IBM DB2 Alphablox

DB2 Alphablox Developer's Guide

Version 8.3

SC18-9434-01

IBM DB2 Alphablox

DB2 Alphablox Developer's Guide

Version 8.3

SC18-9434-01

Note:
Before using this information and the product it supports, read the information in "Notices" on page 269.

Second Edition (November 2005)

This edition applies to version 8, release 3, 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

Chapter 1. DB2 Alphablox applications and the underlying Blox

DB2® Alphablox enables you to create custom business analytic applications—applications that help your end users visualize and analyze live business data and transactions from various data sources. Rather than just providing data in the manner of query and reporting tools, an DB2 Alphablox application typically incorporates business logic and offers guided analysis via an easy-to-use interface.

An DB2 Alphablox application can be any Web application containing DB2 Alphablox building blocks known as Blox. The application can be as simple as one JSP page, or as complex as a whole collection of web pages that communicate with various application servers and data sources.

Blox are reusable software components that you can add to your JSP pages using JSP tags or Java™ code to connect to data sources, perform data transformation and calculations, and provide interactive, data analysis functionality.

The focus of this section is to highlight the key characteristics that are common to DB2 Alphablox applications. With graphical representation and sample scenarios, this section demonstrates how the features and components in DB2 Alphablox make these characteristics possible.

For details on DB2 Alphablox application program flow and development approaches, see Chapter 2, "DB2 Alphablox application program flow," on page 9.

Key characteristics of DB2 Alphablox applications

A DB2 Alphablox application typically has the following characteristics. Each characteristic may be implemented using various combinations of features in DB2 Alphablox:

Real-time data access and analysis

A DB2 Alphablox application can drive analysis of data from multiple data sources, both relational and multidimensional. Through native access to the database (MDX for Microsoft® Analysis Services, Report Script for DB2 OLAP Server™ and Essbase, and JDBC for relational databases), DB2 Alphablox exposes the analytic functionality in the database engine, such as ranking, derived calculations, ordering, filtering, percentiles, variances, standard deviations, correlations, trending, statistical functions, and other sophisticated calculations.

There are different ways live data can be presented to your users. If your users need data presented in grids and charts, first you add a DataBlox to your application and specify the data source to use for that instance of DataBlox. You immediately have access to all the analytic functionality inherent in the database

engine. Then add a PresentBlox, which embeds a GridBlox and a ChartBlox, to use the data from that DataBlox. Now your users can interact with up-to-date data through the Blox user interface to meet their data analysis needs.

For example, for the CFO, the first screen she sees when she logs in may be an executive dashboard that contains a monthly income statement and a summary on market profit ranking. The data is live, and the CFO can choose to drill down on the data if she wants to find out which customer is buying which product.

For creating reports from a relational database, you can use Relational Reporting. At the core of Relational Reporting is the ReportBlox component, which renders a relational result set to a DHTML-based report. Other Blox components support data access, data transformation, calculation, and formatting in Relational Reporting. Each of these Blox performs the specific task that its name suggests.

An relational report can be static or interactive. If you offer to render the report in interactive mode, your users can sort, filter, or reorder data on the fly using Relational Reporting's Report Editor user interface to design their own relational reports.

Interactive user interface

A DB2 Alphablox application typically has grids and charts that can be served in a DHTML rendering, accessible using supported web browsers.

The grids and charts rendered in the DHTML client have an easy-to-use user interface that shields the users from the complexity of analyzing data. When you add a PresentBlox, it can nest a GridBlox, a ChartBlox, a ToolbarBlox, a PageBlox, and a DataLayoutBlox to offer users interactive data analysis, bookmarking, data exporting, and view customization capability. As a developer you can customize and personalize various components in the interface to meet your design needs.

The DataLayoutBlox appears as a data layout panel, enabling users to interactively move and view dimensions among axes. The ToolbarBlox appears as a toolbar, providing quick access to commonly performed data analysis tasks with a click of the mouse. The menu bar offers all the options and actions available to the users. Users can bookmark a view, hide and show the grid or chart, sort data, export the data to PDF or Excel, and navigate the data. The PageBlox appears as a page filter, allowing users to filter data to appear in the GridBlox and ChartBlox. All these Blox are nested inside a PresentBlox to simplify application development and conserve screen real estate.

The components in the user interface can be customized using JSP tags provided in the DB2 Alphablox Tag Libraries. For example, you can specify the colors to use, add or remove buttons in the toolbar, add or remove menus from the menu bar, or add or remove data navigation options. You can also specify a set of criteria for highlighting cells, a feature called Cell Alerts (such as displaying cells in red if they have a value lower than the minimum specified).

A key strength of this user interface is that it does not involve page refreshes every time a user interacts with the data. The user does not have to wait for the entire page to be downloaded, which means longer wait and the likelihood to lose track of the original context. In a portal environment, this is a huge benefit since refreshing the whole portal page involves reloading all the portlets on the page and can take up significantly more time.

DB2 Alphablox themes

DB2 Alphablox offers several themes out of the box for the DHTML client with different associated style sheets and GIF images that you can use immediately. You can also create your own theme by copying an existing DB2 Alphablox theme, then modifying the style sheet and images used in that theme.

Member Filter

Sometimes users want to see more specific data rather than drill up and down one level at a time. They may want to see data for specific members from different parents in the dimension hierarchy. For example, a user may just want to compare data from a representative state within each region.

The Member Filter interface allows them to navigate the Market dimension and select New York from the East region, California from the West region, Illinois from the Central region, and Texas from the South region.

Member Filter is available from right-click and pop-up menus in GridBlox, DataLayoutBlox, and PageBlox when data navigation options are offered. The dimension listed in Member Filter dialog window depends on where the user right-clicks to bring up the menu.

Relational Reporting user interface

When you use ReportBlox and its supporting Blox to create an interactive report, your users can sort, hide, or reorder columns, create break groups, and add summary data for each break group via the Report Editor user interface.

Report Editor consists of three context-sensitive pop-up menus. All these are supported with DHTML. The reports and the interactive menus are rendered with specific CSS style classes. You can customize the colors and fonts by specifying your styles to use.

Personalization

As each user has different data and business needs, a DB2 Alphablox application often needs to be personalized. For example, depending on the users, the first screen they see when they log in may be different. You may want to control data navigation so users in the West region will not see data in the East region. Or, you may want to let your users specify their preference for chart types, or what the threshold numbers they want for highlighting data in the grids.

Custom properties

DB2 Alphablox supports personalization through custom properties. You can define your custom user properties and specify the valid values for each property. Then for each user, you can assign a value for each of the defined properties. Based on the user logged in and the property values associated with the user, you can dynamically specify what to display, how the data should be displayed, or what data navigation functions should be enabled or disabled.

Sharing and collaboration

Some of the common features of DB2 Alphablox that are used to support sharing and collaboration are bookmarking, commenting on data, and PDF conversion.

Bookmarking

A key feature of DB2 Alphablox is its Bookmarks functionality. Through the user interface, users can bookmark a data view and later retrieve the same view with

up-to-date data. Bookmarks can be private, available to users in a specific group, or public to all users that have access to the server.

BookmarksBlox provides an extensive API for managing and manipulating bookmarks. For example, you can programmatically update all bookmarks to reflect changes in the data outline.

Commenting on data

To support collaborative analysis, you can utilize CommentsBlox to support cell-level, page-level, or application-level annotations. Users can add comments to data cell in a GridBlox by right-clicking the cell and select Add Comments. Cells with comments have a comment indicator on the corner so users can quickly spot them and choose to view them.

Convert to PDF

Users often want to save their work or share their view of the data. DB2 Alphablox has a Convert to PDF option that enables you to offer the capability to save the data in Blox to PDF format. This solves a host of problems common to printing or saving web pages using the browser, such as improper page breaks, inappropriate page width to display charts, cross-browser printing differences, and having to e-mail all HTML and images used in the report.

The Convert to PDF option gives you, or users if you choose to, control of the page layout, margins, page orientation, font sizes, colors, header and footer texts, and where the header and footer should be positioned.

Real-time planning

An analytic application may extend from historical analysis to forward looking forecasting and proactive resource allocation. You can build real-time planning applications such as budgeting, sales forecasting, and collaborative demand planning through use of the DB2 Alphablox data writeback capability.

For example, you can extract the data from the data source into a GridBlox, allow regional managers to enter sales forecast numbers within the GridBlox, and upon submitting the data, write the data back to the data source. Together with custom properties, the application can dynamically create the sales forecast worksheet based on the region to which a user belongs.

Underlying Blox components

The key components underlying an DB2 Alphablox application are Blox components. Blox are reusable software components that enable you to connect to data sources, perform various data manipulation and presentation tasks, and build dynamic, personalized data analytic applications.

One Blox may provide several of the above functionality through its properties and associated methods. These properties and methods enable you to specify and control Blox appearance and behavior. There are also event filters for handling user events such as drilling up/down, pivoting, changing the page filter, or loading a bookmark.

The following table provides a brief description of each of the Blox.

Blox	Description
DataBlox	• "DataBlox" on page 6: the key Blox that provides data access, retrieval, and manipulation functionality for all data presentation Blox. • StoredProceduresBlox: allows you to create a connection to a relational database and prepare a stored procedure statement for use.
User Interface Blox	There are six Blox that enable you to present data in a grid or chart format with interactive user interfaces with which your users can analyze data, change page filters, add or load a bookmark, specify grid or chart layout, and more. These Blox are: • "GridBlox" on page 7 • "ChartBlox" on page 7 • "DataLayoutBlox" on page 7 • "PageBlox" on page 7 • "ToolbarBlox" on page 7 • "PresentBlox" on page 7 Each of these Blox has a user interface component and an application programming interface (API) that gives you control over their presentation and actions allowed. Many return a result sets that you can get meta information from.
Analytic Infrastructure Blox	These Blox provide means to building analytic infrastructure. • RepositoryBlox: provides a means for developers to save and retrieve application properties in the DB2 Alphablox Repository. • 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. • AdminBlox: provides programmatic access to information on the server, users, groups, roles, data sources, and applications set through the Administration pages in the DB2 Alphablox home page
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"
FormBlox	A series of FormBlox are available to provide a familiar HTML form interface and handle state management for you. These FormBlox are data-aware and allow users to select a data source, dimensions, members, or other options you offer to create personalized queries.
ContainerBlox	The ContainerBlox, available only in the DHTML mode, can be used to create custom Blox components and manages persistence during your user sessions.
ReportBlox	ReportBlox, rendered in an interactive HTML format, is the core Blox for building reports from relational data sources. Details on ReportBlox and its supporting Blox are in the *Relational Reporting Developer's Guide*.

The focus of following sections is on what DataBlox and each of the user interface Blox enables and how they work together to provide the following:

- programmatic control to developers
- the visual data analysis experience to users

DataBlox

DataBlox is the Blox that specifically offers the needed functionality for data access. It has no graphical user interface. Instead, it is at the heart of all the Blox that provide a graphical user interface for users to interact with the data. It has an extensive application programming interface (API). For example, you can detect if the data source needed has been successfully connected to or whether the current database operation is complete, etc. You can prevent a user from performing certain data navigation actions or seeing certain data in the result set based on who the user is. The following table shows some of properties and methods associated with DataBlox to demonstrate the extensiveness of its API.

Categories of DataBlox Properties/Methods	Description	Examples
Data sources	Properties related to the data source	`aliasTable, catalog, query, schema, connectOnStartup, userName, password, dataSourceName`
Data manipulation	Properties related to data manipulation such as calculated members, sorting, and drilling	`calculatedMembers, columnSort, rowsort, hiddenMembers, keepOnly, parentFirst`
Data appearance	Related to data appearances such as whether or how duplicate, missing, zero data or prefix in member names should be displayed	`memberNameRemovePrefix, memberNameRemoveSuffix, suppressDuplicates, suppressMissing, suppressNoAccess, suppressZeros`
Write back	Related to data update	`commitData()`
Result set	Related to the result set containing the data	`clearResultSet(), getResultSet()`
MetaData	Related to the MetaData object of the underlying data source for the current DataBlox	`dimensionRoot, getMetaData()`
MDB result set	Related to the axes, dimensions, tuples, and cells in the multidimensional data result set	`((MDBResultSet) getResultSet())`
RDB result set	Related to the columns and rows in the relational data result set	`((RDBResultSet) getResultSet())`
MDB metadata	Related to the multidimensional metadata for the result set	`((MDBMetaData) getMetaData())`
RDB metadata	Related to the relational metadata for the result set	`((RDBMetaData) getMetaData())`
Event filters	Related to the server-side event filters	`addFilter(), removeColumnSort()`

GridBlox

A GridBlox displays relational or multidimensional data in an advanced grid format, enabling users to drill, pivot, sort, and explore the data. It has an extensive set of properties and associated methods to let you control its appearances, numeric formatting, and others. By default, a standalone GridBlox has:

- a menu bar that offers all options and functionality available to the GridBlox and the underlying DataBlox
- a ToolbarBlox that offers users quick access to common functionality by clicking a button

ChartBlox

ChartBlox displays relational or multidimensional data in a variety of chart formats, enabling users to change chart appearances and explore the data. ChartBlox needs a DataBlox to provide data access and data manipulation functions. By default, a standalone ChartBlox has:

- a menu bar that offers all options and functionality available to the ChartBlox and the underlying DataBlox
- a ToolbarBlox that offers users quick access to common functionality by clicking a button

DataLayoutBlox

DataLayoutBlox displays available data dimensions and the axes on which they currently reside, enabling users to move dimensions between axes. DataLayoutBlox nests within a PresentBlox. It cannot nest within a standalone GridBlox or a standalone ChartBlox.

When users move a dimension from one axis to another, data in both GridBlox and ChartBlox within the same nesting PresentBlox will automatically reflect the changes.

PageBlox

PageBlox displays dimensions residing on the page axis, effectively filtering data in the chart or grid, enabling users to change data filters. PageBlox nests within a PresentBlox. It cannot nest within a standalone GridBlox or a standalone ChartBlox. When the user makes a selection from the dimension in the PageBlox, the data in the GridBlox and ChartBlox within the same nesting PresentBlox will reflect the filter selected.

ToolbarBlox

ToolbarBlox displays buttons, enabling user access to various Blox functionality. ToolbarBlox needs to nest within a PresentBlox or a standalone GridBlox or ChartBlox. By default, ToolbarBlox is turned on in these user interface Blox and appears as two toolbars. These toolbars are customizable. You can add or remove buttons in the existing toolbars. You can even add or remove a toolbar.

PresentBlox

PresentBlox combines all the above Blox into a single Blox to simplify application development and conserve screen real estate. All Blox nested within PresentBlox interact with each other. They use the same data source, and data navigation actions done in PageBlox, for instance, affect the data displayed in both the nested GridBlox and ChartBlox. Data options specified are reflected in all its nested Blox

when applicable. For example, if you specify to use aliases for member names, aliases will be used in GridBlox, ChartBlox, and PageBlox.

DB2 Alphablox FastForward

DB2 Alphablox FastForward is a sample application framework for quickly developing, deploying, and sharing custom analytic views. Out-of-the-box, the FastForward framework delivers common application services, including security, collaboration, customization, and personalization. Application administrators, typically OLAP administrators, can create new versions of an FastForward application, publish reports by selecting report templates and configuring report parameters, and then deploy the new application without ever looking at code. JSP developers can further modify or extend the application framework and add new custom report templates for application administrators to configure and deploy. See the Working with DB2 Alphablox FastForward topic in the *Developer's Guide* for more information.

Chapter 2. DB2 Alphablox application program flow

This section describes the file structure of an DB2 Alphablox application, how an application is processed by the DB2 Alphablox and the application server, and how an application developer develops an application using standard web technologies to achieve the desired end-user interaction and program control.

Application file structure

Since DB2 Alphablox runs in a Java 2 Enterprise Edition (J2EE) web application server environment, this section describes the file structure in the underlying application server when you create an DB2 Alphablox application.

Application context

When you create an application from the DB2 Alphablox home page, you are asked to specify information such as application context, display name, Home URL, default saved state, and write privileges security role. Based on this set of information, DB2 Alphablox creates the application definition in the DB2 Alphablox repository as well as the application directory structure. A directory with the name of the application context that you specified is created, and is usually referred to as the application "docroot," application context, or application directory.

Where this application directory physically resides depends on the application server. When DB2 Alphablox is installed using WebSphere®, the application directory is in WebSphere's `installedApps` directory. For more information, refer to the *Administrator's Guide*.

All files for the application that you create must reside within this application directory structure. Typically, these are a combination of JSP, HTML, CSS, JavaScript™, and image files. Also, Java classes or other Java Archive files that contain servlets, beans, or other utility classes are typically placed in subdirectories in the `WEB-INF` directory, in `classes` and`lib` directories as suggested for J2EE specification.

DB2 Alphablox Repository

The DB2 Alphablox Repository is a store of objects that DB2 Alphablox uses to keep track of applications, users, groups, bookmarks, and other such information. Physical files associated with the DB2 Alphablox Repository reside in the `<db2alphablox_dir>/repository` directory (when you use the DB2 Alphablox filesystem repository), where `<db2alphablox_dir>` is where DB2 Alphablox is installed.

For example, when you create an application called "MyApp1" from the DB2 Alphablox home page, a folder named "MyApp1" for that application is created under the `<db2alphablox_dir>/repository/applications/` directory. When you define a custom application property, the application properties descriptor file `appprodesc.properties` is updated to store the information.

Likewise, when you add a user, a folder with that username is created under the `<db2alphablox_dir>/repository/users/` directory. Each user has an associated user

property file that stores information such as password, email address, and group association, as defined through the DB2 Alphablox home page.

By using the RepositoryBlox API, you can get, set, save, or delete an application state, or get the user name and groups to which the user belongs.

Working with Blox in JavaServer Pages

In a J2EE environment, to serve dynamic content, the key technology to use is JavaServer Pages (JSP). The JSP technologies allow for the combination of HTML, JavaScript, and Java code in one physical file.

Since Blox are typically Java beans, to add a Blox, you use a JSP tag to include the bean as you normally would with any Java bean, by using the `<jsp:useBean>` tag. You can also take advantage of DB2 Alphablox custom JSP tags to add Blox using XML-like syntax.

Request processing

This section describes how an HTTP request for an DB2 Alphablox application is processed by the underlying application server and DB2 Alphablox. The following sections provide a high-level, simplified view of the process. For a more complete picture, see a book on JavaServer Pages technology.

The description is based on an application with an application context of MyApp1 with the following files:

- `welcome.html`: the application entry page. This page has a link that points to `intro.jsp` and `firstGrid.jsp`.
- `intro.jsp`: a JSP file with some general Java and JavaScript code.
- `firstGrid.jsp`: a JSP file with a GridBlox in it, similar to the one shown earlier in the section "Working with Blox in JavaServer Pages."

The description also assumes that the application server is responsible for serving web pages without a separate web server.

User Request 1 (http://myAppServer/MyApp1/welcome.html)

1. User "dave" accesses `http://myAppServer/MyApp1/welcome.html` through his browser.
2. The application server goes to `MyApp1/` and looks at the security information defined in the application deployment descriptor file `web.xml` in the `WEB-INF/` directory.
3. Based on the security constraints defined, the application server challenges the request for username and password.
4. A J2EE session is started upon authentication. The application server sends a cookie in the response back to the browser. The cookie that is sent contains a session ID.
5. The application server parses through `welcome.html` and sends it back to the browser.

User Request 2 (http://myAppServer/MyApp1/intro.jsp)

1. Dave clicks on the link that points to `intro.jsp`. An HTTP request for `http://myAppServer/MyApp1/intro.jsp` is sent.

2. The application server accesses the cookie and header information to look up the J2EE session ID and verify the security.
3. The application server has a JSP engine that compiles and executes the JSP file. The application server first checks to see if this file has been compiled or has changed since it was last compiled.

   If compilation is needed, the engine processes and compiles the file into a Java class file. It checks whether or not the classes and package referenced in the JSP file exist, and whether or not the syntax is correct.
4. The application server executes the complied file and issues a response back to the browser.

User Request 3 (http://myAppServer/MyApp1/firstGrid.jsp)

1. Dave goes back to `welcome.html` and clicks a link that points to `firstGrid.jsp`. An HTTP request for `http://myAppServer/MyApp1/firstGrid.jsp` is sent.
2. The application server accesses the cookie and header information to look up the J2EE session ID and verify the security.
3. The application server first checks to see if this file has been compiled or has changed since it was last compiled.
4. If compilation is needed, its JSP engine processes and compiles the file into a Java class file. The application server checks whether the classes, packages, and tag library descriptor files (TLD) as referenced in the `<%@ ...%>` directive exist, whether the Java methods and custom tags used are valid, and whether the syntax is correct. The application server then execute through `firstGrid.jsp`.
5. It encounters the following scriptlet and processes it. The variable `banding` gets a value of `true` or `false`:

   ```
   <% String banding =
    (Math.random() >= 0.5) ? "true" : "false"; %>
   ```
6. Then it encounters a tag it is unfamiliar with—`<blox:grid...>`. The prefix `blox` matches what is specified in the taglib directive `<%@ taglib uri="bloxtld" prefix="blox" %>`.
7. The application server goes to the tag library as defined in the taglib directive. Tags are “macros” that are replaced by actual Java code that creates and initializes beans. The application deployment descriptor file `/MyApp1/WEB-INF/web.xml` tells the application server where the tag library descriptor file (TLD) is located:

   ```
   <taglib>
      <taglib-uri>bloxtld</taglib-uri>
      <taglib-location>/WEB-INF/tlds/blox.tld</taglib-location>
   </taglib>
   ```
8. DB2 Alphablox is now called into duty. DB2 Alphablox initializes the bean, user session, application instance, and peers, and sends the results back to the application server. The details on how Blox are processed and served are discussed in the next section, “DB2 Alphablox program flow” on page 12.
9. The application server continues to process the lines in `firstGrid.jsp` until it reaches the end.
10. The application server sends the result back to the browser.

The role of the application server

In summary, the application server is responsible for the following tasks:

- user authentication and security

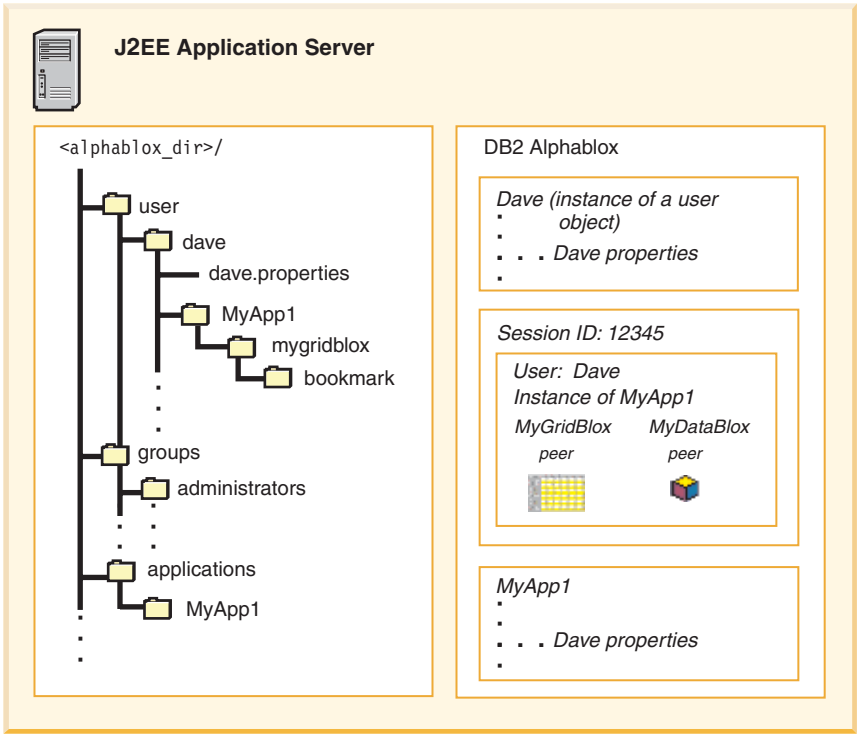
- processing and serving HTML files
- processing and compiling JSP files with help from its servlet/JSP engine, then serving the entire response generated back to the browser

DB2 Alphablox program flow

When the application server encounters custom tags such as `<blox:grid>`, it calls upon the tag library specified. Based on specifications in the application's `web.xml` file, the application server knows which Java package to use to replace the tags into Java code:

At this point, DB2 Alphablox performs the following tasks:

1. DB2 Alphablox gets the user from the request object (an API in J2EE) and checks to see if a user object for Dave has already been created. If this is the first request from the user, DB2 Alphablox creates the DB2 Alphablox user.
2. DB2 Alphablox loads the user profile from the DB2 Alphablox Repository and creates a user instance.
3. DB2 Alphablox next creates a session instance, assigning a new session ID that is included in the response header. An instance of the user object is added to the session.
4. DB2 Alphablox then creates the application instance.
5. Next, DB2 Alphablox retrieves the application name from the request object and checks to see if an application object that matches the application name already exists. If not, DB2 Alphablox creates the application object and an instance of that application is added to the session instance.
6. Now that instances for the user, the session, and the application have been created, DB2 Alphablox creates peers.
   a. `firstGrid.jsp` has a GridBlox with an ID of `MyGridBlox`. DB2 Alphablox checks to see if a grid peer for `MyGridBlox` already exists. If it doesn't exist, DB2 Alphablox creates one.
   b. The grid peer looks for an associated data peer. If it doesn't exist yet, DB2 Alphablox creates one.
7. DB2 Alphablox sends the rendered result back to the application server. The application server takes the output sent by DB2 Alphablox, merges it into the rest of the file before it sends the result back to the browser.



If another request comes from the same user for the same application in the same session, the existing peers are reused.

The role of DB2 Alphablox

In summary, DB2 Alphablox is responsible for the following tasks:

- data access and manipulation
- building and deploying interactive analytic applications
- personalization of data views (more detail in the following sections)

Bookmarking, application states, and the DB2 Alphablox Repository

When user "dave" bookmarks a data view, depending on whether the bookmark is saved as private, public, or visible to a named group, a folder with the name of that instance of the presentation Blox (usually a PresentBlox, GridBlox, or ChartBlox) will be created under either the user's, the application's, or the group's folder in the repository.

Information pertaining to each bookmark stored in the repository includes its visibility (private, public, or a specified group), the application name, width, and height of the presentation Blox, data source and data query for that view, description of the bookmark, as well as the color schemes and other data display options associated with that view.

Through the APIs provided, you can programmatically get names of bookmarks with a specified visibility. You can save, delete, rename or restore bookmarks, and detect bookmark saving and loading events. You can also programmatically create bookmarks or change all data queries saved with bookmarks to reflect changes in the data outline.

The repository also stores the state of an application if you specify to automatically save an application's state through the application definition page on DB2 Alphablox Administration pages. DB2 Alphablox will save the information on all Blox (if you have multiple PresentBlox or multiple independent Blox) in the application, including the query result sets, grid and chart appearance, and other changes made by the user.

Application development and programming model

The *Developer's Guide* covers the setup of your application development environment, JavaServer Pages, the Blox Tag Library, and task-based implementation steps and details. Before delving into those details, it will be helpful to understand the following concepts.

Blox components

Blox components are built on Java beans. An extensive API is available for you to access and control Blox and Java objects on the server using Java, scriptlet, or Blox custom JSP tags. The server-side API gives you control over Blox presentation and behavior, prevents your business logic from being exposed to users (through source viewing or file saving options from the browser), and shields the programming complexity from page designers on your development team.

JSP and custom tags

JSP is the key technology in J2EE that enables the combination of static and dynamic content in one file. By default, the application server will only invoke its JSP engine to process a request and generate dynamic content if the file has the `.jsp` extension. An HTML page will not go through the servlet compilation and request generation process. The application server will serve the page as a static HTML page, and browsers will ignore the JSP code and Java scriptlet. Therefore, in order to use the functionality offered by DB2 Alphablox, Blox should be added to JSP pages.

Since Blox are built on Java beans, you can expect them to have the same attributes Java beans have. For example, they have properties, and setter and getter methods for the properties. You can use the standard `<jsp:useBean>` tag to add a Blox, and then use the `<jsp:getProperty>` and `<jsp:setProperty>` tags to get/set a Blox property. However, you should use the DB2 Alphablox Tag Libraries whenever possible. When you use the custom Blox tags, the scope is automatically set to "session" and DB2 Alphablox takes care of the session management and automatically cleans up unused/expired resources for you.

For information on both approaches, and on the Blox Tag Library, see "Blox display tag" on page 41.

Server-side API versus client-side API

Since Blox are Java beans, you can access Blox and their peers on the server to get information and control Blox behavior and appearances. The processing is done on the server before the output is sent to the client. This allows you to use other resources on the server, reuse components, and reduce the discrepancies and inconsistencies often found among different browsers or browser versions. Typically, on the server side you can do the following using JSP, Java scriptlets, or custom tags:

- create an instance of a Blox
- set the properties of the Blox dynamically

- get the properties of the Blox

In some case where you want users to be able to make a selection such as choosing the region for which they want to see the data or specifying some parameters for displaying a grid, you will need to call some JavaScript functions that communicate the choices to the server. The DHTML client has a straightforward client-side API that allows you to call a JSP page on the server or a server-side bean and set its property. The client-side API is detailed later in this book.

Chapter 3. Your development environment

DB2 Alphablox solutions are based on the open standards of the World Wide Web, enabling you to have many options for development tools which you can use to build analytic applications using DB2 Alphablox components.

If you already have web development tools that you are comfortable with, you can most likely continue to use them to develop analytic applications using DB2 Alphablox. In this section, some important developer issues are discussed in hopes of maximizing your success.

Choosing application development tools

The DB2 Alphablox solution was intentionally designed to support the open standards technologies of the Internet, including HTML, CSS, JavaScript, and others. By not requiring a proprietary development environment, DB2 Alphablox allows you to choose the tools with which you are most familiar or comfortable. Experienced Java developers may already be comfortable using IBM® Rational® Application Developer, Rational Web Developer, Eclipse with the Web Tools Platform plug-ins, or other IDEs. If you're not familiar with Java, or frequently author web pages, you may be familiar with your favorite HTML editor. Some of you will be more comfortable using powerful text editors, such as Visual SlickEdit or jEdit. If you haven't already found your ideal development environment yet, you can explore the many choices available, and know that the one you choose can likely be used to develop analytic applications based on DB2 Alphablox.

Web browsers

DB2 Alphablox applications are supported using Microsoft Internet Explorer only (see the *Installation Guide* for specific requirements).

As a developer, you will be working back and forth between coding in your development environment and testing your code in a web browser and should be aware of some commonly encountered issues that may affect your work. First, there are some general browser considerations and issues to be aware of during development. Second, to enhance your development experience, you will want to optimally configure your testing browsers for use during your development. These issues are covered in the following sections.

General considerations

During application development, you may find it advantageous to configure your browsers for what could be considered "development mode." In the tasks below, steps are given for configuring Microsoft Internet Explorer to be optimized for developer efficiency. Keep in mind that you should always test your final applications and their behavior using web browsers and configurations that the users will be using. Most likely, the differences in configurations between development mode and end user modes will not affect the end result, but it is always a best practice to test your applications using the full range of possible browsers and configurations that are likely to be encountered by end users.

Working with DHTML mode

There are a couple of important points that you should be aware of when you are working with the DHTML client during application development. It is not the intent here to completely discuss how to code using DHTML technologies, but to explain a few of the frequently encountered behaviors that you should understand when working with DB2 Alphablox applications.

Modifying Blox tags

One of the first lessons you'll learn when working with the DHTML client is that modifications to Blox tags will not take effect during your current browsing session. Instead, in order to see the changes made after modifying Blox tags, the browser must be closed, and a new browsing session started. This is expected behavior that results from how the DHTML client works with the server-side code.

Another frequent error that you might make when developing applications is to inadvertently create multiple Blox components with the same `id` attribute. Typically, this will happen when you copy and paste code, including a Blox definition tag to create another Blox on the same or a different page, but forget to change the `id` attribute of the new Blox. If two Blox have the same `id`, then the first one loaded into the browser memory will determine what the second Blox will look like -- the property settings on the second Blox are ignored.

Configuring and developing with Microsoft Internet Explorer

The following steps apply to Microsoft Internet Explorer version 5 or later.

1. Open your Microsoft Internet Explorer browser.
2. Click on the Tools menu and select Internet Options on the submenu to open the Internet Options window.
3. In the Temporary Internet Files section, click on the Settings button.
4. The default setting for "Check for newer versions of stored pages" is "Automatically." Change this setting to "Every visit to the page." This selection will make it more likely that the web page you open will be the newest version of a page you are working on.
5. Close this dialog by clicking on the OK button, but do not close the Internet Options dialog window.
6. Now select the Advanced tab in the Internet Options. A long, scrollable list of check boxes should be visible. The following sections cover different portions of this long options window. The settings below are optional, but are recommended for enhancing your troubleshooting web pages.

   JavaScript Error Notification
7. [Optional] To help you recognize JavaScript errors, it is recommended that you check the "Display a notification about every script error" in the Browsing section of the Advanced options. What this does is to pop up a dialog box that you cannot miss stating that a JavaScript error has occurred. If you do not enable this, you will be have to pay attention and notice any JavaScript alert message that appears in the lower left corner status window.

Tip: When viewing DB2 Alphablox application pages within Microsoft Internet Explorer, the browser may not process the page you are attempting to display in a way you expect. Sometimes the browser may re-display a cached page instead of the new updated page. The setting above is supposed to help prevent this, but can be unreliable. Even when you click on the browser's Reload button, the page displayed may continue to be a cached page. When

you think this might be the problem, you have a couple of other options. First, you can force a hard refresh the page (getting a fresh copy from the server instead of a cached copy) by using the Control-Refresh technique: hold down the Control key while clicking on the Refresh button. The second option is to close and reopen the browser. This results in a new browser session, and is the most certain way to guarantee that the page being displayed is the newest page.

Web browsers - known Mozilla issues

Known Mozilla issues that differ from Microsoft Internet Explorer are highlighted.

It is a best practice to test your applications using web browsers supported by your organization. The table below highlights known Mozilla and Mozilla Firefox web browser behaviors for Blox UI components that differ from supported Microsoft Internet Explorer browsers.

Table 1. Notable Mozilla issues

Issue	Notable Mozilla issues
Chart resizing in splitter container	Chart stays the same size, then repaints. Firefox maintains the aspect ratio of the chart, which alters the container size.
Edit Copy (Blox Model API and UI feature)	Not supported. There are no methods available for copying to the clipboard in the Gecko engine.
Edit field selection and caret position	Not supported. The window.getSelection() method used in the Gecko engine does not return text in an edit field. This is a known limitation and may be fixed in a future release.
ComboBox auto-complete highlighting	Not supported. There are no methods available for highlighting the completed text, although the completion works.
Line breaks in tool tips	Not supported. There are no methods for adding line breaks to tool tips. Line break characters display unique characters. DHTML view code will instead replace these with spaces in Firefox.
Pop-up menus, pull right menus, toolbar drop downs	Restricted to the frame. Pop-up menus and pull right menus must stay inside of the frame in Firefox and automatically adjust to allow as much of the menus to fit as possible.
Drag and drop differences	In Firefox, no changed cursors appear for dragging and cannot drop. Select fields can not be dragged. Users cannot select text using the cursor in edit fields that are draggable.
Resizeable dialogs	No border. Removed border from resizable dialogs in Firefox. The main problem stems from the use of margins with percentage sized dialog contents which is not usable in Firefox - the contents always is too large.
Sizing static text component	Sizes set on static components are ignored. To work around this, you can set the size on the component's parent for compatibility in both browsers.

Table 1. Notable Mozilla issues (continued)

Keyboard support (accelerators)	Not supported. Not supported in Firefox as `tabindex` cannot be set on `div` elements. This will be supported in Mozilla Firefox 1.8.
Accessibility	Not supported
Right-to-left (RTL) grid display	Not supported
Grid.setFixedScrollbarPosition(boolean) and Grid.isFixedScrollbarPosition()	Not supported

Application Studio

Application Studio has examples and other tools that can be useful for learning and development purposes. The Application Studio can be accessed through the Assembly tab on the DB2 Alphablox home page. To examine and reuse the sample code for your application, the files are located under the Application Studio directory at:

```
<db2alphablox_dir>\system\ApplicationStudio\Examples
```

The Blox Sampler example set, referenced throughout this Guide, demonstrates many of the techniques discussed and resides in the `Examples` directory.

Chapter 4. Design considerations

As with any application development, you need to clearly identify the requirements before you can proceed with the design and development, and subsequently evaluate the success of your application. This topic includes some general requirement gathering guidelines that will help you identify the needs of your users and other issues you will want to take into consideration before you begin.

Defining application requirements

The goal of application design is to ensure the application provides appropriate information and functionality to meet the particular needs of a specific user audience. As you gather the requirements, you need to look into three areas: data, user interface, and application logic.

- "Data requirements"
- "User interface requirements" on page 22
- "Application logic requirements" on page 24

Data requirements

The specific class of application that DB2 Alphablox supports is online analytical processing (OLAP). In contrast to online transactional processing (OLTP) applications that generate and access data in transactional data sources, OLAP applications access data in data sources. These data sources usually contain data consolidated from transactional detail and stored in a multidimensional architecture.

Part of the application design process is identifying required data, as well as any security issues surrounding access to that data. Answering the following questions can help define application data requirements.

1. What information do the users want or need?

   Answering this question as precisely as possible is the first step toward locating the information and defining efficient queries against it. It is important to determine if users already have access to the required information, if there are security issues involved in enabling access, and so forth. For example, regional sales managers may be able to view data for all regions, while regional sales representatives may be able to view data for only their region.

