicked. It should be added within a component tag such as <bloxui:menuItem> and <bloxui:toolbarButton>. It has the following tag attributes.

```
<bloxui:clientLink
   features=""
   link=""
   target="" />
```

Attribute	Description
features	A comma-separated browser feature string. When the target attribute is specified, this attribute sets the features for the new browser window. For example, features="toolbar=no,status=no". The

	browser feature string should be specified the same way as the JavaScript's `window.open()` method.
`link`	An URL. This can be one of the following: • An absolute URL. The string should begin with "`http://`". • A relative URL: – Starting the string with a slash (/) indicates that the URL is relative to the server root. Note that 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. • a JavaScript function name. The string should begin with a "`javascript:`" prefix.
`target`	The name of the browser window to load the URL specified in the `link` attribute with the browser features specified in the `features` attribute. If this attribute is not specified, the URL is loaded into the current browser window.

See "Example 3: Creating a menu item" on page 822 and "Example 2: Adding a custom toolbar" on page 828 for examples on how this tag is used in conjunction with the other Blox UI tags.

The <bloxui:setProperty> Tag

The `<bloxui:setProperty>` tag allows you to set the value of a property for a layout or analysis class. For example, when you specify to use a class in your `<bloxui:customAnalysis>` or `<bloxui:customLayout>` tag, you can set the value for a named property using this `<bloxui:setProperty>` tag. It has the following tag attributes:

```
<bloxui:setProperty
   name=""
   value="" />
```

where:

Attribute	Description
`name`	The name of the property.
`value`	The value of the property.

setPropertyTag Example

The following example shows how to set the default number of members to show in the popup dialog triggered by the TopN analysis class:

```
<bloxui:customAnalysis
   analysis="<%= new TopN()" >
   <bloxui:setProperty name="number" value="15" />
</bloxui:customAnalysis>
```

See "The <bloxui:customAnalysis> Tag" on page 800 for a complete example.

Model Constants and Their Values

This section lists the common model constants and their values. For a complete list, see the `ModelConstants` interface in the `com.alphablox.blox.uimodel` package in the Javadoc. *Names for the constants are all in uppercase. Their values, when specified in your Blox UI tag attributes, should all be in lowercase with no underscores ("_"), with the first letter of second and each subsequent word in uppercase.*

Chart Elements

Constant	Value
CHART	chart
CHART_FILTER	chartFilter
CHART_FILTERS_CONTAINER	chartFiltersContainer
CHART_TOTALS_FILTER	chartTotalsFilter

Menus

Constant	Value
MAIN_MENU	mainMenu
HELP_MENU	helpMenu
TOOLS_MENU	toolsMenu
DATA_MENU	dataMenu
DATA_ADVANCED_MENU	dataAdvancedMenu
FORMAT_MENU	formatMenu
BOOKMARK_MENU	bookmarkMenu
VIEW_MENU	viewMenu
CHART_MENU	chartMenu
FILE_MENU	fileMenu
EDIT_MENU	editMenu
VIEW_TOOLBAR_MENU	viewToolbarMenu
CHART_TYPES_MENU	chartTypesMenu
DATA_COMMENTS_MENU	dataCommentsMenu
FORMAT_LAYOUT_MENU	formatLayoutMenu
TOOLS_MANAGE_MENU	toolsManageMenu

Menus Elements

Constant	Value
BOOKMARK_ADD	bookmarkAdd
BOOKMARK_LOAD	bookmarkLoad
BOOKMARK_ORGANIZE	bookmarkOrganize
CHART_COMBO_TYPES	chartComboTypes
CHART_AXIS_PLACEMENT	chartAxisPlacement
CHART_DATA_VALUES	chartDataValues
CHART_ALL_DATA	chartAllData
CHART_SELECTED_DATA_ONLY	chartSelectedDataOnly
CHART_OPTIONS	chartOptions
CHART_TYPES_PIE	chartTypePie
CHART_TYPES_LINE	chartTypeLine
CHART_TYPES_BAR	chartTypesBar
CHART_TYPES_MENU	chartTypesMenu
CHART_TYPES_3DPIE	chartType3DPie
CHART_TYPES_MORE	chartTypesMore
CHART_TRENDLINES	chartTrendlines
HELP_HELP	helpHelp
HELP_ABOUT	helpAbout
DATA_SORT	dataSort
DATA_SORT_ASCENDING	dataSortAscending
DATA_SORT_DESCENDING	dataSortDescending
DATA_DRILL_UP	dataDrillUp
DATA_DRILL_DOWN	dataDrillDown
DATA_EXPAND	dataExpand
DATA_COLLAPSE	dataCollapse
DATA_EXPAND_ALL	dataExpandAll
DATA_PIVOT	dataPivot
DATA_SHOW_ONLY	dataShowOnly
DATA_REMOVE_ONLY	dataRemoveOnly
DATA_KEEP_ONLY	dataKeepOnly
DATA_HIDE	dataHide
DATA_UNHIDE_ALL	dataUnhideAll
DATA_SWAP_AXES	dataSwapAxes
DATA_MEMBER_FILTER	dataMemberFilter
DATA_CALCULATION_EDITOR	dataCalculationEditor
DATA_OPTIONS	dataOptions
DATA_ADVANCED_DRILL_THROUGH	dataAdvancedDrillThrough
DATA_ADVANCED_FORMAT_MASK	dataAdvancedFormatMask
DATA_ADVANCED_MERGED_HEADERS	dataAdvancedMergedHeaders

Constant	Value
DATA_ADVANCED_SHOW_BOTTOM_LEVEL	dataAdvancedShowBottomLevel
DATA_ADVANCED_SHOW_SIBLINGS	dataAdvancedShowSiblings
DATA_ADVANCED_SET_HIDDEN_MENU	dataAdvancedSetHidden
DATA_ADVANCED_SET_HIDDEN_ROWS	dataAdvancedSetHiddenRows
DATA_ADVANCED_SET_HIDDEN_COLUMNS	dataAdvancedSetHiddenColumns
DATA_ADVANCED_SET_HIDDEN_MEMBERS	dataAdvancedSetHiddenMembers
DATA_ADVANCED_TRAFFIC_LIGHTS	dataAdvancedTrafficLights
DATA_COMMENTS_DISPLAY_COMMENTS	dataCommentsDisplayComments
DATA_COMMENTS_ADD_COMMENT	dataCommentsAddComment
EDIT_UNDO	editUndo
EDIT_REDO	editRedo
EDIT_COPY	editCopy
EDIT_DELETE	editDelete
EDIT_SELECT_ALL	editSelectAll
EDIT_FIND	editFind
EDIT_HISTORY	editHistory
FILE_OPEN	fileOpen
FILE_SAVE_AS	fileSaveAs
FILE_EXPORT_TO_PDF	fileExportToPDF
FILE_EXPORT_TO_EXCEL	fileExportToExcel
TOOLS_GRID_OPTIONS	toolsGridOptions
TOOLS_MANAGE_TRAFFIC_LIGHTS	toolsManageTrafficLights
TOOLS_PRESENT_OPTIONS	toolsPresentOptions
VIEW_GRID	viewGrid
VIEW_CHART	viewChart
VIEW_PAGE_FILTER	viewPageFilter
VIEW_DATA_LAYOUT	viewDataLayout
VIEW_POPPED_OUT	viewPoppedOut
VIEW_TOOLBAR_CUSTOMIZE	viewToolbarCustomize
LOGO	logo
DATA_NAVIGATE_BUTTON	dataNavigateButton
EDIT_UNDO_BUTTON	editUndoButton
EDIT_REDO_BUTTON	editRedoButton

Dialog Buttons

Constant	Value
OK	ok
CANCEL	cancel
YES	yes
NO	no

Constant	Value
APPLY	apply
HELP	help

Toolbars

Constant	Value
STANDARD_TOOLBAR	standardToolbar
NAVIGATION_TOOLBAR	navigationToolbar

General Elements

Constant	Value
HEADER_CONTAINER	headerContainer
BODY_CONTAINER	bodyContainer
PRESENT_SPLITTER	presentSplitter
TREENODE_LABEL	treeNodeLabel
GRID_CELL_VALUE	gridCellValue
DATA_LAYOUT_LIST	dataLayoutList
DATA_LAYOUT_TREE	dataLayoutTree
DATA_LAYOUT_ROW_CONTAINER	dataLayoutRowContainer
DATA_LAYOUT_COLUMN_CONTAINER	dataLayoutColumnContainer
DATA_LAYOUT_PAGE_CONTAINER	dataLayoutPageContainer
DATA_LAYOUT_OTHER_CONTAINER	dataLayoutOtherContainer

Chapter 27. XML Resource Files Reference

This chapter provides general reference for writing XML resource files used to create Blox UI model containers. These XML files allow you to create model containers such as dialogs, toolbars, menus and menubars using the predefined elements and attributes. You can then write your own controller to control these components.

Resource Files Overview

Resource files are language localizable descriptions of compound model containers. They are standard XML files with predefined elements and attributes. The Blox UI Model reads the resource files and constructs all of the listed Java model objects and sets the specified attributes on each object. These resource files are how the user interface is built in the DHTML client.

You can write your own XML resource files using the formats described in this section. You can add the model UI components listed under the `com.alphablox.blox.uimodel.core` package in an XML resource file and have DB2 Alphablox construct all the Java model objects that can be further extended or modified in code as needed. Typically, you need to attach a controller for models that are created from a resource file to handle updates to and events from a model component.

For a complete example that demonstrates how to write a controller that launches a custom dialog created using an XML resource file, see the topic on Dialogs in the DHTML Client UI Extensibility section in the *Developer's Guide*.

The Top-Level Elements

Only one top-level container can be defined per resource file. The top-level element has to be one of the following:

- Dialog
- Menubar
- Menu
- Toolbar
- ComponentContainer

The UI created this way is usually either a dialog box, or menubar, a menu (such as the right-click menu), or a toolbar. These component containers then contain other components. For example, a dialog box may contain some texts (`Static`), an edit text box (`Edit`), a few check boxes (`CheckBox`), a set of radio buttons (`RadioButton`), and an OK button and a Cancel button (`Button`). Besides being a top-level element, the `ComponentContainer` element is often used to group a set of elements for better manipulation of the layout and style.

All the visual UI components descend from the Component base class and are arranged in a hierarchy that provides both formatting control as well as a way to centrally manage sets of primitive components. In an XML resource file, the `ComponentContainer` element is often used in other component containers to "group" several components. This allows you to better manipulate the layout and set attributes for all the elements in the same ComponentContainer.

For more information on the Blox UI Model, see "Blox UI Model" on page 10.

Locations for Storing XML Resource Files

The resource files you created can reside anywhere on your disk either by specifying the file path (such as `c:\path\to\yourFile.xml`), a class in the classpath, or a XML filename in com.alphablox.resource.uimodel (typically found under `<alphablox_dir>/system/AlphabloxPlatform/AlphabloxServer/abxclasses/`). When specifying the resource name in your code, if the path is not included, DB2 Alphablox automatically looks under the com.alphablox.resource.unimodel package. For instructions on how set your class path, see the *Administrator's Guide*.

Supported Argument Types

Only the following argument types are supported for models created using an XML resource file:

- string
- boolean
- integer
- Layout (horizontal, vertical, or grid)
- Style
- alignment

The supported argument types are related to the attribute types as these have corresponding "setter" methods on the Component. Only setters that take these types can be used in the XML file. For information on attributes, see "Attributes" on page 843.

Caching of Resource Files

Model resource files are cached by default. After the initial request to load a resource, resource file changes will require restarting the DB2 Alphablox. To disable caching, place a `cache="false"` attribute on the top-level resource element and restart the server if it has already started. For example;

```
<Dialog name="myDialog" title="My Own Dialog"
   cache="false" modal="true"
   height="420" width="450" layout="vertical">
   <!--other components omitted -->
</Dialog>
```

Localization

Resource files follow the same localization naming convention as resource bundles. The specified locale's language code is appended to the resource filename before it is loaded. For example, if the locale is set to French, the resource filename should have an appended `_fr` suffix. If the resource file does not exist, the unmodified resource filename is used.

Elements for XML Resource File

All the model UI components in `com.alphablox.blox.uimodel.core` package are elements you can add to a resource file. Each element has a set of attributes you can specify. As these UI components inherit the same set of properties from the `Component` class, these elements have similar attributes you can specify. The common ones include `name`, `title`, `alignment`, `valignment`, `height`, `width`, `layout`, and more. For a list of these common attributes, see "Common Attributes to All UI Elements" on page 847.

List of Elements

The element names for the UI components have the same names as the classes in the `com.alphablox.blox.uimodel.core` package, with an uppercase first letter for each word in the names. The following is a list of the elements:

Component	Description
`Button`	Push button component.
`CheckBox`	CheckBox component.
`ComponentContainer`	Generic container for UI model objects
`ControlbarContainer`	Container for Controlbars, the base class for control bars (menus and toolbars) that can be contained in a ControlbarContainer.
`Dialog`	Dialog component. Dialogs are floating containers that are used to collect input from and/or show status to users. Create a dialog and then populate the dialog with components such as Buttons, CheckBoxes and RadioButtons to present users with option lists or to make a decision.
`DropDownList`	Drop down list component. A DropDownList consists of a single displayed option with a mechanism to select from a list of other choices. Only one choice may be selected at a time. Use a DropDownList when space is limited and the constant display of possible choices is not required.
`DropDownToolbarButton`	Drop down toolbar button component. The DropDownToolbarButton has both a drop down list of selections as well as an action button to invoke the currently displayed drop down list. The control generates a ClickEvent when the selection is changed or the action button is clicked.
`Edit`	Edit field (text field) component. An Edit component allows the user to enter and modify one or more lines of text. Text can be copied, moved, and inserted into the Edit field using the standard user UI mechanisms.
`GroupBox`	GroupBox component for providing a titled container for dialogs and other models. The GroupBox component is primarily used to group components in dialog boxes. For example, if there are a number of components dedicated to setting

	options for a chart, then these should be grouped together in a GroupBox with the title ″Chart Options.″ RadioButton components will behave differently inside of a GroupBox if they are not named. All unnamed RadioButtons in a named group will become automatically grouped. Pressing one radio button will unselect others in the group.
`Image`	Image component for displaying GIF, JPEG, or other compatible image. Unlike StaticImage, an Image component will generate a ClickEvent when clicked.
`ListBox`	ListBox component.
`Menu`	Menu component consisting of MenuItems and other Menus. Menus inside of Menus will be treated as submenus with the appropriate submenu behavior. MenuItems will display as selections and will generate a ClickEvent when selected. By default, a MenuItem's name is used to construct a handler method in controllers. For example, a MenuItem with the name ″drillDown″ will map to a method called ″actionDrillDown″ in controllers. All new menus are assigned a vertical layout by default.
`Menubar`	Menubar component used in conjunction with ControlbarContainer to display top level menus.
`MenuItem`	MenuItem component. This is a selectable item in Menu.
`RadioButton`	RadioButton component.
`Spacer`	Spacer component for adding fixed height and width spacing among components
`SpinnerButton`	Spinner component that accepts integer input from the user and provides buttons to increase/decrease the value. You can set the initial value, increment, and low and high values.
`SplitterContainer`	SplitterContainer component for displaying two components with a splitter bar between them that the user can adjust. Use either the `HorizontalLayout` or the `VerticalLayout` to control the orientation of the splitter.
`Static`	Static component for displaying simple static text such as instructions, labels, and values where interaction is not needed.
`StaticImage`	Component to render a static image which does not respond to user input.
`TabbedContainer`	Container window with tabs for all child containers. This container is used to display a list of tabs corresponding to all child containers. A typical use is to implement a tabbed dialog box.

	This container can only contain other component containers. The style attached to this container is applied to the tabs. The title of the child container is used for that container's tab label. The selection state of the child containers is used to determine the selected tab. If no child containers are selected, then the first container is automatically selected. If multiple child containers are marked as selected, then the first one encountered is considered selected. The order in which the child containers are added to the tabbed container determines the tab order. For top and bottom (horizontal) layout, the first container is on the right. For left and right (vertical) layout, the first container is on the top.
`Toolbar`	Toolbar component used in conjunction with ControlbarContainer to display toolbars.
`ToolbarButton`	ToolbarButton component; can be used anywhere in the component model but are primary designed to work in ControlbarContainers. The `name` of the component is used to construct the image name by appending a `".gif"` extension.