2. Where does the information reside?

   DB2 Alphablox applications can access data from a wide variety of multidimensional and relational databases. Before beginning to develop applications, make sure you consider which data sources you will need to access and verify that DB2 Alphablox supports your requirements. (For details on support for specific data sources, see the *Installation Guide*) Many analytic applications rely on multidimensional data sources that organize information into a hierarchy of dimensions and members. Blox are specifically designed to exploit this data hierarchy; their user interfaces enable drilling up and down through the hierarchy, filtering data based on dimensions and members, moving one or more dimensions to a different axis, and so forth.

DB2 Alphablox also supports relational data sources that organize information into a row-and-column format. One use for relational data is "drill to detail," enabling users to move from a multidimensional data source of consolidated information into its underlying detail in a relational data source.

The DB2 Alphablox Cube Server component of DB2 Alphablox enables administrators to create multidimensional data cubes from information residing in relational data sources. The DB2 Alphablox Cube Server is useful for applications that do not require the scalability and overhead of full-featured OLAP data sources. The DB2 Alphablox Cube Server contains dimensional metadata so that users can perform such operations as drilling, pivoting, and filtering. For information on how to transform relational data into multidimensional data, see the *DB2 Alphablox Cube Server Administrator's Guide*.

Note that when data appears in an DB2 Alphablox application, the underlying data format is not apparent to the user. However, user actions that require multidimensional format (such as drilling) are disabled if the data is in relational format.

Beginning with DB2 Alphablox, you can use ReportBlox to develop interactive reports from relational data sources, allowing your users to add break groups, reorder columns, sort data, rename columns, and edit cell and header styles through the built-in Report Editor user interface. For information on ReportBlox and its support Blox, see the *Relational Reporting Developer's Guide*.

User interface requirements

The user interface is key to application usability. The application should include content presentation, application navigation, and user assistance. While a comprehensive discussion of effective user interface and web page design is beyond the scope of this document, this section provides some guidelines in the following areas:

User groups

When defining the requirements for user groups, keep the following considerations in mind:

- Users often have different usability requirements. One way to address these differences is to place users within small groups within a larger group. For example, the Financial user group might contain an Analyst group and an Administrative group. An application's interface can change dynamically based on the group to which the user belongs.
- Users also have different data access requirements. Typically, the security facilities of the data sources themselves support user- and group-level access restrictions. To ensure easy user access to data, DB2 Alphablox user groups should parallel those implemented on the data source.

Content presentation

DB2 Alphablox enables you and, to a lesser degree, your users to control content presentation. You have considerable latitude in organizing and presenting information. The look and feel of an application page might be an executive dashboard, an intranet portal, or a printable report.

You also have choices in data presentation. By selecting appropriate Blox and setting property values on those Blox, you can choose whether data appears in a grid, a chart, or a grid/chart combination. Because DB2 Alphablox provides many different chart types, you can experiment with the most effective data presentation.

Where appropriate, you can permit users to change chart types, toggle between grid and chart presentations, and so forth. For example, some users may prefer a pie chart that quickly conveys percentages and trends, while others may prefer a grid that supplies numeric values and supports complex analyses. You should clearly understand an application's target audience to facilitate the design of appropriate and effective content presentations.

User instructions

DB2 Alphablox provides online help that includes comprehensive instructions on using each Blox. The default mechanism for accessing DB2 Alphablox help pages is clicking the question mark on the toolbar or the Help > Help... menu option in the menu bar. However, to reduce the users' learning curve, you may find it helpful to provide application-level user instructions right on each application page.

If this approach is not appropriate, or if the user instructions are quite extensive, you can edit and expand on the DB2 Alphablox online help. For more information, see Chapter 25, "Adding user help," on page 249.

User navigation

The simplest application is a single JSP page with one or more Blox. However, if an application calls for several Blox with which the user interacts, two situations may occur. If multiple Blox reside on a single page, some Blox may scroll out of the browser window when others appear. Therefore, consider providing links within the page that move quickly from area to area.

More typically, several JSP pages comprise an application. This application design should include backward and forward links between pages. An application "home page" might link to all other pages in the application and provides an appropriate place for notifying users of application features and enhancements.

Data manipulation

You can produce applications that range from fully interactive analytic and what-if scenarios to static presentations for quick management snapshots. In fact, simply by enabling or disabling Blox interactivity and toolbars, you may be able to develop a single application that serves multiple user needs. Successful application design requires knowledge of how the target audience will interact with the data.

Saving and restoring work

Users performing complex analyses often want to save their work at a certain point, or make a particular view of the data available to other users. DB2 Alphablox supports these requirements through toolbar buttons.

Application logic requirements

Another major design area is application logic. While Blox provide for data access, presentation, and manipulation, most DB2 Alphablox applications provide logic to meet specific user needs, such as:

- offering a list of predefined queries from which users make a selection
- enabling users to construct queries dynamically through a series of related selections
- setting the initial query, delivery format, and application appearance based on user login
- highlighting exception data based on user-entered values
- toggling content presentation based on user actions
- performing "what-if" scenarios and optionally writing the results back to the data source

You can use a combination of JSP, HTML forms, JavaScript functions, Java and DB2 Alphablox custom properties to implement application logic.

Custom properties

Through the DB2 Alphablox administration pages, you can define custom properties that are available on applications, data sources, and users. After defining a custom property, you can use it via RepositoryBlox server-side Java code. For more information, see "Creating custom user properties" on page 218.

Note: Portlets rely on the portal infrastructure to access user profile information. Keep in mind that portal user profile and DB2 Alphablox user information are maintained separately.

Planning for portlet development

When using Blox components in a portal application, you need to consider the following design requirements during the planning phase. Because Blox-enabled portlets need to be on the same page with other portlets, there are things you need to be aware of in order for all the portlets on a page to work properly together.

- Blox components requires Internet Explorer v5.5 and above. Make sure this requirement is consistent with other portlets served by your portal server. Check the specific browser requirements in the *Installation Guide*.
- Caching should be turned off for applications that have Blox components. Blox components' powerful interactive user interface and support for live data require communications to the server. These abilities do not work if caching is turned on. Because of this, all portlets on the same page need to work in a non-cached environment.
- For the same reason as described above, Blox-enabled portlets do not work in offline mode.
- Data sources used by Blox components need to be defined to DB2 Alphablox through the DB2 Alphablox administration pages. Users with DB2 Alphablox administrative rights need to be set up separately from your portal administration. You may need to work with your DB2 Alphablox administrator and database administrator to get your data sources ready.

Designing an accessible application

For users with disability, it is crucial to provide keyboard equivalents for all actions. For users with limited vision, you also need to take into consideration the limitation of text browsers and screen readers. The UI components in DB2 Alphablox support accessibility for Internet Explorer, with built-in keyboard shortcuts and accelerators.

As you develop and customize your analytic applications, here are some things you need to keep in mind to support accessibility:

- Create accelerator keys for your custom menu options. For custom menu items that appear on the menu bar, you should specify your own accelerator access key. This can be achieved by setting the `accesskey` tag attribute for the `<bloxui:menu>` and `<bloxui:menuItem>` tags. See the Blox UI Tag Reference in the *Developer's Reference* for details.
- Disable or limit the use of the chart component. Non-graphical browsers and screen readers cannot reveal images to visually impaired users. The chart component, given its graphical nature, is not accessible using the keyboard. It is recommended that you remove the chart component from a PresentBlox (`chartAvailable="false"`) or use only the grid component for users with limited vision.
- Render Blox in the provided high contrast theme. Blox user interface rendered using the high contrast theme not only reduces the use of colors to shades of gray, but also honors the font size display preference set in the browser. To use the high contrast theme, set the `theme` URL attribute to `highcontrast`. For example: `http://server/application/file.jsp?theme=highcontrast`.
- Make it easy for visually impaired users to get to the data contents without having to tab through the menu bar or toolbars. The `<bloxui:accessibility>` tag is designed to enhance the user experience for users with disabilities. It provides a tag attribute to allow users to skip the menu bar or toolbars and get right to the data. It also lets you provide keyboard access to open dialogs that have lost focus. See the Blox UI Tag Reference in the *Developer's Reference* for details.

Designing an accessible application usually involves personalization via a user profile. One way to achieve this is through a custom user property in DB2 Alphablox. For example, you can specify a custom user property called `accessible` through the DB2 Alphablox Admin Pages. For visually impaired users, this property will be assigned a specific value such as "Yes." You can then access the value using the RepositoryBlox `getUserProperty()` method, and programmatically hide the chart component in a PresentBlox.

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<html>
<head>
   <blox:header/>
</head>
<body>
<blox:data id="myData" dataSourceName="QCC-Essbase"
   query="!" />
<blox:repository id="myRepository" />
<blox:present id="myPresent" visible="false">
   <blox:data bloxRef="myData" />
</blox:present>

<%
   String accessible = myRepository.getUserProperty("accessible");
   if (accessible.equals("Yes")) {
```

```
      myPresent.setChartAvailable(false);
   } else {
      myPresent.setChartAvailable(true);
   }
%>

<blox:display bloxRef="myPresent"/>
</body>
</html>
```

For more information about general application accessibility issues, visit the IBM Accessibility Center. This site provides many developer resources and guidelines that help you learn and develop accessible applications.

Designing for bidirectional languages

Bidirectional (also known as BiDi) languages are languages that are read from right to left, and still have numbers read from left to right. By default, Web browsers interpret the code in an HTML page and display the visual components from left to right. For bidirectional languages such as Arabic and Hebrew, depending on the settings in the browsers or on the users' Windows® systems, visual components might not display automatically from right to left. Internet Explorer users can set the viewing direction through the **View** > **Encoding** menu option, but not all browser versions provide that option. Web page designers can also specify the direction by adding the `dir` attribute to the `<body>` tag and setting its value to `rtl`.

Because the server's locale determines the language in which the Blox components are displayed, when DB2 Alphablox is installed on a BiDi system, the Blox user interface will be in that language. However, by default, the display direction for the individual components in the user interface is still from left to right. In a ChartBlox, a left-to-right direction means the X axis labels are displayed on the left hand side of the chart. A right-to-left direction will put the X axis labels to the right. In a GridBlox, a left-to-right direction means the row headers are displayed to the left hand side of data cells. A right-to-left direction will put the row headers to the right. Depending on your design goal, there are different ways for you to ensure that Blox components are displayed from right to left.

Setting the dir attribute in HTML code

Because the Blox user interface is rendered in dynamic HTML, you can utilize the browsers' ability to interpret the directional instruction specified in the HTML code. This `dir` attribute can be set in the `<body>` tag (`<body dir="rtl">`) or an inner `<div>` tag (`<div dir="rtl">`). This approach instructs the browsers to display the Blox components from right to left, and yet still allows the users to change the display direction through the provided **Grid Options** and **Chart Options** dialogs. In these two dialogs, users of your application can still set the display back to left-to-right.

If you need to dynamically change the direction, one way to do so is by passing your JSP with a parameter. You then set the direction for the output based on the value passed in for that parameter. For example, if you have a `test.jsp` file, you can call it with a parameter as follows:

```
http://myServer/myApp/test.jsp?dir=rtl
```

The `test.jsp` file gets the parameter value and sets the direction in the `<body>` tag:

```
<!--test.jsp-->
<% taglib uri="bloxtld" prefix="blox"%>
<%@ page contentType="text/html;charset=utf-8" %>

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

<html>
<head>
<blox:header/>
</head>

<body dir="<%= request.getParameter( "dir" ) %>">

<blox:present id="myPresent" width="700" height="500" menubarVisible="true">
   <blox:data bloxRef="dataBlox" />
</blox:present>
</body>
</html>
```

Setting the direction in the UI component

The Component base class for all visual components in the Blox UI model has a `setDirection()` method. The default is `DIRECTION_DEFAULT`, which means to respect the direction setting in the HTML code or the browser. If you set a component's direction to `DIRECTION_RTL`, its subcomponents will always be written from right to left, regardless of languages, browser settings, or the direction specified in the HTML code. Your users can still change the display direction through the provided **Grid Options** and **Chart Options** dialogs.

Chapter 5. Using JavaServer Pages and the Blox Tag Library

The use of JavaServer Pages technology in DB2 Alphablox applications enables developers to rapidly create and easily maintain web-based analytic applications. In addition to developing with DHTML technologies (including HTML, JavaScript, and CSS), JSP technologies add dynamic scripting elements that let you tap into the power of Java without having to master Java. This topic explains how JavaServer Pages technology are used within DB2 Alphablox applications.

JavaServer Pages Technology

JavaServer Pages (JSP) technology allows developers to rapidly create and easily maintain web-based analytic applications using familiar DHTML technologies, including HTML, CSS, and JavaScript, along with dynamic scripting elements that enable developers to use Java and server-side processing. This technology also helps developers create applications that are less prone to the vulnerabilities of cross-browser idiosyncrasies. In a nutshell, the primary advantages of JSP technology are:

- Separating content generation from presentation

  Using JSP technology, web page developers can use HTML or XML tags to design and format web application pages. JSP tags and scriptlets allow web page developers to use familiar tag syntax and scripting capabilities to generate pages, with the core program logic being hidden within custom tag libraries and Java beans. Advanced Java developers can use Java to create these reusable components that can be used by web page designers and application developers.
- Emphasis on reusable components

  Most JSP pages rely on the use of cross-platform, reusable components, such as Java beans and servlets. Using JSP makes it easier for web page designers and developers to generate content using Java beans and servlet components. For example, Blox are Java beans that interact with server peers, but developers can use simple tags to define these beans.
- Simplification of web development with tags

  JSP technology enables dynamic content generation by encapsulating much of the functionality in easy-to-use, JSP-specific XML tags. These standard JSP tags are used to interact with JavaBeans™ components, set and get bean attributes, and perform other functions that would otherwise be more difficult and time-consuming to code. The use of JSP custom tag libraries allow DB2 Alphablox and others to create easy-to-use tags that can be used by web page designers and developers, while hiding the complexity that they don't need to be concerned with.

The *Developer's Guide* assumes basic familiarity with JavaServer Pages technology, but even without this knowledge, you can still create some basic DB2 Alphablox applications. The remainder of this topic focuses on explaining how to use JSP with DB2 Alphablox.

To learn more about the JavaServer Pages technology, the following books and web sites are recommended by Alphablox:

Book recommendations

Bergsten, Hans. 2004. *JavaServer Pages*. Sebastapol, CA: O'Reilly & Associates.

An excellent guide to JavaServer Pages and application development without having to be a hard-core developer. The first part, targeted for web page designers and developers, discusses JSP concepts and how JSP fits into web application development. The later programming-oriented parts discuss how to create JSP components and custom JSP tags.

Fields, Duane K.; Kolb, Mark A.; and Bayern, Shawn. 2001. *Web Development with JavaServer Pages (2nd edition)*. Greenwich, CT®: Manning Publications.

Another excellent guide to JavaServer Pages, intended for both web page designers and Java developers. Includes discussions on using the JSP 1.2 and Servlet 2.3 specifications and examples for common web application tasks.

Falkner, Jason (editor). 2001. *Beginning JSP Web Development*. Birmingham, UK: Wrox Press.

This introduction to JavaServer Pages assumes no previous programming experience and only prior HTML experience. While introducing how to build web-based applications, it explains the relevant JSP and Java concepts. By the third chapter, you are creating simple Java beans.

Web sites

JavaServer Pages (Sun) - http://java.sun.com/products/jsp/

Sun invented the JavaServer Pages technology and this is the place for the latest information, including news, specifications, software, and tutorials. In the Technical Resources section, you can get PDF versions of quick syntax reference cards and guides.

JavaBeans (Sun) - http://java.sun.com/products/javabeans/

Includes the JavaBeans specifications, tutorials, and the latest news about JavaBeans technology.

Servlets (Sun) - http://java.sun.com/products/servlets/

Sun's site for the latest in servlet technology, including news, specifications, and tutorials.

JSP Insider - http://www.jspinsider.com/

An excellent source for JSP articles, reference guides, and links to other JSP resources. The *JSP Buzz* newsletter offers news and articles, and is a good way to stay current on JSP products and features.

JGuru - http://www.jguru.com/

This site has many articles on topics relevant to JSP web application development. There are also useful FAQs about JSP and servlets.

Using JavaServer Pages with DB2 Alphablox

With JavaServer Pages technology and DB2 Alphablox, you can rapidly create and more easily maintain sophisticated analytic applications. Although JSP is a server-based technology, it allows you to incorporate standard client-side technologies, including HTML, JavaScript, and Cascading Style Sheets. This allows you, as an DB2 Alphablox developer, to use these technologies to build flexible and extensible applications.

Analytic applications typically employ both client-side and server-side techniques, using the best of both technologies to deliver your applications. The entire Blox API, including the Blox Client API and the server-side Java API, are detailed in the *Developer's Reference*.

Most often Blox, including the presentation Blox, will be defined in JSP files using the Blox tag library. The Blox tag library, developed using JavaServer Pages technology, includes easy-to-use tags for specifying Blox and their properties. Other Blox tags can be used to handle common developer tasks, including application and business logic debugging. Although you could use standard JSP actions (including the `jsp:useBean`, `jsp:setProperty`, and `jsp:getProperty` tags) to develop JSP applications with DB2 Alphablox, the Blox tag library offers almost identical functionality with less effort. The remainder of this topic focuses on the use of the core Blox tags in the Blox Tag Library to define Blox. Other Blox tag libraries, including the Blox Form Tag Library, the Blox Logic Tag Library, and the Blox UI Tag Library are discussed later in this guide.

Server-side programming with DB2 Alphablox

The DB2 Alphablox server-side programming model (SSPM) emphasizes processing application and business logic, whenever possible, on web application servers. DB2 Alphablox offers a rich set of server-side functionality with its support for JavaServer Pages and the Java programming language. In conjunction with powerful application servers, like BEA WebLogic and IBM WebSphere, Alphablox is able to offer a powerful J2EE-compliant environment for developers. DB2 Alphablox offers a Blox Java API that gives developers the full power of Java, JavaServer Pages, JavaBeans components, and Java Servlets technologies.

This guide focuses on teaching you (even if you have limited or no Java experience) how you can tap into the power of Java server-side programming models to rapidly deliver analytic applications. By being task-focused, this guide will help you quickly learn to use the power of DB2 Alphablox for solving immediate business needs.

Using the Blox Tag Libraries

Developed with JavaServer Pages technology supporting JSP custom tags, the DB2 Alphablox Blox Tag Library includes easy-to-use tags that can be used by web page authors and Java developers.

The core Blox tags are used to define the common user interface Blox, including ChartBlox, DataBlox, DataLayoutBlox, GridBlox, PageBlox, PresentBlox, and ToolbarBlox. Blox tags are also available for defining Blox used specifically in building relational reporting applications. Other Blox tag libraries are useful for creating powerful form elements (Blox Form Tag Library), extending the Blox UI (Blox UI Tag Library), handling complex business logic (Blox Logic Tag Library), and supporting URL-based client-side links in a portal application (Blox Portlet Tag

Library). For relational reporting requirements, the Blox Reporting Tag Library, including the ReportBlox and other associated Blox, is discussed in the *Relational Reporting Developer's Guide*.

Before describing the advantages of using the Blox Tag Library, take a look at the following code examples and you should be able to see for yourself advantages in using Blox tags. Don't worry right now about understanding the details of how the Blox tags work — that will be explained shortly. Instead, focus on the layout and readability of the code examples.

If you already know how to use standard JSP syntax to define Java beans, you should be able to understand the following code example without difficulty. If this syntax is new to you, however, you may have lots of questions and be concerned that you have lots to learn before you can begin making any progress. Keep in mind that this example highlights the more difficult way to code Blox, using standard JSP syntax to define a PresentBlox:

```
<jsp:useBean id="myPresentBlox"
  scope="session"
  class="com.alphablox.blox.PresentBlox">
<%
  BloxContext context = BloxContextFactory.getBloxContext(request, response);
  myPresentBlox.init(context,"myPresentBlox");
  myPresentBlox.setProperty("width","540");
  myPresentBlox.setProperty("height","350");

  DataBlox myDataBlox=myPresentBlox.getDataBlox();
  myDataBlox.setProperty("dataSourceName","TBC");
  myDataBlox.setProperty("query","<ROW(Market) <ICHILD Market
     <COLUMN(Year) Year !");
  myDataBlox.connect();
%>
</jsp:useBean>
```

The *Developer's Guide* spends little time on this syntax. If this syntax is new to you, don't worry the Blox tag library provides an easier way to build Blox components and applications. On some occasions standard JSP syntax will be the only way to code a solution, but most of the time you'll be able to use the Blox tags instead. Now compare the previous code example with the following example, using Blox tags to define the same PresentBlox:

```
<blox:present id="myPresentBlox"
  width="540"
  height="350">
  <blox:data
    dataSourceName="TBC"
    query="<ROW(Market) <ICHILD Market <COLUMN(Year) Year !"/>
</blox:present>
```

As you can see, the code is easier to read and maintain. Here are the primary reasons you should be interested in using the JSP custom tag approach:

- Easier to read

  In the PresentBlox example, code created using the Blox tag library is much easier to read— the properties are mapped using tag attributes as simple name/value pairs, which can be formatted for easy readability.

- Easier to code

  Since there is less to type, Blox created using Blox tags can be more rapidly coded. There is no need to add a number of additional lines for initializing and connecting to the data source—it is handled automatically by the Blox tags.

- Easier to maintain

As a result of being easier to read and code, Blox tags should be easier to maintain. Details, including initialization of the Blox (Java beans) and connecting to the data source, are automatically handled for you. You focus on defining the Blox and its properties, and the tag library takes care of making it all work. Also, one of the advantages of encapsulating Java code with tag libraries is that it makes future updates to the Java code easier to manage for both Alphablox and you.

So, let's get started learning about how to use these Blox tags to define the Blox you will be using in your analytic applications.

Accessing the Blox Tag Library

By default, a web page will ignore tags that it doesn't know about. This means that if you put Blox tags on a page without telling the page where to find information about them, it will ignore them.

So, before you take advantage of Blox tags, you need to add a single line, called a taglib directive, to the top of your JSP pages. Here is the JSP `taglib` directive that you should use:

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

This line of code tells the JSP compiler that you intend to use a custom tag library that is located at the URI (uniform resource identifier) specified as `bloxtld`. This URI, `bloxtld`, is a shorthand label that defines the location where your DB2 Alphablox can find the tag library descriptor file (`blox.tld`) for the Blox custom tag library. The `blox.tld` file is located within each application you create, typically at the following location:

```
/webapps/<applicationName>/WEB-INF/tlds/blox.tld
```

The `blox.tld` file determines which tags and tag attributes are supported in DB2 Alphablox applications, and is automatically created in new applications created using the DB2 Alphablox home pages.

Note: If you are curious to learn more about tag library descriptor (`.tld`) files, see one of the recommended JavaServer Pages technology resources listed above in the JavaServer Pages Technology section. To learn more about how DB2 Alphablox applications are created, see the *Administrator's Guide*.

The `taglib` directive can be placed anywhere on a JSP page, as long as it occurs before you use Blox tags on that page. The best practice, however, is to place the taglib directive at the top of your JSP page, above the `<html>` tag, like this:

```
<%@ taglib uri="bloxtld" prefix="blox" %>
<html>
<head>
...
```

Once again, the `taglib` directive notifies your web application server that you intend to use Blox tags, and need the library to be available. The JSP Engine then parses through the JSP page, looking for any tags on a page that begin with the `blox` prefix, as defined in the `taglib` directive, and when it finds one, it executes the Java code in the tag library— you don't need to see it.

Using the Blox header tag

After you've added the `taglib` directive to the top of your page, an important tag you need to include on the page is the Blox header tag (`<blox:header>`). This tag manages the rendering of Blox on your pages, making critical external JavaScript and Cascading Style Sheets (CSS) files available. Also, it adds a few lines of code that manage file caching.

The Blox header tag should be placed somewhere within the `<head>` section of your JSP page, but after the `taglib` directive:

```
<%@ taglib uri="bloxtld" prefix="blox" %>
<html>
<head>
  <blox:header/>
  ...
</head>
```

Minimally, you need to add the shorthand `<blox:header>` tag, as shown in the example above. Depending on your particular application, though, you may need to add nested tags within this tag for performing other important Blox actions. This usage will be explained later in this guide. For details about the syntax and usage of the `<blox:header>` tag, see the *Developer's Reference*.

Note: When using `<jsp:include>` to include a file that has a Blox on a JSP page, you need to add a `<blox:header>` tag to the top of that page so that the Blox doesn't hang at the initializing stage.

Note: When developing applications using framesets and multiple frames from more than one application, you can use the `<blox:session>` tag to help manage user sessions properly. See the *Developer's Reference* for further information on this tag.

The next section, on defining Blox, explains how to use Blox tags from the Blox Tag Library to define Blox on your application pages.

Defining Blox

The following table lists the user interface Blox and their JSP custom tags:

Blox Name
: Blox Tag

ChartBlox
: `<blox:chart>`

DataBlox
: `<blox:data>`

DataLayoutBlox
: `<blox:dataLayout>`

GridBlox
: `<blox:grid>`

PageBlox
: `<blox:page>`

PresentBlox
: `<blox:present>`

Note: Details on these Blox and complete syntax for their tags, attributes, and usage can be found in the *Developer's Reference*.

Note: There are no spaces between the `blox` prefix, the colon, and the name of the tag. If you put a space after the colon, you will generate JSP compiler errors.

Note: In discussions about Blox tags throughout the *Developer's Guide*, you will see references to the shorthand syntax for tags. This is to help you be clear when the tag is being discussed instead of the Blox itself. For example, instead of referring to the Blox GridBlox tag, this guide will frequently refer to the `<blox:grid>` tag.

As discussed earlier in Chapter 1, "DB2 Alphablox applications and the underlying Blox," on page 1, Blox can be standalone or nested as children within other parent Blox, depending on which Blox is being used and their particular usage. By default, a standalone Blox includes nested Blox set to their default values. A PresentBlox, for example, includes a nested ChartBlox, DataBlox, DataLayoutBlox, GridBlox, PageBlox, and ToolbarBlox.

The following table lists the nested Blox components for each standalone presentation Blox:

Standalone Blox
: Nested Blox Components

ChartBlox
: DataBlox, ToolbarBlox

DataBlox
: CommentsBlox [optional]

DataLayoutBlox
: DataBlox

GridBlox
: DataBlox, ToolbarBlox

PageBlox
: DataBlox

PresentBlox
: ChartBlox, DataBlox, DataLayoutBlox, GridBlox, PageBlox, ToolbarBlox

Rather than have to include nested tags for the nested Blox, you can just include just the top-level parent Blox tag (the nested Blox listed above are implicitly included).

A minimal PresentBlox, for example, defined with `<blox:present>` tag would look like this when coded with opening and closing tags:

```
<blox:present id="myPresentBlox"></blox:present>
```

or like this, when using the shorthand method:

```
<blox:present id="myPresentBlox"/>
```

The shorthand method is recommended most of the time — opening and closing tags are only required when content (called the body) needs to be added between the tags.

Note: The minimal examples above include an `id` attribute, since this is the minimum definition required for a Blox to be rendered properly. All parent Blox within an application must be uniquely identified, but nested Blox don't require `id` attributes.

If you included all of the possible nested Blox tags that are implicit to the PresentBlox, this is what the same PresentBlox would look like:

```
<blox:present id="myPresentBlox">
   <blox:grid/>
   <blox:chart/>
   <blox:toolbar/>
   <blox:page/>
   <blox:dataLayout/>
   <blox:data/>
</blox:present>
```

Since the nested Blox are implicitly included, even when you do not explicitly add the nested Blox tag, include nested Blox tags only when you need to include required tag attributes or tag attributes for properties that you need to change from their default settings.

If you placed a PresentBlox tag on a page without any defined nested Blox or defined tag attributes, a PresentBlox would be rendered on the page, but the Blox would not do anything interesting — without defining a data source and query on the nested `<blox:data>` tag at least, no data would be retrieved. In order for Blox to do something useful, you usually need to add tag attributes or nested property tags.

At this point, you should understand how to get a Blox to appear on a JSP page and how to include nested Blox. The next section explains how to add simple tag properties to Blox.

Setting Blox properties using tag attributes

The initial Blox properties that are used when a Blox is instantiated on a JSP page are determined by the settings defined in Blox tag attributes or nested property tags. Property tags will be explained in the next section, "Setting Blox Properties Using Property Tags."

Blox have both common and unique properties. And, just like the implicit Blox that occur within other Blox, all of the properties on Blox have default values. Most of these default values may never need to be changed by you, but when you need to change their values, most of these properties can be exposed and changed using tag attributes. Details about the hundreds of tag attributes that can be used to customize Blox are described in the *Developer's Reference*, but let's take a look at two of the commonly modified attributes as examples.

Without defining a data source and an initial query for a PresentBlox, it would render properly and display `No data available` messages in the grid and chart sections. To retrieve data for display, you must define two DataBlox properties, using the `dataSourceName` and the `query` attributes. As mentioned earlier, to access and modify a nested Blox, you need to add the nested Blox tag within the top-level Blox, then add the required tag attributes or nested property tags. In the following PresentBlox example, a nested DataBlox tag is added with two tag attributes, `dataSourceName` and `query`:

```
<blox:present id="uniqueName">
  <blox:data
    dataSourceName="definedDataSource"
    query="query"/>
</blox:present>
```

If you wanted to change the default chart type setting on a PresentBlox, you would have to add a nested ChartBlox tag, defining an alternate chart type using the ChartBlox `chartType` attribute. In the following example, the PresentBlox will now display a line chart instead of the default bar chart:

```
<blox:present id="uniqueName">
  <blox:data
    dataSourceName="definedDataSource"
    query="query"/>
  <blox:chart chartType="Line"/>
</blox:present>
```

To customize Blox to meet your user requirements, you'll be adding and modifying many Blox properties using the tag attributes. In addition to the tag attributes, some Blox require special property tags of their own to define properties. The next two sections discuss how these "nested" property tags are used to define some specific Blox style properties.

Setting Blox properties using style property tags

While most Blox properties are exposed and defined using tag attributes, other properties, including all styles and indexed properties (properties that use an index value to allow multiple instances within the same Blox) are relatively more complex and most include sub-properties that also need to be defined. Some of these properties require the use of their own tags while others can be used as property tags or tag attributes on a Blox.

The following table lists all of the non-indexed Blox properties (properties that cannot have multiple instances, each identified by an index value) that are exposed and defined using property tags:

Property	Associated Sub-properties or Attributes	Applies To
`titleStyle`	`foreground font`	ChartBlox
`footnoteStyle`	`foreground font`	ChartBlox
`labelStyle`	`foreground font`	ChartBlox
`axisTitleStyle`	`foreground font`	ChartBlox

For each of the style properties listed above, a Blox property tag defines the property and its associated sub-properties. Attributes on the tags are used to set the individual sub-properties of the style property. Like the nested Blox definition tags, the Blox property tags are nested within the Blox to which their properties apply. Unlike the Blox definition tags, though, these tags do not define objects, but are used instead to define properties on Blox.

The following GridBlox tag example includes several GridBlox tag attributes, including `id`, `bandingEnabled`, and `defaultCellFormat`. In the body of the GridBlox

tag, you can see a nested property tag (<blox:titleStyle>), being used to define how all of the cells in the grid should appear:

```
<blox:grid id="myGridBlox"
  bandingEnabled="false"
  defaultCellFormat="#,###.00"/>
  <blox:titleStyle
    foreground="red"
    font="Helvetica:10"/>
</blox:grid>
```

Property tags enable developers to code complex properties without having to put all of these sub-properties on a single value string. This is a convenience for readability and coding, helping to reduce the likelihood of coding errors. Also, since long value strings cannot contain line breaks, they cannot be formatted as nicely as the examples above. Using a single value string, the previous example would look like this:

```
<blox:grid id="myGridBlox"
  bandingEnabled="false"
  defaultCellFormat="#,###.00"
  titleStyle="foreground=red,font=Helvetica:10;"/>
</blox:grid>
```

Whether to define styles using tag attributes or nested property tags is up to your personal preference, although being consistent in your approach makes debugging easier. Also, using nested quotation marks in a style tag attribute is trickier.

Usage of the style property tags is explained further under each style property of the *Developer's Reference*.

Setting indexed Blox properties using property tags

Indexed Blox properties, also defined using Blox property tags, include an index value to allow multiple instances of these properties to be used within a Blox. Unlike the previous style property tags, these property tags include `index` attributes to allow handling multiple instances of the same tag within a Blox tag.

There are two important differences between indexed and non-indexed property tags:

- You can have multiple instances of the same indexed property tag within a parent Blox tag.
- The order in which you place indexed Blox property tags in your code affects the outcome, unless you explicitly define the index values in your attribute.

Indexed property tags have a common `index` attribute for defining the order of interpretation when multiple examples exist. The index attribute allows you to:

- script to the indexed properties defined in these property tags
- assign the order of interpretation, so that you don't need to reorder these property tags within a nested Blox (although keeping them in order should help you better interpret the expected behaviors).

If the `index` attribute is not defined, an implicit index value is assigned automatically. The first `index` attribute is assigned a value of 1. If you intend to use multiple indexed property tags and will be scripting to these tags, you should consider adding the index attribute to your tags and assigning values that you can see in your code. This will help ensure that you are scripting to the right tag.

The following table lists all indexed properties, their sub-properties, and the Blox to which they belong:

Index Property	Associated Sub-properties or Attributes	Applies To
cellAlert	index enabled condition value value2 description font foreground background apply format align valign link image image_align scope	GridBlox
cellFormat	index format scope	GridBlox
cellEditor	index scope	GridBlox
cellLink	index description image image_align link scope	GridBlox
generationStyle	index foreground background font align valign	GridBlox

As previously mentioned, when you script to an indexed property tag, it has either an implied or defined index attribute. In the following example, the GridBlox has two GridBlox `cellAlert` tags, but neither of them have `index` attributes defined:

```
<blox:grid id="myGridBlox">
  <blox:cellAlert
    condition="GT"
    value="50"
    scope="{Scenario:Variance}"/>
  <blox:cellAlert
    condition="GT"
    value="50"
    scope="{Scenario:Variance}"/>
</blox:grid>
```

To modify the second `<blox:cellAlert>` tag, your Java method might look like this:

```
myGridBlox.setCellAlert(2,"condition=GT,value=50, background=red,
   scope={Scenario:Variable}")
```

Even though the GridBlox tag doesn't explicitly show an index attribute, the `index` property for the second `cellAlert` property is automatically set to 2. While this works, it would be better to define your cell alerts by setting explicit `index` attributes, like this:

```
<blox:grid id="myGridBlox">
  <blox:cellAlert index="1"
    condition="GT"
    value="50"
    scope="{Scenario:Variance}"/>
  <blox:cellAlert index="2"
    condition="GT"
    value="50"
    scope="{Scenario:Variance}"/>
</blox:grid>
```

Doing this, especially with many more cell alerts defined, would make it easier to know the index values on each of the cell alerts.

Controlling the visibility of Blox components

The `visible` common Blox property allows developers to control the rendering, or display, of a Blox on a JSP page. This property can be applied to the following Blox: ChartBlox, DataBlox, DataLayoutBlox, GridBlox, PageBlox, PresentBlox, RepositoryBlox, and the nested ToolbarBlox. By default, the value for `visible` on these Blox is `true`. You can set the `visible` property to `false` and later use the `<blox:display>` tag to display it after you are done with some processing logic.

Note: When using the DHTML client, Blox JavaScript objects are created on the client page, allowing the page to communicate with DB2 Alphablox using the Client API. But, if the visible property is set to false, the client-side JavaScript Blox object will not be created.

Details about the visible tag attribute can be found in *Developer's Reference*.

Processing logic before rendering

In more advanced analytic applications, you may find the need to use Java methods in scriptlets to process some business logic before making a Blox visible on a JSP page. In these instances, you can set the `visible` attribute of the Blox to `false`, then use the Blox display tag (`<blox:display>`) to control the visibility of a Blox.

Before rendering a view to a JSP page, you can set the `visible` attribute of the Blox to `false`, include your processing logic code, then render Blox after the processing has been completed.

The following example shows a PresentBlox with the `visible` attribute set to `false`, followed by a scriptlet with some processing logic using Java, and finally the `<blox:display>` tag, resulting in the PresentBlox being displayed on a page:

```
<blox:present id="myPresentBlox"
  visible="false"
  ...
/>

<%
  your processing logic would go here
%>

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

In this example, if you did not set the PresentBlox's `visible` attribute to `false`, the JSP container would have rendered two Blox on the page, one before the processing logic was done and one after. And, the first PresentBlox would not show the effects of the logic you performed.

For Blox such as the RepositoryBlox and the DataBlox, that are not visible on a page anyway, setting the `visible` property to `false` will have no effect. For both of these Blox, the `visible` property is ignored since these Blox are never visible.

Rendering Blox on multiple pages

The <blox:display> tag also comes in handy when you need to create a Blox on one page, or a frame in a frameset, but render the same Blox on a different page. There are two common instances where this might be useful: when you have a Blox on a page and you would like to have custom pages for printing or exporting to Microsoft Excel. When a Blox view is exported to Microsoft Excel, for example, you can define your Blox on one page that has an "Export to Excel" button. When a user clicks on that button, you could load a page into Excel displaying only the grid and the chart, without unnecessary text or buttons from the originating page appearing.

Also, once a Blox has been instantiated during a session, its current view can be made available to other pages using the <blox:display> tag.

For details about the <blox:display> tag, see the *Developer's Reference* and the "Creating custom print pages using the <blox:display> tag" on page 156 of this guide.

Blox utility tags

Some Blox tags are not used to define Blox that appear on a page, but instead provide access to additional functionality. Here is a brief description of the Blox utility tags, but details can be found in the *Developer's Reference*.

Blox header tag

The Blox header tag (<blox:header>) was described earlier in "Using the Blox header tag" on page 34.

Blox context tag

The <blox:bloxContext> tag is similar to the <blox:header> tag in that it creates the appropriate BloxRequest, BloxResponse, and BloxContext objects depending on the actual request and response type (HTTP or portlet-based). However, it does not output any themes or JavaScript code for rendering. An example of the use of this tag is when you do not have a Blox on a certain JSP page but need to access Blox context information on the portlet that contains other JSP pages with Blox. Since both the <blox:bloxContext> tag and the <blox:header> tag attempt to declare the same variables, they cannot coexist in a JSP page.

Blox debug tag

The Blox debug tag (<blox:debug>), another special tag, can be added to a JSP page to have useful debugging information sent to the system console.

More information about the use of the Blox debug tag can be found in the Troubleshooting section. For information on using the system console, see the *Administrator's Guide*.

Blox display tag

The <blox:display> tag, discussed earlier, is useful for either rendering a Blox after some processing logic has occurred or rendering a Blox on different pages than it was originally created on. Details about using the <blox:display> tag can be found in Chapter 15, "Presenting data," on page 151 and the *Developer's Reference*.

Using standard JSP syntax

Blox can be defined using the standard JSP syntax instead of the Blox custom tags. As discussed earlier, Blox tags almost always offer the best method for defining Blox. In some situations, however, you may find that the only alternative is to use standard JSP syntax.

If you have only coded using the standard JSP syntax before and have never used tag libraries, you may prefer to use that syntax than Blox tags because it is familiar. But before you decide to use standard JSP syntax, you should try to use the Blox custom tags for the reasons described earlier in this topic.

Note: If you use both standard JSP syntax and the Blox tags on the same page, you need the following two lines at the top of your page:

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

Note: Blox tags define Blox (Java beans) using a `scope` set to `session`. If you are using standard JSP syntax, you almost always should use your bean (in this case, a Blox) with a session scope. The default for the `useBean` syntax sets the scope to `page`. Here is an example of what this might look like:

```
<jsp:useBean id="regionsPresentBlox"
   class="com.alphablox.blox.PresentBlox"
   scope="session">
```

Next steps

Throughout this section, you've learned how to use the core Blox tags to define presentation Blox and their properties. You've also learned about the Blox utility tags, including `<blox:display>`, `<blox:debug>`, and `<blox:header>`. To learn specifics about the syntax and usage of these tags, see the *Developer's Reference* and this guide.

In the rest of this *Developer's Guide*, you will be introduced to commonly encountered tasks and some potential solutions. The *Developer's Guide* and the *Developer's Reference*, along with some good JavaServer Pages references, should take you a long way towards helping you develop sophisticated analytic applications for your users.

Chapter 6. Blox Form Tag Library

The Blox Form Tag Library includes FormBlox and other tags for generating HTML form elements with built-in enhancements. Some of the tags automatically generate selection lists for data sources, dimensions, and dimension members. Others can be used to manage radio buttons and checkboxes, or to create tree controls for navigation and other purposes. And, when you use these tags, persistence of state is handled during the session. As a result, you do not need to write additional Java or JavaScript code to manage the persistence of selections during a user's browsing session. The checkboxes maintain their checked state, the last radio button selected stays selected, and tree menus maintain their state even if a user leaves that page and returns to it later during the same session.

Using the Blox Form Tag Library

The FormBlox and related tags are defined in the `bloxform.tld` file. When you create a new DB2 Alphablox application, this file is automatically included in the following directory:

<application_dir>/WEB-INF/tlds/bloxform.tld

Note: If the TLD file is not found, or is accidentally deleted, a copy of the current version of this TLD file can be found in the following directory:

```
<db2alphablox_dir>/bin/
```

The FormBlox and related tags are collectively defined in the `bloxform.tld` file. When you create a new DB2 Alphablox application, this file is automatically included in the following directory:

```
/<applicationContext>/WEB-INF/tlds/bloxform.tld
```

To use Blox form tags, the following taglib directive must be included at the top of JSP pages for the tag library to be recognized.

```
<%@ taglib uri="bloxformtld" prefix="bloxform" %>
```

Overview of FormBlox components

The FormBlox components are briefly described below and some simple examples of usage are described. The FormBlox components have been built using the same lower-level Blox UI components that you have access to as a developer, but the work of building these convenient Blox components has already been done for you.

Detailed information (including syntax, usage, and examples) about the FormBlox API and other form-related tags can be found in the Blox Form Tags Reference section of the *Developer's Reference* and in the *Blox API Javadoc* documentation.

FormBlox component categories

The FormBlox components of the Blox Form Tag Library can be grouped into four categories: form controls, metadata selection lists, time schema selection lists, and tree controls. Below is a brief overview of these component groups.

Basic form controls

This group of FormBlox components create basic HTML form controls, including checkboxes, text fields or text areas, radio buttons, and selection lists. For the most part, these Blox components offer similar capabilities as the standard HTML elements, but also have the benefit of maintaining state throughout a session. When a user event occurs, such as checking a checkbox or selecting an item in a selection list, the changed value is sent to the appropriate object without refreshing the page.

FormBlox Component
Description

CheckBoxFormBlox
Creates checkboxes (either individual or grouped) for selecting or deselecting items using HTML `<input type="checkbox">` tags.

EditFormBlox
Creates text fields or text areas using HTML `<input type="text">` or `<textarea>` tags.

RadioButtonFormBlox
Creates radio button form controls with `<input type="radio">` tags.

SelectFormBlox
Creates drop-down and scrolling selection lists using the HTML `<select>` and `<option>` tags.

Metadata selection lists

This group of FormBlox components create specialized HTML selection lists that generate selection list options from data source metadata. These include selections lists for data sources, multidimensional databases, cubes, dimensions, and members. Since these lists are dynamically generated, you do not need to worry about remembering to add or remove list options.

Unlike other basic HTML selection lists, these metadata selection lists maintain state throughout a user session. When a user event occurs, such as checking a checkbox or selecting an item in a selection list, the changed value is sent to the appropriate object without refreshing the page.

FormBlox Component
Description

DataSourceSelectFormBlox
Creates a dynamically-generated HTML selection list of DB2 Alphablox data sources.

CubeSelectFormBlox
Creates a dynamically-populated HTML selection list containing the available cubes in a specified DB2 Alphablox data source.

DimensionSelectFormBlox
Creates a dynamically-populated HTML selection list of the available dimensions in a specified multidimensional cube.

MemberSelectFormBlox
Creates a dynamically-populated HTML selection list including the available members in a selected dimension.

Time schema selection lists

The time schema-related Blox Form tags allow developers to dynamically generate selection lists for users to choose common business time periods and time units.

These selection lists can be used to drive analytic views that users see. Here is a summary of the two major time schema tags, the TimePeriodSelectFormBlox and the TimeUnitSelectFormBlox.

Blox Component
Description

TimePeriodSelectFormBlox
Creates selection lists offering common business time periods, including current week, current month, month to current, quarter to current, year to current, last quarter, last two quarters, four quarters, last two months, last three months, last six months, last year, and last two years. Can include custom time periods.

TimeUnitSelectFormBlox
Creates HTML selection lists offering time unit options, which can include day, week, month, quarter, and year.

Tree controls

This category includes one item, the TreeFormBlox. The TreeFormBlox creates tree controls composed of folders and items. These folders and items, when selected, which can have actions associated with them. A TreeFormBlox can be used to create hierarchical selection lists and navigation menus. It can also use the HTML form POST method. If enabled, items and folders can be dragged and dropped within the tree control.

Blox Component
Description

TreeFormBlox
Creates DHTML tree controls that can be used for creating hierarchical selection lists as well as navigation menus.

Getting and setting Blox and JavaBeans component properties

The Blox Form Tag Library includes two nested tags, `<blox:getChangedProperty>` and `<blox:setChangedProperty>`. The `<blox:getChangedProperty>` is useful for passing pass values, or linking, between two FormBlox. The `<blox:setChangedProperty>` can be used between two FormBlox, between a FormBlox and another Blox component (e.g., a DataBlox), or between a FormBlox and other custom JavaBeans components. The `<blox:setChangedProperty>` also includes a `callAfterChange` attribute that can invoke a server-side Java method after a change has happened. The boolean `debugEnabled` attribute can be useful in debugging unexpected behavior.

Here is a summary of these two nested FormBlox tags:

Blox Component
Description

`<bloxform:getChangedProperty>`
Nested within a FormBlox to target another FormBlox. The specified property value from the target FormBlox is passed to the FormBlox with this tag.

`<bloxform:setChangedProperty>`
Nested within a FormBlox to target any other JavaBeans components, including other FormBlox, Blox, or custom beans. The specified property value from the owning FormBlox is passed to the target bean.

Detailed information (including syntax, usage, and examples) about the `<bloxform:getChangedProperty>` and `<bloxform:setChangedProperty>` tags can be found in the Blox Form Tags Reference section of the *Developer's Reference* and in the *Blox API Javadoc* documentation.

FormBlox event model

If you find it necessary, you can write your own event handlers for FormBlox components. The simple FormBlox event model provides before and after values from a control whenever it changes. The FormBlox components are not intended to be comprehensive solutions for development, but offers a simple event model that can be used to handle basic events, which occurs most of the time. If you need to create more sophisticated components that can handle more complex requirements, you can build your own using the Blox UI Model components, which support a richer event model as well.

Examples using FormBlox tags

Many examples in Blox Sampler and elsewhere include the use of FormBlox components. And, throughout this guide, examples appear using FormBlox. The FormBlox components are a great resource for developers, handling many of the onerous tasks involving coding for dynamic generation of lists and state management. Highlighted below are some examples that make use of various FormBlox components.

Ad Hoc Analysis using DataSourceSelectFormBlox
In Blox Sampler, under the Using FormBlox and Logic Blox section, there is an example of using a DataSourceSelectFormBlox to create a simple view for users to select a data source, then use a fully interactive PresentBlox to analyze using the cube specified. A step-by-step description of the code used in this example can be found in "Setting different data sources using DataSourceSelectFormBlox" on page 112, in the Connecting to Data topic.

Query Builder
The Query Builder, found under the Workbench section of the Assembly tab in the DB2 Alphablox Admin pages, is an interface that helps developers generate multidimensional query statements for use with DB2 OLAP Server, Hyperion Essbase, and Microsoft Analysis Services. Behind the scenes, if you view the source code for Query Builder, you will find instances of FormBlox components being used, including the DataSourceSelectFormBlox, the CubeSelectFormBlox,

Specifying Report Options using FormBlox
Another good example of usage of FormBlox components can be found in the Using HTML Form Elements example under the Interacting with Data section of Blox Sampler. In this example, the RadioButtonFormBlox and the CheckboxSelectFormBlox are used to select report options.

Navigation Menu Using TreeFormBlox
When you open the Blox Sampler application, the navigation menu that opens in the left frame uses the TreeFormBlox. See the `navigation.jsp` file in the Blox Sampler application for an example of a large navigation menu created using this FormBlox.

Report Templates in FastForward Applications
The Alphablox FastForward application makes extensive use of FormBlox components to build report templates and for the navigation menu. To examine the code used in the Alphablox FastForward application, create a new copy of the application. Code examples of the use of FormBlox

components are also described in Chapter 26, “Working with DB2 Alphablox FastForward,” on page 251.

Chapter 7. Blox Logic Tag Library

The Blox Logic Tag Library includes more easy-to-use tags that can be used for handling time period selections, manipulation of multidimensional database queries without a user needing to know how to create Essbase report scripts (for use with DB2 OLAP Server and Essbase) or MDX statements (for use with Microsoft Analysis Services) and member security.

Using the Blox Logic Tag Library

Tags available in the Blox Logic Tag Library are defined in the `bloxlogic.tld` file. When you create a new DB2 Alphablox application, this file is automatically included in the following directory of your application:

<application_dir>/WEB-INF/tlds/bloxlogic.tld

Note: If the TLD file is not found, or is accidentally deleted, a copy of the current version of this TLD file can be found in the following directory:

```
<alphabloxDirectory>/bin/
```

To access the Blox Logic Tag Library on a page, the following JSP taglib directive needs to be included:

```
<%@ taglib uri="bloxlogictld" prefix="bloxlogic" %>
```

Blox Logic Tag Library components

The major Blox Logic components in the Blox Logic Tag Library include the MDBQueryBlox, the MemberSecurityBlox, and the TimeSchemaBlox summarized below.

Detailed information (including syntax, usage, and examples) about the Blox Logic Tag Library components can be found in the Business Logic Blox and TimeSchema DTD Reference section of the *Developer's Reference* and in the *Blox API Javadoc* documentation.

Logic Blox
: **Description**

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 with the data query re-executed.

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

Creates a time table for a given data source based on your definition of a time schema. 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), generation levels (for Year, Quarter, Month and Week), start date of the time period in the cube, whether normal calendar time or weekly time should be applied, and if the length of a year is exceptional (such as 48-week year).

Using MDBQueryBlox components to select products

Through easy-to-use tags, the MDBQueryBlox can be used to manipulate multidimensional queries without having any logic specific to DB2 OLAP Server, Hyperion Essbase, or Microsoft Analysis Services.

In Blox Sampler, under Using Logic Blox, there is an example of using the MDBQueryBlox with a PresentBlox to allow users to select a product from a select list (created using a MemberSelectFormBlox). Here we'll quickly go through the major steps in creating a similar page:

1. Add the JSP `taglib` directives for the Blox tag libraries to be used on the page.

```
<%@ taglib uri="bloxtld" prefix="blox" %>
<%@ taglib uri="bloxlogictld" prefix="bloxlogic" %>
<%@ taglib uri="bloxformtld" prefix="bloxform" %>
```

2. Define the DataBlox that will be used, setting the `visible` attribute to false and enabling the use of alias member names by setting `useAliases` to `true`.

```
<blox:data id="dataBlox"
     visible="false"
     dataSourceName="QCC-Essbase"
     useAliases="true" />
```

3. Specify the lists of tuples to be used on the column, row, and page axes.

```
<!-- Column Time tuples -->

<bloxlogic:tupleList id="timeTuples">
  <bloxlogic:dimension>All Time Periods</bloxlogic:dimension>
  <bloxlogic:tuple>
    <bloxlogic:member>Qtr 1 01</bloxlogic:member>
  </bloxlogic:tuple>
  <bloxlogic:tuple>
    <bloxlogic:member>Qtr 2 01</bloxlogic:member>
  </bloxlogic:tuple>
</bloxlogic:tupleList>

<!-- Column Measures tuples -->

<bloxlogic:tupleList id="measuresTuples">
  <bloxlogic:dimension>Measures</bloxlogic:dimension>
  <bloxlogic:tuple>
    <bloxlogic:member>Sales</bloxlogic:member>
  </bloxlogic:tuple>
  <bloxlogic:tuple>
    <bloxlogic:member>Sales % of All Locations</bloxlogic:member>
      </bloxlogic:tuple>
</bloxlogic:tupleList>

<!-- Page tuples -->

<bloxlogic:tupleList id="pageTuples">
  <bloxlogic:dimension>Scenario</bloxlogic:dimension>
  <bloxlogic:dimension>All Products</bloxlogic:dimension>
  <bloxlogic:tuple>
```

```
    <bloxlogic:member>Actual</bloxlogic:member>
    <bloxlogic:member>All Products</bloxlogic:member>
  </bloxlogic:tuple>
</bloxlogic:tupleList>
```

4. Add a MemberSelectFormBlox for users to be able to select products from the Product dimension.

```
<!-- MemberSelect FormBlox for the Product dimension. On change event,
     MemberSelect FormBlox will change the pageTuples.
-->

<bloxform:memberSelect id="selector"
  visible="false"
  dataBloxRef="dataBlox"
  dimensionName="All Products"
  rootMemberName="100"
  selectedMemberName="100">
  <bloxform:setChangedProperty
    formProperty="selectedMembers"
    targetRef="pageTuples"
    targetProperty="listFromMetadataMembers"
    callAfterChange="changed"/>
</bloxform:memberSelect>
```

5. Add an MDBQueryBlox.

```
<!-- The MDBQuery creates a query from the 2 column tuples,
     the page tuple and the row query fragment
-->

<bloxlogic:mdbQuery id="query"
  dataBloxRef="dataBlox">
  <bloxlogic:axis type="columns">
    <bloxlogic:crossJoin>
      <bloxlogic:tupleList tuplesRef="timeTuples" />
      <bloxlogic:tupleList tuplesRef="measuresTuples" />
    </bloxlogic:crossJoin>
  </bloxlogic:axis>
  <bloxlogic:axis type="rows"
     queryFragment='<ROW ("All Locations") <ICHILD "All Locations"' />
  <bloxlogic:axis type="pages">
    <bloxlogic:tupleList tuplesRef="pageTuples" />
  </bloxlogic:axis>
</bloxlogic:mdbQuery>
```

6. Add the PresentBlox, referring to the DataBlox specified earlier

```
<blox:present id="presentBlox"
  visible="false">
  <blox:data
    bloxRef="dataBlox" />
</blox:present>
```

7. Add the rest of the page to render the Blox and layout the view

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

<body>

<table width="100%" height="400">
  <tr>
    <td align="center" height="10">Product: <blox:display
       bloxRef="selector" /></td>
  </tr>
  <tr>
    <td>
       <blox:display bloxRef="presentBlox" width="100%" height="100%" />
```

```
    </td>
  </tr>
</table>
</body>
</html>
```

See Blox Sampler, under Using Blox Logic Tags, for the complete code and a working example using the MDBQueryBlox and MemberSelectFormBlox.

Listing cube members using MemberSecurityBlox

The MemberSecurityBlox tag allows you to list members in a dimension based on access permission rights. It uses the DataBlox `suppressNoAccess` property to filter members and can take multiple root members. It can also be used to specify multiple `dimension:members` pairs for filtering.

Here's an example of how a MemberSecurityBlox can be used to :

1. Add the JSP page directive at the top of the file specifying the Java class that needs to be accessed.

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

2. Add the JSP taglib directives for the Blox tag libraries you will be using.

   ```
   <%@ taglib uri="bloxtld" prefix="blox"%>
   <%@ taglib uri="bloxformtld" prefix="bloxform"%>
   <%@ taglib uri="bloxlogictld" prefix="bloxlogic"%>
   ```

3. Remember to add the `<blox:header>` tag to the head section of the page.

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

4. Add a DataBlox to the page.

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

5. Add the MemberSecurityBlox tag.

   ```
   <bloxlogic:memberSecurity id="memberSecurityMsas"
     dataBloxRef="myDataBlox"
     cubeName="QCC"
     dimensionName="[Products].[Category]">
     <bloxlogic:memberSecurityFilter
       dimensionName="[Measures]"
       memberName="[Measures].[Sales]" />
     <bloxlogic:memberSecurityFilter
       dimensionName="[Measures]"
       memberName="[Measures].[COGS]" />
   </bloxlogic:memberSecurity>
   ```

6. Add a SelectFormBlox.

   ```
   <bloxform:select id="members"
     visible="false"
     multiple="true"
     size="5" >
   <%
     members.setItems(memberSecurityMsas.getDisplayMemberNames());
   %>
   </bloxform:select>
   ```

7. Add a `<blox:display>` tag to the page where you want the selection list to appear

   ```
   <body>
     <blox:display bloxRef="members" />
   </body>
   ```

TimeSchemaBlox component

The TimeSchemaBlox creates a time table for a given data source based on your definition of a time schema. Using the TimeSchema Data Type Definition (DTD), you can define the structure of the Time dimension by specifying: names of the time dimensions, generation levels (for Year, Quarter, Month and Week), start date of the time period in the cube, whether normal calendar time or weekly time should be applied, and if the length of a year is exceptional (such as 48-week year).

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 XML file containing the definition of the TimeSchema should be named `timeschema.xml` and stored in your application's `WEB-INF` directory. The Data Type Definition (DTD) used to define the TimeSchema XML is described in TimeSchema XML DTD.

Details about the syntax and usage of TimeSchemaBlox and the TimeSchema XML DTD can be found in the Business Logic Blox and TimeSchema DTD Reference section of the *Developer's Reference*.

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.

```
<blox:data id="dataBlox"
  dataSourceName="QCC-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>
```

Chapter 8. Blox Portlet Tag Library

The Blox Portlet Tag Library provides tags that let you define a portlet link or action link in your Blox or UI components. These links allow you to use the Portlet API for portlet-to-portlet messaging. This section introduces the Blox Portlet Tag Library and shows how it can be used to attach a ClientLink-based link or action link to a top-level Blox or UI Component in a portlet.

Note: A top-level Blox is the outmost Blox that nests other Blox. For example, if you have a PresentBlox with nested GridBlox and ChartBlox, the PresentBlox is the top-level Blox.

Overview of Blox Portlet tags

Often times you want a click of a link in one portlet to trigger an update in another portlet. The `ClientLink` object of the Blox UI Model lets you attach a link to a specific Blox UI component. This URL-based link is handled by the browser when the component is clicked. The `ClientLink` object causes the view layer to handle the user's click using its own logic rather than sending the action back to the server. In addition to the URL to load, `ClientLink` also allows you to specify the target window in which the new page is to be loaded and a browser window feature string (for example, `features= "scrollbars=yes,width=300,height=300"`). However, in the portal environment, the link will only work the first time. After the link triggers a page reload, the portlet link becomes stale due to the way the portal server treats each request. Subsequent clicking of the link will not submit a real action. The Blox Portlet Tag Library lets you add a PortletLinkDefinition or ActionLinkDefinition, which provides the following functionality:

- If a PortletLinkDefinition is added, it is used to create a PortletLink object. The PortletLink object is then used to define the actual link to invoke the URI with the specified parameter values. The URL is re-encoded by the portlet each time the page is refreshed, preventing it from getting stale.
- If an ActionLinkDefinition is added, it can be used to create a PortletLink. Or it can be used to obtain a portlet URI for this link by passing an action name to `BloxResponse.getActionURL()` .

When a PortletLinkDefinition or ActionLinkDefinition is combined with Blox, the definition is assigned to the Blox while the PortletLink is used to generate a ClientLink for use within the Blox UI Model. You can use the Blox Portlet tags inside any data presentation Blox, FormBlox, ReportBlox, or any Blox UI Model components that have the concept of a clicked event. While both PortletLinkDefinition and ActionLinkDefinition can be used to create a PortletLink, ActionLinkDefinition also lets you set the action name for this link definition to create the PortletURI. However, you cannot pass the action name to `BloxResponse.getActionURL()` after this definition has been used to generate a PortletLink.

For details on essential code structure and development tips on how to add Blox to your portlet JSP, start with the portlet tutorials in the Getting Started section.

Using the Blox Portlet Tag Library

Tags available in the Blox Portlet Tag Library are defined in the `bloxportlet.tld` file. For portlet projects, you should use a portlet development tool to properly set up the structure and the deployment descriptor file. See the tutorial on building portlets using Rational Application Developer in the Getting Started section for information on how to create a project so all the needed Alphablox Tag Libraries and servlet mapping are properly referenced and specified. For details on how to add Blox components to your portlet, see the portlet tutorials in the Getting Started section.

Note: If the TLD file is not found, or is accidentally deleted, a copy of the current version of this TLD file can be found in the following directory:

`<alphabloxDirectory>/bin/`

Blox Portlet tags are added to a Blox or Blox UI component by nesting the `<bloxportlet:actionLinkDefinition>` or the `<bloxportlet:portletLinkDefinition>` tag inside the Blox tag. This attaches the link definition to the component. Parameters and their values are specified using the nested `<bloxportlet:parameter>` tag:

```
<%@ page contentType="text/html"%>

<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxportlettld" prefix="bloxportlet" %>
<%@ taglib uri="/WEB-INF/tld/portlet.tld" prefix="portletAPI" %>

<portletAPI:init/>

<%
  String bloxName = portletResponse.encodeNamespace("buttonContainer");
%>

<head>
  <blox:header />
</head>

<blox:container id="myButtonContainer" bloxName="<%= bloxName %>"
   width="40" height="20">
   <bloxportlet:actionLinkDefinition action="showData">
      <bloxportlet:parameter name="a" />
      <bloxportlet:parameter name="b" value="2" />
      <bloxportlet:parameter name="c" />
   </bloxportlet:actionLinkDefinition>

<%
   BloxModel model = myButtonContainer.getBloxModel();
   model.clear();
   Button myButton = new Button("button1", "Show Data");
   model.add(myButton);
   model.changed();

%>
</blox:container>
```

You can then get to PortletLink from the named action, and set the link information or parameter values in a scriptlet:

```
<%
   // programmatically set the parameter values for the named Portlet
   PortletLink plink = myButtonContainer.getPortletLink("showData");
```

```
   plink.setParameterValue("a","1");
   plink.setParameterValue("c","xyz");
   myButton.setClientLink(plink.getClientLink());
%>
```

PortletLink is used to generate markup to call into a PortletLinkDefinition. Some of the PortletLink methods include:

- `getClientLink()`: Returns a ClientLink representing this PortletLink to be used within the Blox UI Model
- `getLinkHRef()`: Returns a String representing an `HREF` that could be used in an HTML anchor tag
- `setParameterValue()`: Sets the parameter's value within this link

For more information on ClientLink, see "The DHTML Client API framework" on page 94.

Blox Portlet Tag Library examples

This section includes examples demonstrating the use of Blox Portlet tags inside a Button (a Blox UI component) and a ReportBlox. Each example uses the basic approach to adding an action link or portlet link:

1. Add the `<bloxportlet:actionLinkDefinition>` tag inside the Blox or UI component to attach this link definition, and specify a name for the action using the `action` attribute.
2. Use the nested `<bloxportlet:parameter>` tag to specify the name of the parameter and its value.

You can then get the PortletLink for the named action and set the link information or parameter values in a scriptlet. For details on the APIs, see the com.alphablox.blox.portlet package in the Javadoc documentation. For more examples on how to add a portlet link to a GridBlox and a TreeFormBlox, see the Blox Portlet Tag Reference topic in the *Developer's Reference*.

Adding links to buttons

The following example defines an action named "showData" for a Button with three parameters. The PortletLink's ClientLink is hooked up with the button, so when the button is clicked, values for two of the three parameters for this PortletLink are set. Additional code is needed to use this information, such as in another portlet. This example only demonstrates how to set the link.

```
<%@ page contentType="text/html"%>

<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxportlettld" prefix="bloxportlet" %>
<%@ taglib uri="/WEB-INF/tld/portlet.tld" prefix="portletAPI" %>

<portletAPI:init/>

<%
  String bloxName = portletResponse.encodeNamespace("buttonContainer");
%>

<head>
  <blox:header />
</head>

<blox:container id="myButtonContainer" bloxName="<%= bloxName %>"
```

```
   width="40" height="20">
   <bloxportlet:actionLinkDefinition action="showData">
      <bloxportlet:parameter name="a" />
      <bloxportlet:parameter name="b" value="2" />
      <bloxportlet:parameter name="c" />
   </bloxportlet:actionLinkDefinition>

<%
   BloxModel model = myButtonContainer.getBloxModel();
   model.clear();
   Button myButton = new Button("button1", "Show Data");
   model.add(myButton);
   model.changed();

   // programmatically set the parameter values for the named Portlet
   PortletLink plink = myButtonContainer.getPortletLink("showData");
   plink.setParameterValue("a","1");
   plink.setParameterValue("c","xyz");
   myButton.setClientLink(plink.getClientLink());
%>
</blox:container>
```

Adding links to ReportBlox components

The following example defines an action named ″selectProductCode″ for a ReportBlox. The link is attached to the Product column. When a product name in the report is clicked, the value for the parameter ″code″ is set to the product's code. Additional code is needed to use this information, such as in another portlet. This example only demonstrates how to set the link.

```
<%@ page contentType="text/html" %>
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxreporttld" prefix="bloxreport"%>
<%@ taglib uri="bloxportlettld" prefix="bloxportlet" %>
<%@ taglib uri="/WEB-INF/tld/portlet.tld" prefix="portletAPI" %>

<portletAPI:init/>

<head>
   <blox:header/>
   <link rel="stylesheet" href="/AlphabloxServer/theme/report.css">
</head>
<%
   String reportName = portletResponse.encodeNamespace("myReportBlox");
%>

<bloxreport:report id="report" bloxName="<%= reportName %>" interactive="false">
   <bloxreport:cannedData />
   <bloxreport:filter expression="Sales < 100"  />
   <bloxreport:group members="Area" />
   <bloxreport:sort member="Week_Ending"  />

   <bloxportlet:actionLinkDefinition action="selectProductCode">
      <bloxportlet:parameter name="code" />
   </bloxportlet:actionLinkDefinition>

<%
    PortletLink link = report.getPortletLink("selectProductCode");
    link.setParameterValue("code", "<value member=\"code\"/>");
    String href = link.getLinkHRef();

    String productLink = "<a href=\""+ href + "\"><value/></a>";
  %>
```

```
  <bloxreport:text>
    <bloxreport:data columnName="Product" text="<%= productLink %>" />
  </bloxreport:text>
</bloxreport:report>
```

The following shows how this action might be processed with an `actionPerformed()` method created to utilize this information through the Portlet API:

```
<%
public void actionPerformed(ActionEvent event) throws PortletException {
    String actionString = event.getActionString();
    PortletRequest request = event.getRequest();

    if (actionString.equals("selectProductCode")) {
       String productCode = request.getParameter("productCode");
       // ... use the value of the parameter accordingly ...
    }
}
%>
```

Chapter 9. Blox UI Tag Library

The DHTML Client UI model provides a library of tags to enable easy access to commonly used UI manipulations. The tags are contained in the library `bloxui.tld` and a full listing of each tag with properties can be found in the Blox UI Tags Reference section of the *Developer's Reference*.

DB2 Alphablox provides a tag library for manipulating the Blox, called the Blox Tag Library. The Blox UI Tag Library is complementary to the Blox Tag Library. Developers should use the Blox Tag Library to set data properties, perform general UI manipulations (such as chart/grid orientation, making menu bars visible, and adjust the split pane), and access general DB2 Alphablox functionality, such as cell alerts and calculated members. If you are using the DHTML client and need a higher level of control over the user interface that cannot be provided by the Blox library, the Blox UI Tag Library may provide the functionality that you need.

Blox UI Tag Library categories

The Blox UI tags are grouped into four categories:

Category
: **Description**

Component Customization
: For customizing the UI at the component level. Examples: customizing menus and toolbars.

Custom Layout
: Provide control over the layout of the grid, such as adding blank rows or columns, or producing the grid in a butterfly layout.

Analysis
: Used to incorporate analysis features into your application.

Utility Convenience tags for facilitating the processing of actions. Developers can use utility tags to intercept user selections or take action when the grid changes.

Blox UI tag examples

In this section, a few examples of the many Blox UI tags are shown to give you a flavor of the power of these easy-to-use tags.

Blox UI component customization

The following example shows a component customization tag. These tags can be used to add and remove, or enable and disable, menus in the user interface.

For example, you could disable the tools menu and remove the bookmarks menu by using the following Blox and Blox UI tags:

```
<blox:grid id="testGridBlox"
    width="600"
    height="700"
    bandingEnabled="true"
    rowIndentation="None"
```

```
    commentsEnabled="false">
    <blox:data bloxRef="dataBlox" />
    <bloxui:menu name="toolsMenu" disabled="true" />
    <bloxui:menu name="bookmarkMenu" visible="false" />
</blox:grid>
```

Custom layout tags

This next example shows the use of a custom layout tag to change the display of the grid to a butterfly layout, which moves the row header into the middle of the grid as shown in the following picture:

```
<blox:present id="bfpresent"
  visible="true"
  width="600"
  height="400"
  chartAvailable="false">
  <blox:grid bandingEnabled="true" />
  <blox:data bloxRef="bfdata" />
  <bloxui:butterflyLayout
    scope="{ Scenario:Budget }"
    showOnLayoutMenu="true"
    addSeparatorColumns="false" />
</blox:present>
```

Using the properties of this tag, developers can specify the position of the row headers as well as whether or not separator columns should be introduced between the header and the data.

Analysis tags

Developers can also incorporate analytics directly into their applications using an analysis tag. For example, the assembler might want to incorporate a 'bottom N' calculation into their application:

This view can be achieved using the following PresentBlox tag and the Blox UI `bottomN` tag:

```
<blox:present id="tbnpresent"
  width="600"
  height="500"
  chartAvailable="false">
  <blox:grid bandingEnabled="true" />
  <blox:data bloxRef="tbndata" />
  <bloxui:bottomN
    prompt="true"
    showRank="true"
    number="7"/>
</blox:present>
```

Blox UI analysis tags also include a general tag that enables developers to incorporate calculations into their application.

Utility Tags

Finally, the Blox UI Tag Library offers tags to make processing of user input much easier. The following code sample uses a utility tag to intercept the user clicking on the Pivot menu item to display a dialog.