Beside the elements listed above, there are `Item` and `ClientLink`, which are described next.

The Item Element

In the XML resource file, when a `ListBox`, `DropDownList`, or `DropDownToolbarButton` element is added, use the `Item` element to specify the individual items:

```
<DropDownList name="selectList"
      title="Undo"
      tooltip="Select an option" >
      <Item value="A" />
      <Item value="B" />
      <Item value="C" />
</DropDownList>
```

The ClientLink Element

The `ClientLink` element defines a URL-based link that will be handled by the browser when the component is clicked. This element allows you to specify a link, the target window in which the new page is to be loaded, and a browser window feature string (for example, `"toolbar=no,scrollbars=yes"`) in much the same way as the JavaScript's `window.open()` method.

Attributes

Attribute names have lowercase first word, with the first letter in each subsequent word in uppercase, such as `name`, `title`, `width`, `height`, `themeClass`, `imageURL`, and `themeBasedImage`. Since all the UI components derive from the `Component` class, they share many common attributes. These attributes correspond to the "setter" methods on the Component object (for example, `setName()`, `setTitle()`, `setWidth()`, and `setHeight()`). For a listing of these common attributes, see "Common Attributes to All UI Elements" on page 847. For details on their methods, see the Javadoc under the `com.alphablox.blox.uimodel.core` package.

The following example of a simple dialog demonstrates how to add different elements, specify their attributes, and manipulate the layout to create a dialog box using the Blox UI model components.

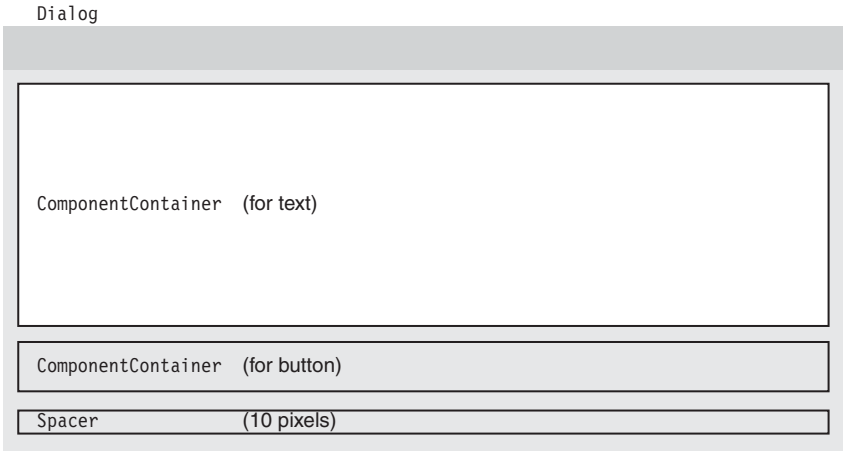
Examples of Resource XML Files

Example 1: An "About" Dialog Box

The first sample XML resource file creates an "About MyApp" dialog:

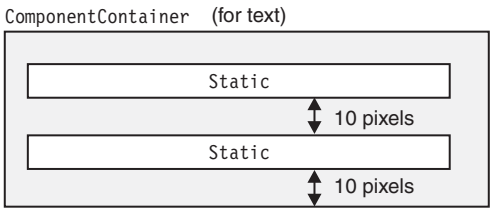
```
<?xml version="1.0" ?>
<Dialog name="aboutDialog" title="About MyApp"
   height="150" width="400" layout="vertical">

   <ComponentContainer layout="vertical" alignment="center">
      <Static name="credit"
       title="Brought to you by the Information Technology group"
       themeClass="csLblFnt csThmClr csAbtTxt" />

      <!-- Add a 10px space in between two Static components -->
      <Spacer />
      <Static name="company"
       title="Copyright 2003 Your company name here."
       themeClass="clsIbar" />
      <Spacer />
   </ComponentContainer>

   <!-- Add another ComponentContainer to have the button aligned in
        the center -->
   <ComponentContainer layout="horizontal" alignment="center">
      <!--If button text is less than 7-8 characters, add width="70" -->
      <Button name="ok" title="OK" />
   </ComponentContainer>
   <!-- Add 10px margin from bottom -->
   <Spacer />
</Dialog>
```

- The dialog box has a height of 150 pixels and a width of 400 pixels. The elements contained in this dialog are to be stacked vertically (`layout="vertical"`). If the width or height of a spacer is not specified, the default is 10 pixels in height and 10 pixels in width.



- The first `ComponentContainer` contains a credit statement (`Static`) and a company/copyright statement (`Static`).
- The two Static components are stacked vertically (`layout="vertical"`) and aligned in the center (`alignment="center"`).

- Some CSS classes are applied to the `Static` components. The theme classes are described in the Presenting Data chapter in the *Developer's Guide*.



- The second `ComponentContainer` contains a Button. In order to align the button in the center, it needs to be in its own `ComponentContainer` or it will align on the left with the two `Static` components.

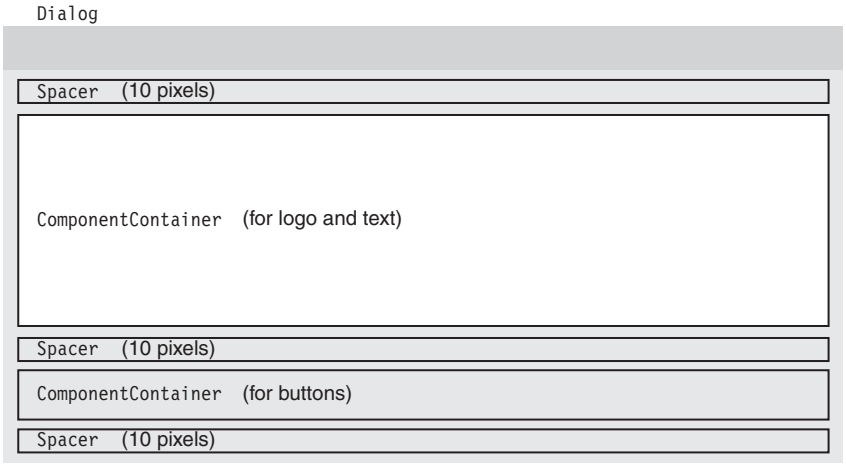
Example 2: A Confirmation Dialog Box

The second sample XML resource file creates a confirmation dialog:

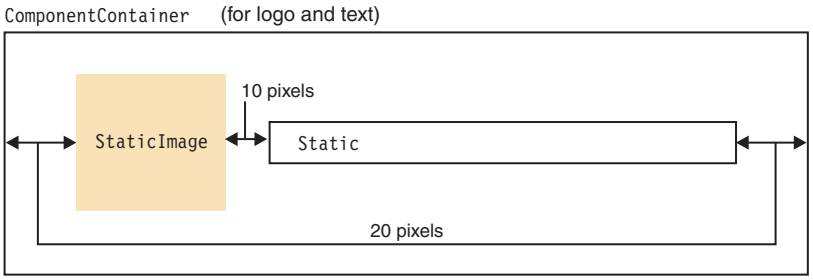
The XML code that produces the above dialog is as follows

```
<?xml version="1.0" ?>
<Dialog name="myDialog" title="Confirmation" modal="true"
   height="140" width="400" layout="vertical">

   <!-- Add 10px margin from top -->
   <Spacer />
   <!-- Need a horizontal layout in the main area in order to add 20px
        margin on each side -->
   <ComponentContainer layout="horizontal">
      <!-- Add 20px margin on left side -->
      <Spacer width="20" />

      <!-- CONTENT AREA-->
      <StaticImage imageURL="/SalesApp/images/logo.gif" />
      <Spacer width="10" />
      <Static name="credit"
         title="Do you want to apply the change?"
         themeClass="csLblFnt csThmClr csAbtTxt" />

      <!-- Add 20px margin on right side -->
      <Spacer width="20" />
   </ComponentContainer>

   <!-- Add 10px margin between content area and buttons -->
   <Spacer />

   <ComponentContainer name="buttonContainer" layout="horizontal"
    alignment="right">
      <!--If button text is less than 7-8 characters, add width="70" -->
      <Button name="ok" title="Yes" width="70" />

      <!-- Add 5px margin between buttons -->
      <Spacer width="5" />
      <Button name="cancel" title="Cancel" width="70" />
      <!-- Add 20px margin on right side, matching main content area -->
      <!-- Only put this in if there is equivalent or greater margin on the
           left already -->
      <Spacer width="20" />
   </ComponentContainer>

<!-- Add 10px margin from bottom -->
   <Spacer />
</Dialog>
```

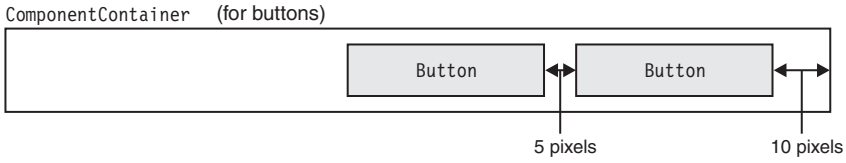
- The dialog box has a height of 150 pixels and a width of 400® pixels. The elements contained in this dialog are to be stacked vertically (`layout="vertical"`).



- A `ComponentContainer` is added to include a logo (`StaticImage`) and text (`Static`).
  - The `ComponentContainer's` layout is set to `horizontal` so components inside this container are stacked from left to right.
  - Some CSS classes are applied to the components. The theme classes are described in the Presenting Data chapter in the *Developer's Guide*.



- Another `ComponentContainer` is added to contain two buttons (`Button` components). The two buttons are right aligned (`alignement="right"`) with 10 pixels at the end to add some space before the border.



Element Attributes

This section lists the attributes common to all elements:

- "Common Attributes to All UI Elements" on page 847
- "Additional Attributes for CheckBox and RadioButton" on page 849
- "Additional Attributes for ControlbarItem, MenuItem, and ToolbarButton" on page 849

Common Attributes to All UI Elements

The following table lists the attributes common to all elements:

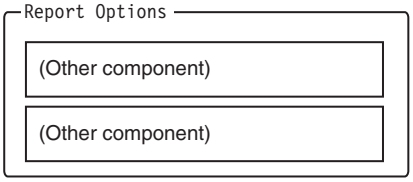
Attribute	Description
`name`	Specifies the name of the component. This is used to identify the component and provides the action name for the component. For example, if the component name is `myButton`, a ClickEvent on the component will invoke the method `actionMyButton`. **Important:** When naming your components, avoid reserved component names. Component names are the same as the constants in the *ModelConstants* interface in the com.alphablox.blox.unimodel package.
`title`	Specifies the title of the component. This is the displayed text. See "Use of the title Attribute" on page 848 for more information on the usage of the title attribute in each component.
`alignment`	Specifies the horizontal alignment of the component. Valid values are `right`, `left`, and `center`.
`batchEvents`	Specifies whether a corresponding event should be sent to the server immediately as the user takes an action that causes the componenet to change state. When this is set to `false`, as soon as the user takes an action that causes this component to change state, the client sends the event to the server. When this is set to `true`, events generated by this component are batched on the client until some other action causes the client to connect to the server (such as sending events from other components). This setting only effects events sent by components in dialogs.
`setBusyAfterEvent`	Specifies if the UI should be set to busy state each time the component generates an event. When set to `true`, UI of this component is set to busy when an event is generated. Valid values are `true` and `false`. This is useful in stopping user input during long or sensitive operations. This setting controls the behavior of the UI immediately after an event is generated.

`clickable`	Specifies if the component should generate mouse click events. Valid values are `true` and `false`.
`disabled`	Specifies whether the component is to be disabled. Valid values are `true` and `false`.
`height`	Specifies the height of the component in pixels.
`layout`	Specifies the layout to attach to the container. Valid values are `vertical`, `horizontal`, and `grid(`*numOfColumns*`)`. This attribute only applies to ComponentContainer and components deriving from it. For example, to create a layout of a four-column grid: layout=″grid(4)″
`setRightClickMenu`	Sets the right-click menu for this component. This menu will be displayed when users right-click the component.
`style`	Specifies the style to be attached to the component. The style can be expressed as a CSS-like style string. It can also be the name of a Style object.
`tabStop`	Sets the component as a tab stop. Valid values are `true` and `false.` The default is `true`. Note, however, that tab stop does not work with radio buttons. This is a browser behavior.
`themeClass`	Specifies the name of the theme class or classes that should be used for the component.
`tooltip`	Specifies the tool tip (popped-up text when users mouse over the component) to be attached to the component.
`valignment`	Specifies the vertical alignment of the component. Valid values are `top`, `bottom`, and `center`.
`visible`	Specifies the visibility of the component. Valid values are `true` and `false`.
`width`	Specifies the width of the component in pixels.

Use of the title Attribute

The following table describes the use of the `title` attribute in each element:

Element	**Use of the title Attribute**
`Button`	The button label.
`CheckBox`	The text displayed after the checkbox.
`ComponentContainer`	The title for top-level elements. Otherwise, it is ignored.
`Edit`	Ignored.
`GroupBox`	The title of the GroupBox. For example, with `title="Report Options"`, the display is as follows:



`Image StaticImage`	Ignored.
`ListBox DropDownList`	Ignored.
`Menu MenuItem`	The menu label.
`Menubar Toolbar`	Used in menus to refer to Toolbars. Otherwise, it is ignored.
`Spacer`	Ignored.
`Static`	The text to display.

Additional Attributes for CheckBox and RadioButton

Besides the common attributes described in "Common Attributes to All UI Elements" on page 847, the `CheckBox` and `RadioButton` have the following attributes:

Attribute	Description
`checked`	Specifies if the checkbox or radio button should be checked (selected). Valid values are `true` and `false`.

Additional Attributes for ControlbarItem, MenuItem, and ToolbarButton

Besides the common attributes described in "Common Attributes to All UI Elements" on page 847, the `ControlbarItem`, `MenuItem`, and `ToolbarButton` elements have the following attributes:

Attribute	Description
`checked`	Specifies if the MenuItem should be checked (selected). Valid values are `true` and `false`.
`checkBox`	Specifies if the MenuItem should act like a checkbox, with a check mark next to it when it is selected. Valid values are `true` and `false`.
`imageURL`	Specifies the image URL: • 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. Note that 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 theme directory. Typically, this directory is located at: `<alphablox>/repository/theme/<themeName>/i/`.
`separator`	Specifies that the item is a toolbar or menu separator. Valid values are `true` and `false`.

themeBasedImage	Specifies whether this image is located in the directory of the current theme. Valid values are true and false. When this attribute is set to true, the server looks for an image file that has the same name as the component, with a .gif extension.

Additional Attributes for Dialog

Besides the common attributes described in "Common Attributes to All UI Elements" on page 847, Dialog has the following attributes:

Attribute	Description
defaultButton	Specifies the default button that is clicked when users press the Enter key. All dialogs are created with the default button set to the OK button. Once users click another button, that button becomes the default.
modal	Specifies that the dialog is a model dialog. A model dialog lives in its own separate movable window and stops the rest of the UI from accepting input until it is dismissed.
resizable	Specifies that the user can resize the dialog window. Valid values are true and false. The default is false.

Additional Attributes for Image and StaticImage

Besides the common attributes described in "Common Attributes to All UI Elements" on page 847, the Image and StaticImage elements have the following attributes:

Attribute	Description
themeBasedImage	Specifies whether this image is located in the directory of the current theme. Valid values are true and false. When this attribute is set to true, the server looks for an image file that has the same name as the component, with a .gif extension in the theme's image directory. Typically, this directory is located at: <alphablox>/repository/theme/<themeName>/i
imageURL	Specifies the image URL as follows: • 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. Note that 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 theme directory if themeBasedImage is set to true. If themeBasedImage is set to false, starting the string without a slash indicates that the URL is relative to the current application directory.

Additional Attributes for Static

Besides the common attributes described in "Common Attributes to All UI Elements" on page 847, the Static element has the following attributes:

Attribute	Description
`wrapText`	Specifies whether the text specified in the `title` attribute of a Static component should be wrapped. Valid values are `true` and `false`. The default is `false`.

Special Attribute for Top-level Component Containers

The five top-level component containers—Dialog, Menu, Menubar, Toolbar, and ComponentContainer— have one special attribute to specify caching of the resources:

Attribute	Description
`cache`	Specifies whether resources will be cached after loading. The default is `true`, meaning any changes made to the resource files require restarting the server.

Attributes for Item

The `Item` Element has only one attribute:

Attribute	Description
`value`	Specifies the item to be added to the list.

Attributes for ClientLink

The `ClientLink` element has three attributes. These attributes allow you to specify the same arguments you can pass in a `window.open()` JavaScript method.

Attribute	Description
`link`	Specifies the client link (a URL) for the component. Specification of this attribute is required.
`target`	Specifies the target window for the URL to load. This is optional. If not specified, the new URL is loaded into the same window.
`features`	Specifies the window features in a comma-separated string if the `target` attribute is specified. For example: `features="scrollbars=yes,width=300,height=300"`

Examples for Top-Level Elements

This section provides an example XML resource file for each of the top-level elements. For a complete example that demonstrates how to write a controller to control components created using an XML resource file, see the dial chart example in the Blox Sampler (DHTML version) under the UI Extensibility section.

ComponentContainer Element

The following XML is the actual resource file used by the Chart Types and Configuration dialog. This dialog has three tabs, one of which is Chart Types. When users click this tab, the following resource file is called:

```
<?xml version="1.0" ?>
<ComponentContainer name="chartTypesTab"
   title="Chart Types"
   layout="vertical"
   alignment="left">
   <Spacer />
   <ComponentContainer layout="horizontal">
      <Spacer width="20" />
      <ComponentContainer layout="vertical">
         <ComponentContainer layout="horizontal" alignment="left">
            <Static title="Chart Type" alignment="left" />
            <Spacer width="10" />
            <DropDownList name="chartTypesSelector" />
         </ComponentContainer>
         <Spacer height="5" />
         <Image name="chartTypeImage" />
      </ComponentContainer>
      <Spacer width="20" />
   </ComponentContainer>
</ComponentContainer>
```

The following is the part of the XML in the Chart Types and Configuration dialog resource file that loads the above resource file:

```
<?xml version="1.0" ?>
<Dialog name="chartTypesDialog" title="Chart Types and Configuration"
   modal="false" height="420" width="450" layout="vertical">
   <TabbedContainer name="ChartTypesTabContainer"
      themeClass="csChrtCntnr" height="360" width="440"
      layout="horizontal" alignment="left">
   </TabbedContainer>
   <!--other components omitted -->
</Dialog>
```

Menu Element

The following XML creates a menu called `myFormatMenu`, with two menu items. This menu could then be, for example, attached to a component as a right-click menu.

```
<?xml version="1.0" ?>
<Menu name="myFormatMenu" title="Format" valignment="top">
   <MenuItem name="layout1" title="Special Layout 1"
      valignment="top" />
   <MenuItem name="layout2" title="Special Layout 2"
      valignment="top" />
</Menu>
```

Menubar Element

The following XML creates a View menu under a custom menubar called `myMenubar`. The View menu has four menu options: Grid, Chart, Page Filter, and Data Layout. Following the View menu is a StaticImage that does not send any ClickEvent when clicked.

```
<?xml version="1.0" ?>
<Menubar name="myMenubar" layout="horizontal"
   themeClass="csCmpBg csThmClr csCmpBrdr csMnbr" >
   <Menu name="view" layout="vertical" title="View"/>
      <MenuItem name="viewGrid" title="Grid" checkBox="true"
          themeBasedImage="true" setBusyAfterEvent="true" />
```

```
      <MenuItem name="viewChart" title="Chart" checkBox="true"
          themeBasedImage="true" setBusyAfterEvent="true" />
      <MenuItem name="viewPageFilter" title="Page Filter"
          checkBox="true" themeBasedImage="true"
          setBusyAfterEvent="true" />
      <MenuItem name="viewDataLayout" title="Data Layout"
          checkBox="true" themeBasedImage="true"
          setBusyAfterEvent="true" />
   </Menu>
   <StaticImage name="logo" imageURL="smallLogo.gif"
      tooltip="Copyright (C) 2004 My Company"
      valignment="top" themeClass="clsStaticImage"
      themeBasedImage="true" />
</Menubar>
```

Dialog Element

See "Examples of Resource XML Files" on page 844 for a detailed layout discussion.

Toolbar Element

This sample XML resource file creates a Toolbar with the Excel, PDF, and Help buttons, with a separator between the PDF button and the Help button.

```
<Toolbar name="myToolbar" title="Exporting" layout="horizontal">

   <ToolbarButton name="fileExportToExcel" title="Excel"
    tooltip="Export to Excel"
    Bookmark" themeBasedImage="true" />

   <ToolbarButton name="fileExportToPDF" title="PDF"
    tooltip="Export to PDF"
    themeBasedImage="true" />

   <ToolbarButton separator="true" />

   <ToolbarButton name="helpHelp" title="Help" tooltip="Help"
    themeBasedImage="true" />

</Toolbar>
```

In this case, since `fileExportToExcel`, `fileExportToPDf`, and `helpHelp` are built-in ToolbarButtons (see "Custom Toolbar Tags" on page 823 for toolbar constants and values), we do not need to write our own controller to control these components. If you create your own ToolbarButton, a controller to handle the updates to and events coming from these components is needed. The images used for the buttons are theme-based, meaning they exist in the `<alphablox_dir>/repository/theme/i` directory, with different image files available for active, inactive, and disabled modes. You can also specify the URL to the image using the `imageURL` attribute. For more information on `themeBasedImage` and `imageURL`, see "The <bloxui:toolbarButton> Tag" on page 825.

Chapter 28. Using the Alphablox XML Cube

The Alphablox XML Cube defines XML tags and attributes for representing query result sets returned from application data sources or DB2 Alphablox cubes. When a result set is transformed into an Alphablox XML Cube document, the document presents an open, predictable data structure with known elements, regardless of the layout of the underlying data source.

The Alphablox XML Cube presents a tree view of the data, as does the W3C XML DOM standard, where nodes correspond to data elements. The W3C XML DOM includes functions for manipulating document elements; Alphablox has extended the DOM to provide convenience methods particularly suited for manipulating analysis cube data. (For more information on these extensions, see Chapter 29, "Extended DOM API Reference," on page 863.)

Application programmers can access the XML Cube document and process its data to implement custom logic or data layouts. This chapter explains the Alphablox XML Cube by using a familiar data representation often used in Alphablox applications.

Data Representation

Application programmers familiar with the DB2 Alphablox representation of an analysis cube result set will quickly understand the organization of the Alphablox XML Cube. This section reviews the key concepts of the DB2 Alphablox representation, using the following simple example:

Product	EAST	West	South	Market
Audio	12,460	15,507	0	27,967
Visual	33,138	26,605	24,565	84,308
Product	45,598	42,112	24,565	112,275

The result set includes descriptive elements (names of dimensions and members) and associated data values. A typical DB2 Alphablox representation, shown in the example, organizes the descriptive elements into row and column axes, and the data values into data cells. Note the following about the example:

- The Market dimension resides on the column axis, and includes three members: East, West, and South, as well as the Market rollup.
- The Product dimension resides on the row axis, and includes two members: Audio and Visual, as well as the Product rollup.
- Multiple dimensions can reside on the same axis. When this occurs, there is an implied grouping, with one dimension grouped within another.

- A *tuple* represents a set of members, one from each dimension on an axis. In the example, because there is only one dimension on each axis, each tuple represents a single member.
- Data values appear in cells at the intersection of tuples. For example, a value of 12,460 appears at the intersection of the Audio tuple and East tuple.

Note: In GridBlox or PresentBlox, unused dimensions reside on the Other axis. In the Alphablox XML Cube, each unused dimension resides on a separate slicer axis.

The next section explains how to render a query result set into XML format.

Sample Alphablox XML Document

Below is the example result set rendered as an XML document. In some cases, line breaks have been added for readability.

```
<?xml version="1.0"?>

<!DOCTYPE cube SYSTEM '/alphablox/AnalysisServer/xml/dtd/cube.dtd'>

<cube>
   <bloxInfo>
     <bloxID>15</bloxID>
     <bloxName>MyDataBlox</bloxName>
     <appName>MyXMLDoc</appName>
   </bloxInfo>
   <data>
   <slicer>
     <slicerDimension name="Period">Period</slicerDimension>
     <slicerMember name="Period" gen="1"
                   leaf="false">Period</slicerMember>
     </slicer>
     <slicer>
       <slicerDimension name="Accounts">Accounts</slicerDimension>
       <slicerMember name="Accounts" gen="1"
                     leaf="false">Accounts</slicerMember>
     </slicer>
     <slicer>
       <slicerDimension name="Scenario">Scenario</slicerDimension>
       <slicerMember name="Scenario" gen="1"
                     leaf="false">Scenario</slicerMember>
     </slicer>
  <axis name="columns" index="0">
       <dimensions>
        <dimension name="Market" index="0">Market</dimension>
       </dimensions>
       <tuple index="0">
        <member name="East" index="0" gen="2" spanInHierarchy="1"
            spanIndexInHierarchy="0" leaf="false">East</member>
       </tuple>
       <tuple index="1">
        <member name="West" index="0" gen="2" spanInHierarchy="1"
            spanIndexInHierarchy="0" leaf="false">West</member>
</tuple>
<tuple index="2">
        <member name="South" index="0" gen="2" spanInHierarchy="1"
            spanIndexInHierarchy="0" leaf="false">South</member>
</tuple>
<tuple index="3">
        <member name="Market" index="0" gen="1" spanInHierarchy="1"
            spanIndexInHierarchy="0" leaf="false">Market</member>
</tuple>
  </axis>
```

```
  <axis name="rows" index="1">
<dimensions>
        <dimension name="Product" index="0">Product</dimension>
</dimensions>
<tuple index="0">
        <member name="Audio" index="0" gen="2" spanInHierarchy="1"
            spanIndexInHierarchy="0" leaf="false">Audio</member>
</tuple>
<tuple index="1">
        <member name="Visual" index="0" gen="2" spanInHierarchy="1"
            spanIndexInHierarchy="0" leaf="false">Visual</member>
</tuple>
<tuple index="2">
        <member name="Product" index="0" gen="1" spanInHierarchy="1"
            spanIndexInHierarchy="0" leaf="false">Product</member>
</tuple>
  </axis>
  <cells>
    <row>
      <column>
        <cell>13438.0</cell>
      </column>
      <column>
        <cell>22488.0</cell>
      </column>
      <column>
        <cell>0.0</cell>
      </column>
      <column>
        <cell>35926.0</cell>
      </column>
    </row>
    <row>
      <column>
        <cell>33138.0</cell>
      </column>
      <column>
        <cell>40351.0</cell>
      </column>
      <column>
        cell>24565.0</cell>
      </column>
      <column>
        <cell>98054.0</cell>
      </column>
    </row>
    <row>
      <column>
        <cell>46576.0</cell>
      </column>
      <column>
        <cell>62839.0</cell>
      </column>
      <column>
        <cell>24565.0</cell>
      </column>
      <column>
        <cell>133980.0</cell>
      </column>
    </row>
  </cells>
 </data>
</cube>
```

Alphablox XML Tags

DB2 Alphablox uses the XML tags described in this section to represent the elements in a query result set returned from an application data source. The tags support analysis cubes with an unlimited number of axes.

As with all XML tags, the following rules apply to the Alphablox tags:

- An XML document must begin with the XML declaration:
  `<?xml version="1.0"?>`
- There can be one and only one root element in an XML document. All other tags that define the document's content are contained within the root element. The following tags define the root element for the Alphablox XML Cube:
  `<cube>...</cube>`
- Tags can nest but cannot overlap. For example, the following is valid:
  `<bloxInfo><bloxName>myBlox</bloxName></bloxInfo>`
  but the following is not valid:
  `<bloxInfo><bloxName>myBlox</bloxInfo></bloxName>`

The tags are listed in their order of appearance in an XML document.

XML Tag	
`<?xml version="1.0"?>`	Identifies an XML document and names the W3C specification that it uses. The Alphablox XML Cube uses the 1.0 specification.
`<!DOCTYPE cube...>`	Identifies the document type (by naming its root element) and specifies the associated DTD file.
`<cube> </cube>`	Identifies the document root element for the Alphablox XML Cube. There can be one and only one cube element in an XML document. All the other elements are contained within it.
`<bloxInfo> </bloxInfo>`	Provides information about this Blox.
`<bloxID> </bloxID>`	Identifies this Blox instantiation (the value is automatically provided by DB2 Alphablox).
`<bloxName> </bloxName>`	Identifies the Blox used for this instantiation (the value is taken from the `bloxName` property).
`<appName> </appName>`	Identifies the application (the value is taken from the Blox `applicationName` property).
`<data> </data>`	Identifies the entire data area of the XML document, including axis definitions and data cells.
`<slicer> </slicer>`	Identifies the area of the XML document that defines a slicer axis. A slicer axis provides a "slice through" the cube. With OLAP data sources, each dimension that does not appear in the data area is placed on a separate slicer axis, along with one of its members.
`<slicerDimension> </slicerDimension>`	Names the dimension on the slicer axis.
`<slicerMember> </slicerMember>`	Names the member in a slicer dimension.

<axis> </axis>	Identifies the axis type (typically column or row) and defines its contents. The five named axes (in low-to-high sequence) are column, row, page, chapter, and section. An axis can also be referred to as Axis[index], where index is in the range of 0 to N. For example, one way to refer to the row axis is Axis[1]; the way to refer to the first unnamed axis, is Axis[5].
<dimensions> </dimensions>	Identifies an area within the <axis> </axis> tags where the dimensions on the axis are named.
<dimension> </dimension)	Identifies a specific dimension.
<tuple> </tuple>	Identifies an element containing a set of all the dimension members on one axis.
<member> </member>	Identifies a member that belongs to a tuple.
<cells> </cells>	Identifies the area of the XML document that contains data values (as opposed to dimension and member headings).
<axisCells> </axisCells>	Identifies an unnamed axis (also referred to as Axis[5] through Axis[N]) and includes its data cells.
<section> </section>	Identifies a section axis (also referred to as Axis[4]) and includes its data cells.
<chapter> </chapter>	Identifies a chapter axis (also referred to as Axis[3]) and includes its data cells.
<page> </page>	Identifies a page axis (also referred to as Axis[2]) and includes its data cells.
<row> </row>	Identifies a row axis (also referred to as Axis[1]) and includes its data cells.
<column> </column>	Identifies a column axis (also referred to as Axis[0]) and includes its data cells.
<cell> </cell>	Identifies a data value at the intersection of tuples.

Alphablox XML Tag Attributes

Alphablox XML tags use the following attributes. Remember that indexes are 0-based.

Attribute	Description
gen	Identifies the generation level of this element within its data hierarchy (specifically, the level of a member within its dimension). For example, in the Product dimension, Product has a gen value of “1”, Audio and Visual have a gen value of “2”, and VCR and TV have a gen value of “3”.
index	Identifies the position of this element in a series of like elements. For example, the following lines indicates that this is the first tuple: <tuple index=″0″>...</tuple>

leaf	Specifies if this is a leaf node (true) or not (false).
name	Provides a unique name for this element. For example, the following line provides a name (as well as a data value) for a dimension element: <dimension name=″memberName″ index=″0″>AliasName</dimension>
span	Identifies the number of tuples that a member spans. For example, a member named Qtr1 would have a span of “3” (for January, February, and March). For MemberElement, use spanInHierarchy rather than span.
spanInHierarchy	Identifies the number of tuples that a member spans within a hierarchy defined by the same root parent. For example, a member named March sould have a spanInHierarchy value of 3. For MemberElement, use spanInHierarchy rather than span.
spanIndex	Indicates the zero-based position of this member in a series of spanned members. For example, January would have a spanIndex of “0”; February, “1”; and March, “2”. For MemberElement, use spanIndexInHierarchy rather than spanIndex.
spanIndexInHierarchy	Indicates the zero-based position of this member in a series of spanned members, but relative to the root parent in its hierarchy. For example, if April has a spanIndex of 3, but occurs within the column Qtr2, the spanIndexInHierarchy value would be 0. For MemberElement, use spanIndexInHierarchy rather than spanIndex.