```
<blox:grid id="testActionFilter"
  width="80%"
  height="500"
```

```
  bandingEnabled="true">
  <blox:toolbar visible="true" />
  <blox:data bloxRef="dataBlox" />
  <bloxui:actionFilter
    className="<%= MyActionFilter.class.getName() %>"
    componentName="dataPivot" />
</blox:grid>

<%!
  public static class MyActionFilter implements IActionFilter
  {
    public void actionFilter( DataViewBlox blox,
                        Component component ) throws Exception
    {
        MessageBox.message( component, "Action Filter",
                                        "Item clicked!" );
    }
}
%>
```

In this example, the `actionFilter` class is defined in the JSP file and then associated with the grid and the pivot menu item. Event filters are further discussed in the Utility Tags section of the Blox UI Tags Reference in the *Developer's Reference*.

More Examples

Check out Blox Sampler for running examples, with source code, of these and other Blox UI Tag Library tags.

Chapter 10. DHTML Client UI Extensibility

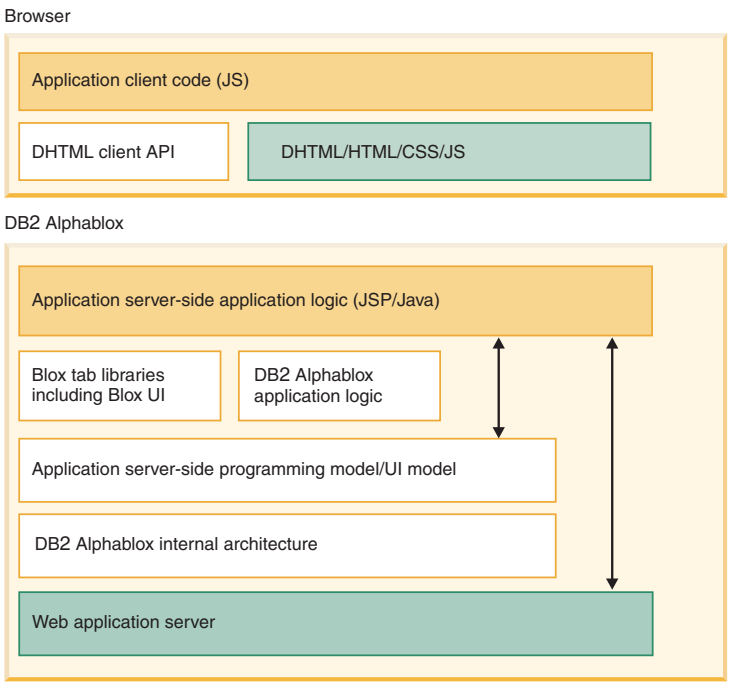
This advanced topic is for developers comfortable working with server-side Java APIs. The Blox UI Model, described in depth here, offers a rich and powerful API for customizing user interfaces beyond the use of simple Blox UI tags. The next topic, Chapter 11, "DHTML Client API," on page 93, discusses the DHTML Client API, which helps application developers bridge the gap between client-side and server-side programming models.

The Blox UI Model

The DHTML User Interface Model is a programmatic API allowing application developers to examine and control the end user's user interface. This API provide direct control over all aspects of the user interface, including what gets displayed as well as the processing of user input. The UI Model API also provides a level of control over the user interface similar to the control developers have over metadata and result set data using the data API. Together, the data and user interface APIs provide a developer total control and customization of all aspects of an application.

As shown in the diagram below, the DB2 Alphablox framework includes:

- DB2 Alphablox internal architecture
- DB2 Alphablox server-side programming model, which includes the UI Model
- DB2 Alphablox application logic
- Blox Tag Libraries, including the Blox UI Tag Library
- DHTML Client API



The Blox UI Model API is a key part of the DB2 Alphablox server-side programming model and is supported by all of the presentation Blox, including

PresentBlox, GridBlox, ChartBlox, DataLayoutBlox, and PageBlox. The server-side `getBloxModel()` method allows the application developer to access the model for a specific Blox instance.

When we refer to the Blox UI Model, we are actually referring to three distinct user interface concepts: components, controllers, and events. These components are summarized here:

Concepts
Description

Components
The individual controls and containers that make up the user interface such as buttons, list boxes, edit fields, grids, and charts. Components exist in a hierarchy of containers to provide structure to the user interface. The resulting model is a logical representation of the user interface presented to the user.

Controllers
Used to process events from components, translating generic component behaviors into application-defined behaviors. For example, a chart may be displayed in response to a user selecting a CheckBox. In this case, a controller interprets the checking of the checkbox as a signal to display the chart. All application logic should reside in the one or more controllers attached to model Components (or containers).

Events
Communicate state changes from the user interface, the underlying application logic, and from the model itself to the Model's components and controllers. Each component and controller has a predefined set of events that it recognizes and understands. Recognized events usually result in modifying the locally stored state of the Component. Application code should use events to trigger application logic based on user actions. For example, when the state of CheckBox is changed on the browser, a `ClickEvent` is generated and sent to the server. When an application's custom controller receives the `ClickEvent` it can perform the associated application behavior such as displaying a chart.

The main points to remember about the UI Model include:

- The Model is a server-side representation of the state of user interface objects on the client. This allows server-side Java code to set up components, deal with interactions between components, and process user input without having to actually write any actual client-side code. The Model itself doesn't really have an input button; instead it allows server-side code to control the state of such a button that is present to the user on the browser. For example, this allows server-side code to determine which grid cells a user has selected.
- Programming directly to the UI Model is optional. Application developers only need to interact with the model if and when they want to add new features or modify existing features.

Purpose of the Blox UI Model

Each presentation Blox already provides a wide array of features and properties that change the Blox's appearance and behavior. These include:

1. **Blox tag attributes** (`bandingEnabled`, `chartFirst`, etc.)
2. **Blox properties**, manipulated in JSP scriptlets by method calls
3. **Blox UI modifier tags**, such as butterfly layout, compressed headers, etc.

If the built-in behavior is what the application developer desired then all that is required is to add an attribute to the Blox tag or call a server-side Java method.

In some cases, the application developer desires a new feature that is not available as a built-in property or wishes to tweak the implementation of built-in feature. Prior to the introduction of the UI Model, the only recourse was to request a change and wait for it to appear is a future product release.

Using the UI model, an application developer can change the behavior of built-in features as well as add new features to the UI. Some examples of these customizations include:

- Adding a new toolbar that provides a custom calculation operations
- Changing the appearance of grid
- Adding checkbox or other controls to grid cells and providing the logic to process the user's selections
- Modifying the right click menu depending on what user interface component the user clicks on

The UI Model does not take the place of easy-to-implement built-in product features, but it does provide application developers with practically unlimited customization possibilities when needed.

Blox UI components overview

Every component in the model corresponds to a particular control on the user interface. These controls include buttons, grids, trees, check boxes, list boxes, charts, menus, toolbars, etc. Changing the state of a component in the UI Model will affect the state of the control in the user interface. Likewise, as the user interacts with the controls in the user interface, the UI Model is updated to reflect the state of the control.

Since the model maintains the state of all the components, even if the user refreshes the page, the server-based model will maintain the state of the components and will use that state when refreshing the page. Server-based application code can, at any time, inspect the state of any of the components used in the user interface and does not need to store or mange state information separately. For example, if a Checkbox component is added to the UI Model, the `Checkbox.isChecked()` method will provide up-to-date information regarding the checked state.

Within the UI Model, Components are arranged in a hierarchy that provides both formatting control as well as a way to centrally manage sets of primitive Components. This hierarchy is made possible by using one or more ComponentContainers, which, in turn, can contain Components as well as other ComponentContainers.

The resulting hierarchy might look something like this for a simple dialog:

Dialog ComponentContainer ComponentContainer CheckBox RadioButton RadioButton ComponentContainer Button Button

UI Model components can be changed, modified, added, or deleted at any time during the lifetime of a Blox. This allows the user interface to change as the user interacts with the interface, selecting options and features. When changing

components after the initial page has been delivered to the browser, the developer must invoke the `changed()` method as follows:

- If a component is modified either directly or indirectly (i.e., its style is changed), then `changed()` should be invoked on the component itself
- If a component is added or deleted, then `changed()` should be invoked on the parent container.

The `changed()` method has no effect (either positive or negative) when called on components before the initial page is delivered to the user.

Components

Every component in the model corresponds to a particular control on the user interface (except those that are hidden). These controls include buttons, grids, trees, check boxes, list boxes, charts, menus, toolbars, etc. In the UI Model, `Component` is the base class for all visual components and containers. Thus, every visual component is derived from Component. The Component class provides the base set of properties and behaviors needed for a visual component to participate in the UI Model framework. Likewise, all non-visual model objects such as styles, layouts and chart axis definitions do not descend from the `Component` base class.

Component names

All components can be assigned a name. The name is changeable during the lifetime of the Component, but most likely it will be set once and remain unchanged for the lifetime of the component. The name serves two purposes:

1. The name is used to identify the component and its role in the model. For example, each menu item has a name that is used to identify its specific function. If components did not have names, then it would be very difficult to identify the components purpose – especially when components are moved around inside of the model. Named components can be moved around inside of the model and still operate normally.
2. The name of a component serves as its "action code." Components that generate action events (for example, `ClickEvent`) use the name of the component, if available, to map the action to a method. This is fully described later in the Controller section.

Handling non-unique component names

Names do not have to be unique and in the default Blox models, some names are shared by different components. This is very handy when multiple components map to the same action and also allows components to be freely moved between different Models.

Since names are not guaranteed to be unique, code that searches for components by name should be prepared to deal with multiple results. To reduce the chances of multiple results from a name-based search, start the search as deep in the component hierarchy as possible. For example, if you are looking for a toolbar button named "sort" you should start the search in each toolbar rather than at the top of the model.

The example below shows you how to lookup all components with the same name in order to perform some action on them (in this case to hide them).

```
ArrayList components =
    myBlox.getBloxModel().searchForAllComponents("componentName");
```

```
for (int i=0; i < components.size(); i++) {
  Component component = (Component)components.get(i);
  component.setVisible(false);
  }
```

Finally, there is no written "law" that names have to not be unique. Custom model code can easily stick to a unique naming convention for the components it adds and not worry about multiple results from component searches.

Built-in names

All components added by the Blox Model's have a standard set of names. Avoid using the same names for unrelated functions. Standard names for all major Blox components are defined in the `com.Alphablox.blox.uimodel.ModelConstants` class file in the *Blox API Javadoc* documentation as well as in the Blox UI Tags Reference section of the *Developer's Reference*. Always use the defined constants in your code rather than using hardcoded strings.

Component titles

The title is used to describe the component to the user. For example, the title added to a CheckBox would be used to tell the user why they are checking the box. Each component uses the title in a slightly different manner, but the purpose is the same. Below is a list of each component and how the title is used.

Component Type
: **How the "title" is used**

Static The displayed value

ComponentContainer
: Title for top-level containers, otherwise ignored

Checkbox
: Displayed after the CheckBox

RadioButton
: Displayed after the RadioButton

Edit Ignored

GroupBox
: Title of the GroupBox

ListBox, DropDownList
: Ignored

Image, StaticImage
: Ignored

Toolbar, Menu bar
: Used in menus to refer to Toolbars, otherwise ignored

Menu, MenuItem
: The menu label

Button
: The button label

Spacer
: Ignored

Containers

Every component must be in a container in order to be displayed. Containers can be arranged in a hierarchy to provide encapsulation of sets of components as well as for layout control. ComponentContainers provide services such as searching for components inside of the container as well as searching for components anywhere in a container's hierarchy.

ComponentContainers are descendents of Component which allows containers to be added to other containers and to share the base Component capabilities. For example, containers can have names, UIDs, borders and background colors.

Components inside of a container are displayed according to the order in which the components were added to the container. The container's layout defines how this order should be interpreted.

Layout

ComponentContainer layouts are limited to specifying the orientation used to display Components in the container. Attach a `VerticalLayout` to a container to cause the components to be stacked vertically. Attach a `HorizontalLayout` to a container to cause the components to be displayed left to right.

```
// Show the components in the container vertically stacked
ComponentContainer.setLayout(new VerticalLayout());

// Show the components in the container left to right
ComponentContainer.setLayout(new HorizontalLayout());
```

Compound components

The Model provides a number of core user interface Components. But, in many cases, it may be desirable to create higher-level Components consisting of some number of core components working in harmony. These "compound components" can then be treated as any other component and can be added to the UI as needed.

Creating a compound Component is as simple as extending the `ComponentContainer` class and adding the desired user interface Components.

For example:

```
Class MyComponent extends ComponentContainer {
    public MyComponent() {
    add(new Static("label:"));
    add(new ListBox());
    }

// Deal with events and add custom behaviors
}
```

The above `MyComponent` class can then be added to any `ComponentContainer` as easily as any of the core Component classes:

```
myContainer.add(new MyComponent());
```

Create compound Components to create reusable custom Components with built-in behavior that can be used as easily as one of the core Components.

Using ContainerBlox

In order to display any UI Model components on the page, they must be placed inside of a Blox frame. ContainerBlox is essentially an empty Blox frame with no predefined DB2 Alphablox application logic. It provides developers with an area on the page to create custom user interfaces using the UI Model's user interface components. Since the ContainerBlox has no predefined behavior, the developer needs to manually add all required Components including menus, toolbars, grids, etc.

An application developer would use a ContainerBlox when the application requires a custom user interface which is not provided by any of the presentation Blox. For example, use a ContainerBlox to place a UI Model Tree component on the page to assist in user navigation. In this example, it is desirable not to inherit any of the existing Blox behaviors since the tree operation is 100% defined by the application.

ContainerBlox can be used like any other Blox with a UI Model such as:

```
<blox:container id="myComponent" >
<%
  BloxModel model = myComponent.getBloxModel();

// Add user interface components and handlers
// Keep in mind that the model is empty
%>
</blox:container>
```

Alternatively, the ContainerBlox can be subclassed in order to create a self-contained custom Blox component based on the Model.

```
class MyComponent extends ContainerBlox
{
   public MyComponent( )
   {
          BloxModel model = myComponent.getBloxModel();

// Add user interface components and handlers
   }
}
```

The above class could then be used in a `<jsp:useBean>` tag, such as:

```
<jsp:useBean id="myBlox" class="MyComponent" scope="session" />
```

Controllers

Controllers provide the application logic that defines the behavior of one or more components. The UI Model's base controller class also provides a number of services, which make processing events easier, as well as provide a framework to easily override standard controller behavior. Any Component or Component-derived class can have an attached controller, however, not all Components need to have a controller and in most cases this will be the norm.

When a Component receives an event, the event is dispatched to the Controller attached to the component. If a controller is not available or the attached controller indicates that the event should be passed along (by returning `false` from the event handler), the event is sent to the component's parent. The process repeats until the event is handled or the component has no parent, as in the case of the top-level container, and the event is just ignored.

Since controllers typically provide application logic that translates the state of many components into a single action, they are most likely to be attached to containers rather than the individual components. A prime example of this are Dialogs where many components will be controlled by the controller attached to the dialog.

Even though containers are more likely to have an attached controller, there is nothing preventing a component from having a dedicated controller. Typically components will have their own controller in the following situations:

- The Component is an addition to the user interface that does a specific task. Most application developer added menu items and toolbar buttons fall into this category and it makes sense to simply add a controller to the added component.
- The Component is part of a container with a controller, but the component does some special processing that affects its state. For example, an edit control could have a controller that specifically validates user input.

The Controller base class

All controller classes must descend from the `Controller` class. This base class provides a number of services, including:

- Converting all received events to method calls. Each event received by the controller causes a method of the form `public boolean handle`*`EventType`*`(`*`EventType`* `event) throws Exception` to be invoked. `EventType` should be replaced with the actual event class such as `SelectionChangedEvent`. If the method does not exist, then the controller will ignore the event.
- Converts `ClickEvents`, which are the primary user action events, into method calls based on the name of the Component the user clicked. For example, a `ClickEvent` on the Button Component named "myButton" will cause the method `public void actionMyButton(ModelEvent event) throws Exception` to be invoked. If the method does not exist, then the controller will ignore the event and pass it along to the next interested party, if any.
- Invokes the `closedDialogName()` method when a dialog created by a component associated with the controller is closed.
- Provides the infrastructure that allows custom code to add event handlers to override the controller's built-in event behavior.

Implied controllers

A number of the core Components have implied controller that cannot be overridden. These implied controllers handle internal state changes so that the Component reflects the correct state before the attached controller can examine the Component's state. For example, when a check box is checked, the CheckBox component will immediately check itself when it receives a `ClickEvent` before any controllers receive the `ClickEvent`.

When a controller receives an event, the controller can safely interrogate the Component for state information and be assured that the Component's model on the server accurately reflects the state of the control on the client.

Blox UI Model events

Events are used to communicate component state changes and actions between the browser and the UI Model. They are also used inside the UI Model to notify controllers of model and property changes.

The main points about UI Model events:

- Most events convey granular user interface actions. For example, button clicks, menu clicks, scrolling, etc.
- Events can be intercepted by application developer code. For example, code can intercept a users click on the drill down menu item.
- Events in the UI Model are similar to JavaScript events in that they are both concerned with user interface actions
- An event is dispatched to the component which generated the event and then to its parents
- Events are also used to transmit information between controllers and components inside of the model
- These internal events allow code to intercept key actions inside of the model as well user interface actions. For example, when the grid creates a cell it generates an event that allows code to customize the cell.

Events are dispatched in a specific order that lets all related components and controllers participate in event processing as follows:

1. Model object – the specific model component generating the event (such as an edit field content change).
2. Model object controller – if the component has a controller, that controller will receive the event
3. Model object's parents - Steps 1 and 2 above are repeated until the top-level container is reached.
4. Model controller – The top-level controller for the model will be the last stop for the event. If the controller does not handle the event, it will be discarded.

In all cases, an event handler can do one of the following:

- Ignore the event and do nothing which causes the event to continue being dispatched
- Absorb the event and optionally take some action, which may include generating additional events. No further dispatching of the event will take place.
- Modify the event and allow it to continue being dispatched.
- React to the event, but allow it to continue being dispatched
- Access the component which generated the event using `getComponent()`
- Access any other properties specific to the event

The following examples describe how you can intercept an event from a Button Component. For each example, the component is a button named `MyButton` and it is being added to a Blox named `blox`. All buttons generate a `ClickEvent` when the user presses the button.

Adding dedicated controllers to components

In this example, we will add a controller to the button itself, which will process the button click. This makes it simple to add behavior when single Components are added to the model, but can make it difficult to coordinate behavior across

multiple Components. You would use this method of handling events when a component does not already contain a controller or when you wish to replace the preexisting controller.

```
<blox:grid ... >
<%
  BloxModel model = blox.getBloxModel();
  Button button = new Button("MyButton");

  button.setController(new Controller() {
    public boolean actionMyButton(ModelEvent event) throws Exception
    {
    // Do something
    }
  });

  model.add(button);
%>
</blox:grid>
```

Adding listeners to preexisting controllers

This example adds the button handler to a Blox model controller. Here we are adding our listener to a controller higher in the model hierarchy. The event will be sent to the component that caused the event (i.e., the Button component), and then percolate up the component hierarchy.

```
<blox:grid .... >
<%
  BloxModel model = blox.getBloxModel();
  model.add( new Button("MyButton"));
  model.getController().addEventHandler(new IEventHandler() {
    public boolean handleClickEvent(ClickEvent event)
        throws Exception {
      if ("MyButton".equals( event.getComponent().getName())){
         // Do something
         return true;
      }
      return false;
   }
  });
%>
</blox:grid>
```

Notice that in all of the above examples, the event handler is being added inside of the Blox tag. This is an important point because you do not want the handler added every time the page is refreshed. All code inside of the Blox tag is only executed when the Blox is initially created and not on every page refresh. There is a similar convention when `<jsp:useBean>` is used with session scope.

Model Dispatcher

Once a component is attached to a Blox model, that component can be used to obtain a model dispatcher. The dispatcher is a service point provided by the top-level container in the model, and offers a number of model-related services. The following services are available from the dispatcher (see `BloxModel` and the `IModelDispatcher` interface):

- Displaying a dialog – causes a dialog to be displayed on the client using `showDialog()`
- Closing a dialog – cause the dialog to be closed on the client using `closeDialog()`

- Obtaining a reference to the top-level container with `getTopLevelContainer()`
- Dispatching events – dispatches an event inside the model using `dispatchEvent()`
- Displaying a browser window – causes the browser to display a new window with the provided URL with `showBrowserWindow()`
- Sending JavaScript commands to the client with `sendClientCommand()`
- Displaying a right click menu – causes the browser to immediately display a right click menu at the specified location using `setAttachedRightClickMenu()`
- Controlling the busy state – allows code to put the browser UI into a busy state until release using `setBusy()`

The most common use of the dispatcher is to display a Dialog as follows:

```
component.getDispatcher().showDialog(myDialog);
```

Components not attached to a Model will not have model dispatcher.

Dialogs

Dialogs are used to collect input from users in order to set options or clarify user intentions. The UI Model makes it easy for application developers to quickly construct and display a dialog to the user. A `Dialog` is a container which extends the base `ComponentContainer` model object by adding two special abilities:

1. The Dialog lives in its own separate, sizable, moveable window on the browser
2. The dialog can optionally stop the rest of the user interface from accepting input until it is dismissed.

Otherwise, Dialogs work like other `ComponentContainers`. Most, if not all Dialogs will also require a `Controller` to interpret a user's selections and take action.

To focus the user's attention on a Dialog, set the dialog's modal property using the `Dialog.setModal(boolean)` method. Modal Dialogs prohibit interaction with other parts of the user interface until they are dismissed. Modeless Dialogs do not prohibit user interface interaction and are best used for Dialogs having an "apply" feature. Multiple Dialogs can be simultaneously displayed with the last displayed Modal Dialog in control of the user interface.

Dialogs will redisplay if the user refreshes the browser page.

Creating simple dialogs

Using the Blox UI Model, you can create dialogs for capturing input from users. To create a dialog, follow these basic steps:

1. Create the dialog from a UI Model resource file.
2. Create a controller which extends `DialogController` to handle all user interactions with the dialog. In most cases OK, cancel, and apply will be the only user actions that the controller is interested in.
3. Attach the controller to the Dialog object.
4. Instruct the model dispatcher to display the dialog.

In the example that follows, the JSP page adds a custom menu item labeled "My Menu Choice," available under Data in the menu bar of a PresentBlox. When a user clicks on "My Menu Choice," the dialog defined in an XML resource file is displayed. In this simple example, the dialog asks the user a question with an OK

or Cancel response. Shown below is the code for the two files, the JSP page displaying the PresentBlox and custom menu item and the XML resource file used to define the dialog window. The JSP page expects the XML resource file to be found at the application's root directory.

JSP page (customDialog.jsp)

The following JSP file will display a PresentBlox on a page, with a custom "My Menu Choice" menu option available at the bottom of the Data menu. To understand what is happening, read the comments in the JSP file.

```
<%@ page import="com.alphablox.blox.*,
  com.alphablox.blox.uimodel.*,
  com.alphablox.blox.uimodel.core.*,
  com.alphablox.blox.uimodel.core.event.*,
  com.alphablox.blox.uimodel.core.Component.*,
  com.alphablox.blox.uimodel.core.grid.*,
  com.alphablox.blox.uimodel.tags.IActionFilter,
  com.alphablox.blox.uimodel.tags.internal.ActionFilterAdapter,
  com.alphablox.blox.data.*,
  com.alphablox.blox.data.mdb.*" %>

<%@ page import="java.io.*,
  java.io.File.*,
  java.util.*" %>

<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxuitld" prefix="bloxui"%>

<%!
  // class needs to be static in order to be used by the
  // bloxui:actionFilter tag

  public static class MyActionFilter implements IActionFilter {
    String dialogPath;

    public MyActionFilter(PageContext pageContext) {
      File ctxPath =
        new File(pageContext.getServletContext().getRealPath(""));
      dialogPath = ctxPath.getAbsolutePath() + File.separator +
         "MyDialog.xml";
    }

    // handle the action for the MyMenuChoice component
    public void actionFilter(DataViewBlox blox, Component component)
      throws Exception {
      System.out.println("actionMyMenuChoice() was called");
      try {
        Dialog dialog = Dialog.createFromResource(dialogPath);
        DialogController dialogController = new MyDialogController(dialog);

        // Attach the controller to the dialog
        dialog.setController(dialogController);

        // Get component from the event so we can get model dispatcher
        // The dispatcher is used to send the dialog to the client
        component.getDispatcher().showDialog(dialog);
      }
      catch(Exception e) {
     System.out.println("actionMyMenuChoice() exception" + e.getMessage());
        throw e;
      }
    }
  }

  public static class MyDialogController extends DialogController {
```

```
    public MyDialogController(Dialog dialog) {
      super(dialog); // must be the first thing in this constructor
      System.out.println("MyDialogController () was called");
    }
    public void actionOk() {
      // Take some action
      System.out.println("actionOk() was called");
      // Invoke default OK handler after taking the action
      super.actionOk();
    }
  }
%>

<blox:data id="analyticsDataBlox"
  dataSourceName="QCC-Essbase"
  query="!">
</blox:data>

<blox:present id="analyticsBlox"
  visible="false"
  width="95%"
  height="45%"
  splitPane="false"
>
<blox:data bloxRef="analyticsDataBlox"/>
  <bloxui:menu name="dataMenu">
    <bloxui:menuItem separator="true" />
    <bloxui:menuItem name="MyMenuChoice"
      title="My Menu Choice" />
  </bloxui:menu>
  <bloxui:actionFilter
    filter="<%= new MyActionFilter(pageContext) %>"
    componentName="MyMenuChoice" />
</blox:present>

<html>
<head>
  <blox:header/>
</head>
<body>
<blox:display bloxRef="analyticsBlox" />
</body>
</html>
```

XML resource file (MyDialog.xml)

In the example above, we created the dialog using a resource file. Alternatively, we could create the dialog by creating and adding the individual components that make up the dialog. The resource file consists of a language localizable XML representation of Components in the dialog and is a simple way to create dialogs, menu bars, and toolbars.

The dialog resource `MyDialog` for the example would look something like this:

```
<?xml version="1.0" ?>
<Dialog name="MyDialog"
   title="My Dialog"
   cache="false"
   modal="true"
   height="150"
   width="500"
   layout="vertical">
<Static title="Do you really want to do this?" />
<ComponentContainer layout="horizontal" alignment="center">
   <Button name="ok" title="OK" />
```

```
   <Button name="cancel" title="Cancel" />
</ComponentContainer>
<Spacer />
</Dialog>
```

Any of the core Model Components can be added to the resource file as children of a container. For more information about resource files, refer to the XML Resource Files Reference section of the *Developer's Reference*.

MessageBox

A `MessageBox` is a modal Dialog with a simple API to enable developers to quickly and easily display a text-based message to the user. The `MessageBox` can collect simple input from the user in the form of `OK`, `Yes/No`, `Yes/No/Cancel`, and `OK/Cancel` responses. The MessageBox is always modal in nature so that the user must respond before continuing to interact with other parts of the user interface.

If the application logic needs to inform the user about some situation, then a `MessageBox` can be sent to the user and the user's response can be ignored.

```
MessageBox.message(myBlox.getBloxModel().getModelDispatcher(),
        "Attention User",
        "Some situation has occurred you should know about");
```

When the application logic is interested in the user's response, it can use the callback mechanism provided by the `MessageBox` class to be informed of the user's wishes. In this case, the `MessageBox` invokes a method on the `IMessageCallback` interface to communicate the user's response back to the application code. Any class can implement this interface in order to receive notification when user responds to the MessageBox.

An example of a Class displaying a `MessageBox` and providing a response handler:

```
class MyClass implements IMessageCallback
{
  public void ask(IModelDispatcher dispatcher) {
    MessageBox.message(dispatcher, "MessageBox Title",
                    "MessageBox message",
                     MessageBox.MESSAGE_OKCANCEL,this);
    }
    public boolean action(MessageBox messageBox,String action){
        // handle the user response
    }

}

// To invoke the above MessageBox

MyClass mine = new MyClass();
mine.ask(myBlox.getBloxModel().getModelDispatcher());
```

DHTML client application logic and flow

The UI Model controls a user interface that is split across multiple tiers - the DB2 Alphablox on one tier and the browser on the other. This is similar to the tiered nature of HTML, but there are some critical differences.

The UI Model updates the DHTML user interface without page refreshes. As the user interacts with the UI, changes are made to the page without refreshing the entire page or frameset.

The UI Model keeps a representation of the user interface on the server that maintains the state of the UI and provides a server-based programmatic interface to the UI. This means that the two tiers need to be kept in sync with each other and changes on the server need to be reflected on the client and vise versa.

Having the user interface split across multiple tiers and based on the HTTP protocol impacts the way server-side code is written and how it handles user actions. Server side code cannot wait around for user responses because those responses happen on different threads and each thread is a limited server resource. Most of this is not all that different than the way other user interfaces operate whether it is done with message loops, callbacks, or handlers. In the UI Model, controllers are used to handle user actions.

Code will typically be structured as follows:

Thread 1 (assume the user hit a button on the user interface)

1. Intercept the `ClickEvent`
2. Create a `Dialog` object
3. Populate the dialog with the required components
4. Attach a controller to the dialog to handle events from the dialog
5. Use the dispatcher to show the dialog (the dialog is not really shown at this point, rather it is queued to be show as soon as possible)
6. Return from the `ClickEvent` handler

Some time passes ...

Thread 2 (assume the user hit OK on the dialog created by in thread 1)

1. Intercept the `ClickEvent` for the OK button in the dialog controller
2. Act on the state of the components in the dialog
3. Cause the dialog to close (again, the dialog is queued to be closed as soon as possible)
4. Return from the `ClickEvent`

The example uses the case of displaying and collecting input from a Dialog, but the general structure is the same for all Model components. The processing of user actions and the creation of the model are always on different threads.

DHTML client is theme-based

The UI Model is driven by the same theme mechanism as the HTML client, although it uses different CSS files and class names (i.e., the DHTML client CSS files do not affect the HTML client and vise versa). All of the CSS class names used by the DHTML client are published and available for override by application developers who what to create their own custom themes. Keep in mind that the UI Model's chart object does not use the theme's CSS setting since it is an image (see "Charting" on page 81).

The CSS files are organized so that the common visual attributes such as color, font, font size, and background image are easy to locate and change. All of these attributes are located in each theme's directory in a file named *themeName*`_dhtml.css` (for example, `coleman_dhtml.css`). Refer to the CSS themes documentation ("CSS themes" on page 157) for a list of the CSS class names and their functions.

The theme's layout is applied to each Blox immediately before it is rendered. If you have custom code that dramatically modified the layout of the UI Model, you should turn off the theme layout application for that Blox. If you do not turn off the theme layout application, it is very likely that your changes will be undone in favor of the theme's default layout.

The `setApplyThemeLayout(boolean)` method controls the application of the theme's layout to the Blox Model. Set this to false to stop the server from applying the layout.

Styles

All UI Model components allow the developer to apply styles to override the default styles provided by the controlling theme. Styles which may be applied to a component include the foreground and background colors, fonts, borders, and other text attributes such as underline, bold and italic.

UI Model styles work in conjunction with the styles defined in the controlling theme's CSS classes. When a style is applied to a component, only the attributes specifically set in the style are applied to the component. Attributes not set will continue to inherit the values supplied by the theme. This allows a developer to set the foreground or background of a component and not worry about all the other attributes such as fonts and borders.

Chart components use a slightly different style mechanism since charts are based on images files which are not affected by a theme's CSS styles. See the chart section for a description of the chart-specific styles.

To apply a style to a UI Model component, you create a Style object and set it in the component as follows:

```
// Make the text in a component bold
Style myStyle = new Style();
myStyle.setBold(true);
myComponent.setStyle(myStyle);
```

Alternatively, you can create a `Style` object using a CSS-like shorthand notation. Please note that the shorthand notation is a limited subset of CSS and only supports those CSS styles that are specifically supported by the Blox model Style object.

```
// Make the text in a component bold
Style myStyle = new Style("font-weight:bold;");
myComponent.setStyle(myStyle);
```

In addition to the `Style` object, developers can also specify a specific CSS class to be used for each UI Model component. You should use this when you want to apply CSS styles that are not supported by the Blox model `Style` object. The class specified can be located in a CSS file or specified on the HTML page using the <Style> element.

```
// Set the CSS class to be used by tha component
myComponent.setThemeClass("myCssClass");
```

Care should be exercised when taking this approach since it also has the effect of overriding the default DHTML theme classes specified for some components. Further, some model components modify the theme class names depending on the state of the component. For example, appending `Disabled` or `Enabled` to the CSS class name.

Setting multiple theme classes

If you want to extend the CSS class used for a component, it is possible to add multiple theme class names separated by a space. For example, to add your CSS class to a component which already has a class defined do the following:

```
button.setThemeClass("myThemeClass "+ button.getThemeClass());
```

Charting

One particularly interesting feature of the UI Model is charting. There are Chart components that allow the assembler to create a chart or modify existing charts. The purpose of this section is to provide a general structure of the chart UI Model and several examples of common tasks so that an assembler can get started with charting extensibility.

Key Terms related to charting

DataSeries

A data series is a list of data values each of which generally represents a single member. For instance, if I have a grid with Sales versus Region (East, West, North, and South), then I would have a data series that has 4 data values (1 for East, West, North, and South) and my data series would have a name ″Sales″. Then, when the data is plotted on the chart, it would make one line with 4 points on it. Different charts accept different types of data series objects. One type of data series used by Bar, Line, Area, and Pie charts is a SingleValueDataSeries where each point on the chart is just a single value. BarDataSeries and LineDataSeries both extend SingleValueDataSeries. Additionally there are multi-value data series as well (such as ScatterDataSeries and BubbleDataSeries). For instance, a ScatterDataSeries has an X and Y value for each data point.Some chart types support only 1 data series. Currently the PieChart only supports a single pie and there is only 1 data series per pie. Most chart types support multiple data series. Imagine a line chart with 3 lines on it. Each line represents a single data series (might be Sales, COGS, and Inventory for instance) Each line has 4 points (East, West, North, South). Each data series is plotted against 1 or more Axis. Axes can either be an OrdinalAxis or a NumericAxis. See the bar chart code sample for how you might set up your own BarChart.

OrdinalAxis

An OrdinalAxis is essentially an axis with string based labels. This is the axis that contains the labels for groups of data. For instance, if I am plotting a data series with values 3, 5, 4, 6 and these values come from the East, West, South, and North Sales numbers, then the OrdinalAxis would contain labels ″East″, ″West″, ″South″, and ″North″. The reason it is called Ordinal is because the order of the data series matches the order of the labels. Each label is essentially a bucket for data points.

NumericAxis

A NumericAxis is an axis on the chart where the actual data values (from the data series) are plotted. For instance, if I have a data series with values 3, 5, 4, 6 then the NumericAxis would have a range of 0 to 10 (controllable) and have tic marks every 1 (controllable) and would show up on the left hand side of the chart (controllable).

The Chart component

There is no single ″Chart″ component that represents all charts. The chart UI Model has a different component for every basic class of charts (`PieChart`, `BarChart`, `LineChart`, `ScatterChart`, `DialChart`, etc.). There are slightly different APIs for each basic class of charts. For example, on a bar chart, several visual properties can be set such as bar border styles, and bar width. These visual properties have no analog in a `LineChart` (lines don't have borders or width although they do have thickness). So `BarChart` has a `setBarBorder(borderStyle)` method while a `LineChart` does not.

All chart classes descend from the `ChartObject`. There are other logical groupings where various chart classes can be treated similarly. For instance, there could be code that works for all `RectangularChart` objects to put grid lines in the chart region.

One important point in using the Chart UI Model involves casting the `Chart` object to the appropriate type (see the above diagram). In order to get to the `BarChart` APIs, you should first check that the `Chart` object is actually a `BarChart`, then cast it accordingly:

```
Component chart = bloxModel.searchForComponent(ModelConstants.CHART);

  // Checks to see if the chart is actually a BarChart
  if (chart != null && chart instanceof BarChart) {
  BarChart barChart = (BarChart) chart;
  // Now we can use the specific BarChart APIs
  barChart.setBarWidth(20);
}
```

Chart objects can be created from scratch (see BarChart example below) but more frequently, an assembler may want to modify an existing `Chart` object which has been created from a ChartBlox. To illustrate how this is best done, see the Change Context (Right-Click) Menu example below) which places a custom context (right-click) menu on the `Chart` object.

Controlling chart settings

There are some common items that an assembler may want to configure. For more details on the specific APIs, refer to the *Blox API Javadoc* documentation, available from the Help menu in the DB2 Alphablox Admin Pages.

NumericAxis

Attribute
Description

`axisTitle`
The title for the axis. Will be displayed if possible (not possible for pies).

`formatMask`
Sets the format mask for how the numbers along the tic marks will be displayed.

`scale` Sets the minimum, maximum and step size values for this axis.

OrdinalAxis

Attribute
Description

axisTitle
The title for the axis. Will be displayed if possible (not possible for pies).

labels Sets the text labels displayed below each tic mark.

DataSeries

Attribute
Description

seriesName
The title for the data series. Will be displayed in the legend.

dataValues
Sets the data values.

Legend

Attribute
Description

legendTitle
The legend title is displayed just above the legend items.

position
Along with Right and Bottom, TopLeft, TopRight, BottomLeft, BottomRight, and Center. Center puts the Legend inside the chart.

legendLayout
Vertical or Horizontal orientation for the legend items. Horizontal means all the legend items go on one line. Vertical puts 1 on each line.

ChartTitle, Footnote, AxisTitle

These are all `ChartStatic` objects.

Attribute
Description

displayText
The displayed text

tooltip
The tooltip that will be displayed if dwellLabelsEnabled is turned on

textStyle
Gives the ability to control the foreground color, font name, font size, and font angle

regionStyle
Gives the ability to control the background color/image, border width, border type, and border color.

Chart event handling

The chart component itself has the same event handling mechanisms as any other Component. However, end-users can click on various portions of the chart (a data point, a label, a legend item, etc.) and these events are handled differently since these are not full-fledged Components (they don't extend the `Component` class). Instead, there is a `ChartComponent` that serves as a superclass to Axis, Legend, AbstractDataSeries, and ChartStatic (title, footnote). One `ChartComponent` can be selected on a Chart at a time. An assembler can intercept this selected event in the Chart's Controller (or any parent Controller) using the example below. Note in the example that only SelectedEvents which contain a special attribute (`Chart.EVENT_ATTR_TARGET`) are processed. This "target" is the `ChartComponent` that was selected:

```
chart.setController( new Controller() {
  public void handleSelectedEvent(SelectedEvent event) {
  Chart theChart = (Chart) event.getComponent();

  if (event.getAttribute(Chart.EVENT_ATTR_TARGET) != null) {
  ChartComponent chartComponent =
    theChart.getSelectedChartComponent();

  // If the user clicked on an OrdinalAxis, then figure
  // out the label and print it out
  if (chartComponent instanceof OrdinalAxis) {
  OrdinalAxis axis = (OrdinalAxis) chartComponent;
  Label label = axis.getLabels()[ axis.getSelectedIndex() ];
  MessageBox.message( theChart, "OrdinalAxis", "Label "
   + axis.getSelectedIndex() + ": " + label.getDisplayText());
  }

  if (chartComponent instanceof Legend
  && theChart instanceof OrdinalChart)
  {

  // Remember that the legend items map to 1 to each data series.
  Legend legend = (Legend) chartComponent;
  SingleValueDataSeries dataSeries = ((OrdinalChart)
    theChart).getAllDataSeries()[legend.getSelectedIndex()];
    MessageBox.message( theChart, "Legend", "Legend Item "
    + legend.getSelectedIndex() + ": " +
      dataSeries.getSeriesName());
      }
  if (chartComponent instanceof AbstractDataSeries
    && theChart instanceof OrdinalChart)
  {

  SingleValueDataSeries dataSeries = (SingleValueDataSeries)
     chartComponent;

  Number value = dataSeries.get(dataSeries.getSelectedIndex());
  MessageBox.message( theChart, "DataSeries", "Data Point "
    + dataSeries.getSelectedIndex() + ": " + value);
    }
   }
  }
});
```

Code samples

Included below are a couple of code examples of working charting.

Custom context (right-click) menus for charts

This example demonstrates attaching a custom right click menu to an existing `Chart` object. When the end user right-clicks on chart data points, axis labels, legends, etc. the custom contextual (right-click) menu will appear. One interesting point to note is that any time an end-user does a data operation (e.g. drill down) or changes chart types, the `Chart` object is rebuilt completely. The custom right-click menu needs to be reattached every time this occurs. The `ComponentRebuiltNotify` event is sent anytime the `Chart` object is rebuilt. A handleComponentRebuiltNotify event handler is to reattach the context menu in these cases. It is crucial that any modifications to the `Chart` object are done inside the `handleComponentRebuiltNotify()` event handler otherwise as soon as the chart type is changed (or there is some data operation), your customizations will be lost:

```
<blox:present id="customRightClick"
  width="80%"
  height="500">
<%
  PresentBloxModel bloxModel =
    customRightClick.getPresentBloxModel();

  // Find the chartBrixModel and its controller
  bloxModel.addEventHandler( new IEventHandler() {
  public boolean handleComponentRebuiltNotify(
    ComponentRebuiltNotify event)
      throws Exception
  {
  Component component = event.getComponent();
  if (component instanceof ChartBrixModel) {
  ChartBrixModel chartBrixModel = ((ChartBrixModel) component);
  Component chartMaybe =
     chartBrixModel.searchForComponent(ModelConstants.CHART);

    if (chartMaybe != null) {
    Chart chart = (Chart) chartMaybe;

  /*** Make the menu ***/

  Menu headerMenu = new Menu();
  headerMenu.add( new MenuItem("headerItem",
    "Header Menu Item ..."));

  // Add a dedicated controller to the header menu
  headerMenu.setController(new Controller() {
   public void actionHeaderItem(ModelEvent event) {
   MessageBox.message(event.getComponent(), "Right Click",
     "Test");
    }
  });
  chart.setRightClickMenu(headerMenu);
    return true;
  }
 }
  return false;
  }
});
%>

<blox:data
  dataSourceName="qcc-essbase"
  useAliases="true"
  query="<ROW (\"All Products\") <ICHILD \"All Products\"
     <COLUMN (\"All Time Periods\") <CHILD \"All Time Periods\" !"/>
</blox:present>
```

Javadoc documentation

See the DB2 Alphablox Information Center for complete UI Model Javadoc documentation. The server-side Javadoc documentation lists all of the UI Model classes and their methods. The *Blox API Javadoc* documentation is available by clicking on the Help menu option in the DB2 DB2 Alphablox Admin Pages.

Blox UI Model examples

Single toolbar

In this example, the two default DHTML client toolbars are combined into a single toolbar. This saves vertical space at the expense of horizontal width. Instead of constructing an entirely new toolbar, the code appends all of the navigation toolbar buttons onto the end of the standard toolbar and then removes the empty navigation toolbar. Since UI Model components can only exist in a single container at one time, adding a component to a container also removes it from its old container.