XML Data Islands

An XML data island is a block of valid XML code embedded inside an HTML document. Data islands enable programmers to script against the XML document without having to load it (through script or the <OBJECT> tag). Currently, XML data islands are supported only in Microsoft Internet Explorer 5.5 and later.

Definition Syntax

The syntax for defining an inline data island in a page appears below. Note the use of the <XML> and </XML> tags:

```
<XML ID="DataIslandID">
  <XMLDATA>
    <DATA>TEXT</DATA>
  </XMLDATA>
</XML>
```

For example, the following lines define a data island with three data values:

```
<XML ID="MyDataIsland">
  <dataSources>
    <dataSource name="DB2">IBM DB2 OLAP Server 8.1</dataSource>
    <dataSource name="MSOLAP">Microsoft OLAP Services 7.0</dataSource>
    <dataSource name="Essbase">Hyperion Essbase 6.5</dataSource>
  </dataSources>
</XML>
```

The contents of a data island can also reside in an external file. Use the following syntax to include an external XML file as a data island:

```
<XML SRC="http://<server>/MyXmlFile.xml"></XML>
```

XMLDocument Property

The XMLDocument property returns the root node of the inline or external XML data island. Programmers can use the standard XML DOM to navigate the data island from this root. For example, the following function returns all the data from MyDataIsland.

```
function returnXMLData(){
   return document.all("MyDataIsland").XMLDocument.nodeValue;
}
```

The following syntax is also valid. Using the example from "Definition Syntax" on page 860, the line returns a value of "IBM DB2 OLAP Server 8.1":

```
MyDataIsland.XMLDocument.documentElement.childNodes.item(0).text
```

DataBlox as an XML Data Island

The following lines define a standard DataBlox as a data island:

```
<XML ID="MyDataBlox">
   <blox:data id="MyDataBlox"
      dataSourceName = "qcc">
      query = "!"
      render = "XML">
   </blox:data>
</XML>
```

By setting the `render` attribute to `XML` causes the DataBlox result set to be rendered into XML format. When the page is processed, the lines defining the Blox are replaced by the rendered XML lines.

Elsewhere in the page, a JavaScript function could gain access to the contents of the data island through syntax like the following:

```
MyDataBlox.getCube.getUniqueName
```

Chapter 29. Extended DOM API Reference

This chapter provides the API reference for the methods available on the classes used for the DB2 Alphablox extended DOM.

DB2 Alphablox Extended DOM Overview

A Document Object Model (DOM) uses a standard syntax to describe a document as a series of objects. The W3C XML DOM Specification defines the basic XML API implemented in the Alphablox XML Cube DOM.

Alphablox has extended the DOM to describe the objects found in an analysis cube result set, and to provide methods for locating, retrieving, and manipulating those objects. The API is for use with both Java and JavaScript. For example, the API enables programmers to request a completely new DOM after performing such actions as drill down or pivot.

This chapter describes the DB2 Alphablox extensions.

Note: All indexes are 0-based, the convention in both Java and JavaScript.

Important: Modifications to abstract XML elements throw exceptions (org.w3c.dom.DOMException). The following modification methods throw exceptions: appendChild, removeChild, replaceChild, insertBefore, and setAttribute.

For more information about the DB2 Alphablox extended DOM, see Chapter 28, "Using the Alphablox XML Cube," on page 855.

AASCubeXMLDocument

Package	com.alphablox.blox.xml
Inherits	None
Description	The class for the DB2 Alphablox extended DOM to represent the result set as XML

The following method is available on the `AASCubeXMLDocument` class:

- `getCube()`

getCube()

Get the cube element.

Syntax

Java Method

```
CubeElement getCube( );
```

AbstractXMLElement

Package	com.alphablox.blox.xml
Inherits	None
Description	The abstract class for all DB2 Alphablox XML elements; a super class from which common methods are inherited

The following method is available on the `AbstractXMLElement` class:

- getIntAttribute()

getIntAttribute()

Get the value for the named attribute as an Integer; throw exceptions if the value is not an integer or if the attribute is not a valued attribute of the element.

Syntax

Java Method

```
int getIntAttribute(String attrName);
        throws NumberFormatException, IllegalArgumentException;
```

where:

Argument	Default	Description
`attrName`	none	A string representing a named attribute.

AbstractDimensionElement

Package	com.alphablox.blox.xml
Inherits	AbstractXmlElement
Description	The abstract class for the dimension element

The following method is available on the `AbstractDimensionElement` class:

- getUniqueName()
- getDisplayName()

getUniqueName()

Get the unique name of the element. (For IBM DB2 OLAP Server or Hyperion Essbase data sources, the unique name is replaced by an alias if the query specifies to use aliases.)

Syntax

Java Method

```
String getUniqueName();
```

getDisplayName()

Get the display name of the element.

Syntax

Java Method

```
String getDisplayName();
```

AbstractMemberElement

Package	com.alphablox.blox.xml
Inherits	AbstractXmlElement
Description	The abstract class for the member element

The following methods are available on the `AbstractMemberElement` class:

- getUniqueName()
- getDisplayName()
- getGenerationLevel()
- getIsLeaf()

getUniqueName()

Get the unique name of the element. This returns the outline name from the database, not the alias.

Syntax

Java Method

```
String getUniqueName();
```

getDisplayName()

Get the display name of the element. (For IBM DB2 OLAP Server or Hyperion Essbase data sources, the display name is replaced by an alias if the query specifies to use aliases.)

Syntax

Java Method

```
String getDisplayName();
```

getGenerationLevel()

Get the value of the member's generation level (gen) attribute. A value of zero (0) indicates the member has no parent.

Syntax

Java Method

```
int getGenerationLevel();
```

getIsLeaf()

Return `true` if the member has no children.

Syntax

Java Method

```
boolean getIsLeaf();
```

CubeElement

Package	com.alphablox.blox.xml
Inherits	AbstractXMLElement
Description	The class for the cube element

The following methods are available on the `CubeElement` class:

getSlicerCount()

Get the number of slicers. (A slicer is an axis used to filter data. Each slicer can name one dimension and member, such as Market, East, New York.)

Syntax

Java Method

```
int getSlicerCount();
```

getSlicer(int n)

Get the *n*th slicer. Return null if the slicer does not exist.

Syntax

Java Method

```
SlicerElement getSlicer(int n);
```

getAxisCount()

Get the number of axes.

Syntax

Java Method

```
int getAxisCount();
```

getAxis(int index)

Get the nth axis. Return null if the axis does not exist.

Syntax

Java Method

```
AxisElement getAxis(int index);
```

getAxis(String axisName)

Get the named axis. Return null if the named axis does not exist. (axisName can be one of the `AxisElement` static constants described under "AxisElement" on page 870.)

Syntax

Java Method

```
AxisElement getAxis(String axisName);
```

getBloxInfo()

Get the BloxInfo element.

Syntax

Java Method

```
BloxInfoElement getBloxInfo();
```

getCells()

Get the cells element.

Syntax

Java Method

```
CellsElement getCells();
```

BloxInfoElement

Package	com.alphablox.blox.xml
Inherits	AbstractXmlElement
Description	The class for the BloxInfo element

The following methods are available on the `BloxInfoElement` class:

- getBloxName()
- getBloxID()
- getApplicationName()

getBloxName()

Get the unique name of the Blox.

Syntax

Java Method

```
String getBloxName();
```

getBloxID()

Get the system-assigned Blox ID.

Syntax

Java Method

```
int getBloxID();
```

getApplicationName()

Get the application name.

Syntax

Java Method

```
String getApplicationName();
```

SlicerElement

Package	com.alphablox.blox.xml
Inherits	AbstractXmlElement
Description	The class for the slicer element

The following methods are available on the `SlicerElement` class:

- getDimension()
- getMember()

getDimension()

Get the dimension element of the slicer.

Syntax

Java Method

```
SlicerDimensionElement getDimension();
```

getMember()

Get the member element of the slicer.

Syntax

Java Method

```
SlicerMemberElement getMember( );
```

SlicerDimensionElement

Package	com.alphablox.blox.xml
Inherits	AbstractDimensionElement
Description	The class for the slicerDimension element

The following methods are available on the `SlicerDimensionElement` class:

- getDisplayName()
- getSlicer()
- getMember()
- getUniqueName()

getDisplayName()

Gets the display name of the element.

Syntax

Java Method

```
String getDisplayName();
```

getSlicer()

Get the slicer element of this dimension.

Syntax

Java Method

```
SlicerElement getSlicer();
```

getMember()

Get the member element of the dimension.

Syntax

Java Method

```
SlicerMemberElement getMember();
```

getUniqueName()

Gets the unique name of the element.

Syntax

Java Method

```
String getUniqueName();
```

SlicerMemberElement

Package	com.alphablox.blox.xml
Inherits	AbstractMemberElement
Description	The class for the slicerMember element

The following methods are available on the `SlicerMemberElement` class:

- getDimension()
- getDisplayName()
- getSlicer()
- getUniqueName()

getDimension()

Get the slicer dimension of the member.

Syntax

Java Method

```
SlicerDimensionElement getDimension();
```

getDisplayName()

Gets the display name of the element.

Syntax

Java Method

```
String getDisplayName();
```

getSlicer()

Get the slicer element of this member.

Syntax

Java Method

```
SlicerElement getSlicer();
```

getUniqueName()

Gets the unique name of the element.

Syntax

Java Method

```
String getUniqueName();
```

AxisElement

Package	com.alphablox.blox.xml
Inherits	AbstractXmlElement
Description	The class for the axis element

The following methods are available on the `AxisElement` class:

- "getDimensionCount()" on page 871
- "getDimension()" on page 871
- "getTupleCount()" on page 871
- "getTuple()" on page 871
- "getIndex()" on page 872

AxisElement Constant Fields

The following table shows the constant fields used in the methods for the `AxisElement` class.

Field	Description
public static final String COLUMNS_AXIS="columns"	Constant for the name of the columns axis.
public static final String ROWS_AXIS ="rows"	Constant for the name of the rows axis.
public static final String PAGES_AXIS ="pages"	Constant for the name of the pages axis. (See Note below.)
public static final String CHAPTERS_AXIS="chapters"	Constant for the name of the chapters axis. (See Note below.)
public static final String SECTIONS_AXIS="sections"	Constant for the name of the sections axis. (See Note below.)

Note: Pages, chapters and sections are not valid axis names when using the CubeElement getAxis method with an IBM DB2 OLAP Server or Hyperion Essbase data source. With IBM DB2 OLAP Server or Hyperion Essbase data sources, you only have access to the row and column axes (and slicers). You

do not have access to the other axes, as you do with other data sources (i.e., Microsoft Analysis Services and DB2 Alphablox cubes).

getDimensionCount()

Get the number of dimensions.

Syntax

Java Method

```
int getDimensionCount();
```

getDimension()

Get the *n*th dimension of this axis. Returns `null` if it is not available.

Syntax

Java Method

```
DimensionElement getDimension(int n);
```

getTupleCount()

Get the number of tuples.

Syntax

Java Method

```
int  getTupleCount();
```

getTuple()

Gets the specified tuple. Returns `null` if the tuple is not available.

Syntax

Java Method

```
TupleElement getTuple(int n);
TupleElement getTuple(String memberNames);
TupleElement getTuple(String [] memberNames);
```

Usage

The `int n` form gets the *n*th tuple for this axis.

The `String memberNames` form gets the tuple element identified by the `memberNames` String. The `memberNames` String is a comma-separated string of unique (and case-sensitive) member names:

```
"QTR1, BUDGET"
```

The order of member names must match exactly the order of members in the tuple, and the number of members must match the number of dimensions on the axis.

The `String[] memberNames` form gets the tuple element identified by the array of strings, where every array element is a unique (and case-sensitive) member name. Return null if the tuple is not available. The order of the array elements must match exactly the order of members in the tuple, and the number of members must match the number of dimensions on the axis.

getIndex()

Get the index of the axis.

Syntax

Java Method

```
int getIndex();
```

TupleElement

Package	com.alphablox.blox.xml
Inherits	AbstractXmlElement
Description	The class for the tuple element

The following methods are available on the `TupleElement` class:

getMemberCount()

Get the number of members.

Syntax

Java Method

```
int getMemberCount();
```

getMember()

Get the *n*th member element of the tuple. Returns `null` if the element is not available.

Syntax

Java Method

```
MemberElement getMember(int n);
```

getindex()

Get the index of the tuple.

Syntax

Java Method

```
int getIndex ();
```

getAxis()

Get the tuple axis.

Syntax

Java Method

```
AxisElement getAxis();
```

DimensionElements

Package	com.alphablox.blox.xml
Inherits	AbstractDimensionElement
Description	The class for the dimension element

The following method is available on the `DimensionElements` class:

- getDisplayName()
- getIndex()
- getUniqueName()

getDisplayName()

Gets the display name of the element.

Syntax

Java Method

```
String getDisplayName();
```

getIndex()

Gets the index of the dimension relative to other dimensions in the same axis to which this dimension belongs.

Syntax

Java Method

```
int getIndex();
```

getUniqueName()

Gets the unique name of the element.

Syntax

Java Method

```
String getUniqueName();
```

DimensionsElement

Package	com.alphablox.blox.xml
Inherits	AbstractXMLElement
Description	The class for the dimensions element

The following methods are available on the `DimensionsElements` class:

- getDimension()
- getDimensionCount()

getDimension()

Get the Dimensions element at a specific position.

Syntax

Java Method

```
DimensionElement getDimension(int index);
```

getDimensionCount()

Returns the number of dimensions in an axis.

Syntax

Java Method

```
int getDimensionCount();
```

MemberElement

Package	com.alphablox.blox.xml
Inherits	AbstractMemberElement
Description	The class for the member element

The following methods are available on the `MemberElement` class:

MemberElement Constant Fields

The following table shows the constant fields used in the methods for the `MemberElement` class.

Field	Description
public static final String DRILL_DOWN ="dd"	Constant for the drill down method
public static final String DRILL_UP ="du"	Constant for the drill up method
public static final String PIVOT ="p"	Constant for the pivot method
public static final String KEEP_ONLY ="ko"	Constant for the keep only method
public static final String REMOVE_ONLY ="ro"	Constant for the remove only method
public static final String DATA_SORT ="ds"	Constant for the data sort method
public static final String SWAP_AXES="sw"	Constant for the swap axis method

getDimension()

Get the dimension to which this member belongs.

Syntax

Java Method

```
DimensionElement getDimension();
```

See Also

"DimensionElements" on page 873

getDisplayName()

Get the display name of the element.

Syntax

Java Method

```
String getDisplayName();
```

getGenerationLevel()

Get the member generation level.

Syntax

Java Method

```
int getGenerationLevel();
```

Usage

Returns an integer representing the member generation level.

getIndex()

Get the position of this member compared to other members in the parent tuple. The index is in the range of 0 and the count of members in the tuple.

Syntax

Java Method

```
int getIndex();
```

getIsLeaf()

Determine if the member is a leaf member with no children.

Syntax

Java Method

```
boolean getIsLeaf();
```

Usage

Returns `true` if it is a leaf member; `false` otherwise.

getSpan()

Get the value of the member span attribute (the number of attached members in the same tuple that have the same unique name).

Syntax

Java Method

```
int getSpan();
```

getSpanIndex()

Get the span index for the member element, where the index is the location of this member within the span of similar members. The index is in the range of 0 and the span attribute value minus one.

Syntax

Java Method

```
int getSpanIndex();
```

getTuple()

Get the member tuple.

Syntax

Java Method

```
TupleElement getTuple();
```

getUniqueName()

Get the unique name of the element.

Syntax

Java Method

```
String getUniqueName();
```

getURL()

Get the member URL.

Syntax

Java Method

```
String getURL(String baseUrl);
String getURL(String baseUrl, String method);
```

Usage

For the method name constants, see the table "MemberElement Constant Fields" on page 874.

Examples

Get the member URL (with no method attached). For example:

```
getURL("/main.jsp?render=dhtml");
```

returns:

```
/__PeerRequest__/main.jsp?render=html&AN=MyApp&BI=1&AX=0&DI=0&
IX=14&MN=
```

Get the member URL (with method name attached). For example:

```
getURL("/main.jsp?render=html", MemberElement.DRILL_DOWN);
```

returns:

```
__PeerRequest__/main.jsp?render=html&AN=MyApp&BI=1&AX=0&DI=0&IX=14&MN=dd
```

setURL()

Set the URL of the member element.