```
<blox:present id="task1present"
  width="800"
  height="400">
<%
  // Get the model
  PresentBloxModel model = task1present.getPresentBloxModel();

  // Get the two default toolbars
  Toolbar standard = model.getStandardToolbar();
  Toolbar navigation = model.getNavigateToolbar();

  // Add a separator
  standard.add( ToolbarButton.separator() );
  // Move the buttons (don't use an iterator because
  // the component is changing)
    while (navigation.size() > 0)
      standard.add(navigation.get(0));

// Remove the navigation toolbar and update the toolbar menu
    navigation.delete();
    model.populateToolbarMenu();
%>
</blox:present>
```

Disabling context (right-click) menus

This example will intercept the right click event on a Blox's grid and, rather than displaying the right click menu, it will display a message to the user. Since the grid already has an attached controller, the example adds an event handler to that controller which intercepts all RightClickEvents. Returning true from the event handler will stop future processing of the event (return false to allow other handlers to handle the event).

A Blox property exists which will disable the right click menu, but this example is important because it is the foundation for customizing right clicking behavior. For example, the handler could enable or disable the right click menu based on the type of cell (row header, column header, data) or based on the contents of the cell the user has selected.

```
<blox:present id="task2present"
  width="800"
  height="400">
<%
  // Get the model
  PresentBloxModel model = task2present.getPresentBloxModel();

  // Find the grid and its controller
  GridBrixModel grid = model.getGrid();
  Controller controller = grid.getController();

  //Add custom event handler to intercept the right-click event
  controller.addEventHandler(new IEventHandler() {
  public boolean handleRightClickEvent(RightClickEvent event) {
  MessageBox.message( event.getComponent(), "Not allowed",
    "Right clicking the grid is not allowed" );

  // Return true to stop the processing this event
    return true;
    }
  });
%>
</blox:present>
```

Customized context (right-click) menu

When a static header or cell right-click menu is not provided (that is, when getHeaderCellRightClickMenu() or getCellsRightClickMenu() returns null), the grid Brix's contextual (right-click) menu is a copy of the Data menu found on the menu bar of the Blox component. This is the default behavior for a newly created grid Brix in either a PresentBlox or GridBlox. There are several ways of customizing the right-click menu ranging from full replacement to adding items to the default menu. Use the table below to help decide which method is best for the task.

Table 2. Possible solutions for customization of menus

Requirement	Solution
Replace the default right-click menu with a static menu which is not affected by the current cell selection.	Use setHeaderCellRightClickMenu() and setCellsRightClickMenu() methods
Replace the default right-click menu with a dynamic menu based on the current cell selection.	Intercept the RightClickEvent at the GridBrixController level and supply a menu based on the cell selection. Be sure to return ″true″ from the event handler to prevent default handling of the right-click event. See the IModelDispatcher interface for details on launching the right-click menu.
Add menu items to the default right-click menu where the items are not dependant on the current cell selection.	Add the menu items to the Data menu on the menu bar. This only needs to be done once and they will be automatically replicated on the right-click menu.
Add menu items to the default right-click menu where the items are dependant on the current cell selection	Intercept the RightClickEvent at the BloxModelController level, get the current right-click menu using the IModelDispatcher interface, and then add the custom menu items.

This example overrides this default behavior by setting a custom right-click menu for grid header cells and another for grid data cells. The menu items will display a MessageBox to the user indicating the type of cell selected as well as the cell's value.

Although the right click menus added by this example are static (i.e., they do not change based on the cell selected) the action that results from selecting the menu item is very dynamic as the message displayed is adjusted for the cell(s) selected. To do this, the controller attached to each menu item examines the current grid cell selection and tailors the message using that selection.

For simplicity, this example only examines the first cell in the selection list which may not be the actual clicked cell if multiple cells are selected.

```
<blox:present id="task3present"
  width="800"
  height="400">
<%
  // Get the model
  PresentBloxModel model = task3present.getPresentBloxModel();

// Find the grid and its controller
    final GridBrixModel grid = model.getGrid();
    Controller controller = grid.getController();

    // Make and add the header right click menu
    Menu headerMenu = new Menu();
    headerMenu.add(new MenuItem("headerItem","Header Menu Item ..."));
    grid.setHeaderCellRightClickMenu( headerMenu);

// Add a dedicated controller to the header menu
    headerMenu.setController( new Controller() {
    public void actionHeaderItem( ModelEvent event ) {

    // Get the selected cell(s)
    GridCell[] cells = grid.getSelectedCells();
    MessageBox.message(event.getComponent(),"Right Click",
    "You right clicked on a header cell - " +
      cells[0].getValue());
    }
  });

    // Make and add the data cell right click menu
    Menu cellMenu = new Menu();
    cellMenu.add(new MenuItem("cellItem","Cell Menu
      Item ..."));
    grid.setCellsRightClickMenu(cellMenu);

    // Add a dedicated controller to the cell menu
    cellMenu.setController( new Controller() {
    public void actionCellItem(ModelEvent event) {

    // Get the selected cell(s)
    GridCell[] cells = grid.getSelectedCells();
    MessageBox.message(event.getComponent(),"Right Click",
      "You right clicked on a data cell - " +
         cells[0].getValue());
    }
  });
%>
```

Custom grid layout

This example shows you how to create a custom grid layout that can modify the contents of individual grid cells when the grid cell is created. In this example, the layout modifies the cell by adding the Microsoft Analysis Services cell attributes as a tool tip to each header cell (this example will only work for Microsoft Analysis Services data sources).

The custom layout tag provides application developers with a way to customize the entire grid or individual cells by taking care of most of the details of hooking into the grid. These details include dealing with grid rebuild notifications and handling cell modifications as the cells are needed rather than up-front when the grid is built.

By default, the cells in a grid are not actually created until either (1) the user requests a page containing the cell or (2) server-side code requests the cell from the grid. In general, it is better to not cause the grid to create all cells.

The first part of this example demonstrates the `<bloxui:customLayout>` tag, which hooks the custom layout class to the Grid. The tag will create an instance of the layout class and attach it to the grid. It will also manage the layout's appearance on the layout menu which is an optional feature letting the user turn the layout on and off. Further, the tag will save the user's setting in all bookmarks saved by the user.

```
<blox:present id="task5present" width="800" height="400" visible="true">
    <bloxui:customLayout
        className="CustomLayout"
        showOnLayoutMenu="true" />
</blox:present>
```

The class specified by the `className` attribute in the tag must exist on the server's class path or the tag will not be able to create an instance of the layout class. The class itself must extend `com.alphablox.blox.uimodel.layout.AbstractLayout`. Refer to the `AbstractLayout` Javadoc documentation for a complete listing of all the methods and services provided. In this example, our layout class is only concerned with cell creation and ignores grid creation.

The second part of the example is the actual class which extends `AbastractLayout` and performs all of the work. The `getFormatName()` method returns the name of the layout and will be used as the menu item if the layout is added to the menu system.

All of the real work is done in the `layoutCell()` method. Each time a grid cell is created; the `layoutCell()` method is invoked with a reference to the newly created cell. The method checks the cell to see if it is a header cell and if so adds a tool tip to the cell containing the member attribute information.

```
public class CustomLayout extends AbstractLayout {
  protected String getFormatName() {
    return "Custom Layout (show MSAS member attributes)";
  }

  protected void layoutCell(GridCell gridCell,DataViewBlox dataViewBlox)
     throws Exception {

    // Make sure this is a header cell, and it belongs to the grid Brix

    // (i.e. not added by another layout)
    if (!gridCell.isColumnHeader() && !gridCell.isRowHeader())
```

```
        return;
    Property[] properties = getHeaderCellProperties(gridCell,
       dataViewBlox);

    if ( properties.length > 0 ) {
    // Create a tooltip with the properties and add to the cell
    StringBuffer buffer = new StringBuffer(200);
    for ( int i=0; i < properties.length; i++ ) {
      if (i > 0)
        buffer.append("\r\n");
        buffer.append(properties[i].getName());
        buffer.append("=");
buffer.append(properties[i].getValue());
    }
    gridCell.setTooltip( buffer.toString());
  }
}
private Property[] getHeaderCellProperties(GridCell gridCell,
    DataViewBlox dataViewBlox) throws ServerBloxException {
    if (!(gridCell instanceof GridBrixCellModel))
      return new Property[0];

    GridBrixCellModel cell = (GridBrixCellModel)gridCell;

    MDBResultSet results =
       (MDBResultSet)dataViewBlox.getDataBlox().getResultSet();

    Axis axis = results.getAxis(cell.isColumnHeader() ?
        Axis.COLUMN_AXIS_ID : Axis.ROW_AXIS_ID);

    Tuple tuple  = axis.getTuple(cell.isColumnHeader() ?
        cell.getNativeColumn() : cell.getNativeRow());

    TupleMember tupleMember = tuple.getMember(cell.isColumnHeader() ?
           cell.getNativeRow() : cell.getNativeColumn());

MDBMetaData meta =
    (MDBMetaData)dataViewBlox.getDataBlox().getMetaData();

Property[] properties =
    meta.getPropertiesOfMember(tupleMember.getUniqueName());

    return properties;
    }
}
```

Mapping grid cells to underlying result sets

This example of a custom grid layout adds a tool tip to each cell with information about the unique name of header cells and the value of data cells. The example demonstrates two important concepts:

1. The grid layout class can be put directly in the JSP page which may be appropriate if the layout is only going to be used with a single Blox. Placing the class code in the JSP file can quicken the development debugging cycle for all layouts. However when a layout is developed in this manner it should be placed in a separate class file after debugging.
2. The layout uses the `MDBResultSet` to UI Model conversion methods available on the GridBrixModel to map UI Model grid cells to `MDBResultSet` objects.

The first part of this example shows you how to reference a class defined in the actual JSP page. Since most web servers mangle the class name each time the JSP

file is compiled, the code obtains the layout's class name directly from the class itself. The application developer does not need to worry about the class path when the layout class is placed in the JSP file.

```
<blox:present id="lookupGridCell"
  width="700"
  height="500">
  <bloxui:customLayout
    className="<%= CustomLayout.class.getName() %>"
    showOnLayoutMenu="true"/>
</blox:present>
```

The second part of this example demonstrates the Java class that implements the layout. When each UI Model grid cell is created the custom layout invokes `findGridBrixCell()` to obtain the `MDBResultSet` object corresponding to created cell. The value returned from this method will depend on the result set object matching the UI Model cell (this object will be a `Cell`, `TupleMember` or `null`). `GridBrixModel` also provides methods to map cells from the result set to cells in the UI Model.

Keep in mind when implementing custom layouts that the order of cells in the UI Model grid may not match the order of cells in the actual `MDBResultSet`. This is especially true for layouts such as the butterfly layout which moves the row headers.

```
<%!
  public static class CustomLayout extends AbstractLayout {
    protected String getFormatName() {
      return "Custom Layout";
  }
  protected void layoutCell(GridCell gridCell, DataViewBlox dataViewBlox)
    throws Exception {
  MDBResultSet results =
     (MDBResultSet)dataViewBlox.getDataBlox().getResultSet();
  GridBrixModel grid = (GridBrixModel)gridCell.getGrid();
  Object object = grid.findGridBrixCell( results, gridCell);
  if (object == null) {
   gridCell.setTooltip("This cell is not from the MDB result set");
}
  else if (object instanceof Cell) {
    gridCell.setTooltip("Cell\r\nValue:" + ((Cell)object).getDoubleValue());
   }
   else if (object instanceof TupleMember) {
   TupleMember member = (TupleMember)object;
   gridCell.setTooltip("TupleMember" + \r\nUniqueName: " +
      member.getUniqueName() +
   "\r\nDimension: " + member.getDimension().getUniqueName() +
   "\r\nAxis :" + member.getDimension().getAxis().getIndex());
  }
  else
   gridCell.setTooltip("Unexpected object: "
     + object .getClass().getName());
 }
}
%>
```

Chapter 11. DHTML Client API

This topic covers the DHTML Client API, which enables easy access to server-side application logic and APIs using JavaScript methods and allows the assembler to leverage client-side scripting to add value to the application by offloading navigation, some UI manipulation, and entry validation from the server.

DHTML Client API overview

The core logic for an application built for the DHTML client is made up of server-side components such as the UI Model, scriptlets, beans and other supporting classes. As a result, the focus of the DHTML Client API is to enable easy access to server-side application logic and APIs rather than exposing a large, RPC-based API on the client. It also allows the assembler to leverage client-side script to add value to the application by offloading navigation, some UI manipulation and entry validation from the server.

The DHTML Client API is a client framework that provides services such as event processing, error handling, communications services and an RPC mechanism for JavaScript code.

Using the DHTML Client API

The DHTML Client API is used when HTML or JavaScript on the surrounding page needs to interact with one or more Blox on the page. The main uses of the client API include:

- **Invoking server-side application logic**: The DHTML Client API provides methods to directly invoke methods on server-side beans. In addition, return values from the server side beans are returned to the client and converted to the appropriate JavaScript object.
- **Event processing**: Through the API, assemblers can send events to the server to simulate a user interaction with the UI. JavaScript can be used to create event objects such as click events and methods are provided to send the event to the server. This allows HTML buttons and other controls outside of the Blox framework to simulate user interactions. Events may also be used to change the state of components within the model.
- **Intercepting events**: JavaScript methods can be registered as listeners for all events generated by all Blox on the page. The event listener can choose to ignore the event or let the event be processed normally. JavaScript can be written which changes the behavior of the UI and/or processes some user selections on the client.
- **Polling the server for changes**: The client API framework, in conjunction with the server, automatically handles all UI updates and transfer of information between client and server. However, the assembler does have the ability to explicitly poll for changes. This is most often useful if the application is making changes outside of the client framework. Common examples of this would include communicating with the server through other frames or by using an HTTP communications facility such as the XMLHTTP object.

- **Handling errors returned by the server or communications layer**: The client framework will invoke a JavaScript method to handle server and communications errors. Client code can register its own error handler to process these errors.

Later in this section, examples of common uses of the DHTML Client API will be provided.

The DHTML Client API framework

The client framework consists of two main objects, the `BloxAPI` object and the `Blox` object, that power the UI and handle communications with the server. Some related utility objects are also available.

BloxAPI Object

The `BloxAPI` JavaScript object contains a number of generic services used by all Blox on the page. It provides communications services between the client and server as well as a convenient RPC mechanism for JavaScript code. There is exactly one `BloxAPI` object per frame controlling all incoming and outgoing traffic between the server and all Blox in that frame.

The `BloxAPI` object handles the following:

- Polling the server for changes
- Providing APIs for event and error management
- Dispatching changes to the various Blox in the frame
- Handling communication and server errors
- Providing APIs for RPC access
- Providing an API to send events

Blox Object

Each Blox in the frame has an associated JavaScript `Blox` object. The Blox object is responsible for the following:

- Responding to change notifications from the server
- Responding to and handling busy state and busy indication
- Providing DB2 Alphablox 4.x compatibility methods: `isBusy()`, `updateProperties()`, `flushProperties()`, `call()`, and `setDataBusy()`
- Handling and managing Dialogs associated with the Blox
- Handling the right click menu associated with the Blox

Utility objects

In addition to the `BloxAPI` and `Blox` objects, the framework also supports some utility objects, the most important of which include:

- *xxx*`Event` – Specific objects for each type of model event that can be issued by the client. Example: `ClickEvent`
- `Grid` – Provide read-only access to some grid properties such as the list of visible selected cells
- `Exception` - Object used to communicate server exceptions to the client

Sending events

The UI model exposes a number of events that can be issued by the client, such as a `ClickEvent`. For each of these events, the DHTML Client API defines JavaScript objects. As a result, JavaScript can be used to create event objects and sent the event to the server. This allows HTML buttons and other controls outside of the Blox framework to simulate user interactions. Events may also be used to change state on custom model components. For example, the following HTML code will send a `ClickEvent` to a model component with a UID of UID:

```
<input type=button value="Show Dialog"
    onclick="bloxAPI.sendEvent(new ClickEvent('container', UID));" >
```

For a list of events that are exposed to the client, see the *Developer's Reference*.

Initiating Blox UI Model events from JavaScript

It is possible to generate and send UI Model events from JavaScript back to the server. Doing this allows regular HTML controls on the page to simulate the user clicking on the Blox used interface. For example, the Blox's menu can be turned off, but HTML buttons can be placed on the page to give a user to some features of the UI.

The example below will create an HTML button which invokes the data options menu item when clicked.

```
<blox:present id="samplePresent"
  width="700"
  height="500">
</blox:present>
<%
/* In order to send an event from the client, we need the component's UID */
  BloxModel model = samplePresent.getBloxModel();
  Component component = model.searchForComponent(
    ModelConstants.DATA_OPTIONS );
    int uid = component.getUID();
%>

<input type=button value="Data Options"
   onclick="bloxAPI.sendEvent(new ClickEvent('samplePresent',<%= uid %>));">
```

The `input` element's `onclick` event handler is using the BloxAPI to send a `ClickEvent` to the server. The `ClickEvent` is a JavaScript object which takes the Blox name and the UID of the target component. Since the UID is dynamically assigned, the code has to look it up in the model when the page is requested.

Intercepting events

There are two facilities available for intercepting events on the client side:

1. JavaScript code can register a listener for all client-side events. This means that every action a user takes can be intercepted, examined, and either ignored or processed. This approach provides control over just about every user interaction with the UI. For example, each time the user selected a menu item a `ClickEvent` is generated containing the Blox, the UID of the menu item, and the name of the menu item.
2. The UI Model provides a `ClientLink` object that can be attached to most Model Components that have the concept of a clicked action (i.e., generate that `ClickEvent`). The `ClientLink` object causes the view layer to handle the user's

click using its own logic rather than sending the action back to the server. For the DHTML client, any `Component` which has an attached `ClientLink` will be processed on the client in the form of a JavaScript call or the opening of a new browser window.

Intercepting client-side events

This example demonstrates how to intercept UI Model events on the client. A developer would do this when the UI Model event performs some client-side action that does not require server involvement.

The JavaScript `eventHandler()` function will be invoked for all events generated by the UI. Since the handler sees all events, the code must examine the type of event as well as the destination UID (or name) in order to intercept specific UI events. Returning `false` from the handler will allow the event to be processed and sent to the server. Return `true` to stop all further processing of the event.

The example is client-side JavaScript code:

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

bloxAPI.addEventListener(eventHandler);
```

Invoking JavaScript directly from the user interface

Rather than intercepting every event generated by the UI on the client, components can be instructed to invoke client-side JavaScript directly. In this case, the component will not sent a `ClickEvent` event to the server.

The code for assigning JavaScript methods to clickable components resides on the server. The example below finds the options menu item on the Data Menu and forces the menu item to invoke a JavaScript method (alternatively, the menu item can be set to load a browser URL).

```
<blox:present id="samplePresent" width="700" height="500">
<%
  // Find the component
  BloxModel model = samplePresent .getBloxModel();
  Component component = model.searchForComponent(
    ModelConstants.DATA_OPTIONS );

// Create a client link using javascript:protocol method
   ClientLink link = new ClientLink("javascript:
      myJavaScriptFunction( );" );

  // Set the link on the component
  component.setClientLink( link );
%>
</blox:present>
```

When the Data Options menu item is clicked, the client will invoke the JavaScript `myJavaScriptFunction()` function rather than send the event to the server. It is assumed that the JavaScript function has been already defined on the page.

Exception handling

When using `callBean` to invoke server-side code your code should be prepared to handle Java exceptions if your server-side method has the possibility of throwing exceptions. Given the `exceptionThrower()` method on the client bean `myBean`, your JavaScript code should examine the return value to determine if an exception has been thrown before processing the result as follows:

```
var retval = myBean.exceptionThrower();
if (retval.constructor == Exception ) {
    alert( "Exception returned: " + retval );
} else {
    // Process the response
}
```

Invoking server-side logic using the DHTML Client API

There are essentially three methods for invoking server side logic from the DHTML client. The method you choose depends on how much of the process you want the server to automate. The methods are listed here in order of least automation to the most automation.

BloxAPI.call() and Blox.call() methods

This is the same `call()` method that was available in DB2 Alphablox 4. It allows you to invoke URLs on the server, passing arguments as URL parameters. This method can call `rmi.jsp` to provide automated argument passing and bean method invocation. Values returned to JavaScript must be parsed and converted into the desired data types.

For example, the following code uses the `bloxAPI.call()` method to invoke a method on a bean (`MyBean`) that toggles the visibility of the data layout panel.

```
<%@ taglib uri='bloxtld' prefix='blox'%>
<%@ taglib uri='bloxuitld' prefix='bloxui'%>
<blox:present id="callpresent"
  visible="false"
  width="600"
  height="500"
  chartAvailable="false" >
  <blox:grid bandingEnabled="true" />
  <blox:data bloxRef="calldata" />
</blox:present>

<jsp:useBean class="MyBean" scope="session" id="myBean">
<%
  myBean.setBlox(callpresent);
%>
</jsp:useBean>

<html>
<head>
  <blox:header />
<script>
// Use call to invoke method on the bean
function showDataLayout(show) {
  var result =
     bloxAPI.call("rmi.jsp?bean=myBean&method=showDataLayout&arg1="+show);
  alert("Result type: " + typeof result + "\r\n\r\n" + result);
}
</script>
```

```
</head>
<body>
<blox:display bloxRef="callpresent" />

<input type="button" value="Hide Data Layout"
   onclick="showDataLayout(false);" >

<input type="button" value="Show Data Layout"
   onclick="showDataLayout(true);" >
</body>
</html>
```

BloxAPI.callBean() method

This method will call a Java bean on the server similar to the combination of the `call` method and `rmi.jsp` described above. It differs from that combination as follows:

- It works directly with the server in finding and invoking a bean method. There is no extra JSP file involved (i.e., you don't need `rmi.jsp`).
- You can specify data types for outgoing method arguments.
- The return value is converted to a real JavaScript object.
- Most simple data types and arrays are supported as arguments and return values.

To use `callBean` in the above example, simply replace the `call()` invocation with the following:

```
var result=bloxAPI.callBean("myBean","showDataLayout",new Array(show));
```

The clientBean (<blox:clientBean>) tag

The Blox clientBean tag (`<blox:clientBean>`) can be nested inside of the `<blox:header>` tag, and results in the server generating a JavaScript object for the specified Java bean (or Blox). To use `<blox:clientBean>` in the above example, incorporate the client bean tag into the Blox header. At that point, the developer can make normal JavaScript method calls:

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

<script>

// Use ClientBean to invoke method on the bean

function showDataLayout( show ) {
  var result = myBean.showDataLayout( show );
  alert( "Result type: " + typeof result + "\r\n\r\n" + result );

}
</script>
```

Important: In this example, no method was specified in the `<blox:clientBean>` tag. As a result, a JavaScript method will be generated for each public method in the bean. This can result in significant overhead for beans with more than a few methods. As a result, it is recommended that assemblers explicitly list the methods for which JavaScript is generated, and that assemblers keep the number of methods to a minimum.

Note that the only restriction on the use of <blox:clientBean> is that the arguments passed and returned need to be supported by JavaScript, which effectively limits the supported arguments to primitives and arrays.

Using <blox:clientBean> with server-side Blox components

The <blox:clientBean> tag can be used to access server-side Blox from the client as well. Here is an example of how a JavaScript object for a PresentBlox could be generated:

```
<blox:header>
  <blox:clientBean name="myPresentBlox">
    <blox:method name="setDividerLocation">
    <blox:method name="setChartFirst"/>
  </blox:clientBean>
</blox:header>
```

Important: In this example, two methods are exposed. Given the number of methods available on most of the SSPM Blox objects, it is mandatory that the methods used be explicitly listed in the clientBean tag in the header.

In order to use a server-side Blox in the header, define the Blox with visible="false" and then use the <blox:display> tag to render the Blox in the body of the HTML.

When an server-side Blox is used with clientBean, the following special processing takes place:

- The name of the bean on the client has API appended to the end. This is done regardless of the type of server-side Blox. In the above example, the actual JavaScript object would be named myPresentBloxAPI. This is done because, in most cases, there will already be a JavaScript object on the page added by the DHTML client.
- As a convenience, if the DHTML client finds a JavaScript object with the Blox name ending in API, it will allow developers to call the methods directly on the main DHTML client's Blox object. So, even though the client bean is called myPresentBloxAPI, you can call methods on the myPresentBlox Blox object directly. For example, both myPresentBlox.setChartFirst() and myPresentBloxAPI.setChartFirst() will set the chart first.
- If a server-side Blox has a DataBlox or other nested Blox, such as within a PresentBlox, you can access the nested Blox on the client without having to create a separate client bean section. To do this, add methods to the parent Blox's list prefixed by data, grid, chart, dataLayout, toolbar, and page. To invoke the method, use the appropriate getter with the main Blox for example, myPresentBlox.getDataBlox().connect().

The example below is a full JSP page which demonstrates the use of embedded Blox and the API suffix. Note that the data.setQuery in the GridBlox's client bean section which makes that DataBlox method available to JavaScript code.

```
<%@ page import="com.alphablox.blox.uimodel.*"%>
<%@ taglib uri='bloxtld' prefix='blox'%>
<%@ taglib uri='bloxuitld' prefix='bloxui'%>

<blox:data id="gridDB" ... />

<blox:grid id="grid" width="700" height="500">
  <blox:data bloxRef="gridDB" />
</blox:grid>
```

```
<html>
<head>
  <blox:header>
    <blox:clientBean name="grid">
      <blox:method name="setBandingEnabled" />
      <blox:method name="isBandingEnabled" />
      <blox:method name="data.setQuery" />
      <blox:method name="data.connect" />
    </blox:clientBean>
    <blox:clientBean name="gridDB">
      <blox:method name="setQuery" />
      <blox:method name="connect" />
    </blox:clientBean>
</blox:header>
</head>
<body>
...
<!--
   Calling DataBlox methods via the GridBlox. Since the GridBlox
   is a DHTML Blox and appears on the page, the API suffix is optional.
-->
<input type="button" value="Set query via grid"
   onclick="grid.getDataBlox().setQuery('!');
grid.getDataBlox().connect( );">

<!-- Calling datablox methods directly on the DataBlox. Note that
   here the API suffix is mandatory because there is not DHTML client
   Blox for the DataBlox.
-->

<input type="button" value="Set query via datablox"
   onclick="gridDBAPI.setQuery('!'); gridDBAPI.connect();">
<input type="button" value="Toggle grid banding"
   onclick="grid.setBandingEnabled(!grid.isBandingEnabled());">
<blox:display bloxRef="grid" />
</body>
</html>
```

The DHTML Client DOM API

The Internet Explorer DOM is used extensively by the DHTML Client. The client updates portions of the DOM as the user interacts with the client. As this is part of the implementation the DHTML client, the DOM objects and attributes created by the DHTML client will change in future versions.

Important: Developers should not write client-side code that manipulates or traverses the DOM generated by the DHTML client, as the implementation will likely change going forward.

The DHTML Client DOM API is included in Chapter 27, “DHTML client DOM API,” on page 267.

Using multiple frames

The DHTML client treats each frame in an application as a separate entity. This means that each frame containing a `<blox:header>` tag will have its own Client API framework `BloxAPI` object. As far as the server and client API are concerned, Blox in separate frames may as well be in different browsers.

Because Blox in separate frames are treated as separate entities, unexpected results can occur if:

1. Blox in different frames refer to a common DataBlox. In this case, drilling or other navigation operations performed on the Blox in one frame will not cause the immediate update of a Blox dependent on the same DataBlox sitting in a different frame. In practice, this shouldn't occur too often, if at all.
2. Server-side code executed in one frame modifies or otherwise affects Blox in a different frame.

In both cases, only the Blox in the frame causing the modification will be immediately updated. Blox in other frames will not be updated until those frames perform their automatic polls.

If this situation does occur, the automatic poll in its default state will not be adequate since it may take as long as two minutes to update the Blox in all frames. Some suggested options include:

1. Performing a manual poll using the `BloxAPI` object in each frame with affected Blox.
2. Decreasing the poll timer from its default to a faster interval.
3. Avoiding the situation altogether by keeping Blox in a single frame or by not allowing Blox in different frames to depend on the same DataBlox.

Refreshing pages

The DHTML client updates without refreshing the page by changing the contents of HTML elements. Thus, as a user interacts with the client, HTML is constantly changing in order to present new information. However, the browser does not track any of these HTML changes. Instead, the browser caches HTML received when the page was first requested. The browser also displays the HTML of the initial page if you do a view source.

Important: Due to incremental page updates that occur with the DHTML client, the HTML source code viewed from a browser's View Source option will usually not match the current state of the browser. This can make debugging more difficult.

When a user refreshes a page or returns to a page using the browser's Back button, the browser restores the cached version of the page. Because changes to the DHTML client are maintained on the server, the client and server have a method of detecting and handling this situation to insure that the user is viewing the up-to-date representation of the Blox. A side effect of this mechanism is that when refreshing the page or using the Back button, you may see the original state of the Blox momentarily before they are updated to the current state.

Chapter 12. Capturing events using server-side event filters and listeners

You can capture a data analysis event and perform custom actions either *before* or *after* the event is processed on the server. For example, when a user drills down on a member, you can perform some checks to see if the user has the authority to perform the action before the event is processed on the server. This allows you to cancel the action if necessary. Or you might want to sends mail to the finance department each time someone drills down on certain sensitive parts of the database. Event filters are server-side objects that allow you to capture user data analysis events such as drilling down or pivoting *before* the event is actually processed on the server. Event listeners allow you to be notified *after* the user event has been processed.

An important aspect of 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 action is executed on the database and new data is returned to the client. Event listeners, on the other hand, let you perform additional actions after the event is completed successfully on the server. For example, after a hide-only event is completed, you might 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. This can be achieved using the event listeners. Event listeners will only be triggered when the event is completed with no errors.

Event filter objects

The event filter objects are server-side objects that allow you to capture certain user events such as drilling down or pivoting and perform actions *before* the event is actually processed.

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.

To use the event filters, you need to first add the specific event filter object using the common `addEventFilter()` Blox method. Once you add an event filter to DataBlox, PresentBlox, or other user interface Blox, 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.

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 “Event listener objects” on page 104.

Event listener objects

The event listener objects are server-side objects that allow you to be notified of certain user events such as drilling down or pivoting and perform some actions after the event has been processed. 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. To use the event listener, you need to first add the specific event listener object using the common `addEventListener()` Blox method. Once you add an event listener to DataBlox, PresentBlox, or other user interface Blox, 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.

To perform pre-event processing, you should use the event filters. For a comparison of the usage of the two, see "Event listeners compared to event filters" on page 108.

Using event filters and event listeners

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 filters 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`.

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 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 and remove 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>
```

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 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.

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.

Event listeners compared to 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-event and post-event processing, you should specify the complete class names that include the package information.

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 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
bookmark: load	`bookmarkLoad(BookmarkLoadEvent)` in BookmarkLoadFilter	BookmarkLoadEvent methods
bookmark: rename	`bookmarkRename(BookmarkRenameEvent)` in BookmarkRenameFilter	BookmarkRenameEvent methods
bookmark: save	`bookmarkSave(BookmarkSaveEvent)` in BookmarkSaveFilter	BookmarkSaveEvent methods
collapse	`collapse(CollapseEvent)` in CollapseFilter	CollapseEvent methods
data sort	`dataSort(DataSortEvent)` in DataSortFilter	DataSortEvent methods
drill down/expand all	`drillDown(DrillDownEvent)` in DrillDownFilter	DrillDownEvent methods
drill through	`drillThrough(DrillThroughEvent)` in DrillThroughFilter	DrillThroughEvent methods
drill up	`drillUp(DrillUpEvent)` in `DrillUpFilter`	DrillUpEvent methods
expand	`expand(ExpandEvent)` in ExpandFilter	ExpandEvent methods
hide only	`hideOnly(HideOnlyEvent)` in HideOnlyFilter	HideOnlyEvent methods
keep only	`keepOnly(KeepOnlyEvent)` in KeepOnlyFilter	KeepOnlyEvent methods
select a member (for example, in Member Filter)	`memberSelect(MemberSelectEvent)` in MemberSelectFilter	MemberSelectEvent methods
pivot	`pivot(PivotEvent)` in PivotFilter	PivotEvent methods
data query	`query(QueryEvent)` in QueryFilter	QueryEvent methods
remove only	`removeOnly(RemoveOnlyEvent)` in RemoveOnlyEvent	RemoveOnlyEvent methods
show all	`showAll(ShowAllEvent)` in ShowAllFilter	ShowAllEvent methods
show only	`showOnly(ShowOnlyEvent)` in ShowOnlyFilter	ShowOnlyEvent methods
swap axis	`swapAxis(SwapAxisEvent)` in SwapAxisFilter	SwapAxisEvent methods

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 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
bookmark: load	`bookmarkLoad(BookmarkLoadEvent)` in BoomarkLoadListener	BookmarkLoadEvent methods
bookmark: rename	`bookmarkRename(BookmarkRenameEvent)` in BoomarkRenameListener	BookmarkRenameEvent methods
bookmark: save	`bookmarkSave(BookmarkSaveEvent)` in BoomarkSaveListener	BookmarkSaveEvent methods
filter change in ChartBlox	`changePage(ChartPageEvent)` in ChartPageListener	ChartPageEvent methods
collapse	`collapse(CollapseEvent)` in CollapseListener	CollapseEvent methods
data sort	`dataSort(DataSortEvent)` in DataSortListener	DataSortEvent methods
drill down/expand all	`drillDown(DrillDownEvent)` in DrillDownListener	DrillDownEvent methods
drill through	`drillThrough(DrillThroughEvent)` in DrillThroughEvent	DrillThroughEvent methods
drill up	`drillUp(DrillUpEvent)` in DrillUpListener	DrillUpEvent methods
expand	`expand(ExpandEvent)` in ExpandListener	ExpandEvent methods
hide only	`hideOnly(HideOnlyEvent)` in HideOnlyListener	HideOnlyEvent methods
keep only	`keepOnly(KeepOnlyEvent)` in KeepOnlyListener	KeepOnlyEvent methods
select a member (for example, in Member Filter)	`memberSelect(MemberSelectEvent)` in MemberSelectListener	MemberSelectEvent methods
export data to PDF	`pdf(PdfEvent)` in PdfListener	PdfEvent methods
pivot	`pivot(PivotEvent)` in PivotListener	PivotEvent methods
data query	`query(QueryEvent)` in QueryListener	QueryEvent methods
remove only	`removeOnly(RemoveOnlyEvent)` in RemoveOnlyListener	RemoveOnlyEvent methods
show all	`showAll(ShowAllEvent)` in ShowAllListener	ShowAllEvent methods
show only	`showOnly(ShowOnlyEvent)` in ShowOnlyListener	ShownOnlyEvent methods
swap axis	`swapAxis(SwapAxisEvent)` in SwapAxisListener	SwapAxisEvent methods

Chapter 13. Connecting to data

Before you can do anything useful with DB2 Alphablox applications, the first task you need to do is connect to your data sources. In this section, you'll learn more about creating data sources, connecting to data sources, and how to manage access to data sources.

Creating data sources

Before analytic applications can do anything useful, they need access to data that can be viewed and analyzed by users. One of the first tasks that you need to do is to create Data Source definitions in the DB2 Alphablox Admin pages. These data source definitions point to the relational or multidimensional databases you will be connecting to, and allow you to quickly connect and retrieve result sets from them.

Creating data source definitions is more of an administrative task, but can be done by either server administrators or developers, as long as they have administrative rights. Following is a short description of the task that developers or administrators must perform to define an DB2 Alphablox data source.

Note: All of the examples used in the Developer's Guide and in the Blox Sampler application use the QCC databases, either QCC-Essbase (for DB2 OLAP Server and Essbase) or QCC-MSAS (for Microsoft Analysis Services). To install and configure QCC, see the `readme.txt` file, which is located on the DB2 Alphablox CD under the sample data directory:

```
<cdromDir>/sampledata/qcc/
```

Defining data sources

Defining a data source involves the following steps:

1. Access the DB2 Alphablox Home Page using the Start menu or by entering the following URL in a web browser:

   ```
   http://<serverName>/AlphabloxAdmin/home/
   ```

2. Log in with a user name and password with administrator rights. The DB2 Alphablox Admin Pages with three tabs should appear, defaulting to the Applications page.
3. Click the Administration tab. Then click Data Sources to view the list of available data source definitions.
4. To define a data source, click the Create button below the list of existing data source definitions. (If the data source definition you need for your application already exists, you can skip the rest of these steps.)
5. Complete the entries on the Create Data Source panel. For assistance, click on the Help button on this page.
6. Click Save to save the new definition. The newly defined data source name should appear in the list of available data source definitions.

See the Defining A New Data Source section of the *Administrator's Guide* for complete descriptions of data sources and more details about the steps involved in defining them for supported multidimensional and relational databases.

Defining the DataBlox dataSourceName property

A DataBlox, whether used as a standalone or nested Blox, is used to manage the connection between your presentation Blox and the appropriate data source. DataBlox are also responsible for submitting queries and retrieving result sets from data sources. After you have defined your data source in the DB2 Alphablox Admin pages, you need to tell a DataBlox where to go to get information on accessing the appropriate database. To point a DataBlox to a data source, you use the DataBlox `dataSourceName` property.

Two techniques are available for pointing a DataBlox to a data source:

- setting the DataBlox `dataSourceName` attribute
- setting the DataBlox `setDataSourceName` method, using either the server-side Java method or JavaScript to invoke the server-side method (using the DHTML Client API).

Setting the dataSourceName attribute

The most frequently used technique for defining a data source is to add a `dataSourceName` as an attribute and set its value. The value should be the name of one of the data sources you have already defined in the DB2 Alphablox Admin Pages.

For example, in the following code example the nested DataBlox sets the data source to `QCC-Essbase`:

```
<blox:present id="myPresent" ...>
...
  <blox:data
     dataSourceName="QCC-Essbase"
     query='<SYM <ROW("All Products")
        <COLUMN ("All Time Periods") "2000" Sales !' />
</blox:present>
```

Note: If you forget to add the `dataSourceName` attribute to a DataBlox, your data presentation Blox will display a `No data available` message. Or, if the data source is undefined, the JSP page will not compile properly, resulting in an exception being generated.

Using the setDataSourceName() JavaScript method

Sometimes you may want to change the data source programmatically, using JavaScript or Java, perhaps when a user clicks on a button. The following example shows an example using the Blox JavaScript `setDataSourceName` method:

Setting different data sources using DataSourceSelectFormBlox

Follow these steps to create a JSP page that has a selection list for DB2 OLAP Server and Essbase data sources. When a data source is selected, a default query is executed that loads the available dimensions into the DataLayout panel alloying users to perform ad hoc analysis. A complete version of this example can be found in the Ad Hoc Analysis example in Blox Sampler under the Using FormBlox section. The following example uses the DB2 OLAP Server and Essbase version, but the Microsoft Analysis Services version works similarly.

1. At the top of the page, add a JSP page directive specifying the Java classes that need to be made available:

```
<%@ page import="com.alphablox.blox.form.FormEventListener,
     com.alphablox.blox.DataBlox,
     com.alphablox.blox.form.FormEvent" %>
```

2. Below the page directive, add taglib directives for the Blox tag libraries that will be used on this page, in this case the standard Blox tag library and the Blox Form tag library:

```
<%@ taglib uri="bloxtld" prefix="blox" %>
<%@ taglib uri="bloxformtld" prefix="bloxform" %>
```

3. Specify the DataBlox that will be used, enabling alias member names (DB2 OLAP Server and Essbase only) and telling the DataBlox not to connect to the data source on startup:

```
<blox:data id="AdHocDataBlox"
  connectOnStartup="false"
  useAliases="true" />
```

4. Specify the PresentBlox:

```
<blox:present id="AdHocPresentBlox"
  visible="false"
  width="600"
height="350">
  <blox:grid noDataMessage="Select a data source" />
  <blox:chart noDataMessage="Select a data source" />
  <blox:data bloxRef="AdHocDataBlox" />
</blox:present>
```

   We set the common Blox noDataMessage value to "Select a data source" as a better message than the default No data available message. And, the nested DataBlox tag says to use the previously defined DataBlox.

5. Now we can add a DataSourceSelectFormBlox, which will automatically generate a list of available DB2 OLAP Server and Essbase data sources:

```
<bloxform:dataSourceSelect id="dataSourceSelector"
  type="MDB"
  adapter="IBM DB2 for OLAP"
  visible="false"
  nullDataSourceLabel="Select the data source">
<%
  dataSourceSelector.addFormEventListener(new
     DataSourceFormBloxEventListener(AdHocDataBlox));
%>
</bloxform:dataSourceSelect>
```

   The type attribute says that we're specifying multidimensional data sources only and the adapter attribute setting limits the data sources to DB2 OLAP Server or Essbase only. And, rather than have a data source specified when the page loads, adding the nullDataSourceLabel option will tell the user to "Select the data source."

   Also, the nested Java scriptlet tells the DataSourceSelectFormBlox that it needs to add a FormEventListener included at the bottom of the page. This event listener will allow us to create the DataBlox without specifying the data source until a user selects one.

6. Layout the page and specify where the DataSourceSelectFormBlox and PresentBlox should appear by using the following <blox:display> tags:

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

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

   See the Blox Sampler example for the complete code laying out the page.

7. Finally, add the FormBloxEventListener class:

```
<%!
  public class DataSourceFormBloxEventListener
     implements FormEventListener {

    private DataBlox dataBlox;

    public DataSourceFormBloxEventListener(DataBlox dataBlox) {
      this.dataBlox = dataBlox;
  }

    public void valueChanged(FormEvent event) throws Exception {

      String dataSourceName = event.getFormBlox().getFormValue();
      dataBlox.setDataSourceName(dataSourceName);

      if (dataSourceName != null) {
        dataBlox.setQuery("!");
        dataBlox.updateResultSet();
      }
      else
        dataBlox.disconnect(true);
      }
   }
%>
```

This `DataSourceFormBloxEventListener` class will get the FormBlox value for the data source and set the default query, which will populate the DataLayout panel of the PresentBlox with all of the available dimensions.

Note: For details about the syntax and usage of the FormBlox or DataBlox properties and methods, see the *Developer's Reference*.

Connecting to and disconnecting from data sources

When a standalone or nested DataBlox is instantiated, an implicit `connect` method is invoked. If a query has been specified in the DataBlox, the query is executed and a result set is generated. If the DataBlox `connectOnStartup` property is set to `false` (default is `true`), then the `connect` method will not be invoked and you will have to programmatically connect later.

After a Blox has made a connection to its data source, the connection persists throughout the current session. This default behavior is optimal for performance, preventing an application from repeatedly opening and closing database connections for every query (including initial queries and queries resulting from user interaction with a Blox).

Depending on your task, there are a number of different options available for managing data source connections with DataBlox properties and methods, summarized here:

Goal	DataBlox Properties and Methods	Result
Connect to data source, but do not execute query	`<blox:data ... connectOnStartup="true" ... </blox:data>` [Note: `query` attribute not set]	• no result set retrieved • users see common Blox `noDataMessage` property's message (default: `"No data available"`) which can be customized
	`connect(false);`	• if connection already exists, disconnect, then reconnect • connection is made • defined query is not executed • users see common Blox `noDataMessage` property's message (default: `"No data available"`) which can be customized
Connect to data source and execute query	`connect();`or `connect(true);` [Note: assumes query is already set]	• defined textual query is executed • result set is retrieved
	`setQuery(); updateResultSet();`	• query is set, connection established, and result set is retrieved

Update a result set based on connection property changes	`// Change properties first`
	
	`updateResultSet();`
		• updates the result set after applying result set property changes (e.g., after setting `useAliases` to `true` or `false`)
		
		[Note: Use the `connect` method instead if applying connection properties (such as `dataSourceName`, `username`, `schema`, and `password`).]

Note: For details about the syntax and usage of these DataBlox properties and methods, see the *Developer's Reference*.

Here's an example of what a Java scriptlet would look like for setting a query, then connecting:

```
<%
  String query = "<ROW (\"All Products\") <ICHILD \"All Products\" "+
  "<COLUMN (\"All Time Periods\") <CHILD \"All Time Periods\" "+
(Measures) Sales !";

  PresentBlox3.getDataBlox().setQuery(query);
  PresentBlox3.getDataBlox().connect();

%>
```

Note: The "Initial Query Using JSP Scriptlet" example in the Retrieving Data section of Blox Sampler demonstrates this technique.

Sometimes you may prefer to control when a Blox connects and disconnects from data sources programmatically. For example, you might have a page designed to let the users make a number of selections using selection lists, radio buttons, and checkboxes before they can submit their view request by clicking on a button. There are many ways that this could be implemented, including loading a default

view and presetting HTML form elements or FormBlox with default values or loading a Blox with no view until the users have made their selections. For an example of how this could be done, see "Auto-connecting and auto-disconnecting" below.

Auto-connecting and auto-disconnecting

As described below, the DataBlox `autoConnect` and `autoDisconnect` properties can be used in certain situations with relational and multidimensional data sources for better managing performance and scalability of DB2 Alphablox analytic applications.

Details on the syntax and usage of the `autoConnect` and `autoDisconnect` attributes, see the DataBlox section of the *Developer's Reference*.

Relational data sources

When you have a limited number of ports available and are using relational data sources, you can set the `autoConnect` and `autoDisconnect` properties on DataBlox to manage the use of application connections. The following table summarizes all possible setting combinations of the `autoConnect` and `autoDisconnect` properties and the resulting behavior:

autoConnect	autoDisconnect	Behavior
`false`	`false`	These are the default settings on a DataBlox. Defined queries are executed against defined data sources using an implicit `connect` method. Once a connection is established, it is maintained during the current browsing session.
`true`	`true`	Recommended only when there are limited database ports available. The initial database connection is made and the query executes, followed by the return and display of the result set and automatic disconnection from the database. Remember that many of the user interactions on a Blox will repeat this cycle and require a connection to be reestablished.
`true`	`false`	This is really no different from the default behavior.
`false`	`true`	After the initial result set is displayed, the user will not be able to perform any operations on the result set. While this combination of settings is possible, it is generally not recommended.

Multidimensional data sources

The DataBlox `autoConnect` property has no effect on multidimensional databases, but the `autoDisconnect` property can be used with Microsoft Analysis Services data sources to manage scalability and performance of analytic applications.

For Microsoft Analysis Services data sources only, setting the `autoDisconnect` property to `true` results in data source connections immediately disconnecting after

query executions, which include executing queries, drilling down and up, pivoting, using Keep Only, and Remove Only. Metadata calls are not affected. After each disconnection, the PivotTable Services cache memory is cleared from the `java.exe` process and the DataBlox immediately reconnects using the previous connection information.

Only consider using the `autoDisconnect` property with Microsoft Analysis Service if you are experiencing scalability issues resulting from excessive PivotTable Services cache memory consumption. Each MSAS connection that is maintained can consume up to about 250 MB of memory, rapidly consuming available server memory resources. By setting `autoDisconnect` to `true`, PivotTable Services memory consumption will be prevented. With `autoDisconnect` set to `false` (default value), the PivotTable Services cache is maintained and may result in faster display of frequently accessed data to users.

Single sign-on for Essbase and DB2 OLAP Server

DB2 Alphablox applications support the use of Essbase single sign-on credential objects for accessing Hyperion Essbase and DB2 OLAP Server data sources. This functionality allows Essbase users to sign on once and use the generated credentials to move between DB2 OLAP Server and Essbase data sources.

Single sign-on allows users to connect to multiple DB2 OLAP Server and Hyperion Essbase data sources after logging in only once. When a user is authenticated on a supported data source, an encrypted token is generated which contains user credentials (including the user name and, depending on the configuration, the user password) that can be passed between multiple Essbase data sources. DB2 Alphablox applications can be created to use the credential objects created by the Hyperion Common Security Services and pass this user information through a DataBlox. When Essbase single sign-on is used with DB2 Alphablox applications, user names and passwords do not need to be stored in the DB2 Alphablox Repository.

Passing user credentials using the DataBlox credential attribute

DB2 OLAP Server and Hyperion Essbase user credentials can be passed to a supported data source using the DataBlox `credential` attribute.

The following steps assume you have access to DB2 OLAP Server or Hyperion Essbase data sources that support single sign-on.

To pass a credential object:

1. Add a JSP scriptlet that specifies a variable that obtains the user credentials token.

   Important: This scriptlet must appear above the DataBlox tag for the data source that will be receiving the user credentials.
2. In the DataBlox tag, add the `credential` attribute and set the value to be a JSP expression that retrieves the value for the variable you specified.

In the following example, the required Java packages are made available using JSP page directives, then a JSP scriptlet specifies a variable (`userCredential`) that is used to retrieve the user credentials token. The `credential` attribute of the DataBlox then retrieves this information to access the specified Essbase data source.

```
<%@ page import="com.hyperion.css.CSSAPIIF" %>
<%@ page import="com.hyperion.css.CSSException" %>
<%@ page import="com.hyperion.css.CSSSystem" %>
<%@ page import="com.hyperion.css.application.CSSApplicationIF" %>
<%@ page import="com.hyperion.css.common.CSSUserIF" %>
<%@ page import="java.io.*" %>
<%@ page import="java.net.*" %>
<%@ page import="java.util.*" %>
<%@ taglib uri="bloxtld" prefix="blox" %>
<%@ page contentType="text/html;charset=utf-8" %>

<%!
public class MyCssApp implements CSSApplicationIF {
   //Implements your application contract - code omitted here.
}
%>

<html>
<head>
  <blox:header/>
</head>
<body>
<%
    String credential = request.getSession().getAttribute("SSOToken");

   if (credential == null)
   {
      HashMap css_context = new HashMap();
      String user = request.getParameter("username");
      String password = request.getParameter("password");

      MyCssApp myApp = new MyCssApp();
      CSSSystem system = CSSSystem.getInstance();
      CSSAPIIF css = system.getCSSAPI();

      css_context.put(CSSAPIIF.LOGIN_NAME, user);
      css_context.put(CSSAPIIF.PASSWORD, password);
      css.initialize(css_context, myapp);

      CSSUserIF css_user = css.authenticate(css_context);

      credential = css_user.getToken();
      request.getSession().setAttribute("SSOToken", user1.getToken());
   }
%>


  <blox:data id="myDataBlox"
    credential="<%= credential %>"
    dataSourceName="EssbaseSSO"
    ...
  </blox:data>

  <blox:present id="myPresentBlox"
    width="700" height="500"
    <blox:data bloxRef="myData" />
  </blox:present>

</body>
</html>
```

For more information on the DataBlox credentials attribute, see the Data Reference section of the *Developer's Reference*.

Passing user credentials using the Blox API

Hyperion Essbase user credentials can be passed to an Essbase data source using the Blox API to invoke the DataBlox `setCredential()` method.

The following steps assume you have access to Hyperion Essbase data sources that support single sign-on.

To pass a credential object using the Blox API:

1. Add a JSP scriptlet to specify a variable that obtains the user credentials token.

   Important: This scriptlet must appear above the DataBlox tag for the data source that will be receiving the user credentials.
2. Add the DataBlox tag, but do not add the `credential` attribute.
3. Add a JSP scriptlet below the DataBlox tag to set the user credentials.

In the following example, the JSP scriptlet specifies a variable (`usercredential`) that is used to retrieve the user credentials token. Then, the DataBlox `setCredential()` retrieves the user information and applies it to the specified DataBlox.

```
<%
  String userCredential = myGetCSSToken();
%>
...
<blox:data id="myDataBlox"
  datasourceName="EssbaseSSO"
  ...
</blox:data>

...
<%
  myDataBlox.setCredential(userCredential);
%>
```

Using the Blox API to control the setting of user credentials allows you to perform additional operations you may need to perform before you want the user credentials to be retrieved. For more information on the setCredential() method, see the *Blox API Javadoc* documentation.

Limitations of single sign-on

When using single sign-on with Hyperion Essbase or DB2 OLAP Server data sources, you should be aware of these known limitations and other issues which can affect your application.

Single sign-on support is available in DB2 Alphablox for data sources using Hyperion Common Security Services 2.6 and 2.7 (Hyperion Essbase and Hyperion Essbase Deployment Services 7.1.3, 7.1.2, and 7.1.1). Corresponding DB2 OLAP Server 8.2 versions are also supported by DB2 Alphablox.

- When you use an LDAP server for external authentication, a user credentials token that works in Essbase 7.1.3 will not work in Essbase 7.1.1 or 7.1.2, and vice versa. Corresponding DB2 OLAP Server 8.2 versions will behave the same.
  - Essbase 7.1.3 attempts to look up users using a User ID (UID). If a token is generated with a user's UID, the user credentials token will work with Essbase 7.1.3 data sources but will fail with an "Unknown User" message when applied to Essbase 7.1.1 and 7.1.2.

  - Essbase 7.1.1 and 7.1.2 attempt to look up users using the Common Name (CN). If a token is generated using a user's CN, the token will work when passed to Essbase 7.1.1 and 7.1.2 but will fail with an "Unknown User" message when applied to Essbase 7.1.3.

- If you apply a user credentials token to a DataBlox, any other user name or password associated with the DataBlox will be ignored. If you have specified user names or passwords using DataBlox tag attributes or in the DB2 Alphablox Repository, they will be ignored when user credentials tokens are used.

Chapter 14. Retrieving data

After connecting to a data source, the next task is to retrieve data in result sets generated from your submitted queries. Sometimes, these queries will be provided to you by database administrators or data analysts. More often, you'll be writing query statements on your own, or in collaboration with others. The more familiar you are with the data sources you'll be accessing, the more will you will be able to work independently. In this section, you'll learn only the basics of retrieving data for viewing in DB2 Alphablox applications from various data sources. The goal here is to help you through some of the frequently encountered issues.

Depending on the data source you are accessing, the syntax used for specifying application queries can vary considerably. In DB2 Alphablox applications, query strings can be one of the following:

- Essbase report scripts: for IBM DB2 OLAP Server and Hyperion Essbase data sources
- Multidimensional expressions (MDX): for Microsoft SQL Server Analysis Services and DB2 Alphablox Cube Server
- SQL statements: for relational data sources

If you are familiar with a particular data source and know how to create queries to retrieve data, most of your knowledge is directly applicable in DB2 Alphablox applications. When working with DB2 Alphablox, though, there are some useful tips and techniques that you should know about, so be sure to read the appropriate topics in this section on data sources you'll be working with.

If you are not familiar with a particular data source, the sections that follow should help give a brief overview of syntax and DB2 Alphablox issues you might encounter when working with data sources. For details on working with data sources, see the appropriate sections below for where to find more information.

The Query Builder, included in the Application Studio Workbench, can be used to enter and test queries against data sources you will be using in your analytic applications. This tool connects to any data source defined in DB2 Alphablox. You can use Query Builder in several ways to develop queries:

- enter a text string and see the resulting analysis view
- invoke the last query against the data source and see the resulting analysis view
- execute the data source's default query (if one exists) and see the resulting analysis view
- use the GridBlox user interface to move dimensions among axes; pivot, drill, and filter data; and perform other actions to arrive at the application's required analysis view. Then retrieve the query string required to generate that view.

After arriving at the appropriate query string, you can cut and paste it into a DataBlox's query value, or save it in a text file for later use. For more information, see "Query Builder" on page 143.

The application design determines where to specify an application's queries. An DB2 Alphablox application can issue a query request based on:

- Blox instantiation (through the DataBlox query property)

- a user selecting from a list of predefined queries, perhaps though HTML form buttons
- a custom property, containing a query string, that is associated with a user profile

Setting the DataBlox query property

The DataBlox' `query` property determines the initial query that should be executed on a database after a DataBlox or a nested DataBlox is loaded. If a query is not defined, the default query is an empty string. By default, a Blox that loads without a result set will display a message stating `No data available`.

To define an initial query for a Blox, there are two options:

- Define an initial query in the value for the `query` attribute of the DataBlox, or
- Use a Java method to set the query and then execute it

To define the query string using the DataBlox `query` attribute, just add a `query` attribute to a DataBlox. The DataBlox `query` attribute should be entered as follows:

```
query="queryString"
```

where *queryString* is a string defining the query to be executed against the data source defined using the `dataSourceName` attribute.

In the following example, the nested DataBlox of a GridBlox will execute the defined *queryString* against the `QCC-Essbase` data source:

```
<blox:grid id="myGrid">
  <blox:data
    dataSourceName="QCC-Essbase"
    query='<ROW("All Products") <CHILD "All Products"
       <COLUMN("All Time Periods") <CHILD "2000" Sales !'/>
</blox:grid>
```

For readability purposes and ease of coding, you may find it useful to define the `query` property in a Java function. The two tasks below show how this can be done. When you use the `query` attribute, the Blox, by default, will take care of connecting to the defined data source and executing the query. When using methods, you need to use the DataBlox `connect()` method to have the defined query executed. The following task shows an example of how to use Java methods to retrieve result sets from defined data sources.

Setting and executing queries using JSP scriptlets

When performance becomes an issue, you may want to use this simple trick to enhance the speed of response of a displayed view by taking advantage of a JSP scriptlet, in which two server-side methods are used to generate a result set. The DataBlox `setQuery()` method can be used to define the initial query, then the `connect()` method will result in the execution of that query and return the result set to the containing Blox.

1. At the top of your JSP page, but after the taglib directive, add the appropriate Blox tags to define your presentation Blox, but setting the `visible` attribute to false so that the Blox is not rendered before the data is available. Include a nested DataBlox with the `dataSourceName` attribute to define the data source, but don't include the `query` attribute.

For example, the following Blox tag defines a PresentBlox with `visible` set to `false` and with a `dataSourceName` set to `QCC`:

```
<blox:present id="PresentBlox3"
  visible="false"
  width="550"
  height="350">
  <blox:data
    dataSourceName="QCC"/>
  <blox:grid
    bandingEnabled="true"/>
  <blox:chart
    chartType="Bar"/>
</blox:present>
```

2. Below the Blox tag defining your presentation Blox, add a JSP scriptlet that does three sub-steps:
   - Declare a query variable
   - Set the query defined in the query variable
   - Connect the Blox to the data source, resulting in the result set being returned

   The following scriptlet example shows a query being defined and executed using Java methods:

   ```
   <%
    String query="<ROW(\"All Products\") <ICHILD \"All Products\""+
                 "<COLUMN(\"All Time Periods\") <CHILD 2000 "+
                 "<PAGE(Measures) Sales !";

    PresentBlox3.getDataBlox().setQuery(query);
    PresentBlox3.getDataBlox().connect();
   %>
   ```

   In this example, the query variable is declared as a string (by placing `String` in front of the variable declaration), then setting that variable equal to the desired query statement (in this example, the query is an Essbase report script). Notice that the query is laid out for easy reading and maintenance using concatenated strings. After the query is declared, two DataBlox methods, `setQuery()` and `connect()`, are used. The `setQuery()` method sets the query in the DataBlox (in this example, notice that the defined query can be substituted in the argument of the method by placing `query` as the argument). Then, using the `connect()` method, the DataBlox is instructed to connect to the data source and execute the set query.
3. Further down in the `<body>` of your JSP page, place a Blox display tag where you want the Blox to be rendered for viewing. Reference the presentation Blox using the `bloxRef` attribute, setting its value to the name of the Blox being rendered.

   For the example here, within the `<body>` tag, you would place the following tag:

   ```
   <blox:display bloxRef="PresentBlox3"/>
   ```

As you can see in this example, you can use this technique even with limited Java knowledge.

Multidimensional data sources

An overview on multidimensional databases is available in the *Administrator's Guide*. See the following sections for more information on:

- OLAP Terms and Concepts
- Multidimensional Analysis

- OLAP Database Terms

IBM DB2 OLAP Server and Hyperion Essbase

IBM DB2 OLAP Server and Hyperion Essbase are multidimensional databases optimized for analysis, typically generating sub-second responses to queries.

To retrieve data from DB2 OLAP Server or Essbase cubes, you need to use the Essbase Report Specification Language to generate report scripts, which can be used for query values in DB2 Alphablox applications.

Following is a basic summary of how to create Essbase report scripts with important tips on how to use report scripts in conjunction with DB2 Alphablox functionality.

For details about using DB2 OLAP Server or Essbase and Essbase report scripts, see your DB2 OLAP Server or Essbase documentation.

Creating Essbase report scripts

To pass a query to a IBM DB2 OLAP Server or Hyperion Essbase data source, use the Essbase Report Specification language to create report scripts.

Tip: For information on the Report Script Specification Language, see the online documentation in the DB2 OLAP Server or Essbase installation directory at `/docs/techref/RPTIND.HTM`. If you have the DB2 OLAP Server or Essbase Application Manager installed on your workstation you can access this documentation through the Help menu.

The following example on a DataBlox specifies that:
- The Market and Accounts dimensions are to appear on the column axis.
- The Scenario and Product dimensions are to appear on the row axis.
- The children of all four dimensions should be included in the result set.
- Any unused dimensions appear on the “Other” axis.

```
<blox:data ...
  query="<SYM <ROW (Scenario,Product)
     <ICHILD Scenario <ICHILD Product <COLUMN (Market, Accounts)
     <ICHILD Market <ICHILD Accounts !"/>
```

Essbase report script commands supported by DB2 Alphablox

The following table lists most Essbase report script commands, whether they are supported by DB2 Alphablox (that is, they work when entered in report scripts), the equivalent or near-equivalent DB2 Alphablox functionality, and report script examples using these commands.

Report Script Command	Report Script Example and Comments
!	This is required to execute a report script query. By itself, the “bang” query returns one dimension on a grid or chart, and a list of all available dimensions in the DataLayout panel. Multiple bang report output commands are not supported by DB2 Alphablox. See note below this table.

Report Script Command	Report Script Example and Comments
&1	&CurrentMonth If defined in IBM DB2 OLAP Server or Hyperion Essbase, server substitution variables can be added to report scripts. They are primarily used to simplify maintenance of scripts.
ALLINSAMEDIM	<ROW (Scenario) <ALLINSAMEDIM Actual !
ALLSIBLINGS	<ROW (Scenario) <ALLSIBLINGS Actual !
ANCESTORS	<ROW (Measures) <ANCESTORS "Marketing" !
ASYM	<ASYM <COL (Scenario, Year) Actual Jan Budget Feb !
ATTRIBUTE	<ROW (Product) <ATTRIBUTE Bottle !
BOTTOM	<ROW (Year) <DIMBOTTOM Year <BOTTOM (6, @DataCol(1)) !
CALCULATECOLUMN	See DB2 OLAP Server or Essbase documentation for examples.
CALCULATEROW	See DB2 OLAP Server or Essbase documentation for examples.
CHILDREN	<ROW (Market) <CHILDREN Market !
CLEARALLROWCALC	See DB2 OLAP Server or Essbase documentation for examples.
CLEARROWCALC	See DB2 OLAP Server or Essbase documentation for examples.
COLUMN	<COLUMN (Year) <CHILD Year !
DESCENDANTS	<ROW (Year) <DESCENDANTS Year !
DIMBOTTOM	<ROW (Year) <DIMBOTTOM Year !
DUPLICATE	<ROW (Year) <Child Year <DUPLICATE Qtr1 !
FIXCOLUMNS	<COL (Year) {FIXCOLUMNS 3} <DIMBOTTOM Year !
GEN	<ROW (Product) gen2,Product !
IANCESTORS	<ROW (Year) <IANCESTORS Jan !
ICHILDREN	<ROW (Product) <ICHILDREN Colas !
IDESCENDANTS	<ROW (Product) <IDESCENDANTS Product !

Report Script Command	Report Script Example and Comments
INCMISSINGROWS	{SUPMISSINGROWS} {INCMISSINGROWS} <PAGE (Market) "New York" <ROW (Product) lev0,Product !
INCZEROROWS	{SUPZEROROWS} {INCZEROROWS} <PAGE (Market) "New York" <ROW (Product) lev0,Product !
IPARENT	<ROW (Year) <IPARENT Jan !
LATEST	<LATEST Aug <ROW (Year) <CHILD QTR3 Q-T-D !
LEV	<ROW (Product) lev0,Product !
LINK	<ROW (Year) <LINK(<DIMBOTTOM(Year) AND <DESCENDANTS(Qtr1)) !
MATCH	<ROW (Market) <MATCH (Market, C*) !
NAMESON	{SUPNAMES} {NAMESON} <ROW (Market) <CHILD East !
NOROWREPEAT	{NOROWREPEAT} <ROW (Market, Product) <CHILD East <CHILD Product !
OFFCOLVCALCS	See DB2 OLAP Server or Essbase documentation for examples.
OFFROWCALCS	See DB2 OLAP Server or Essbase documentation for examples.
OFSAMEGEN	<ROW (Market) <OFSAMEGEN East !
ONSAMELEVELAS	<ROW (Market) <ONSAMELEVELAS East !
ORDER	{ORDER 0 5 4 3 2 1} <COL (Product) <CHILD Product !
ORDERBY	<ROW (Product) <DIMBOTTOM Product <ORDERBY ("Product", @DATACOL(1) ASC) !
PAGE	<PAGE (Market) East <ROW (Product) <CHILD Product !
PARENT	<ROW (Year) <PARENT Jan !
REMOVECOLCALCS	See DB2 OLAP Server or Essbase documentation for examples.
RESTRICT	<ROW (Product) <DIMBOTTOM Product <RESTRICT (@DATACOL(1) > 10000) !
ROW	<ROW (Year) <PARENT Jan !
SCALE	{SCALE 100} <ROW (Product) <CHILD Product !

Report Script Command	Report Script Example and Comments
SETROWOP	See DB2 OLAP Server or Essbase documentation for examples.
SINGLECOLUMN	<SINGLECOLUMN <COL (Year) Year <ROW (Product) <CHILD Product !
SORTALTNAMES	<ROW (Product) <SORTALTNAMES <DIMBOTTOM Product !
SORTASC	<ROW (Market) <SORTASC <DIMBOTTOM Market !
SORTDESC	<ROW (Market) <SORTDESC <DIMBOTTOM Market !
SORTGEN	<ROW (Product) <SORTGEN <DESCENDANTS Product !
SORTLEVEL	<ROW (Product) <SORTLEV <DESCENDANTS Product !
SORTMBRNAMES	<ROW (Product) <SORTMBRNAMES <SORTDESC <DIMBOTTOM Product !
SORTNONE	<ROW (Product) <SORTMBRNAMES <SORTDESC <SORTNONE <DIMBOTTOM Product !
SPARSE	<SPARSE <ROW (Product, Market) <DIMBOTTOM Product <DIMBOTTOM Market !
SUPEMPTYROWS	{SUPEMPTYROWS} <PAGE (Market) "New York" <ROW (Product) lev0,Product !
SUPMISSINGROWS	{SUPMISSINGROWS} <PAGE (Market) "New York" <ROW (Product) lev0,Product !
SUPSHARE	<SUPSHARE <ROW (Product) lev0,Product !
SUPSHAREOFF	<SUPSHARE <SUPSHAREOFF <ROW (Product) lev0,Product !
SUPZEROROWS	{SUPZEROROWS} <COL (Measures) Sales <ROW (Year) Jan Feb Mar !
SYM	<SYM <COL (Measures, Year) Sales COGS Jan Feb !
TOP	<ROW (Market) <DIMBOTTOM Market <TOP(5, @DATACOL(1)) !
UDA	<ROW (Market) <UDA (Market, "Major Market") !
WITHATTR	<ROW (Product) <WITHATTR(Caffeinated,"<>",True) !

Notes®:

1. DB2 Alphablox custom properties can be used.
2. DB2 Alphablox uses `selectableSlicerDimensions` to control page displays, but the `<PAGE` command works in report scripts to slice the data.

3. `suppressMissingOnRows`, `suppressMissingOnColumns`, and `suppressZeros` can be used in DB2 Alphablox for a similar effect.
4. `suppressMissingOnRows` and `suppressMissingOnColumns` in DB2 Alphablox suppresses missing values in rows and columns.
5. `suppressZeros` can be used in DB2 Alphablox, but this property suppresses zeros in both rows and columns.

Note: Multi-bang queries, including multiple bang (!) report output commands are not supported in DB2 Alphablox. You may discover that a few select report scripts employing multiple bang report output commands may display results within Blox, but you use them at your own risk.

Unsupported report script commands with DB2 Alphablox equivalents

The following Essbase report script commands are not supported in DB2 Alphablox. Use the listed DB2 Alphablox equivalents instead of the listed Essbase report script commands.

Report Script Command
DB2 Alphablox Equivalents

AFTER `defaultCellFormat` (GridBlox)

BEFORE `defaultCellFormat` (GridBlox)

COMMAS `defaultCellFormat` (GridBlox)

DECIMAL
`defaultCellFormat` (GridBlox)

EUROPEAN
`defaultCellFormat` (GridBlox)

MISSINGTEXT
`missingValueString` (GridBlox)

NOINDENTGEN
`rowIndentation` (GridBlox)

OUTALT `useAliases` (DataBlox)

OUTALTNAMES
`useAliases` (DataBlox)

OUTALTSELECT
`aliasTable` (DataBlox)

OUTMBRNAMES
`useAliases` (DataBlox)

SUPBRACKETS
`defaultCellFormat` (GridBlox)

SUPCOMMAS
`defaultCellFormat` (GridBlox)

Unsupported report script commands without DB2 Alphablox equivalents

BLOCKHEADERS BRACKETS COLHEADING	SAVEANDOUTPUT SAVEROW SETCENTER SETROWOP
CURHEADING CURRENCY DIMEND DIMTOP	SKIP SKIPONDIMENSION STARTHEADING SUPALL
DIMBOTTOM ENDHEADING FEEDON	SUPCOLHEADING SUPCURRHEADING SUPEUROPEAN
FORMATCOLUMNS HEADING IMMHEADING	SUPFEED SUPFORMATS SUPHEADING SUPMASK
INCEMPTYROWS INCFORMATS INCMASK INDENT	SUPNAMES SUPOUTPUT SUPPAGEHEADING
INDENTGEN LMARGIN MASK NAMESCOL	TABDELIMIT TEXT TODATE UCHARACTERS
NAMEWIDTH NEWPAGE NOPAGEONDIMENSION	UCOLUMNS UDATA UNAME UNAMEONDIMENSION
NOSKIPONDIMENSION PAGEHEADING PAGELENGTH	UNDERLINECHAR UNDERSCORECHAR WIDTH
PAGEONDIMENSIONS PRINTROW	ZEROTEXT

Calc Scripts

Calc (calculation) scripts are text files containing instructions to calculate data in DB2 OLAP Server or Essbase cubes. Calc scripts can be invoked in DB2 Alphablox applications using the following DataBlox methods:

- `executeCustomCalc`
- `executeNamedDBCalcScript`
- `substituteCalcScriptTokens`
- `writeback`

For details on using these methods, see the *Developer's Reference*. For more information on using calc scripts in your DB2 OLAP Server or Essbase cubes, see the DB2 OLAP Server or Hyperion Essbase documentation.

Substitution variables

In IBM DB2 OLAP Server or Hyperion Essbase cubes, substitution variables act as global placeholders for information that changes regularly. Each variable has a value assigned to it and can be changed at any time by the database administrator. The use of substitution variables helps reduce maintenance of report scripts, eliminating the need for manual changes to individual report scripts in DB2 Alphablox applications.

For example, many report scripts refer to reporting periods, such as current month or current quarter. By using substitution variables set on the IBM DB2 OLAP Server or Hyperion Essbase server, such as `CurrentMonth` or `CurrentQuarter`, you can change the assigned value in one place, and the appropriate report scripts are dynamically updated when the report script is executed.

To refer to a substitution variable in your report script, place an ampersand (&) in front of the variable name. For example, use `&CurrentMonth` in your report script to reference the substitution variable `CurrentMonth`. The following DataBlox example shows the use of `&CurrentMonth` in the query attribute:

```
<blox:data
  dataSourceName="QCC-Essbase"
  query='<ROW("All Products") <COLUMN("All Time Periods")
  &CurrentMonth <PAGE(Measures) Sales !'/>
```

When the query is executed, `&CurrentMonth` is substituted with the value defined in the IBM DB2 OLAP Server or Hyperion Essbase server.

While substitution variables help reduce maintenance in report scripts, someone still has to manually change the values in the IBM DB2 OLAP Server or Hyperion

Essbase server. As an alternative in DB2 Alphablox applications, you could use Java methods in your JSP pages to automatically calculate a value for the current month or other reporting period, then substitute that value in your report scripts.

Using DB2 OLAP Server or Essbase aliases

DB2 OLAP Server or Essbase aliases, or alternate names for members defined in DB2 OLAP Server or Essbase cubes, can be used to improve the readability of an analytic view. Aliases can be used to refer to alternate member names, such as in a foreign language, or to refer to product identification values. DB2 Alphablox supports the use of aliases in report scripts and values in various properties.

By default, DB2 Alphablox applications display unique members names and not aliases. If you want to display aliases in your analytic views, set the DataBlox `useAliases` property to `true`. For more information, see the DataBlox Reference section within the *Developer's Reference*.

Working with decimals

When displaying numeric data values with a specified number of decimal places, you may need to add the `{DECIMAL}` report script command to your DB2 OLAP Server or Essbase query statement. For more information, see the detailed descriptions of these properties in the *Developer's Reference*.

Use the following Blox tag attributes:	On this Blox:	And this Report Spec directive:
`<blox:grid id="myGrid" ... defaultCellFormat="#,###.00; -#,###.00" > <blox:cellFormat scope="{Sales}" format="#,###.##"/> <blox:cellAlert background="#3333ff" format="#,###.##;(#,###.##" scope="{Scenario}"/> </box:grid>`	GridBlox	`{DECIMAL 2}`

Microsoft Analysis Services

Microsoft SQL Server includes Microsoft Analysis Services, and can be used to retrieve data from multiple relational data sources, including Microsoft SQL Server, Oracle, and others. While similar to IBM DB2 OLAP Server and Hyperion Essbase in functionality, Microsoft uses the Multidimensional Expressions Language (MDX) to query the Microsoft Analysis Services multidimensional data cubes.