Syntax

Java Method

```
void setURL(String url);
```

where

url is the string representing the URL of the member.

AxisCells

Package	com.alphablox.blox.xml
Inherits	AbstractXmlElement
Description	The class for the cells element

The following method is available on the `AxisCells` class:

- getChildrenElement(int *n*)

getChildrenElement(int *n*)

Get the n-th child of the Node. Several methods of the CellElement Class simplify searching for cells. However, when there are more than five axes, the getChildElement method is the only way to search for cells.

Syntax

Java Methods

```
AxisCells getChildElement(int n);
```

Examples

These examples illustrate using the `getChildElement()` and `getCell()` methods to search for a cell.

Example 1:: An XML document has three dimensions (pages, rows, and columns) and the cells element labeled "cells". Search for the cell with page=3, row=2, column =5.

Using the `AxisCells getChildElement()` method:

```
CellsElement cells = doc.getCube().getCells();
AxisCells page = cells.getChildElement(3);
AxisCells row = page.getChildElement(2);
AxisCells column = row.getChildElement(5);
CellElement cell = (CellElement) column.getChildElement(0);
```

Using the `getCell()` method:

```
CellElement cell = doc.getCube().getCells().getCell(3,2,5);
```

Example 2:: An XML document has six dimensions: axis(5), chapters, sections, pages, rows, and columns, and the cells element labeled "cells". Search for the cell with axis(5)= 4, chapter=6, section= 1, page=3, row=2, column =5. (Line breaks are used in the example to ensure that the example fits within the margins of a printed page.)

Using the `getChildElement()` method:

```
CellsElement cells = doc.getCube().getCells();
CellElement cell = cells.getCellElement(4).getCellElement(6).
   getCellElement(1).getCellElement(3).getCellElement(2).
   getCellElement(5).getChildElement(0);
```

CellsElement

Package	com.alphablox.blox.xml
Inherits	AxisCells
Description	The class for the cells element

The following methods are available on the CellsElement class:

- getCell() (this is an overloaded method)

getCell()

Get the specified cell. Return null if the cell is not available or if an axis does not exist.

Syntax

Java Methods

```
CellElement getCell();
CellElement getCell(int col);
CellElement getCell(String colMemberNames);
CellElement getCell(int row, int col);
CellElement getCell(String rowMemberNames, String colMemberNames);
CellElement getCell(int page, int row, int col);
CellElement getCell(String pageMemberNames, String rowMemberNames,
                    String colMemberNames);
CellElement getCell(int section, int page, int row, int col);
CellElement getCell(String sectionMemberNames,
      String pageMemberNames,String rowMemberName,
      String colMemberName);
CellElement getCell(int chapter, int section, int page,int row,
                    int col);
CellElement getCell(String chapterMemberNames,
      String sectionMemberNames, String pageMemberNames,
      String rowMemberName, String colMemberName);
```

Usage

Returns null if the cell is not available. The method that takes no arguments get the cell when there are zero axes. The methods that take int arguments get the cell at the numbered location, the cells that take a String argument get the cell at the intersection of the specified axes members.

Examples

Member names are comma-separated strings:

```
getCell("USA, East, Cars")

getCell("USA, East, Cars", "QTR1, JAN");

getCell("Page1", "USA, East, Cars", "QTR1, JAN");

getCell("SectionX", "Page1","USA, East, Cars", "QTR1, JAN");

getCell("Chapter1","SectionX","Page1","USA, East, Cars",
         "QTR1, JAN");
```

CellElement

Package	com.alphablox.blox.xml
Inherits	AxisCells
Description	The class for the cell element

The following methods are available on the CellElement class:

- "getChildElement()" on page 879
- "getCoordinates()" on page 879
- "getDoubleValue()" on page 879
- "getIndex()" on page 879
- "getTuple()" on page 879
- "getValue()" on page 879

- "setCoordinates()" on page 880

getChildElement()

Get the child element with the specified index.

Syntax

Java Method

```
AxisCells getChildElement(int index);
```

where *index* is an integer representing the position of the desired child element.

getCoordinates()

Get the cell coordinates as a String.

Syntax

Java Method

```
int[] getCoordinates();
```

Usage

Returns an array of integers representing the cell coordinates. The length of the array is the same as the number of axes.

getDoubleValue()

Get the cell value as a Double; throw an exception if the value cannot be returned as a double.

Syntax

Java Method

```
double getDoubleValue()
       throws NumberFormatException;
```

getIndex()

Get the cell index for the specified axis index. If the axisIndex is for the rows axis, return the column number. If the axisIndex is for the columns axis, return the row number. If the axisIndex is invalid, return -1.

Syntax

Java Method

```
int getIndex(int axisIndex);
```

getTuple()

Get the tuple of the cell for the specified `axisIndex`.

Syntax

Java Method

```
TupleElement getTuple(int axisIndex);
```

getValue()

Get the cell value as a String.

Syntax

Java Method

```
String getValue();
```

setCoordinates()

Set the coordinates of the cell.

Syntax

Java Method

```
void setCoordinates(int[] coord);
```

where *coord* is the coordinates of the cell as an array of integers. The length of the array must be the same as the number of axes.

Appendix A. JSP Custom Tag Copy and Paste

This appendix contains versions of the custom tag libraries for each blox. You can use these versions to copy and paste into your JSP files and remove the attributes you do not need and fill in the appropriate values for the ones you do need.

AdminBlox JSP Custom Tag

The following shows the entire AdminBlox custom tag library. You can copy and paste this into your JSP files and remove the attributes you do not need and fill in the appropriate values for the ones you do need. You must fill in values for the attributes in your JSP file or the page will not compile.

```
<blox:admin
       id=""
       bloxName=""
/>
```

BookmarksBlox JSP Custom Tag

The following shows the entire BookmarksBlox custom tag library. You can copy and paste this into your JSP files and remove the attributes you do not need and fill in the appropriate values for the ones you do need. You must fill in values for all the attributes in your JSP file or the page will not compile.

For descriptions of each of the attributes, including the syntax for their values, see "BookmarksBlox Properties and Associated Methods" on page 145.

```
<blox:bookmarks
       id=""
       bloxName=""
/>
```

ChartBlox JSP Custom Tag

The following shows the entire ChartBlox custom tag library. You can copy and paste this into your JSP files and remove the attributes you do not need and fill in the appropriate values for the ones you do need. You must fill in values for all the attributes in your JSP file or the page will not compile.

For descriptions of each of the attributes, including the syntax for their values, see "ChartBlox Properties and Associated Methods" on page 207.

```
<blox:chart
       id=""
       absoluteWarning=""
       applyPropertiesAfterBookmark=""
       areaSeries=""
       autoAxesPlacement=""
       axisTitleStyle=""
       backgroundFill=""
       barSeries=""
       bloxEnabled=""
       bloxName=""
       bookmarkFilter=""
       chartAbsolute=""
       chartCurrentDimensions=""
       chartFill=""
       chartType=""
       columnLevel=""
       columnSelections=""
       comboLineDepth=""
       dataTextDisplay=""
       dataValueLocation=""
       depthRadius=""
       dwellLabelsEnabled=""
       filter=""
       footnote=""
       footnoteStyle=""
       formatProperties=""
       gridLineColor=""
       gridLinesVisible=""
       groupSmallValues=""
       height=""
       helpTargetFrame=""
       histogramOptions=""
       labelStyle=""
       legend=""
       legendPosition=""
       lineSeries=""
       lineWidth=""
       localeCode=""
       logScaleBubbles=""
       markerShape=""
       markerSizeDefault=""
       maxChartItems=""
       maximumUndoSteps=""
       menubarVisible=""
       mustIncludeZero=""
       noDataMessage=""
       o1AxisTitle=""
       pieFeelerTextDisplay=""
       quadrantLineCountX=""
       quadrantLineCountY=""
```

```
        quadrantLineDisplay=""
        removeAction=""
        render=""
        rightClickMenuEnabled=""
        riserWidth=""
        rowHeaderColumn=""
        rowLevel=""
        rowSelections=""
        rowsOnXAxis=""
        seriesColorList=""
        showSeriesBorder=""
        smallValuePercentage=""
        title=""
        titleStyle=""
        toolbarVisible=""
        totalsFilter=""
        useSeriesShapes=""
        visible=""
        width=""
        x1AxisTitle=""
        x1LogScale=""
        x1ScaleMax=""
        x1ScaleMaxAuto=""
        x1ScaleMin=""
        x1ScaleMinAuto=""
        XAxis=""
        XAxisTextRotation=""
        y1Axis=""
        y1AxisTitle=""
        y1FormatMask=""
        y1LogScale=""
        y1ScaleMax=""
        y1ScaleMaxAuto=""
        y1ScaleMin=""
        y1ScaleMinAuto=""
        y2Axis=""
        y2AxisTitle=""
        y2FormatMask=""
        y2LogScale=""
        y2ScaleMax=""
        y2ScaleMaxAuto=""
        y2ScaleMin=""
        y2ScaleMinAuto=""
>
</blox:chart>
```

Nested Tags Inside <blox:chart>

```
<blox:chart ...>
   <blox:footnoteStyle
      font=""
      foreground="" />
   <blox:labelStyle
      font=""
      foreground="" />
   <blox:seriesFill
      index=""
      value="" />
   <blox:titleStyle
      font=""
      foreground=""/>

   <blox:dial
      borderColor=""
      borderType=""
      color=""
```

```
      radius=""
      showLabels=""
      startAngle=""
      stopAngle=""
      ticPosition="">
      <blox:needle
         color=""
         endType=""
         endWidth=""
         needleWidth=""
         scope=""
         tooltip=""
         value="" />
      <blox:scale
         maximum=""
         minimum=""
         scope=""
         stepSize="" />
      <blox:sector
         color=""
         innerRadius=""
         outerRadius=""
         scope=""
         startValue=""
         stopValue=""
         tooltip="" />
   </blox:dial>

</blox:chart>
```

CommentsBlox JSP Custom Tag

The following shows the entire CommentsBlox custom tag library. You can copy and paste this into your JSP files and remove the attributes you do not need and fill in the appropriate values for the ones you do need. You must fill in values for all the attributes in your JSP file or the page will not compile.

For descriptions of each of the attributes, including the syntax for their values, see "CommentsBlox Properties and Methods by Category" on page 287.

```
<blox:comments
       id=""
       bloxName=""
       bloxRef=""
       dataSourceName=""
       collectionName=""
       userName=""
       password="" >
       <blox:sortComments
          field=""
          order="" />
</blox:comments>
```

ContainerBlox JSP Custom Tag

The following shows the entire ContainerBlox custom tag library. You can copy and paste this into your JSP files and remove the attributes you do not need and fill in the appropriate values for the ones you do need. You must fill in values for all the attributes in your JSP file or the page will not compile.

```
<blox:container
       id=""
       bloxName=""
       enablePoppedOut=""
       height=""
```

```
        poppedOut=""
        poppedOutHeight=""
        poppedOutTitle=""
        PoppedOutWidth=""
        render=""
        visible=""
        width=""
>
</blox:container>
```

DataBlox JSP Custom Tag

The following shows the entire DataBlox custom tag library. You can copy and paste this into your JSP files and remove the attributes you do not need and fill in the appropriate values for the ones you do need. You must fill in values for all the attributes in your JSP file or the page will not compile.

For descriptions of each of the attributes, including the syntax for their values, see "DataBlox Properties and Associated Methods" on page 334.

```
<blox:data
        id=""
        bloxRef=""
        aliasTable=""
        applyPropertiesAfterBookmark=""
        autoConnect=""
        autoDisconnect=""
        bloxName=""
        bookmarkFilter=""
        calculatedMembers=""
        catalog=""
        columnSort=""
        connectOnStartup=""
        dataSourceName=""
        dimensionRoot=""
        drillDownOption=""
        drillKeepSelectedMember=""
        drillRemoveUnselectedMembers=""
        enableKeepRemove=""
        enableShowHide=""
        hiddenMembers=""
        hiddenTuples=""
        leafDrillDownAvailable=""
        memberNameRemovePrefix=""
        memberNameRemoveSuffix=""
        mergedDimensions=""
        mergedHeaders=""
        onErrorClearResultSet=""
        parentFirst=""
        password=""
        performInAllGroups=""
        query=""
        retainSlicerMemberSet=""
        rowSort=""
        schema=""
        selectableSlicerDimensions=""
        showSuppressDataDialog=""
        suppressDuplicates=""
        suppressMissingColumns=""
        suppressMissingRows=""
        suppressNoAccess=""
        suppressZeros=""
        textualQueryEnabled=""
        useAASUserAuthorization=""
```

```
        useAliases=""
        useOlapDrillOptimization=""
        userName=""
</blox:data>
```

DataLayoutBlox JSP Custom Tag

The following shows the entire DataLayoutBlox custom tag library. You can copy and paste this into your JSP files and remove the attributes you do not need and fill in the appropriate values for the ones you do need. You must fill in values for all the attributes in your JSP file or the page will not compile.

For descriptions of each of the attributes, including the syntax for their values, see "DataLayoutBlox Properties/Tag Attributes" on page 458.

```
<blox:dataLayout
        id=""
        applyPropertiesAfterBookmark=""
        bloxEnabled=""
        bloxName=""
        bookmarkFilter=""
        height=""
        helpTargetFrame=""
        hiddenDimensionsOnOtherAxis=""
        interfaceType=""
        maximumUndoSteps=""
        noDataMessage=""
        render=""
        visible=""
        width=""  >
</blox:dataLayout>
```

GridBlox JSP Custom Tag

The following shows the entire GridBlox custom tag library. You can copy and paste this into your JSP files and remove the attributes you do not need and fill in the appropriate values for the ones you do need. You must fill in values for all the attributes in your JSP file or the page will not compile.

For descriptions of each of the attributes, including the syntax for their values, see "GridBlox Properties and Associated Methods" on page 546.

```
<blox:grid
        id=""
        applyPropertiesAfterBookmark=""
        autosizeEnabled=""
        bandingEnabled=""
        bloxEnabled=""
        bloxName=""
        bookmarkFilter=""
        columnWidths=""
        commentsEnabled=""
        defaultCellFormat=""
        drillThroughEnabled=""
        drillThroughWindow=""
        editableCellStyle=""
        editedCellStyle=""
        enablePoppedOut=""
        expandCollapseMode=""
        gridLinesVisible=""
        headingIconsVisible=""
        headingsEnabled=""
        height=""
        helpTargetFrame=""
```

```
        informationWindowName=""
        maximumUndoSteps=""
        menubarVisible=""
        missingValueString=""
        noAccessValueString=""
        noDataMessage=""
        paginate=""
        poppedOut=""
        poppedOutHeight=""
        poppedOutTitle=""
        PoppedOutWidth=""
        relationalRowNumbersOn=""
        removeAction=""
        render=""
        rightClickMenuEnabled=""
        rowHeadersWrapped=""
        rowHeadingWidths=""
        rowHeadingsVisible=""
        rowHeight=""
        rowIndentation=""
        showColumnDataGeneration=""
        showColumnHeaderGeneration=""
        showRowDataGeneration=""
        showRowHeaderGeneration=""
        toolbarVisible=""
        visible=""
        width=""
        writebackEnabled="" >
</blox:grid>
```

Nested Tags Inside <blox:grid>

```
<blox:grid ...>
   <blox:cellAlert
      index=""
      apply=""
      background=""
      condition=""
      description=""
      enabled=""
      font=""
      foreground=""
      format=""
      group=""
      link=""
      image_align=""
      image=""
      scope=""
      value=""
      value2="" />]
   <blox:cellEditor
      index=""
      scope="" />
   <blox:cellFormat
      index=""
      background=""
      font=""
      foreground=""
      format=""
      group=""
      scope="" />
   <blox:cellLink
      index=""
      description=""
```

```
      link=""
      scope=""
      image_align=""
      image="" />
   <blox:drillThroughWindow
      height=""
      locationbarVisible=""
      menubarVisible=""
      name=""
      resizable=""
      scrollbarVisible=""
      statusbarVisible=""
      toolbarVisible=""
      url=""
      width="" />
   <blox:editableCellStyle
      background=""
      font=""
      foreground="" />
   <blox:editedCellStyle
      background=""
      font=""
      foreground="" />
   <blox:formatMask
      index=""
      mask="" />
   <blox:formatName
      index=""
      name="" />
```

MemberFilterBlox JSP Custom Tag

The following shows the entire MemberFilterBlox custom tag library. You can copy and paste this into your JSP files and remove the attributes you do not need and fill in the appropriate values for the ones you do need. You must fill in values for all the attributes in your JSP file or the page will not compile.