For more information on Microsoft Analysis Services, see the following resources:

Books

Spofford, George. 2001. *MDX Solutions: With Microsoft SQL Server Analysis Services.* New York: John Wiley & Sons.

An excellent and thorough tutorial/reference guide on how to use MDX to access and analyze data for decision support. Cover basic and advanced MDX statements, offering clear solutions to the most commonly-encountered problems. Strongly recommended for serious developers.

Jacobsen, Reed. 2000. *Microsoft SQL Server Analysis Services Step by Step.* Redmond, Washington: Microsoft Press.

A good tutorial introduction to all aspects of Microsoft Analysis Services, including database administration, building databases, and basic MDX usage.

Newsgroups

If you cannot get your questions answered in the books above or the Microsoft documentation, you can turn to another great resource— Internet newsgroups. For MSAS, though, there is really only one newsgroup to go to get answers to your questions: `microsoft.public.sqlserver.olap`. Use this peer-to-peer newsgroup to discuss MSAS with other administrators and developers. Many great contributors on this newsgroup make it one of the most useful computing newsgroups. George Spofford, the author of MDX Solutions, has been a frequent contributor, asking many tough questions. Also, several Microsoft Analysis Services team members follow this newsgroup and contribute insights that may not be found elsewhere.

To join, point your news server to:

```
news:msnews.microsoft.com/microsoft.public.sqlserver.olap
```

Alternatively, you can access the OLAP newsgroup from the following link, which lists all Microsoft SQL Server-related newsgroups:

```
http://www.microsoft.com/sql/support/newsgroups/
```

And, to search the archives of this newsgroup, point your web browser to Google Groups, a search engine for Usenet newsgroups, available at:

```
http://groups.google.com/
```

In the search field on the home page for Google Groups, enter the following:

```
microsoft.public.sqlserver.olap
```

and click the Google Search button. Almost immediately, you will be presented with the most recent postings. You can also narrow your search further by searching on keywords while limiting your search to just this newsgroup. This is a great resource when you need answers in a hurry.

Creating MDX query statements

To pass a query to Microsoft® SQL Server 2000 Analysis Services, use a valid MDX SELECT statement. MDX syntax is somewhat similar to SQL syntax. In the following syntax for simple queries, note the use of the SELECT, FROM, and WHERE keywords:

```
SELECT axis specification ON COLUMNS,
axis specification ON ROWS
FROM cube_name
WHERE slicer_specification
```

Note: MDX statements queries on rows are not valid unless a column is defined.

The following expression queries the Sales cube and returns a summary of the measures dimension for all the stores in California and Washington. The Measures dimension appears on the column axis; the Store dimension on the row axis:

```
SELECT Measures.MEMBERS ON COLUMNS, {[Store].[Store
State].[CA], [Store].[Store State].[WA]} ON ROWS
FROM [Sales]
```

To obtain the detail for the members (stores) in each of these states, add the CHILDREN key word:

```
SELECT Measures.MEMBERS ON COLUMNS, {[Store].[Store State].
[CA].CHILDREN, [Store].[Store State].[WA].CHILDREN} ON ROWS
FROM [Sales]
```

Note that the approach for obtaining a unique member name is to cascade down the dimension hierarchy. For example, assume a dimension named Stores with the following hierarchy:

```
All Stores
   Canada
   USA
     CA
     OR
   Mexico
```

The following are valid unique member names in that hierarchy:

```
[Store].[All Stores]
[Store].[All Stores].[USA]
[Store].[All Stores].[USA].[CA]
```

For more information on the subset of MDX syntax that DB2 Alphablox supports, please see the *DB2 Alphablox Cube Server Administrator's Guide*.

An Introduction to Multidimensional Expressions (MDX) is available online through the Microsoft site: http://msdn.microsoft.com. The Introduction provides these and many more examples.

Clearing PivotTable Services caches using the autoDisconnect property

If you have a large MSAS cube and are experiencing scalability issues resulting from excessive PivotTable Services memory cache consumption that occurs from MDX queries returning large result sets, you may be able to use the DataBlox autoDisconnect property to help manage the scalability and performance of MSAS-based analytic applications. See "Auto-connecting and auto-disconnecting" on page 116 for details about using this property.

DB2 Alphablox Cube Server

The DB2 Alphablox Cube Server supports queries generated using a limited subset of MDX commands. For complete information on these commands and their use, see the *DB2 Alphablox Cube Server Administrator's Guide*.

Using SAP Business Information Warehouse (SAP BW) with DB2 Alphablox

Before using SAP BW with DB2 Alphablox, the prerequisites discussed must be met.

SAP Business Information Warehouse (SAP BW), an enterprise business intelligence solution from SAP, can be accessed by DB2 Alphablox analytic applications. SAP BW data sources can be accessed using the OLE DB for OLAP data adapter provided by DB2 Alphablox. Using DB2 Alphablox, sophisticated analytic applications can be custom-built to meet the needs of business users in a SAP BW environment. DB2 Alphablox uses the existing OLE DB for OLAP infrastructure to connect to SAP BW via the SAP OLE DB for OLAP provider.

Before DB2 Alphablox can be used with SAP BW, the following prerequisites must be met:

- SAP BW 3.5 Frontend components and the Microsoft Data Access Components (MDAC) must be installed on the same machine as DB2 Alphablox. SAP BW 3.5 Frontend is included in the SAP NetWeaver software packages..
- Microsoft's MDAC 2.7.1, 2.8.0, and 2.8.2 have been tested with DB2 Alphablox.
- The SAP Logon Frontend component should be installed by you SAP Basis Administrator.

Once the above conditions have been met, you can define DB2 Alphablox data source definitions for your SAP BW data sources.

Creating SAP BW data source definitions

Before creating analytic applications that you can use with SAP BW, you need to create data source definitions in DB2 Alphablox.

In order to create SAP BW data source definitions in DB2 Alphablox, make sure you have met the prerequisites described in "Using SAP Business Information Warehouse (SAP BW) with DB2 Alphablox" on page 132.

To create a DB2 Alphablox data source definition:

1. In the DB2 Alphablox Admin Pages click on the Administration tab and then click on the **Data Sources** menu item.
2. Below the list, click on the **Create** button. The **Create Data Source** window appears.
3. Enter a name for your SAP BW data source in the Data Source field.
4. From the **Adapter** selection list, choose the OLE DB for OLAP option. The screen will refresh with fields relevant to this adapter.
5. In the **OLAP Server** field, enter the value from the **Description Field** from the **Frontend** configuration system.
6. For the **Default Database** field, enter the name of your InfoCube.
7. Leave the **Default Schema** field blank.
8. In the **Provider** field, enter the value for your SAP OLE DB for OLAP provider. Ask your SAP Basis Administrator for the SFC_CLIENT and SFC_LANGUAGE values. The provider string should be:

   ```
   MDrmSAP;SFC_CLIENT=clientValue;SFC_LANGUAGE-languageValue
   ```

   where clientValue and the languageValue are the ones you obtained from the SAP Basis Administrator.
9. Fill in the **Default Username** and **Default Password** fields with values for the UserID and password provided to you by your SAP Basis Administrator.
10. Click the **Save** button.

You have now created a DB2 Alphablox data source that can be used with DB2 Alphablox analytic applications.

Creating MDX queries for use with SAP BW

Using the MDX syntax supported by SAP BW, you can begin building DB2 Alphablox applications.

The stage needs to be set just so.

- Using the Blox API, you can set SAP BW MDX queries using the DataBlox setQuery() method. For details on the use of this method, see the *Blox API Javadoc* documentaiton. Here is an example of setting an MDX query using thesetQuery() method:

  ```
  DataBlox.setQuery("SELECT DISTINCT( crossjoin ( {[0CALYEAR].[2001]},
  {[0D_SALE_ORG].[All]})) ON AXIS(0), DISTINCT( {[Measures].[0D_COST],
  [Measures].[0D_INV_QTY], [Measures].[0D_NETVLINV], [Measures].[0D_TAXAMOUN]})
  ON AXIS(1) FROM [$0D_DECU]");
  ```

- Use unique names only in queries and arguments when using the DB2 Alphablox APIs since the SAP OLE DB for OLAP provider requires them. Here is an example of a DB2 Alphablox method using the unique name from an SAP BW data source:

  ```
  ODBOMetaData.resolveMember("[Measures].[0D+NETVLINV]");
  ```

- For more information on SAP BW and its use of the MDX expression language, see the SAP Help Portal web site (http://help.sap.com/) or your SAP BW documentation.
- DB2 Alphablox does not currently support native drillthrough and writeback on SAP BW.

Drillthrough support for DB2 OLAP Server and Hyperion Essbase (using EIS)

OLAP data sources offer business analysts and line of business users deep insights into trends in data, but do not give these users ready access to the raw data unless some mechanism is provided to drillthrough into the underlying data sources. Drillthrough support, if available in an OLAP data source, allows users to reach deeper into the raw data contained in the underlying fact table records for selected cells in the OLAP database.

DB2 OLAP Server or Essbase Integration Services allows DB2 OLAP Server or Essbase administrators to map multidimensional data to more detailed relational data. Out-of-the box, Integration Services can be used with Microsoft Excel to view any predefined drillthrough reports available on the EIS server. The reports generated using Excel are basic reports, offering:

- rows and columns only
- no data formatting
- no control over user interactions
- users cannot view multiple reports or sheets simultaneously

Using DB2 Alphablox drillthrough support for EIS, users can drill from summarized and calculated data stored in DB2 OLAP Server (or Essbase) into detailed data stored in a relational warehouse (using a star schema). DB2 Alphablox drillthrough support for Integration Services leverages predefined Integration Services drillthrough relational reports, is easy to enable and configure, offers powerful functionality and flexible customization.

Out-of-the-box Integration Services drillthrough support

With minimal effort, you can begin using the native DB2 OLAP Server or Hyperion Essbase Integration Services drillthrough support by setting the GridBlox `drillThroughEnabled` property to true (default is `false`). Once enabled, a Drill Through menu option is added to the contextual (right-click) menu, available when right-clicking on a data cell in a grid. DB2 Alphablox automatically generates a dialog window offering a list of the available Integration Services drillthrough

reports (predefined by the EIS administrator). After a user selects a report, a built-in default JSP page returns the report data displayed in a basic interactive Alphablox Relational Reporting view within a separate browser window.

Detailed information about DB2 Alphablox relational reporting functionality is available in the *Relational Reporting Developer's Guide*.

Controlling EIS drillthrough window styles

While the default drillthrough window may be adequate for your purposes, DB2 Alphablox also allows you to customize the display window by use of the nested GridBlox `<blox:drillThroughWindow>` tag. When this tag is nested in a GridBlox that has drillthrough support enabled, it overrides the default out-of-the-box behavior and allows custom browser window properties to be defined.

The tag attributes on the `<blox:drillThroughWindow>` are modeled after the most commonly used window definition properties defined in the `features` argument of the JavaScript `window.open(url,windowName,features)` method that is used to open browser windows. The supported properties include the following:

Tag Attribute
Description

`url` Defines the location of the JSP for the drillthrough window

`name` Name of the drillthrough window

`height` Height of the window

`width` Width of the window

`resizable`
Boolean property determining if the drillthrough window can be resized by users. True by default.

`statusbarVisible`
Boolean property determining if the drillthrough browser window's status bar should be visible. True by default.

`scrollbarVisible`
Boolean property determining if the drillthrough browser window's scroll bars should be available. True by default.

`locationbarVisible`
Boolean property determining if the drillthrough browser window's location bar (address bar) should be displayed. True by default.

`toolbarVisible`
Boolean property determining if the drillthrough browser window's toolbar should be displayed. True by default.

`menubarVisible`
Boolean property determining if the drillthrough browser window's menu bar should be displayed. True by default.

For details about the `<blox:drillThroughWindow>` tag and its attributes, see the GridBlox Reference section of the *Developer's Reference*.

Custom EIS drillthrough support using DB2 Alphablox Relational Reporting

DB2 Alphablox Relational Reporting Blox, discussed fully in the *Relational Reporting Developer's Guide*, can be used to generate custom drillthrough support with many desirable features not possible in the native EIS drillthrough support using Microsoft Excel. The flexibility of DB2 Alphablox allows you to customize how your drillthrough behaves. Using the DB2 Alphablox EIS drillthrough support, developers can:

- provide security at the user or role level by permanently hiding columns in the resultset
- control the order of the result set columns
- add calculated columns to the result set
- create break groups and totals
- rename columns
- format data
- open multiple reports simultaneously

Using RDBREsultSetDataBlox and RDBResultSetTag

When using Relational Reporting to display custom reports, consider the following points:

- `RDBResultSetDataBlox` and `RDBResultSetTag` allow you to reference a DataBlox and take its `RDBResultSet` and place it as the data "producer" of the relational reporting pipeline.
- `RDBResultSetDataBlox` works with DataBlox pointing to a relational data source or DataBlox pointing to a drillthrough-enabled DB2 OLAP Server or Essbase data source. If `rowCoordinate` and `columnCoordinate` are specified, the data for the `RDBResultSet` should come from a drillthrough performed on the DataBlox referenced by the `bloxRef` attribute. For drillthrough to work on DB2 OLAP Server or Essbase data sources, a report name must also be set. If `rowCoordinate` or `columnCoordinate` are not specified, the data for the `RDBResultSet` should come from a `getResultSet` call on the DataBlox referenced by the `bloxRef` attribute. In both cases, null will be returned if the action (`drillthrough` or `getResultSet`) cannot be performed correctly.

Supporting multiple reports

To support multiple reports for each cell, you application must be able to handle multiple reports. For example, for each report you may want to group on different columns. Also, each report might also use different CSS style sheets. A `controller.jsp` file could be used to distribute reports to the appropriate JSP pages. Or, if the reports are not complicated, you could use a single JSP file to handle all of the reports. In either case, you need to pass the report name into the JSP page since the report name is required for the RDBResultSetDataBlox to work.

Adding custom menu options

Using the Blox UI Model, you can create your own custom menu option (for example, "Drill to Relational") to appear when a user right-clicks on a data cell. In this case, you can use your own custom options instead of the menu options provided out-of-the-box in DB2 Alphablox.

In the following Java code example, a "Drill to Relational" option is added to the grid's right-click menu:

```
Menu cellMenu = new Menu();
cellMenu.add(new MenuItem("cellItem","Drill To Relational"));
grid.setCellsRightClickMenu(cellMenu);
// Add a dedicated controller to the cell menu
DataBlox db = presentBlox.getDataBlox();
cellMenu.setController(new DrillingController(db,grid,
    abSessionName,appName));
```

Depending on your needs, you could create a DrillController to handle the drillthrough and disable drillthrough on particular cells if the report would return too many rows of data. Or, you could even open two different reports simultaneously from a single cell.

Other custom EIS drillthrough support

If you decide to build your own custom drillthrough support and do not want to display the data using DB2 Alphablox Relational Reporting, you will need to use the following DataBlox methods:

- `RDBResultSet drillThrough(String reportName, Tuple[] coordinates);`
- `RDBResultSet drillThrough(String reportName, int columnCoordinate, int rowCoordinate);`
- `String[] getDrillThroughReportNames(int columnCoordinate, int rowCoordinate);`

Use the `RDBResultSet` that is returned to build your own custom reports, iterating through the rows and columns to get the data. Also, use `getDrillThroughReportNames` to return a list of the available drillthrough reports for the specific cell, then call drillthrough a specific report and cell.

For details about the `RDBResultSet` object and DataBlox methods, see the DataBlox Reference in the *Developer's Reference*.

Drillthrough support for Microsoft Analysis Services

OLAP data sources offer business analysts and line of business users deep insights into trends in data, but do not give these users ready access to the raw data unless some mechanism is provided to drillthrough into the underlying data sources. Drillthrough support, if available in an OLAP data source, allows users to reach deeper into the raw data contained in the underlying fact table records for selected cells in the OLAP database.

An MDX `DRILLTHROUGH` statement can be used to retrieve the source rowsets from the fact table (the relational data source) that was used to create a specified cell, or tuple in a Microsoft Analysis Services cube. Here is an example `DRILLTHROUGH` statement for Foodmart, a sample OLAP database that ships with Microsoft Analysis Services:

```
DRILLTHROUGH SELECT FROM [Inventory] WHERE (
[Product].[ByManufacturer].[All Product].[Acme], [Warehouse].[Whse 8],
[Time].[Aug. 2000] )
```

DB2 Alphablox native DrillThrough support for MSAS retrieves the underlying resultset using the following `DRILLTHROUGH` statement:

```
DRILLTHROUGH [<Max_Rows>] [<First_Rowset>] <MDX SELECT>
```

where <Max_Rows> is equivalent to the MDX MAXROWS value, <First_Rowset> is equivalent to the MDX FIRSTROWSET value, and <MDX SELECT> is the automatically generated SQL query. The value for <Max_Rows> is taken from the setting in the DB2 Alphablox data source definition.

Before being able to retrieve rowsets from the underlying data source using the DRILLTHROUGH statement, you must first enable the MSAS cube to allow drillthrough in the Drillthrough Options dialog and specify which columns you want to be returned. And, your client application must provide drillthrough support. [See Microsoft's SQL Server Books Online documentation, available from the Microsoft SQL Server menu, for details about configuring drillthrough options on a cube.]

While some client applications provide limited drillthrough support, DB2 Alphablox support for MSAS Drillthrough is easy to configure, yet also allows customized drillthrough behavior that other client applications do not offer. With DB2 Alphablox drillthrough support, you can:

- control the order of the result set columns

  While Microsoft Analysis Services allows you to choose which columns you want to be displayed in a drillthrough operation, you cannot order the columns in the resulting records. DB2 Alphablox ReportBlox functionality allows you to define the order of columns displayed. The OrderBlox can be configured to specify the order in which result set columns should be displayed.
- provide security at the user/role level by permanently hiding columns in the result set

  The built-in security of MSAS only allows you to enable drillthrough or not; you cannot control which drillthrough columns are available based on user roles. Using a nested MembersBlox of a ReportBlox, you can permanently hide columns. End users will not be able to show the columns again using the UI, but the developer can reveal columns using the appropriate API calls.
- add calculated columns to the result set

  Using the ReportBlox's CalculateBlox, you can replicate calculations in the relational report, even though MSAS doesn't natively let you return calculated columns.
- open multiple reports in separate windows

  The out-of-the-box drillthrough support provided by DB2 Alphablox allows end users to open multiple report windows to compare drillthrough data from multiple cells in a grid. You can also custom code your own solution for opening multiple windows -- a Blox Sampler example, in the Retrieving Data section under Microsoft Analysis Services, shows one way of accomplishing this.

Out-of-the-box Microsoft Analysis Services Drillthrough support

With minimal effort, you can quickly begin using the native Microsoft Analysis Services Drillthrough support by setting the GridBlox drillThroughEnabled property to true (default is false). Once enabled, the Drill Through option is added to the context (right-click) menu, available when right-clicking on a data cell in a grid. DB2 Alphablox automatically generates the appropriate DRILLTHROUGH statement and executes the query. The returned resultset is displayed in an interactive ReportBlox within a separate browser window.

Controlling drillthrough window styles

While the default drillthrough window may be adequate for your purposes, DB2 Alphablox also allows you to customize the display window by use of the nested GridBlox `<blox:drillThroughWindow>` tag. When this tag is nested in a GridBlox that has drillthrough support enabled, it overrides the default out-of-the-box behavior and allows custom browser window properties to be defined.

The tag attributes on the `<blox:drillThroughWindow>` are modeled after the most commonly used window definition properties defined in the `features` argument of the JavaScript `window.open(url,windowName,features)` method that is used to open browser windows. The supported properties include the following:

Tag Attribute
Description

`url` Defines the location of the JSP for the drillthrough window

`name` Name of the drillthrough window

`height` Height of the window

`width` Width of the window

`resizable`
Boolean property determining if the drillthrough window can be resized by users. True by default.

`statusbarVisible`
Boolean property determining if the drillthrough browser window's status bar should be visible. True by default.

`scrollbarVisible`
Boolean property determining if the drillthrough browser window's scroll bars should be available. True by default.

`locationbarVisible`
Boolean property determining if the drillthrough browser window's location bar (address bar) should be displayed. True by default.

`toolbarVisible`
Boolean property determining if the drillthrough browser window's toolbar should be displayed. True by default.

`menubarVisible`
Boolean property determining if the drillthrough browser window's menu bar should be displayed. True by default.

For details about the `<blox:drillThroughWindow>` tag and its attributes, see the GridBlox Reference section of the *Developer's Reference*.

Custom Drillthrough Support Using DB2 Alphablox Relational Reporting

DB2 Alphablox Relational Reporting Blox, discussed fully in *Relational Reporting Developer's Guide*, can be used to generate custom drillthrough support with many desirable features not possible in the native Microsoft Analysis Services drillthrough support. In Blox Sampler, under Retrieving Data for both DB2 OLAP

Server (and Essbase) and Microsoft Analysis Services, there is an example of custom drillthrough support. Here we walk through the code, explaining the most important code parts:

1. In your JSP file, enable the Drillthrough context (right-click) menu option by adding the drillThroughEnabled attribute to the <blox:grid> tag, setting the value to true:

```
<blox:grid ...
      drillThroughEnabled="true" ... />
```

2. Set up the interception and handling of the right-click event on the Drillthrough context (right-click) menu option by adding a <bloxui:actionFilter> tag to the PresentBlox:

```
<bloxui:actionFilter
  className="<%= MyDrillThroughClass.class.getName() %>"
  componentName="dataAdvancedDrillThrough" />
```

   The componentName attribute is set to the value of dataAdvancedDrillThrough and the className attribute must be set to the name of the class you're adding to handle the right-click event.

3. Add a JSP page directive at the top of the page, specifying the required classes for your page:

```
<%@ page import="com.alphablox.blox.uimodel.ModelConstants,
com.alphablox.blox.uimodel.tags.IActionFilter,
com.alphablox.blox.DataViewBlox,
com.alphablox.blox.uimodel.core.Component,
com.alphablox.blox.uimodel.core.MessageBox,
com.alphablox.blox.uimodel.GridBrixModel,
com.alphablox.blox.uimodel.PresentBloxModel,
com.alphablox.blox.uimodel.core.grid.GridCell,
com.alphablox.blox.uimodel.GridBrixCellModel,
com.alphablox.blox.uimodel.core.ClientLink" %>
```

4. Add the handler class for the actionFilter:

```
<%!
  public static class MyDrillThroughClass implements IActionFilter
  {
    public void actionFilter( DataViewBlox blox, Component component )
      throws Exception {
        GridBrixModel grid =
           ((PresentBloxModel)blox.getBloxModel()).getGrid();
        GridCell[] cells = grid.getSelectedCells();

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

        }
       GridBrixCellModel cell = (GridBrixCellModel)cells[0];
       int rowIndex = cell.getNativeRow();
       int colIndex = cell.getNativeColumn();
       String bloxName = blox.getBloxName();
       String urlStr = "someReportBlox.jsp?bloxRef="+bloxName;
       urlStr += "&colIndex=";
       urlStr += colIndex;
       urlStr += "&rowIndex=";
       urlStr += rowIndex;

        String timestamp = String.valueOf(System.currentTimeMillis());
```

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

5. Set up the custom relational report. In this example, `someReportBlox.jsp`:

```
<%
  String reportName = request.getParameter("reportName");
  if(reportName == null) {
  reportName = "defaultName";
  }
%>
```

6. Set up your custom relational report using the `<blox:RDBResultSetData>` or `<bloxreport:RDBResultSetData>` tag of the Blox Reporting Tag Library.

```
<blox:report id="drillThrough"
  bloxName="<%= reportName %>"
  interactive="true">
  <blox:rdbResultSetData
  bloxRef="<%= request.getParameter(\"bloxRef\") %>"
  columnCoordinate="<%= request.getParameter(\"colIndex\") %>"
  rowCoordinate="<%= request.getParameter(\"rowIndex\") %>">
</blox:rdbResultSetData>

...
```

   [Optional] Setup a default ReportBlox name if you want to support the opening of multiple reports.

   [Optional] Set the `bloxName` attribute for the ReportBlox if you are using multiple reports. Details about the use of the `bloxName` attribute can be found in the Common Blox Reference section of the *Developer's Reference*.

   [Optional] Define a MembersBlox if you want to permanently exclude members from the view.

   [Optional] Define a GroupBlox if you want to create a cleaner layout using break groups and aggregations.

   [Optional] Define a CalculateBlox to add replicated calculations which MSAS does not natively support.

   [Optional] Define an OrderBlox to order the columns. Natively, MSAS DrillThrough support does not let you reorder the columns.

Review the complete code to perform the custom drillthrough in the Blox Sampler example.

For details on creating relational reporting views, see the *Relational Reporting Developer's Guide*. In the following example, we'll cover some important points relevant to using relational reporting for drillthrough. In this example, we'll call the page being retrieved `someReportBlox.jsp` and use the `<blox:RDBResultSetData>` tag to get the drillthrough data into the relational pipeline.

Other custom drillthrough support

To provide your own customization, one option is to intercept the right-click event controlling the drillthrough support provided by DB2 Alphablox, then use a server-side `ClickEvent` to manage drillthrough customization.

If you want to develop a drillthrough report using your own solution, instead of using the Blox Reporting Tag Library, you need to use one of the following DataBlox methods:

- `RDBResultSet drillThrough(Tuple[] coordinates)`
- `RDBResultSet drillThrough(int columnCoordinate, int rowCoordinate)`

These methods will return an `RDBResultSet` containing the relational data for the drillthrough performed at the specified coordinates. Then, you can display the relational drillthrough data as you'd like.

For details about the RDBResultSet object, see the Relational Result Set Methods section of the DataBlox Reference in the *Developer's Reference*.

Relational data sources

DB2 Alphablox supports the viewing of relational result sets using the presentation Blox. Similar to the other supported databases, you need to specify a data source and a query. When relational result sets are displayed, a limited subset of DB2 Alphablox functionality is available in the standard presentation Blox. ReportBlox can also be used to display relational data, and offers support for many reporting purposes. See the *Relational Reporting Developer's Guide* for details on using the new relational reporting functionality to display relational reports.

Below is a quick overview and summary of SQL statements, and how to use them with DB2 Alphablox.

Creating SQL Statements

To pass a query to a relational data source, use the SQL `SELECT` statement syntax supported by your RDBMS. For example, the following SQL syntax is supported by several relational data sources:

```
SELECT... FROM... WHERE... ORDER BY... GROUP BY...
```

- `SELECT (ALL|DISTINCT) [COLUMNS]` to identify the data columns to include in the result set
- `FROM [TABLELIST]` to identify the name of each database table from which to obtain data
- `WHERE [PREDICATE EXPRESSION]` to specify filters and joins on the data
- `ORDER BY [COLUMNNAMES]` to specify a sort sequence
- `GROUP BY [COLUMNNAMES]` to specify a group list

Note: Functions are **not** supported.

The following example specifies that:

- Columns named `SalesQty` and `ProductID` are selected from two tables (named `Actual` and `Projected`)
- Only those rows are selected where the actual quantity sold is less than the projected quantity sold

```
<blox:data query="SELECT Actual.SalesQty, Actual.ProductID,
  Projected.SalesQty, Projected.ProductID FROM Actual,
  Projected WHERE Actual.SalesQty < Projected.SalesQty".../>
```

Query Builder

The Query Builder provides an easy way to develop and test query syntax. The tool uses a point-and-click interface and does not require in-depth knowledge of a data source's query language.

Query Builder supports interactive query development. You simply manipulate a grid that displays a default result set until they achieve the appropriate data view. Clicking the Get Current® Query button displays the query statement in the correct syntax. You can copy and paste the statement into a text file for later use, or directly into the value of a query within an application template.

Another powerful feature of the Query Builder is the Generate Blox Tag button, which allows you to retrieve the Blox tag syntax required to reproduce the PresentBlox showing the same view and result set. Similarly, you can copy and paste the tag into a text file for later use, or paste it directly into your application.

Using Query Builder

The following task shows you how you can use the Query Builder to begin learning about query statements and how they impact views in DB2 Alphablox applications.

To access Query Builder:

1. On the Application Studio page, click Workbench. The Workbench page opens.
2. Click the Query Builder link.
3. Click on the Connection Settings button to open the drop down list. Select the data source for which you want to build a query. Values for the Catalog, Schema, Username, and Password fields are taken from the data source definition and appear in the text boxes.
4. If necessary, change the values for Catalog, Schema, Username and Password. Click the Execute Default Query checkbox if you want to execute the default query upon connection.
5. Click the Connect button. Upon successful connection, a confirming indicator appears in the Status Frame.
6. Do any of the following:
   - Type a query string into the text box and click the Execute Query button. The results appear in the PresentBlox at the bottom of the page.

     Click the Get Current Query button to retrieve the most recent successful query against this data source. The query string appears in the text box.
   - Click the Default Query button. The default query string associated with this data source appears in the text window. If the data source is an Alphablox cube or an Microsoft Analysis Services cube, ensure that the cube name is correct.

     Note: With the Relational Database connection pooling feature, you will not be able use the Default Query button to get the default query on a relational cube.
   - To view the results of the query, click the Execute Query button. The result set appears in the PresentBlox at the bottom of the page.
   - When data appears in the PresentBlox at the bottom of the page, use the standard user interface to swap axes, drill up or down, move dimensions between axes, and so forth.

- Click the Generate Blox Tag button. This opens a text box containing the complete tag to duplicate the view and result set seen in the PresentBlox.
- Expand the Asymmetrical Query Builder pane, deselect any columns to remove, and click the Apply Column Set button.

  Note: The Asymmetrical Query Builder pane only works with DB2 OLAP Server or Essbase data sources, and only on column headers.

7. After developing the appropriate data view, click the Get Current Query button. The text box displays the query string required to develop the current data view.

   After deriving the appropriate query string, you can copy and paste it into a text file for future use, or directly into the value of your query. To exit Query Builder, close its browser window.
8. Click the Generate Blox Tag button. A text box opens displaying the tag to reproduce the PresentBlox layout and result set. You can copy and paste this tag into a text file for later use, or paste it directly into an application.

Working with JDBC data sources

DataBlox lets you connect to multidimensional as well as relational data sources. Once your relational data source is defined to DB2 Alphablox through the DB2 Alphablox Admin Pages, you can use DataBlox to connect to the data source and retrieve data. However, with JDBC data sources, you might need to perform specific JDBC calls, obtain JDBC URL connection string, or take advantage of connection pooling or stored procedures. DB2 Alphablox provides a JDBCConnection bean that allows you to construct JDBC connection strings from DB2 Alphablox JDBC data sources. DB2 Alphablox also offers StoredProceduresBlox for using stored procedures in relational databases.

Using the JDBCConnection Bean

The JDBCConnection bean is a Java bean that allows you to get information about a 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 Example

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>
```

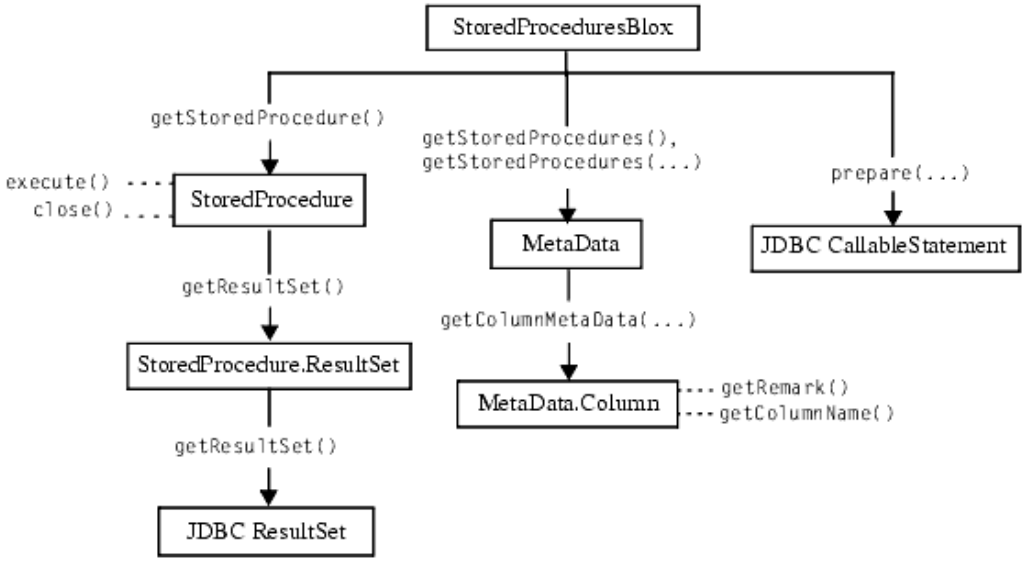
Using StoredProceduresBlox

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 examples

This section includes six examples that demonstrates the use of StoredProceduresBlox. For more examples, see the Javadoc documentation.

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();
%>
```

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();
%>
```

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.

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>
<% }
%>
```

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();
%>
```

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);
%>
```

Chapter 15. Presenting data

The common presentation Blox (GridBlox, ChartBlox, and PresentBlox) offer many alternatives to developers for customizing the analytic views. This topic will help you decide which Blox you should use and how to effectively use presentation Blox.

Choosing Blox for presenting data

In DB2 Alphablox applications, you can use any of the common presentation Blox (GridBlox, ChartBlox, and PresentBlox) to display result sets to end users, but the decision on which particular Blox to use depends on your user requirements.

Presentation Blox can only display the results from one data source at a time. If you need to display results from multiple data sources simultaneously on the same page, you have a couple of options:

- Place multiple Blox on a page, each pointing to a different data source.
- Use a single Blox on a page to display results from multiple data sources by using either the server-side Java API or client-side Blox Client API to change data sources. See the "Setting different data sources using DataSourceSelectFormBlox" on page 112 for an example.

Choosing data presentation Blox components

As discussed earlier in this guide, the primary Blox of interest to users are the Blox which display data: ChartBlox, GridBlox, and PresentBlox. This guide focuses on the use of the presentation Blox available in the DHTML client.

Below is a table summarizing the pros and cons of each of the data presentation Blox:

Blox	Advantages	Disadvantages
GridBlox	• users can readily compare numbers that show small differences • alerts can be created to highlight information in particular cells • information links (using the `cellLink` and `cellAlert` properties) can be added to individual cells or groups of cells • information links (using header links defined in the application definition) can be used to add links to information in row and column headers	• users may have more difficulty spotting large differences between values, when trying to quickly view differences • no access to DataLayout panel

Blox	Advantages	Disadvantages
ChartBlox	• displays multidimensional or relational data in a variety of chart formats	• viewing data visually can obscure the differences between values when values vary only slightly • information links (see GridBlox advantages) cannot be accessed on charts • users cannot see alerts tied to the data • no access to DataLayout panel
PresentBlox	• combines both grid and chart views of data into a single Blox • users can toggle between grid and chart view, or display both views simultaneously (when split pane is enabled) • DataLayout panel allows advanced users to manipulate display of dimensions • with only a grid or a chart enabled, users can have access to the DataLayout panel or the Page panel (these are not available in standalone GridBlox and ChartBlox)	• sometimes users should only be allowed to see only a grid, when data density is too high (too many data points in a chart can be difficult to interpret); the PresentBlox `chartEnabled` property can be set to false, but this eliminates one of the advantages of a PresentBlox

Render formats available to the DHTML Client

The DHTML client's rendering is one of many different Blox rendering formats available. By default, new applications created on the DB2 Alphablox render pages in the DHTML format. Other application default rendering options are definable in the application's definition page. These other available rendering formats, printer and export to Excel, are based on the original DB2 Alphablox rendering model. Depending on the particular rendering format being used, the rendering format may be accessed through the common Blox `render` property, the render URL attribute, or through other mechanisms described below.

Note: DHTML rendering uses the UI Model-based rendering mechanism, allowing the UI to be fully customized. When a Blox is exported to Microsoft Excel, the code responsible for performing the operation is not based on the UI Model and may appear different from the current view in the DHTML client.

A summary of the rendering modes available for use with the DHTML client follows.

DHTML format (render=dhtml)

By default, applications created on the DB2 Alphablox are rendered using DHTML. This format uses standard DHTML technologies (HTML, CSS, DOM, and JavaScript) along with server-side Java technologies to create renderings of data in a highly interactive format that matches Java clients in power, but without the

need to use Java plug-ins or Java-enabled browsers. An additional benefit of the DHTML rendering format is that the user interface, defined by the Blox UI Model, can be extended and customized beyond the built-in support of the DHTML format.

The DHTML rendering format is supported when using either the common Blox `render` property or the `render` URL attribute.

Note: Blox cannot be rendered in DHTML if a page is set to render in a different format since essential JavaScript files will not load.

Printer format (render=printer)

The Printer format generates a view of a Blox's data that is optimized for printing purposes using your browser's built-in printing functionality. The Blox view is generated using HTML tables and CSS styles, and also converts all selectable page filters into a list of selected filters including the dimension names and their selected members. The Printer format generates visual representations based on the original DB2 Alphablox rendering model, and thus will appear different than views presented in the DHTML client. Charts rendered using the Printer format will display using the chart package supported by the DHTML client. To use the NetCharts charting package used with the DHTML client, the application's default rendering mode (specified on the application definition page in the DB2 Alphablox Admin Pages) must also be set to DHTML.

The Printer rendering format is supported as both the common Blox `render` property or using the `render` URL attribute.

Typically, to use this format, you would include an HTML button or link on a page with a Blox view, allowing the user clicks to request a printable copy. In response to a user's request, the DB2 Alphablox renders the page in pure HTML before delivering it to the client. After the page appears in the browser, the user can click the browser's Print button to send the page to the printer. For details, see "Printing Blox output" on page 154.