For descriptions of each of the attributes, including the syntax for their values, see "MemberFilterBlox Properties and Associated Methods" on page 606.

```
<blox:memberFilter
       id=""
       applyButtonEnabled=""
       bloxEnabled=""
       bloxName=""
       dimensionSelectionEnabled=""
       height=""
       selectableDimensons=""
       selectedDimension=""
       visible=""
       width="" >
</blox:memberFilter>
```

PageBlox JSP Custom Tag

The following shows the entire PageBlox custom tag library. You can copy and paste this into your JSP files and remove the attributes you do not need and fill in the appropriate values for the ones you do need. You must fill in values for all the attributes in your JSP file or the page will not compile.

For descriptions of each of the attributes, including the syntax for their values, see "PageBlox Properties and Associated Methods" on page 613.

```
<blox:page
       id=""
       applyPropertiesAfterBookmark=""
       bloxEnabled=""
       bloxName=""
       bookmarkFilter=""
       fixedChoiceLists=""
       height=""
       helpTargetFrame=""
       maximumUndoSteps=""
       menubarVisible=""
       moreChoicesEnabled=""
       moreChoicesEnabledDefault=""
       noDataMessage=""
       render=""
       visible=""
       width="" >
</blox:page>
```

PresentBlox JSP Custom Tag

The following shows the entire PresentBlox custom tag library. You can copy and paste this into your JSP files and remove the attributes you do not need and fill in the appropriate values for the ones you do need. You must fill in values for all the attributes in your JSP file or the page will not compile.

For descriptions of each of the attributes, including the syntax for their values, see "PresentBlox Properties and Associated Methods" on page 623.

```
<blox:present
       id=""
       applyPropertiesAfterBookmark=""
       bloxEnabled=""
       bloxName=""
       chartAvailable=""
       chartFirst=""
       dataLayoutAvailable=""
       dividerLocation=""
       enablePoppedOut=""
       gridAvailable=""
       height=""
       helpTargetFrame=""
       maximumUndoSteps=""
       menubarVisible=""
       noDataMessage=""
       pageAvailable=""
       poppedOut=""
       poppedOutHeight=""
       poppedOutTitle=""
       PoppedOutWidth=""
       removeAction=""
       render=""
       splitPane=""
       splitPaneOrientation=""
       toolbarVisible=""
       visible=""
       width=""  >
</blox:present>
```

RepositoryBlox JSP Custom Tag

The following shows the entire RepositoryBlox custom tag library. You can copy and paste this into your JSP files and remove the attributes you do not need and fill in the appropriate values for the ones you do need. You must fill in values for all the attributes in your JSP file or the page will not compile.

For descriptions of each of the attributes, including the syntax for their values, see "RepositoryBlox Properties and Associated Methods" on page 638.

```
<blox:repository
       id=""
       bloxName=""
       render="">
</blox:repository>
```

ResultSetBlox JSP Custom Tag

The following shows the entire ResultSetBlox custom tag library. You can copy and paste this into your JSP files and remove the attributes you do not need and fill in the appropriate values for the ones you do need. You must fill in values for all the attributes in your JSP file or the page will not compile.

For descriptions of each of the attributes, including the syntax for their values, see "ResultSetBlox Properties and Associated Methods" on page 664.

```
<blox:resultSet
       id=""
       bloxName=""
       dataBlox=""
       resultSetHandler=""
/>
```

StoredProceduresBlox JSP Custom Tag

The following shows the entire StoredProceduresBlox custom tag library. You can copy and paste this into your JSP files and remove the attributes you do not need and fill in the appropriate values for the ones you do need. You must fill in values for all the attributes in your JSP file or the page will not compile.

For descriptions of each of the attributes, including the syntax for their values, see "StoredProceduresBlox Properties and Associated Methods" on page 675.

```
<blox:storedProcedures
       id=""
       bloxName=""
/>
```

ToolbarBlox JSP Custom Tag

The following shows the entire ToolbarBlox custom tag library. You can copy and paste this into your JSP files and remove the attributes you do not need and fill in the appropriate values for the ones you do need. You must fill in values for all the attributes in your JSP file or the page will not compile.

For descriptions of each of the attributes, including the syntax for their values, see "ToolbarBlox Properties and Associated Methods" on page 694.

```
<blox:toolbar
       id=""
       applyPropertiesAfterBookmark=""
```

```
        bloxEnabled=""
        bloxName=""
        bookmarkFilter=""
        helpTargetFrame=""
        removeAction=""
        removeButton=""
        rolloverEnabled=""
        textVisible=""
        toolTipsVisible=""
        visible=""
/>
```

Miscellaneous Tags in blox.tld

The following shows the remaining custom tag libraries. You can copy and paste this into your JSP files and remove the attributes you do not need and fill in the appropriate values for the ones you do need. You must fill in values for all the attributes in your JSP file or the page will not compile.

Display Tag

```
<blox:display
    bloxRef=""
    render=""
    width=""
    height="" />
```

Header Tag

```
<blox:header
    contextPath=""
    pageURL="" >
    <blox:clientBean name="" protected="">
        <blox:method name="" />
    </blox:clientBean>
</blox:header>
```

pdfReport and pdfDialogInput Tags

```
<blox:pdfReport
    header=""
    headerHeight=""
    footer=""
    footerHeight=""
    margin=""
    size=""
    theme=""
    themeListEnabled="" >
    <blox:pdfDialogInput
        index=""
        displayName=""
        defaultValue=""
    />
</blox:pdfReport>
```

Debug Tag

```
<blox:debug />
```

For some examples of using this tag, see “The Blox Debug Tag” in the *Developer’s Guide*.

Logo Tag

```
<blox:logo />
```

Session Tag

```
<blox:session
    key="" />
```

Blox Form-related Custom Tags

This section lists the tag attributes for:

CheckBoxFormBlox Tag

```
<bloxform:checkBox
   id=""
   bloxName=""
   checked=""
   checkedValue=""
   formElementName=""
   themeClass=""
   uncheckedValue=""
   visible=""
/>
```

CubeSelectFormBlox

```
<bloxform:cubeSelect
   id="cubes"
   id=""
   bloxName=""
   dataBlox=""
   dataBloxRef=""
   formElementName=""
   minimumWidth=""
   multiple=""
   selectedCube=""
   selectedCubeName=""
   size=""
   themeClass=""
   visible=""
/>
```

DataSourceSelectFormBlox

```
<bloxform:dataSourceSelect
   id=""
   bloxName=""
   adapter=""
   adminBloxRef=""
   formElementName=""
   minimumWidth=""
   nullDataSourceLabel=""
   selectedDataSourceName=""
   themeClass=""
   type=""
   visible=""
/>
```

DimensionSelectFormBlox

```
<bloxform:dimensionSelect
   id=""
   bloxName=""
   cube=""
   cubeName=""
   dataBlox=""
   dataBloxRef=""
   formElementName=""
   minimumWidth=""
   multiple=""
   selectedDimension=""
   selectedDimensionName=""
   size=""
   themeClass=""
   visible=""
/>
```

EditFormBlox

```
<bloxform:edit
   id=""
   bloxName=""
   charactersPerLine=""
   formElementName=""
   lines=""
   maskInput=""
   maxCharacters=""
   themeClass=""
   visible=""
/>
```

MemberSelectFormBlox

```
<bloxform:memberSelect
   id=""
   bloxName=""
   dataBlox=""
   dataBloxRef=""
   dimension=""
   dimensionName=""
   filterGeneration=""
   filterOperator=""
   formElementName=""
   minimumWidth=""
   multiple=""
   rootMemberName=""
   rootMemberNames=""
   rootMembers=""
   selectedMember=""
```

```
   selectedMemberName=""
   size=""
   themeClass=""
   visible=""
/>
```

RadioButtonFormBlox

The <bloxform:radioButton> tag can have multiple <bloxform:button> tags.

```
<bloxform:radioButton
   id=""
   bloxName=""
   align=""
   borderEnabled=""
   formElementName=""
   themeClass=""
   visible="">

   <bloxform:button
      label=""
      object=""
      selected=""
      value=""
   />

</bloxform:radioButton>
```

SelectFormBlox

The <bloxform:select> tag can have multiple <bloxform:option> tags.

```
<bloxform:select
   id=""
   bloxName=""
   formElementName=""
   minimumWidth=""
   multiple=""
   size=""
   themeClass=""
   visible=""
>
   <bloxform:option
      label=""
      object=""
      selected=""
      value=""
   />

</bloxform:select>
```

TimePeriodSelectFormBlox

```
<bloxform:timePeriodSelect
   id=""
   bloxName=""
   defaultSeriesVisible=""
   formElementName=""
   minimumWidth=""
   selectedSeries=""
   selectedSeriesString=""
   themeClass=""
   timeSchemaBloxRef=""
   visible=""
>
   <bloxform:timeSeries
      expression=""
```

```
      name=""
   />

</bloxform:timePeriodSelect>
```

TimeUnitSelectFormBlox

```
<bloxform:timeUnitSelect
   id=""
   bloxName=""
   formElementName=""
   minimumWidth=""
   multiple=""
   selectedTimeUnit=""
   size=""
   themeClass=""
   timeSchemaBloxRef=""
   visible=""
/>
```

TreeFormBlox

The <bloxform:tree> tag has two nested tags for folders and items. There can be multiple <bloxform:folder> tags and <bloxform:item> tags in a <bloxform:tree> tag.

```
<bloxform:tree
   id=""
   bloxName=""
   draggingEnabled=""
   itemPositoningEnabled=""
   rootVisible=""
   textWrapped=""
   themeClass=""
   visible=""
>
   <bloxform:folder> <%--root folder--%>

      <bloxform:folder
         label=""
         draggable=""
         expanded=""
         href=""
         label=""
         name=""
         object=""
         target=""
         tooltip=""
      >
         <bloxform:item
            draggable=""
            href=""
            label=""
            name=""
            object=""
            target=""
            tooltip="" />

      </bloxform:folder>

   </bloxform:folder>
</bloxform:tree>
```

The <bloxform:getChangedProperty> Tag

```
<bloxform:getChangedProperty
   debugEnabled=""
   formBlox=""
   formBloxRef=""
   formProperty=""
   property=""
/>
```

The <bloxform:setChangedProperty> Tag

```
<bloxform:setChangedProperty
   callAfterChange=""
   debugEnabled=""
   formProperty=""
   target=""
   targetRef=""
   targetProperty=""
/>
```

Blox Logic Custom Tags

The Blox Logic Tag Library has tags for the following Blox:

- "MDBQueryBlox" on page 896
- "MemberSecurityBlox" on page 897
- "TimeSchemaBlox" on page 897

MDBQueryBlox

Tags for MDBQueryBlox has a nesting structure. The nesting structure may vary depending on application needs. For details, see "MDBQueryBlox Tags" on page 746. The following shows the general structure:

```
<bloxlogic:mdbQuery
   id=""
   dataBloxRef=""
   cubeName="" >
   <bloxlogic:axis
      mutable=""
      queryFragment=""
      type="" >
      <bloxlogic:tupleList>
         <bloxlogic:dimension>
            [specify the dimension name here]
         </bloxlogic:dimension>
         <bloxlogic:tuple>
            <bloxlogic:member>
               [specify the member here]
            </bloxlogic:member>
            <bloxlogic:member>
             [specify another member here]
            </bloxlogic:member>
            ...
         </bloxlogic:tuple>
      </bloxlogic:tupleList>
   </bloxlogic:axis>
</bloxlogic:mdbQuery>
```

The `<bloxlogic:tupleList>` tag can also stand alone without being nested:

```
<bloxlogic:tupleList
   id=""
   tuplesRef="" />
```

The <bloxlogic:dimension> tag has one attribute:

```
<bloxlogic:dimension
   list="" />
```

The <bloxlogic:tuple> tag has one attribute:

```
<bloxlogic:tuple
   list="" />
```

The <bloxlogic:crossJoin> tag is a nested tag inside the <bloxlogic:axis> tag. It has no attribute. The <bloxlogic:member> tag also has no attributes.

MemberSecurityBlox

```
<bloxlogic:memberSecurity
   id=""
   cubeName=""
   dataBlox=""
   dataBloxRef=""
   dimensionName="" >
   <bloxlogic:memberSecurityFilter
      dimensionName=""
      memberName="" />
   <bloxlogic:memberSecurityFilter
      dimensionName=""
      memberName="" />
</bloxlogic:memberSecurity>
```

TimeSchemaBlox

```
<bloxlogic:timeSchema
   id=""
   dataBloxRef=""
   name=""
   today=""
/>
```

Blox UI Custom Tags

To use the Blox UI modifier custom tags, import bloxui.tld as follows:

```
<%@ taglib uri="bloxuitld" prefix="bloxui"%>
```

The modifier custom tags include:

- Component customization tags
  - "The <bloxui:calculationEditor> Tag" on page 898
  - "The <bloxui:component> Tag" on page 899
  - "Custom Menu Tags" on page 901
  - "Custom Toolbar Layout Tag" on page 901
- Custom analysis tags
  - "The <bloxui:bottomN> Tag" on page 898
  - "The <bloxui:customAnalysis> Tag" on page 899
  - "The <bloxui:eightyTwenty> Tag" on page 899
  - "The <bloxui:percentOfTotal> Tag" on page 900
  - "The <bloxui:topN> Tag" on page 900
- Custom layout tags
  - "The <bloxui:butterflyLayout> Tag" on page 898

The <bloxui:actionFilter> Tag

```
<bloxui:actionFilter
   componentName=""
   filter="" />
```

The <bloxui:bottomN> Tag

```
<!--Nested within a PresentBlox or a GridBlox-->

<bloxui:bottomN
   description=""
   hideOthers=""
   membersToAnalyze=""
   number=""
   preserveGrouping=""
   prompt=""
   showOtherSummary=""
   showRank=""
/>
```

The <bloxui:butterflyLayout> Tag

```
<!--Nested within a PresentBlox, GridBlox, or ChartBlox -->

<bloxui:butterflyLayout
   addSeparatorColumns=""
   applyLayout=""
   description=""
   position=""
   scope=""
   separatorWidth=""
   showOnLayoutMenu="" />
```

The <bloxui:calculationEditor> Tag

This tag has no attributes.

The <bloxui:clientLink> Tag

```
<!--Nested within a component customization tag-->

<bloxui:clientLink
   features=""
   link=""
   target=""/>
```

The <bloxui:component> Tag

```
<!--Nested within a PresentBlox, GridBlox, or ChartBlox -->

<bloxui:component
   alignment=""
   bloxRef=""
   clickable=""
   disabled=""
   height=""
   name=""
   positionBefore=""
   style=""
   themeClass=""
   title=""
   tooltip=""
   valignment=""
   visible=""
   width="" >

      <bloxui:clientLink
         link=""
         target="" />

</bloxui:component>
```

The <bloxui:compressLayout> Tag

```
<!--Nested within a PresentBlox or a GridBlox-->

<bloxui:compressLayout
   applyLayout=""
   compressColumns=""
   compressRows=""
   description=""
   memberSeparator=""
   showOnLayoutMenu="" />
```

The <bloxui:customAnalysis> Tag

```
<!--Nested within a PresentBlox, GridBlox, or ChartBlox -->

<bloxui:customAnalysis
   analysis="" />
```

The <bloxui:customLayout> Tag

```
<!--Nested within a PresentBlox, GridBlox, or ChartBlox -->

<bloxui:customLayout
   applyLayout=""
   layout=""
   showOnLayoutMenu=""/>
```

The <bloxui:eightyTwenty> Tag

```
<!--Nested within a PresentBlox or a GridBlox-->

<bloxui:eightyTwenty
   description=""
   hideOthers=""
   membersToAnalyze=""
   number=""
   preserveGrouping=""
   prompt=""
/>
```

The <bloxui:gridFilter> Tag

```
<!--Nested within a PresentBlox or a GridBlox-->

<bloxui:gridFilter
   filter="" />
```

The <bloxui:gridHighlight> Tag

```
<!--Nested within a PresentBlox or a GridBlox-->

<bloxui:gridHighlight
   applyLayout=""
   description=""
   includeData=""
   includeHeaders=""
   scope=""
   selection=""
   showOnLayoutMenu=""
   style="" />
```

The <bloxui:gridSpacer> Tag

```
<!--Nested within a PresentBlox or a GridBlox-->

<bloxui:gridSpacer
   applyLayout=""
   axis=""
   description=""
   height=""
   locked=""
   position=""
   scope=""
   showOnLayoutMenu=""
   style=""
   width="" />
```

The <bloxui:percentOfTotal> Tag

```
<!--Nested within a PresentBlox or a GridBlox-->

<bloxui:percentOfTotal
   description=""
   hideOthers=""
   membersToAnalyze=""
   number=""
   preserveGrouping=""
   prompt=""
/>
```

The <bloxui:topN> Tag

```
<!--Nested within a PresentBlox or a GridBlox-->

<bloxui:topN
   description=""
   hideOthers=""
   membersToAnalyze=""
   number=""
   preserveGrouping=""
   prompt=""
   showOtherSummary=""
   showRank=""
/>
```

The <bloxui:setProperty> Tag

```
<!--Nested within a PresentBlox, GridBlox, or ChartBlox -->

<bloxui:setProperty
   name=""
   value="" />
```

The <bloxui:title> Tag

```
<!--Nested within a PresentBlox, GridBlox, or ChartBlox -->

<bloxui=title
   title=""
   style=""
   alignment=""/>
```

Custom Menu Tags

Tags for custom menu layouts include <bloxui:menu> and <bloxui:menuItem>:

The <bloxui:menu> and the <bloxui:menuItem> Tags

```
<!--Nested within a PresentBlox, GridBlox, or ChartBlox -->

<bloxui:menu
   name=""
   bloxRef=""
   disabled=""
   positionBefore=""
   resourceName=""
   title=""
   tooltip=""
   visible=""
   >
   <bloxui:menuItem
      name=""
      checkable=""
      checked=""
      disabled=""
      imageURL=""
      positionBefore=""
      separator=""
      themeBasedImage=""
      title=""
      tooltip=""
      visible=""
      >
      <bloxui:clientLink
         link=""
         target="" />
   </bloxui:menuItem>
</bloxui:menu>
```

Custom Toolbar Layout Tag

Tags for custom Toolbar layouts include <bloxui:toolbar> and <bloxui:toolbarButton>:

The <bloxui:toolbar> and Its Nested <bloxui:toolbarButton> Tags

```
<!--Nested within a PresentBlox, GridBlox, or ChartBlox -->

<bloxui:toolbar
   disabled=""
   bloxRef=""
   name=""
```

```
   positionBefore=""
   resourceName=""
   title=""
   tooltip=""
   visible="">
   <bloxui:toolbarButton
      checkable=""
      checked=""
      disable=""
      imageURL=""
      name=""
      positionBefore=""
      separator=""
      themeBasedImage=""
      title=""
      tooltip=""
      visible="" >
      <bloxui:clientLink
         link=""
         target="" />
   </bloxui:toolbarButton>
</bloxui:toolbar>
```

Relational Reporting Blox Custom Tags

Please see *Relational Reporting Developer's Guide* for custom tags for Blox supporting relational reporting.

Appendix B. DB2 Alphablox XML Cube DTD

Similar to a database schema, a *Document Type Definition* (DTD) defines the data structures that can occur in a document and the sequence in which they can occur.

Knowing a document's structure enables programmers to write code that can traverse the document, extract specific values from it, and perform operations on the values.

DTD Syntax Notes

The following markup appears in the DB2 Alphablox XML Cube DTD:

Element Type Declaration

Names an element and specifies its children.

Syntax

```
<!ELEMENT name (childElement1, childElement2,...childElementN)>
```

Usage

A regular expression character (+, *, or ?) appended to the name of a child element specifies how many of that child element the parent can contain. The absence of one of these characters indicates that the parent element can contain one and only one of that child element.

+	The parent element can contain one or more of the named child element
*****	The parent element can contain zero or more of the named child element
?	The parent element can contain zero or one of the named child element

For example, a line like the following:

```
<!ELEMENT data (slicer*, axis+, cells)>
```

would specify that the data element can have zero or more slicer elements, one or more axis elements, and only one cells element.

Attribute List Declaration

Names an element and specifies its attributes. For each attribute, specifies its name, data type, and required or optional presence.

Syntax

```
<!ATTLIST element-name
        childElementName1 dataType #REQUIRED
        childElementName2 dataType #REQUIRED
        childElementNameN dataType #REQUIRED
>
```

Usage

For example, the following lines:

```
<!ATTLIST dimension
       name CDATA #REQUIRED
       index CDATA #REQUIRED
>
```

specify that the dimension member has two required attributes (name and index), both of which contain plain character data.

Data Types

Associated with an element name, specifies the type of data permitted in the element.

Usage

The DB2 Alphablox XML Cube DTD uses the following two data types:

#PCDATA (Parsed Character Data): plain (non-markup) text that may contain entity references. For example, the string & should be parsed to yield an ampersand sign.

CDATA (Character Data): plain (non-markup) text that does not include entity references. For example, a less-than sign (<), quotation marks (“), or an ampersand (&) are treated as plain text and not parsed.

Tip: A complete discussion of DTD syntax is beyond the scope of this guide.

DTD Elements

For an explanation of each DTD Element that appears in the listing, see “Alphablox XML Tags” on page 858.

DTD Listing

The remainder of this section is a listing of the DB2 Alphablox XML Cube DTD.

```
<!ELEMENT cube (bloxInfo, data)>

<!ELEMENT bloxInfo (bloxID, bloxName, appName)>

<!ELEMENT bloxID (#PCDATA)>
<!ELEMENT bloxName (#PCDATA)>
<!ELEMENT appName (#PCDATA)>

<!ELEMENT data (slicer*, axis*, cells)>

<!ELEMENT slicer  (slicerDimension, slicerMember)>

<!ELEMENT slicerDimension (#PCDATA)>

<!ATTLIST slicerDimension
        name CDATA #REQUIRED
>

<!ELEMENT slicerMember (#PCDATA)>

<!ATTLIST slicerMember
      name CDATA #REQUIRED
      gen CDATA #REQUIRED
      leaf CDATA #REQUIRED
```

```
>

<!ELEMENT axis (dimensions,tuple*)>

<!ATTLIST axis
       name CDATA #REQUIRED
       index CDATA #REQUIRED
>

<!ELEMENT dimensions (dimension*)>

<!ELEMENT tuple (member*)>

<!ATTLIST tuple
        index CDATA #REQUIRED
>

<!ELEMENT dimension (#PCDATA)>

<!ATTLIST dimension
       name CDATA #REQUIRED
       index CDATA #REQUIRED
>

<!ELEMENT member (#PCDATA)>

<!ATTLIST member
       name CDATA #REQUIRED
       gen CDATA #REQUIRED
       span CDATA #REQUIRED
       spanIndex CDATA #REQUIRED
       spanInHierarchy CDATA #REQUIRED
       spanIndexInHierarchy CDATA #REQUIRED
       index CDATA #REQUIRED
       leaf CDATA #REQUIRED
>

<!-- for zero axis, we have cell value only -->
<!ELEMENT cells (axisCells* | section* | chapter* | page*| row*| column*|
                cell)>
<!ELEMENT axisCells (axisCells+ | section+)><!ATTLIST axisCells
       name CDATA #REQUIRED
>

<!ELEMENT section (chapter+)>
<!ELEMENT chapter (page+)>
<!ELEMENT page (row+)>
<!ELEMENT row (column+)>
<!ELEMENT column (cell)>
<!ELEMENT cell (#PCDATA)>
```

Appendix C. Examples Cross References

This chapter contains the additional code examples and cross references to examples in this book:

Category	Example
BookmarksBlox	• "Example 1: Getting a count of all bookmarks" on page 132 • "Example 2: Getting the properties set for a Bookmark" on page 132 • "Example 3: Getting a list of bookmarks that match the specified criteria" on page 134 • "Example 4: Creating a bookmark using BookmarksBlox API" on page 135 • "Example 5: Using server-side bookmarkLoad event filter" on page 136 • "Example 6: Getting a bookmark's query when it is loaded" on page 137
Blox Form Tag Library and FormBlox	• "A CheckBoxFormBlox Example" on page 706 • "A CubeSelectFormBlox Example" on page 708 • "A DataSourceSelectFormBlox Example" on page 710 • "DimensionSelectFormBlox Examples" on page 713 • "An EditFormBlox Example" on page 714 • "A MemberSelectFormBlox Example" on page 718 • "A RadioButtonFormBlox Example" on page 719 • "A SelectFormBlox Example" on page 722 • "A TimePeriodSelectFormBlox Example" on page 726 • "A TreeFormBlox Example" on page 731

Category	Example
Business Logic Blox and the Blox Logic Tag Library	• "An CrossJoin Example" on page 747 • "An MDBQueryBlox Example" on page 748 • "A MemberSecurityBlox Example" on page 761 • "A TimeSchemaBlox Example" on page 769 • "An Sample TimeSchema for an IBM DB2 OLAP Server or Hyperion Essbase Data Source" on page 787 • "An Sample TimeSchema for an Microsoft Analysis Services Data Source" on page 787
Blox UI Tag Library	• "Component Tag Examples" on page 794 • "bottomN Tag Examples" on page 799 • "Menu Tag Examples" on page 821 • "Toolbar Tags Examples" on page 827 • "Toolbar Tags Examples" on page 827
ChartBlox	• "Example 2: Set Chart Properties on the Server Using the bloxAPI.call() Method" on page 911 • Dynamically setting chart type using a SelectFormBlox: "A SelectFormBlox Example" on page 722
CommentsBlox	• "Example 1: Enabling cell commenting" on page 284 • "Example 2: Specifying Field to Sort On and Sort Order" on page 284 • "Example 3: Accessing Cell Comments Using MDBResultSet" on page 285 • "Example 4: Adding a CommentAddedEvent Listener" on page 286
Data Calculation	• "Example 1: Walk Through a Relational Result Set" on page 909 • "Example 1: Adding a calculated member named Profit at the end of the Measures dimension" on page 348 • "Example 2: Specifying the position of the calculated member" on page 349 • "Example 3: Adding a generation number and scope" on page 349 • "Example 4: Replacing missing or null values with the value 0" on page 349 • "Example 5: Calculations involving members from different dimensions" on page 350 • "Example 6: Adding ranking" on page 350 • "Example 7: Adding a separate ranking within each group" on page 351 • "Example 8: Adding running totals within each group" on page 352
Event Filters	• "A Complete drillDownEventFilter Example" on page 465 • "Example 3: Use the server-side ChartPageListener to set the desired data format on the chart when the chart filter is changed" on page 912 • "Example 5: Using server-side bookmarkLoad event filter" on page 136
JDBCConnection Bean	• "JDBCConnection Bean JSP useBean Examples" on page 597
MemberFilterBlox	• "Example 1: Filtering Members for All Available Dimensions" on page 605 • "Example 2: Filtering Members for Specified Dimensions Only" on page 605 • "Example 3: Filtering Members for One Dimension Only" on page 606

Category	Example
StoredProceduresBlox	• "Example 1: Connecting to the data source without a DataBlox" on page 670 • "Example 2: Using the StoredProceduresBlox to connect the data source for use with DataBlox" on page 670 • "Example 3: Getting a list of stored procedures whose name matches a specified pattern" on page 670 • "Example 4: Getting a list of all parameters for each stored procedure" on page 671 • "Example 5: Executing a stored procedure that has one input parameter and two output parameters" on page 672 • "Example 6: Setting a stored procedure result set to a DataBlox" on page 672

Example 1: Walk Through a Relational Result Set

```
<%--  RDBResultSet.jsp
----  Example page to illustrate RDB ResultSet Methods
----
----  Walk a server-side RDB ResultSet and output the column
---   metadata information and the first and last rows of data.
 --%>

<%-- Import the Alphablox taglib --%>
<%@ taglib uri="bloxtld" prefix="blox" %>
<%-- Import the packages for accessing the server-side RDBResultSet--%>
<%@ page import="com.alphablox.blox.data.rdb.*" %>

<%-- creates sqlTypes variable & imports java.sql.Types & java.util.Hashtable--
%>
<%@ include file="SQLTypes.jsp"%>

<blox:data id="relationalDB"
  dataSourceName = "qcc-mssql"
  query = "SELECT * FROM qcc WHERE Sales > 9000 ORDER BY Week_Ending,
Product_Family_Code"
>
</blox:data>
<%
  RDBResultSet rs = (RDBResultSet) relationalDB.getResultSet();

  // Get the schema details
  ResultColumn[] cols       = rs.getColumns(); // column Metadata
  int[]          types      = rs.getTypes();
                              // jdbc/sql data types in the rs
  int            colCount   = cols.length;
                              // num cols in result set

  // each row of data is returned as an array of objects; use types
  // to determine their data types
  Object[]       firstRow   = null;
  Object[]       lastRow      = null;
  int            rowsRead   = 0;

  // iterate through the rows incrementing the row counter and
  // saving the first and last rows of data
  while( rs.hasMoreRows() )
  {
    rowsRead++;
    if( rowsRead == 1 )
    {
      firstRow  = rs.getNextRow( false );
    }
    lastRow = rs.getNextRow( false );
  }
```

```
%>

<html>
<head>
    <title>Relational JSP</title>
    <blox:header/><%-- Blox header tag for standard js and style inclusions --%>
</head>

<body>
<br>
The column count is: <b><%= colCount %></b><br />
The row count is: <b><%= rowsRead %></b><br />
The columns are: <br />

<table border="1" cellspacing="0" cellpadding="0">
  <tr>
    <th>Col Name</th>
    <th>Type</th>
    <th>Type Name</th>
    <th>First Row</th>
    <th>Last Row</th>
  </tr>
  <%
    // Display the column names, their types, typeNames, and
    // the first and last row of data
  for( int i = 0; i < colCount; i++ )
  {
    out.write("\t<tr>");
    out.write("\t\t<td>" + cols[i].getName() + "</td>" ); // Col Name
    out.write("\t\t<td>" + String.valueOf(types[i]) + "</td>"); // Type
    // the names of the sql types is in the sqlTypes hashtable created in
SQLTypes.jsp
    out.write("\t\t<td>" + String.valueOf( sqlTypes.get( new Integer( types[i]
) ) ) + "</td>" ); // Type Name
    out.write("\t\t<td>" + String.valueOf(firstRow[i]) + "</td>"); //First Row
    out.write("\t\t<td>" + String.valueOf(lastRow[i]) + "</td>"); //Last Row
    out.write("\t</tr>");
  }
%>
</table>
</body>
</html>
```

The following code is the SQLTypes.jsp file referenced in the JSP file above:

```
<!-- SQLTypes.jsp
---- Helper page to create a hashtable with all of the SQL data types
---- 2002.03.28  -  YRL & REK
----
 -->

<!-- Imports for standard Java classes used -->
<%@ page import="java.sql.Types.*" %>
<%@ page import="java.util.Hashtable" %>

<%
    Hashtable sqlTypes = new Hashtable();

    sqlTypes.put( new Integer( java.sql.Types.ARRAY ), "ARRAY" );
    sqlTypes.put( new Integer( java.sql.Types.BIGINT ), "BIGINT" );
    sqlTypes.put( new Integer( java.sql.Types.BINARY ),"BINARY" );
    sqlTypes.put( new Integer( java.sql.Types.BIT ), "BIT" );
    sqlTypes.put( new Integer( java.sql.Types.BLOB ), "BLOB" );
    sqlTypes.put( new Integer( java.sql.Types.CHAR ), "CHAR" );
    sqlTypes.put( new Integer( java.sql.Types.CLOB ), "CLOB" );
    sqlTypes.put( new Integer( java.sql.Types.DATE ), "DATE" );
    sqlTypes.put( new Integer( java.sql.Types.DECIMAL ), "DECIMAL" );
```

```
    sqlTypes.put( new Integer( java.sql.Types.DISTINCT ), "DISTINCT" );
    sqlTypes.put( new Integer( java.sql.Types.DOUBLE ), "DOUBLE" );
    sqlTypes.put( new Integer( java.sql.Types.FLOAT ), "FLOAT" );
    sqlTypes.put( new Integer( java.sql.Types.INTEGER ), "INTEGER" );
    sqlTypes.put( new Integer( java.sql.Types.JAVA_OBJECT ) ,
                            "JAVA_OBJECT"   );
    sqlTypes.put( new Integer( java.sql.Types.LONGVARBINARY ),
                            "LONGVARBINARY" );
    sqlTypes.put( new Integer( java.sql.Types.LONGVARCHAR ),
                            "LONGVARCHAR"   );
    sqlTypes.put( new Integer( java.sql.Types.NULL ), "NULL" );
    sqlTypes.put( new Integer( java.sql.Types.NUMERIC ), "NUMERIC" );
    sqlTypes.put( new Integer( java.sql.Types.OTHER ), "OTHER" );
    sqlTypes.put( new Integer( java.sql.Types.REAL ), "REAL" );
    sqlTypes.put( new Integer( java.sql.Types.REF ), "REF" );
    sqlTypes.put( new Integer( java.sql.Types.SMALLINT ), "SMALLINT" );
    sqlTypes.put( new Integer( java.sql.Types.STRUCT ), "STRUCT" );
    sqlTypes.put( new Integer( java.sql.Types.TIME ), "TIME" );
    sqlTypes.put( new Integer( java.sql.Types.TIMESTAMP ),
                            "TIMESTAMP");
    sqlTypes.put( new Integer( java.sql.Types.TINYINT ), "TINYINT" );
    sqlTypes.put( new Integer( java.sql.Types.VARBINARY ),"VARBINARY");
    sqlTypes.put( new Integer( java.sql.Types.VARCHAR), "VARCHAR");
%>
```

Example 2: Set Chart Properties on the Server Using the bloxAPI.call() Method

```
<%--  chartSelect.jsp
----  Example page to illustrate how to use the call method to
----  execute some server-side code.
--%>

<!-- Import the taglib -->
<%@ taglib uri = "bloxtld" prefix = "blox"%>
<html>
<head>
   <title>Change Repository Values</title>
   <blox:header />
</head>

<!-- The JavaScript function that passes the chart type selected and
---- calls another JSP page (setSelection.jsp) on the server to set
---- the chart type.
--%>

<script language="JavaScript">
   function setChartChoice(ChrtType) {
      bloxAPI.call("setSelection.jsp?chart="+ChrtType);
   }
</script>

<body>
<blox:present id = "myPresent"
   height = "400"
   width = "600"
   >

   <blox:chart
      chartType = "Vertical Bar, Side-by-Side">
   </blox:chart>
   <blox:data
      dataSourceName = "QCC-Essbase"
      useAliases = "true"
      query = "<ROW ('All Products')  <ICHILD 'All Products' <SYM
         <COLUMN ('All Time Periods') <Ichild '2001' !"
```

```
   >
   </blox:data>
</blox:present>
<br>
Select a chart type:
<form name = form1>
   <input type="radio" name="chartSelect" value="Bar"
      onclick="setChartChoice(value);"> Bar
   <input type="radio" name="chartSelect" value="Line"
      onclick="setChartChoice(value);"> Line
   <input type="radio" name="chartSelect" value="Pie"
      onclick="setChartChoice(value);"> Pie
</form>
</body>
</html>
```

The JSP file called has the following code:

```
<%--  setSelection.jsp
----  Called by chartSelect.jsp to set the chart type to the one
----  selected.
--%>

<!-- Import the taglib -->
<%@ taglib uri="bloxtld" prefix="blox"%>

<%-- Reference the instance of PresentBlox created in chartSelect.jsp
--%>

<blox:present id="myPresent" />
<%
String chartChoice = request.getParameter("chart");
if (chartChoice != null && chartChoice.trim().length() != 0) {
   myPresent.getChartBlox().setChartType(chartChoice);
   };
%>
```

Example 3: Use the server-side ChartPageListener to set the desired data format on the chart when the chart filter is changed

This example demonstrates how to use the server-side event listener to set the format for Y1Axis (`setY1FormatMask()`) in order to maintain the correct formatting that has been set in the GridBlox when users change the filter in the chart. In this case:

- The Scenario dimension is on the page axis. Both Actual and Variance % have cell formats specified.
- Using the `addEventListener()` method, specify the instance of a ChartPageListener object (`CPListener()`) to call when users change the filter on the chart.
- `CPListener` implements the `ChartPageListener` interface.
- Get the member selected. In this particular example, since the member Variance % has a "%" in the name, we test to see the return string ends with "%". If it does, we set the `Y1FormatMask` accordingly.

```
<%@ page import="com.alphablox.blox.ChartBlox,
                 com.alphablox.blox.event.*,
                 com.alphablox.blox.uimodel.core.MessageBox,
                 com.alphablox.blox.uimodel.BloxModel,
                 com.alphablox.blox.ServerBloxException"%>
<%@ taglib uri="bloxtld" prefix="blox"  %>
<html>
<head>
   <blox:header />
```

```
</head>
<body>
<blox:present id="present" height="400" width="600" >
   <blox:grid defaultCellFormat="#,##0.00;[red](#,##0.00)" >
      <blox:cellFormat index="1" format="#,##0.00;[red](#,##0.00)"
         scope="{Scenario:Actual}" ></blox:cellFormat>
      <blox:cellFormat index="2" format="#,##0.00%;[red](#,##0.00%)"
         scope="{Scenario:Variance %}" ></blox:cellFormat>
   </blox:grid>
   <blox:chart chartType="bar" autoAxesPlacement="false"
      filter="Scenario" XAxis="All Time Periods" legend="All Locations" >
   </blox:chart>
   <blox:data dataSourceName="QCC-Essbase"
      query="{OUTALTNAMES} <ROW (\"All Locations\") <ICHILD \"All Locations\"
<COLUMN (\"All Time Periods\", \"Scenario\") <SYM \"Jan 01\" \"Feb 01\"
\"Mar 01\" \"Apr 01\" \"May 01\" \"Jun 01\" \"Actual\" \"Variance %\" !">
   </blox:data>
   <%-- adds a ChartPageFilter to the ChartBlox --%>
   <% present.getChartBlox().addEventListener(new
CPListener(present.getBloxModel()));%>

</blox:present>
</body>
</html>

<%!
  public class CPListener implements ChartPageListener
  {
    BloxModel model;
    public CPListener (BloxModel model) {
       this.model = model;
    }
    public void changePage(ChartPageEvent cpe) {
      ChartBlox blox = cpe.getChartBlox();
      try {
        if (cpe.getSelection().endsWith("%")) {
          String msg = new String("Setting format mask to be a percentage");
          blox.setY1FormatMask("#%");
           MessageBox msgBox = new MessageBox(msg, "Chart Page Filter",
MessageBox.MESSAGE_OK, null);
          model.getDispatcher().showDialog(msgBox);
        }
        else {
          String msg = new String("Setting format mask to be currency");
          blox.setY1FormatMask("$#K");
          MessageBox msgBox = new MessageBox(msg, "Chart Page Filter",
MessageBox.MESSAGE_OK, null);
          model.getDispatcher().showDialog(msgBox);        }
      } catch (ServerBloxException e) {
        e.printStackTrace();
      }
    }
  }
%>
```

Appendix D. Deprecated APIs

This section lists deprecated properties, methods, classes, and/or URL attributes, the release in which they were deprecated, and the replacements for the deprecated functionality.

Deprecated APIs receive support for a limited time but are no longer a part of strategic product direction. Alphablox recommends eliminating their use as soon as possible. Unless explicitly stated otherwise, a deprecated API receives support for three major releases, including the one in which the release notes announced its deprecation. Major releases are, for example, 3.0.0 or 3.5.0. Minor releases are, for example, 3.0.1.

Warning messages appear in the browser console whenever DB2 Alphablox encounters a deprecated API. Use these messages to identify application pages requiring changes.

Note: For deprecated tags in Relational Reporting, see the *Relational Reporting Developer's Guide*.

There are no deprecated APIs in DB2 Alphablox V8.2. Below are the lists for the previous releases:

Release 8.2 - Deprecated APIs

There are no deprecated APIs in this release.

Release 5.6 - Deprecated APIs

There are no deprecated APIs. There are deprecated fields in DataSourceSelectFormBlox:

Deprecated Constants in DataSourceSelectFormBlox	New Constants in DataSourceSelectFormBlox
`IBMDB2JDBCDriver` Field Value: `IBM DB2 JDBC Driver`	`DB2Driver` Field Value: `IBM DB2 JDBC Driver`
`OracleType4Driver` Field Value: `Oracle Type 4 Driver`	`OracleDriver` Field Value: `Oracle Driver`
`SybaseJConnectDriver` Field Value: `Sybase JConnect Driver`	`SybaseDriver` Field Value: `Sybase SQL Server Driver`

Deprecated Constants in DataSourceSelectFormBlox	New Constants in DataSourceSelectFormBlox
WebLogicMS_SQLServerDriver Field Value: WebLogic MS-SQL Server Driver	MSSQLDriver Field Value: Microsoft SQL Server Driver

Release 5.5 - Deprecated APIs

Deprecated Methods	New Methods
The following server-side methods used to add pre-event processing are deprecated: addBookmarkDeleteFilter(), removeBookmarkDeleteFilter(), addBookmarkLoadFilter(), removeBookmarkLoadFilter(), addBookmarkRenameFilter(), removeBookmarkRenameFilter(), addBookmarkSaveFilter(), removeBookmarkSaveFilter(), addCollapseFilter(), removeCollapseFilter(), addDrillDownFilter(), removeDrillDownFilter(), addDrillThroughFilter(), removeDrillThroughFilter(), addDrillUpFilter(), removeDrillUpFilter(), addExpandFilter(), removeExpandFilter(), addHideOnlyFilter(), removeHideOnlyFilter(), addKeepOnlyFilter(), removeKeepOnlyFilter(), addMemberSelectFilter(), removeMemberSelectFilter(), addPivotFilter(), removePivotFilter(), addQueryFilter(), removeQueryFilter(), addRemoveOnlyFilter(), removeRemoveOnlyFilter() addShowAllFilter(), removeShowAllFilter(), addShowOnlyFilter(), removeShowOnlyFilter(), addSwapAxisFilter(), removeSwapAxisFilter()	"addEventFilter()" on page 48"removeEventFilter()" on page 55
The following server-side methods used to add post-event operations are deprecated: addChartPageFilter(), removeChartPageFilter()	"addEventListener()" on page 49"removeEventListener()" on page 56
The following RepositoryBlox server-side method is deprecated: getUsersGroup()	"getGroupNames()" on page 118 (AdminBlox's User object)

Release 5.1 - Deprecated APIs

The <blox:clustered> tag has been deprecated. This tag was used in a clustering environment under Tomcat running Resonate Central Dispatcher software. The standalone clustering solution is no longer supported.

Release 5.0 - Deprecated APIs

There are no deprecated APIs in this release.

Release 4.1.1 - Deprecated APIs

Deprecated property or method (Client-side)	New property or method (Client-side)
suppressMissing, isSuppressMissing(), setSuppressMissing()	No replacement. Use instead: suppressMissingRows, suppressMissingColumns

Release 4.1 - Deprecated APIs

There are no deprecated APIs in this release.

Release 4.0 - Deprecated APIs

Deprecated property or method (Client-side)	New property or method (Client-side)
cellAlerts, setCellAlerts()	No replacement. Use instead: cellAlert, getCellAlert(), setCellAlert()
dataLayoutVisibleAtStartup	No replacement. Use instead: dataLayoutAvailable, isDataLayoutAvailable( ), setDataLayoutAvailable( )
dataRowsInFirstPage, getDataRowsInFirstPage(), setDataRowsInFirstPage()	No replacement.
datasource, setDataSource( )	bloxDataSource, setBloxDataSource( )
dimensionsOnPageAxis, getDimensionsOnPageAxis(), setDimensionsOnPageAxis()	selectableSlicerDimensions, getSelectableSlicerDimensions(), setSelectableSlicerDimensions()
getAlertEnabled()	isAlertEnabled()
getAlwaysShowLastColumn()	isAlwaysShowLastColumn()
getAlwaysShowLastRow()	isAlwaysShowLastRow()
getAutoAxesPlacement()	isAutoAxesPlacement()
getChartAbsolute()	isChartAbsolute()
getChartFirst()	isChartFirst()
getDataTextDisplay()	isDataTextDisplay()
getDrillKeepSelectedMember()	isDrillKeepSelectedMember()
getDrillRemoveUnselectedMembers()	isDrillRemoveUnselectedMembers()
getDwellLabelsEnabled()	isDwellLabelsEnabled()
getEnableKeepRemove()	isEnableKeepRemove()
getEnableShowHide()	isEnableShowHide()
getExpandCollapseMode()	isExpandCollapseMode()
getGridLinesVisible()	isGridLinesVisible()
getGroupSmallValues()	isGroupSmallValues()
getHeadingIconsVisible()	isHeadingIconsVisible()
getHeadingsEnabled()	isHeadingsEnabled()
getHidePlusMinus()	isHidePlusMinus()
getMustIncludeZero()	isMustIncludeZero()
getOnErrorClearResultSet()	isOnErrorClearResultSet()
getPaginate()	isPaginate() setPaginate()
getParentFirst()	isParentFirst()
getPerformInAllGroups()	isPerformInAllGroups()
getRelationalRowNumbersOn()	isRelationalRowNumbersOn()
getRowHeadingsVisible()	isRowHeadingsVisible()
getRowsOnXAxis()	isRowsOnXAxis()
getShowColumnDataGeneration()	isShowColumnDataGeneration()
getShowRowDataGeneration()	isShowRowDataGeneration()
getSuppressDuplicates()	isSuppressDuplicates()
getSuppressMissing()	isSuppressMissing()
getToolbarFloatable()	isToolbarFloatable()

Deprecated property or method (Client-side)	New property or method (Client-side)
getUseAliases()	isUseAliases()
getUseSeriesShapes()	isUseSeriesShapes()
getWritebackEnabled()	isWritebackEnabled()
headerStyle, setHeaderStyle( ), getHeaderStyle( )	headingStyle, setHeadingStyle( ), getHeadingStyle( )
multipleDimensions, getMultipleDimensions( ), setMultipleDimensions( )	No replacement. Use instead: autoAxesPlacement, isAutoAxesPlacement( ), setAutoAxesPlacement( )
noAccessString	noAccessValueString, getNoAccessValueString( ), setNoAccessValueString( )
splitLocation	dividerLocation setDividerLocation( )
suppressZeroRows, getSuppressZeros(), getSuppressZeroRows(), setSuppressZeroRows()	suppressZeros, isSuppressZeros( ), setSupperssZeros( )
useAASAuthorization, setUseAASAuthorization(), getUseAASAuthorization()	AASUserAuthorizationEnabled, setAASUserAuthorizationEnabled(), isAASUserAuthorizationEnabled()

Deprecated Classes (Server-side)	Use Instead (Server-side)
ServerDataBlox	DataBlox bean
ServerRepositoryBlox	RepositoryBlox

URL Attributes No Longer Supported
bookmark
browser
height
grid_scrollbars
left
top
width

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation, Licensing, 2-31 Roppongi 3-chome, Minato-ku, Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation, J46A/G4, 555 Bailey Avenue, San Jose, CA 95141-1003 U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM	DB2	DB2 OLAP Server
DB2 Universal Database	WebSphere	

Alphablox and Blox are trademarks or registered trademarks of Alphablox Corporation in the United States, other countries, or both.

Microsoft, Windows, 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.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product or service names may be trademarks or service marks of others.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Program Number: 5724-L14

Printed in USA

SC18-9435-00

Spine information:

IBM DB2 Alphablox Developer's Reference Version 8.2