PDF format

DB2 Alphablox offers an alternative print delivery mechanism, Convert to PDF, which can be used to convert individual Blox views into Adobe Acrobat PDF files. Users frequently like having access to PDF files for future reference or for sharing with others. DB2 Alphablox PDF format offers options for limited customization of page layouts, for example, including the generation of headers and footers with logos and defined text. Unlike the other rendering formats covered in this section, the PDF format cannot be selected using a `render` URL attribute or the common Blox `render` property. This format generates PDF files based on the original DB2 Alphablox rendering model, and thus will appear different than views presented in the DHTML client. Charts rendered using the PDF format will display using the chart package associated with the render mode for the application. To use the NetCharts charting package used with the DHTML client, the application's default rendering mode (specified on the application definition page in the DB2 Alphablox Admin Pages) must also be set to DHTML.

To learn more about how Convert to PDF works and to add this feature to your applications, see Chapter 23, "Converting to PDF," on page 237.

Export To Excel format (render=xls)

DB2 Alphablox can deliver the output of a page in an HTML format that is loaded into a Microsoft Excel spreadsheet. When using this format, the MIME type on the page being returned is set to `application/vnd.ms-excel`, the standard MIME type for Microsoft Excel. With this export facility, which uses the `render=xls` URL attribute, users can deliver the grid data to the spreadsheet program for further analysis. For a complete explanation, see "Exporting Blox views to Microsoft Excel" on page 156.

XML format

With an standalone DataBlox (that is, one that is not nested inside of another Blox), you can render a query result set from an application data source into XML format by setting the `render` URL attribute to `xml` (`render=xml`). Using the XML format is explained further in "Exporting to XML" on page 233.

Specifying delivery formats

By default, application pages are delivered in DHTML format. There are no special steps required to enable multiple application delivery formats. An attribute added to an application's URL informs DB2 Alphablox to deliver the page in the specified format. For example, the following URL requests that the MyApplication page be delivered in DHTML format:

```
http://<server>/applications/ThisView.jsp?render=dhtml
```

And, if you want a page to be rendered in Printer format, the URL would look similar to this:

```
http://<server>/applications/ThisView.jsp?render=printer
```

Valid values when using the `render` URL attribute with the DHTML client include:

Render attribute
: **Description**

`dhtml`
: Default. Renders Blox using DHTML technologies on the client.

`none`
: Remove all Blox from the page.

`printer`
: Render in printer-ready format with no interactivity

`xls`
: Render the page to an HTML format and sets the page's MIME type to `application/vnd.ms-excel`, resulting in the page being exported to Microsoft Excel

`xml`
: Render into XML format (applicable only to explicit DataBlox)

Printing Blox output

Two alternatives, the Printer format and the Convert to PDF option, are available for generating printable pages displaying Blox results. The Printer format renders Blox components on a web page using PNG images and HTML tables. Also, any PageBlox page filters (either standalone or nested in a PresentBlox) are converted from interactive HTML selection lists into static lists of dimensions and the selected slicers for each dimension.

The Convert to PDF option generates a PDF file allowing more flexibility over the resulting printable file by the assembler and user. For details on using the Convert to PDF option, see Chapter 23, “Converting to PDF,” on page 237.

Printing with HTML-based printing

Although Blox on HTML pages can be printed using the browser’s Print button, DB2 Alphablox can be used to render Blox output into a more printer-friendly format, as described in the example below. The rendered output appears in a browser window, replacing the interactive Blox with its printable version.

Note: By default, Microsoft Internet Explorer browsers do not print background colors and images. In order for users to print background colors and images in Internet Explorer, the user must manually configure their browser. As a result, you should assume that the majority of users will not have this setting enabled. Since you have no control over this browser setting, you should design printable pages with the assumption that most users have not modified this setting— most will neither know about this setting nor where it can be found in their browser options.

To set this property, from the browser’s menu bar, click View, then Internet Options. Select the Advanced tab and scroll down to the Printing section. Then check the Print background colors and images option, and click Apply.

Creating printable pages using the render=printer URL attribute

The simplest method for rendering printable pages is to use the `render=printer` URL attribute. By appending `?render=printer` to the end of the URL for a page, the Blox on that page will be rendered in a printer-friendly format. If the Blox is a PresentBlox, selectable page filters will appear in a list at the top of the Blox view, indicating to viewers which slicers were active when the printable copy was generated. Instead of applying the URL attribute to the existing page, it is usually preferable to open a new browser window displaying the printable version of a view.

Follow these steps to create printable pages using this method:

1. On a page with Blox on it, add a button or link to generate a printable page.

   For example, the following HTML code creates a button, labeled “Print Preview” in the body of a JSP page, that will open the current page’s view in a new window, rendering it in `printer` mode:

   ```
   <form>
   <input type="button" value="Print Preview"
      onclick="window.open('mView.jsp?render=printer','_new')">
   </form>
   ```

2. Now open your view and test the button. You should see a new window pop open with the current view rendered in a printable format.

You should also notice that this page includes the button you added. Typically, this would not be desired, so you could, for example, place common services buttons in one frame of a frameset, including a print button and export to Excel button, and display your analytic views in a separate frame. Then your buttons could trigger the opening of your view page in a separate, printable page without the buttons appearing on it.

A better alternative would be to open a new page in a separate window. A custom print page offers more potential as a reasonable printable page. The following task gives one approach for generating custom print pages.

Creating custom print pages using the <blox:display> tag

The following steps show you how to create a page that uses the `<blox:display>` tag to render a new page with the same data view, but in a printable format:

1. Create your analytic view and add a button that will be used to open a custom print page for this view. In the following code snippet, a button is created that will open a new window with `MyView-print.jsp` displayed within it.

   ```
   <form>
     <input type="button" value="Print View"
     onclick="window.open('MyView-print.jsp','_new')">
   </form>
   ```

2. Now, create your custom print page. Besides including a `<blox:display>` tag for rendering the view, you might consider including the following items:
   - title for the view being printed
   - summary of content
   - company name or logo graphic
   - date the page was generated
   - warning about usage (i.e., internal or confidential)
   - copyright notice

   The following code snippet is an example of a simple print page, which would generate a printable page that includes a title, Blox rendered in printer mode, and the date it was printed:

   ```
   <h2>My Grid View</h2>
   <blox:display bloxRef="MyGridBlox2" render="printer"/>
   <p>
     Printed: <script>document.write(new Date());</script>
   </p>
   ```

3. Save your custom print page.

Using this approach, you can offer many customized printing options.

A working example of the example above can be seen in Blox Sampler under Presenting Data. In the print page that is generated, users can select the browser's File menu to print the page.

Exporting Blox views to Microsoft Excel

Out of the box, an option for Export to Excel can be found on the Blox menu bars under the File menu. If a menu bar is not displayed, you may decide to offer an export to Excel option through a button or link on an analytic view. When rendering a PresentBlox to Microsoft Excel in `xls` format, only the grid output will be displayed in the spreadsheet. A ChartBlox cannot be rendered using the `xls` format.

The DB2 Alphablox export to Excel option is available for use by end users with the following minimum configurations:
- Microsoft Excel (see *Installation Guide* for supported versions).
- Microsoft Internet Explorer (see *Installation Guide* for supported versions).

CSS themes

DB2 Alphablox uses Cascading Style Sheets (CSS) themes to control aspects of the layout and appearance of pages rendered. Themes enhance the look-and-feel of applications. Custom themes can be created, helping your organization to adopt a corporate appearance in your DB2 Alphablox applications or to create appearances that integrate well with existing web-based applications or portals.

Specifying HTML themes in applications

Themes can be specified for the appearance of Blox in DHTML renderings in the server's Default HTML Client Theme setting in the DB2 Alphablox Admin Pages or using the `theme` URL attribute on a page.

The Default HTML Client Theme value can be specified in the DB2 Alphablox Admin Pages. Upon initial installation, the `coleman` theme is selected. To specify a different server default theme, click on the Administration tab, then under Server Properties on the left menu select System. Although the Default HTML Client Theme option allows many different theme options, only two themes, `coleman` and `financial`, are supported for use with the DHTML client.

To specify the `theme` property as a URL attribute, use the following format:

```
http://.../application/view.jsp?theme=financial
```

When you define a theme using a URL attribute, the `theme` property defined applies to all Blox on that page.

CSS theme files

The files for the two DB2 Alphablox CSS themes supported by the DHTML client, `coleman` and `financial`, can be found in the `theme` directory of the DB2 Alphablox Repository:

```
<alphabloxDirectory>/repository/theme
```

For the DHTML client, an DB2 Alphablox theme consists of the following structure:

File/Folder
: **Description**

<themeName>.css

<themeName>.properties
: Includes theme specifications about file locations, layout, and color. The `<themeName>.properties` file settings are specified below.

<themeName>_dhtml.css
: Includes theme specifications unique to the DHTML rendering format

i
: Folder containing images specific to the theme

DB2 Alphablox theme files are named based on the theme to which they belong (for example, `financial.properties` and `financial_dhtml.css` are the theme files for the `finanical` theme option. These files are located in a theme directory, also named based on the theme option. For example, the financial theme files are located in the following directory:

```
<alphabloxDirectory>/repository/theme/financial
```

Note: For portlet development, use the Portal Theme Utility to create a theme that will combine your selected portal theme with a DB2 Alphablox theme. This tool is available under the Administration tab on the DB2 Alphablox home page.

CSS theme properties defined in themeName.properties files

The theme properties file (`<themeName>.properties`) is a plain text file that defines the following DHTML-related theme properties:

Property
Description

`name =` *<themeName>*
The name of the theme (e.g., `coleman` or `financial`)

`description =` *description*
A description of the theme which appears on the server console

`bgcolor = #3d3d5f`
The background color for areas of ChartBlox not specified in `<themeName>.css`

`fgcolor = white`
The foreground (text) color for areas of ChartBlox not specified in *themeName*`.css`

`css = <themeName>.css`
The name of the *themeName*`.css` file

`background = true`
Causes the Blox background to appear

`layout=string`
Controls the placement of nested Blox within PresentBlox. For more information, see "Layout strings."

`privateimages = true`
For all the images used by this theme, use the theme's private images directory (`<repository>/theme/`*<themeName>*`/i/`)

`windowbgcolor = #655973`
The background color for a chart display area within PresentBlox

`windowfgcolor = white`
The foreground (text) color for the chart display area within PresentBlox

bkgrd_image_chart = *imageFile*
The background image to appear in charts

chart_color_series
The 18 colors used in creating chart colors, including lines, bars, pie slices, etc.

Layout strings

The display of a rendered PresentBlox is divided into multiple components:

Component Name
Contents

`viewarea`
The area presenting the grid and/or chart

`toolbar`
The Blox toolbar

page The PageBlox area

layout The DataLayoutBlox area

Each of these components can accept two options:

- `title` sets the title for the component.
- `orientation` specifies a vertical or horizontal orientation for the component (`viewarea` ignores this option).

Important: Layout strings must be specified as a single unbroken line in the Properties file (*themeName*`.properties`). The format and example shown are broken into multiple lines here for readability.

The simplest format of the layout string is:

```
layout = component1 {option: value; option: value;};
component2 {option: value; option: value;}; ...
componentN {option: value; option: value;};
```

The following example is the default.

```
layout = toolbar {title: Tool Bar; orientation: horizontal;};
minimizearea {title: Minimize Area; orientation: horizontal;};
page {title:Page Filters; orientation: horizontal;};
layout {title:Data Layout; orientation: vertical;},
viewarea {title: View Area;}
```

Note: Note the comma (not semicolon) between the `layout` and `viewarea` components, causing those two components to reside side by side in a single section, rather than in separate sections.

It produces the following PresentBlox layout:

Blox Toolbar
Minimize Area [links rendering mode only]
Interactive Settings [links rendering mode only]
Page Filters
Data Layout View Area

Important points to know about the layout string syntax:

- The PresentBlox display area can be divided into as many as six sections.
- Semicolons are used to separate sections.
- Up to three components can occupy a single section.
- Commas are used to separate components that occupy the same section.
- Layout strings must be specified as a single unbroken line in the Properties file (`<themeName>.properties`). They are broken into multiple lines here for readability.
- The extended format of the layout string is:

```
layout = componentN {option: value; option:
value;}[,componentN {option: value; option: value;}];
componentN
{option: value; option: value;}[,componentN
{option: value; option: value;}];
```

The financial theme uses the following layout string:

```
layout = toolbar { title: Tool Bar; orientation: horizontal; };
   minimizearea { title: Minimize Area; orientation: horizontal; };
   ibar { title: Interactive Settings; orientation: horizontal; };
   page { title: Page Filters; orientation: horizontal; };
   layout { title: Data Layout; orientation: vertical; },
   viewarea { title: View Area; };
```

CSS classes defined in the .css file

The *themeName*.css file is a plain text file that defines the CSS classes described below. To see the values set for a class in a particular theme, open the theme's *themeName*.css file, found here:

```
<alphabloxRepository>/theme/<themeName>/themeName.css
```

Tip: Cascading Style Sheets specifications can be found at the World Wide Web Consortium (http://www.w3c.org/style/css).

The following tables list the application-wide styles, the legacy styles, and the overrides available in the *themeName*_dhtml.css file:

Application-wide styles

Style Class
Description

csApBg Application background color - overall background of a Blox

csCmpBg
Component background color - data area backgrounds of individual Blox

csCmpBrdr
Component border - borders of individual Blox and controls

csThmClr
Base theme color - used by text labels, information text, decorative elements

csFntClr
Default font color - used by data, messages, functional text

csFntSpc
Default font spec - used mainly by menus, toolbars and buttons

csLblFnt
Default label font spec - used by labels for GUI interactive-controls

csGrdFnt
Default grid font specification - used by all grid cells

csSlctBg
Default selection background color

csSlctClr
Default selection font color

csDsbldClr
Default disabled font color

csThrDBrdrRsd
3D raised border

csThrDBrdrLwrd
3D depressed border

Legacy styles

Style Class
Description

csDmnsnHdrs
Dimension headers

csClmnHdrs
Column headers

csRwHdrs
Row headers

csRwHdrsNnBnd
Row headers - non-banded

csRwHdrsBnd
Row headers - banded

csDtCl Data cell

csTdClBnd
Data cell - banded

csDtClNnBnd
Data cell - non-banded

Overrides

Style Class
Description

csBckClr
Background color (Blox components)

csWndwClr
'Window' color (Blox components)

csRwHdrs
Row headers

csRwHdrsNnBnd
Row headers - non-banded

csRwHdrsBnd
Row headers - banded

csDtCl Data cell

csTdClBnd
Data cell - banded

csDtClNnBnd
Data cell - non-banded

Overriding defined styles

One of the results of adding the <blox:header> tag to the head section of JSP pages is to provide an automatic link to the appropriate CSS file.

Note: See also the information on applying styles to data cells and cell alerts, described in "Using format masks to highlight data" on page 171 and "Using cell alerts to highlight data" on page 172)

To override defined styles, you can create an entirely new theme:

1. Copy an existing theme directory, giving it a new name.
2. Rename the *themeName*.css and *themeName*.properties within that directory.
3. Make the appropriate content changes to these files.
4. In the application URL, provide the name of the new theme. For a page named view.jsp, the URL might become:

   /<applicationName>/view.jsp?theme=MyTheme

Note: To load a new theme or reload a modified theme, use the load theme command in the DB2 Alphablox console. Note that this command does not accept a specific theme name as a parameter; the command simply loads all themes from the Repository.

Note: Web browsers frequently cache files, such as buttons and icon GIF files. When testing a new or modified theme, you may need to make sure the browser cache's cache is cleared before you can see changes you've made.

Applying styles to cell alerts

The styles for cell alerts are not defined in a CSS file. Instead, they are defined inline in the HTML, based on the DataBlox cellAlert property. Because CSS has cascading effect, the inline style is the one rendered by the browser, overriding any other styles defined for a data cell.

User interface appearance

Blox can be configured to display different color schemes, fonts, and banding characteristics which can complement the appearance, or "look and feel," of your application pages. Just as you can control the colors of fonts and background colors on web pages using Cascading Style Sheets, you can also use Blox properties and HTML themes to control the appearance of Blox on your application pages. In the following sections, common appearance properties are discussed.

Included in the table below is a list of the most commonly modified appearance features on the presentation Blox:

Blox **Common Appearance Properties**

GridBlox

- missingValueString: determines what to display in place of default #MISSING
- rowHeadingsVisible: specifies whether the row headings to the left of the data values appear on the grid
- gridLinesVisible: turn grid lines on or off

ChartBlox

- `chartType`: change chart type
- `depthRadius`: can be used to create a slight 3D effect on standard bar charts
- labels and placement

PresentBlox

- `dividerLocation`: when splitPane is enabled, determines where the divider should appear on loading of the PresentBlox
- `splitPane`: enables a splitter bar that allows simultaneous display of grid and chart
- `splitPaneOrientation`: determines whether the splitter bar is horizontal or vertical

DataBlox

- `suppressMissing`: suppresses rows or columns where there is no data
- `suppressNoAccess`: suppresses visibility of rows and columns where the user has no access rights to the data
- `suppressZeros`: suppresses rows or columns where all zeros appear

ToolbarBlox

- `removeButton`: removes defined buttons from the toolbar
- button color and size

ReportBlox

- See *Relational Reporting Developer's Guide*. CSS style settings can be modified for most HTML elements

For complete listings of appearance properties, see the listings of appearance properties and methods in the "by Category" sections of each Blox Reference section in *Developer's Reference*.

The next sections, discuss in more detail some of the common appearance properties used in the GridBlox, ChartBlox, and PresentBlox.

Grid Appearance

Grids can be a great source of information, but sometimes reading them or being able to notice important information is difficult. As a developer, you can change many of the appearance properties of grids, but the following are the most commonly changed properties:

- `missingValueString`: determines what to display in place of default `#MISSING`
- `rowHeadingsVisible`: specifies whether the row headings to the left of the data values appear on a grid
- `gridLinesVisible`: turn grid lines on or off

The sections below discuss some of these properties and how to use them to create more usable grids.

Row banding

In a grid with many rows, row banding is enabled by default. If necessary, for aesthetic reasons or to avoid classes with cell alert colors you may have selected, you can changed the background and foreground colors used in row banding by modifying CSS theme properties. For information on CSS themes, including a list of modifiable CSS themes, see "CSS themes" on page 157

Cell appearance

The grid data cells that display the result sets can be customized to display in different foreground and background colors, and fonts. To control these options, use CSS themes. For information on CSS themes, including a list of modifiable modifying CSS themes, see “Printer format (render=printer)” on page 153

Chart Appearance

The appearance of charts within DB2 Alphablox applications can be customized in many ways to meet your users’ particular needs. A few of the commonly changed ChartBlox properties, `chartType`, `depthRadius`, and `chart_color_series` are discussed here.

Chart Types

The most frequently changed property on a ChartBlox is `chartType`, which defaults to `Vertical Bar, Side-by-Side, 3D Effect`. ChartBlox has many other chart types available to cover almost every user’s needs, but four of the most commonly used chart types are `Bar`, `Line`, `3D Bar`, and `Pie`. You can either use these short names or their full chart type names in the `chartType` property setting. For a complete listing of valid names for all of the supported chart types, see the ChartBlox Reference section of the *Developer’s Reference*. The following table lists the four available chart shortcut names and their full chart type names:

Shortcut Name	Full Chart Type Name
`Bar`	Vertical Bar, Side-by-Side
`3D Bar`	3D Bar
`Line`	Vertical Line, Absolute
`Pie`	Pie

Adding 3D appearance to charts

The default `chartType` property for ChartBlox is `Vertical Bar, Side-by-Side, 3D Effect`, which results in a two-dimensional bar chart with a slight 3D effect. A standard bar chart (`Bar` or `Vertical Bar, Side-by-Side`) or line chart (`Line` or `Vertical Line, Absolute`) is flat and two dimensional, but by setting the `depthRadius` property, you can add a subtle 3D effect of your own choice to the these charts, giving them a different look. To add a subtle bit of depth to the flat bar and line charts, combine the `depthRadius` property along with the `chartType` property, for example like this:

```
<blox:chart
   ...
   chartType="Bar"
   depthRadius="5"
   ... />
```

Acceptable values for `depthRadius` are integers between `0` and `100`. The `depthRadius` setting will also affect the appearance of other 2D charts.

Chart colors

The chart colors used to create the lines, bars, and pie slices can be set based on the CSS theme being used. To specify chart colors other than the default colors

used in a particular theme, you can define your own chart color series, consisting of 18 different colors. To specify a different color series to be used in a theme, open the *themeName*.properties file (for example, coleman.properties), found in the following directory:

<alphabloxRepository>/theme/*themeName*/

Find the chart_color_series property and redefine the 18 colors used to define colors. Chart colors are specified using hexadecimal code (e.g., #E0CB68), like the values commonly used in web pages.

PresentBlox appearance

The following topics describe a few ways that you can easily change the appearance of a PresentBlox.

Split panes

By default, the PresentBlox instantiates displaying two panes, one with a grid and the other with a chart. The splitPane property, by default set to true, allows users to view their data in tabular and graphical representations simultaneously. Also, as a user interacts with the data in one pane, the other pane is simultaneously updated to reflect those changes. For example, when the user drills down on the data in the chart, the grid pane reflects the new result set.

With split panes available, the divider is set to vertical by default. When more than a few items will appear on the y-axis of your graph, you may want to consider changing dividerLocation to horizontal, displaying both the grid and chart across the full width of the PresentBlox. Frequently, when using a horizontal setting, you may also find that the chart looks better when it appears above the grid. This can be specified by setting the chartFirst property to true, overriding the default.

Due to a lack of available screen space, or because the data density of the initial result set may make the initial graph unreadable, you may think it would be better to set splitPane to false. While this is a reasonable choice, you may want to consider leaving the split pane option available to the user (consider that this particular option cannot be changed in any of the available Toolbar options), but change the initial display location of the split pane divider, setting it to one side but leaving it available.

The dividerLocation property allows you to set the initial location of the splitter bar. The acceptable values range from 0 to 1, with value of 0 meaning that only the display on the right (or bottom, depending on the splitPaneOrientation setting) should appear, and a value of 1 meaning that only the display on the left (or top) should appear. Try a few different settings to see if it makes sense to change it from the default value of 0.5.

For complete information about the splitPane, splitPaneOrientation, dividerLocation, and chartFirst properties, see the PresentBlox Reference in the *Developer's Reference*.

Modifying DataLayout properties

By default, the DataLayoutBlox, or DataLayout panel, is available, but not visible to end users. For analytic views where you will have mostly advanced users, you

may decide that you want to have the DataLayout panel visible when a PresentBlox loads. To make the DataLayout panel visible when the PresentBlox loads, you need to set the nested DataLayoutBlox's `visible` attribute to `true`, as in this example:

```
<blox:present id="myPresentBlox">
  ...
  <blox:dataLayout visible="true"/>
  ...
</blox:present>
```

If you do not want the DataLayout panel to be available to users, for example if you expect only casual users to view and use a PresentBlox, you can disable the DataLayout panel by setting the PresentBlox `dataLayoutAvailable` attribute to false. This will automatically result in the DataLayout button not appearing on the toolbar. The following code snippet shows the proper usage of this property:

```
<blox:present id="myPresentBlox"
  dataLayoutAvailable="false">
  ...
</blox:present>
```

The availability of the DataLayout panel is determined by the PresentBlox `dataLayoutAvailable` property setting, and is an attribute on the `<blox:present>` tag. The visibility of the DataLayout panel is a property of the DataLayoutBlox object itself and is thus controlled using the `visible` attribute of the `<blox:dataLayout>` tag.

Modifying menu bar properties

The menu bar is the text-based menu appearing at the top of Blox, automatically incorporating relevant menus based on whether the Blox is a GridBlox, ChartBlox, or PresentBlox. By default, the `menubarVisible` tag attribute of the `<blox:grid>`, `<blox:chart>`, and `<blox:present>` is set to true. To remove the menu bar from one of these Blox, add the `menubarVisible` tag attribute to the Blox tag and set the value to `false`.

Other than displaying or not displaying the menu bar, there are no tags or tag attributes to control its appearance. Advanced developers can use the extensibility of the Blox UI model to uniquely customize the menu bar.

Modifying toolbar properties

By default, the toolbar is available to users on a PresentBlox. There are a couple of common appearance settings that are frequently modified by developers. The toolbar is considered always available, but can be set to not be visible to users using the `visible` attribute of the ToolbarBlox tag.

If you decide to keep the toolbar visible, you may still choose to disable, or remove, some of the toolbar buttons. To remove buttons from the toolbar, use the `removeButtons` property of the nested ToolbarBlox to remove any buttons you decide are unwanted. For example, the following example shows the Help would be removed from a ToolbarBlox:

```
<blox:present ...>
  <blox:toolbar removeButtons="Load,Save,Help"/>
</blox:present>
```

Note: Note that you must remember to add `Load` and `Save` to the `removeButtons` tag attribute since they are included in the default value string for this property. If you forget to add them to the string, the Load and Save buttons will appear in the toolbar.

Data appearance

When you retrieve result sets from a data source, the data may either be returned preformatted or not. Once the results are retrieved into an DB2 Alphablox application, you have several options for controlling the appearance and formatting of data. Using DataBlox properties, you can suppress rows or columns with zeros, missing data (or null values), duplicate data, or data that a user does not have access rights to. In a GridBlox, you can format the numbers, including symbols ($, %, etc.) or groupings (showing commas or periods). Also, in a GridBlox, you can display numbers in thousands, millions, or whatever grouping you find appropriate for your data.

GridBlox properties

Three GridBlox properties are useful for changing the formatting of data values, and thus your data's appearance on grids, include `defaultCellFormat`, `cellFormat`, and `formatMask`. In the following tasks, you'll learn how to use these format mask properties to solve some frequently encountered tasks. For complete details on the use of these properties, see the GridBlox Reference in the *Developer's Reference*.

Formatting values in thousands and billions

A common request from end users is to get rid of unnecessary noise in the data by eliminating, or essentially rounding out, irrelevant numbers. If a user is working on a budget that is many thousands of dollars, then seeing the cents and even dollars may not be of any use, and will occupy unnecessary space in the data cells. As a developer, you can help this user out by changing the `defaultCellFormat` property of the GridBlox. For example, to display all grid values in thousands you have two options. To display the value in thousands followed by a "K," representing thousands, enter the following setting:

```
defaultCellFormat="#,###K"
```

The `K` in the value tells DB2 Alphablox that you want the numbers in thousands, but to append a `K` to the end of the value.

While the `K` indicates that the value is displaying in thousands, many users prefer not to see the `K` in every field. To display the grid values in thousands, but without the `K` suffix, you can use a feature, available on all format masks in DB2 Alphablox applications to calculate the numbers in thousands by entering the following setting:

```
defaultCellFormat="#,###/1000"
```

In some situations, you may want to display a special character at end, for example a "B" for billions. To add a "B" to the end of this value, modify the previous setting to show:

```
defaultCellFormat="#,###/1000'B'"
```

For details on use of `defaultCellFormat` and other format mask properties, see the GridBlox Reference in the *Developer's Reference*.

Displaying percentages for specific members

When using the GridBlox `defaultCellFormat` property, the entire grid of values is affected. Frequently, you will want to limit the formatting of values to a particular row or column, or use the `defaultCellFormat` property for all data values except for those for a particular member. To limit number formatting to a specific member, affecting only a single row or column, you can use the `cellFormat` property. Unlike the `defaultCellFormat` property, `cellFormat` is an indexed property and requires the use of its own Blox tag. Because it is a separate tag and represents a GridBlox property, the `cellFormat` tag must be nested within the body of a GridBlox tag. Here is a code snippet showing what a `cellFormat` tag would look in a non-nested GridBlox if you only wanted to display your Variance % values with the percent symbol (%) after each value for that member:

```
<blox:grid id="myGridBlox">
  <blox:cellFormat
    format="#,###.00%"
    scope="{Scenario: Variance %}" />
</blox:grid>
```

In this example, Variance % would show values in percent to two decimal places.

Controlling decimal appearances

While it may not be immediate obvious to you how decimals affect the appearance of your data, here's an example where data display makes the readability of a grid more difficult. In the following grid, notice how the numbers in the column do not line up in a row with any of the value places appearing in alignment:

7.654
3.21
43.21
543.2
3
3.2

As you notice, even in this small sample of numbers, it is difficult to compare values since you have to mentally try to line up the numbers based on the decimal location. You can define format masks so that all of the decimals on these values line up in the column. For example, you could have the `Variance %` column up using a `<blox:cellFormat>` tag, like this:

```
<blox:cellFormat
  format="#,###.000"
  scope="{Scenario:Variance %}"/>
```

The `Variance %` column would now display data like this:

7.654
3.210
43.210

543.200
3.000
3.200

As you have learned in this example, the appropriate format mask will make your data more readable, more meaningful, and better looking. Format masks can also be used to display negative values in parentheses or in red, display currency symbols, and percentage symbols. See the GridBlox Reference in the *Developer's Reference* for other format mask options.

Chapter 16. Highlighting and commenting on information

How can you call attention to information that differs in some significant way from the rest of the data? Commonly referred to as “exception reporting” or “traffic lighting,” a common goal is to alert users to information that may be important to make decisions upon. This topic discusses the use of DB2 Alphablox cell alerts and information links to solve this problem. Cell alerts can be used to highlight information by changing data cell styles, and can also be used to show links based on some criteria. Information links, including header and cell links, can also provide a way to highlight cells, leading users to more information.

The ability for users to add and view comments on specific data in a grid is another powerful way to highlight information. A topic in this section covers how to use CommentsBlox to add this feature your applications.

Overview

Along with access to the wealth of information stored within your company’s databases comes the problem of how to help users find important information quickly. DB2 Alphablox developers can use techniques such as cell alerts and hyperlinks on grids to either highlight information based on some business criteria or to link users to further information relevant to the application they are using. In the following sections, you will learn how to use cell alerts, cell links, and cell alert links to bring attention to important or ancillary information.

The ability for users to add and view comments on specific data in a grid is another powerful way to highlight information. In this topic, you will also learn about how to enable users to add and display comments (or annotations) to data cells in multidimensional databases.

Using format masks to highlight data

Negative values in a grid show minus signs in front of the values by default. As an alternative, you can use the `defaultCellFormat` or other format mask properties to display negative values with parentheses around them or in red. Like Microsoft Excel, format masks in DB2 Alphablox applications can be used to display negative values in red.

Highlighting negative values in red

To highlight all negative values on a grid in red, set the `defaultCellFormat` using one of the format masks that will display negative values in red. All values are formatted according to the values in the `cellStyle`, which by default displays all values in black.

In the following example, the positive values will display all values with two decimal places and groupings will be separated with commas. All negative values (indicated by the format mask to the right of the semicolon) will show the same formatting as the positive values, except that these values will be red:

```
<blox:grid ...
  defaultCellFormat="#,###.00;[red]#,###.00"
</blox:grid>
```

If you only wanted negative values for specific members to be displayed in red, you would need to use the `cellFormat` property. In the following example, the negative values for the member Actual will be displayed in red:

```
<blox:grid ...>
  <blox:cellFormat
    format="#,###.00;[red]#,###.00"
    scope="{Scenario:Actual}"/>
</blox:grid>
```

For details on the use of `defaultCellFormat` and `cellFormat` properties, see the GridBlox section of the *Developer's Reference*.

Highlighting negative values with parentheses

Another alternative, which reflects a common practice in the financial community, is to display negative values within parentheses (but without minus signs). While this is a common number formatting practice, it also may help call attention to negative values in a grid. In the following example, negative values will be surrounded by parentheses:

```
<blox:grid ...
  defaultCellFormat="#,###.00;(#,###.00)"
</blox:grid>
```

If desired, you can combine these two highlighting methods. The following setting will result in negative values being displayed within parentheses and in red:

```
<blox:grid ...
  defaultCellFormat="#,###.00;[red](#,###.00)"
</blox:grid>
```

For details on the use of `defaultCellFormat` and `cellFormat` properties, see the GridBlox section of the *Developer's Reference*.

Using cell alerts to highlight data

Analyzing information is difficult enough to do without having to carefully scrutinize all of the numbers in a large grid to spot deviations or trends that warrant further attention. If an analyst misses an important deviation in even a single value, it could have costly consequences for a company. Since time is scarce, anything that can be done to speed up their work yet help keep them from missing important changes will be a productivity boost. Many analytic applications, including flash reports and executive scorecards, use exception reporting or traffic lighting to signal that attention needs to be given to some data. DB2 Alphablox supplies Blox properties and methods that can be used to highlight this critical information.

In DB2 Alphablox applications, the GridBlox `cellAlert` property can be used to highlight important information that users might be interested in being alerted to:

- significant deviations from expected values
- negative values, pointing out potential profitability issues
- ratios that are out of bounds from acceptable ranges

Two ways of using the `cellAlert` property to highlight information will discussed below: cell formatting and cell alert links.

Cell formats

Depending on the application you are working on, it might be a user requirement for you to perform "traffic lighting" on values based on some business logic. Traffic lighting is a commonly-used term used to describe a form of exception reporting in which different ranges of values are highlighted to users using something as simple as the red, yellow, and green light metaphors of the real traffic lights you encounter when driving your car around town. Traffic lighting is an easy-to-comprehend technique that applies knowledge from one domain of life (driving a car or walking across a street) to another domain (business intelligence). A typical application of traffic lighting in an analytic application is to highlight the backgrounds of data values according to some criteria that are commonly reflected in the three standard traffic light colors. The following table lists the three standard traffic light colors and describes their commonly used meanings:

Background Color
Description

Red Dangerous levels, values which should be of major concern to users.

Yellow
Values are not within acceptable, or desired, ranges and could merit attention.

Green Acceptable values that are "safe" or "good" ranges and do not necessarily need attention.

The following task explains how you can create a simple traffic lighting reporting system to alert your users to important changes in their data.

A simple traffic lighting reporting system

There are many possible variations possible in creating a traffic lighting notification system for reporting. Earlier in this topic, you learned about how you could use the GridBlox `defaultCellFormat` or `cellFormat` properties to highlight negative values in a grid. While this is a good solution for many situations, there will be instances when negative values are actually "good" values, so highlighting them in red using format masks may not work. The GridBlox `cellAlert` property allows you to customize alerting by

- background colors of cells
- data value styles, including font and color
- cell links that appear when criteria are met

For complete details about all attributes that can be used with the `cellAlert` property, see the GridBlox section of the *Developer's Reference*. Follow these steps to create a simple traffic lighting system for a member on a grid:

1. Pick the member on which you want to highlight ranges of values.

   In this example, the member on which traffic lighting will be done is Variance %. While four columns (Actual, Forecast, Variance, and Variance %) will appear in the grid, only the Variance % member will display background colors indicating levels of concern.
2. Add a `<blox:cellAlert>` tag to define values that should appear with red backgrounds.

```
<blox:cellAlert
   scope="{Scenario:Variance %}"
   condition="LT"
   value="0"
   background="red">
</blox:cellAlert>
```

3. Add a <blox:cellAlert> tag to define values that should appear with yellow backgrounds.

```
<blox:cellAlert
   scope="{Scenario:Variance %}"
   condition="between"
   value="0"
   value2="10"
   background="yellow">
</blox:cellAlert>
```

4. Add a <blox:cellAlert> tag for values that should appear with green backgrounds.

```
<blox:cellAlert
   scope="{Scenario:Variance %}"
   condition="GT"
   value="10"
   background="green">
</blox:cellAlert>
```

5. Run your report.

Example: To see a working example of this reporting system, see the Traffic Lighting example in the Highlighting Data section of Blox Sampler.

When you use traffic lighting and exception reporting, here are a few important points to keep in mind:

- Be careful that ranges cover all values. For example, if you have values greater than zero appearing in green and values less than zero appearing in yellow, then when the value is zero, no highlighting will occur.
- For extremes, unless there are fixed limits on the range of values, consider using GT or GTEQ on one end of your range and LT or LTEQ at the other end of the value ranges. This prevents having to perform extra maintenance later, if values exceeded predefined ranges.
- Color schemes used in row banding, generation styles, and cell styles can interfere with your cell alerts, causing an important alert to be missed. For example, if your default row banding displays alternate row backgrounds in yellow, and an alert is also using yellow as its background color, a user would most likely miss the alert.
- Color schemes used in alerts can affect the results of printed pages. Cells with red backgrounds, depending on the printer, may turn black or very dark on printing, obscuring the value in those cells.
- Online readability of cells with alerting background colors may be difficult because of the low contrast between the value color and the background color. Black values with red backgrounds can be particularly difficult to read. And, keep in mind that variations in colors and color contrasts can vary from monitor to monitor.

Cell alert links

In addition to changing the appearance of a cell as the result of a match with a cell alert's criteria, you can also define a link that will appear when certain criteria are

met. For example, when a particular value matches your conditions, then you may want to pop open a text window saying something about that value, and why the cell alert applies in this case.

Creating alert messages for cell alerts

In this task, the goal is to pop open a text window when a user clicks on the link icon for a cell. Here are the steps to follow:

1. Create the window that will be popped up for the alert.

   For example, your window might be a simple HTML window with a short message, no navigation elements, and a Close Window button. It might look like this:

   ```
   <html>
   <head>
   <title>Alert Message</title>
   </head>
   <body>
   Your alert message here
   </body>
   </html>
   ```

   In this example, assume that the file is saved as `alertMessage.html`.

2. Nest a GridBlox `cellAlert` tag within the GridBlox tag.

   ```
   <blox:grid id="myGridBlox"
      ...>
      <blox:cellAlert
         condition="LT"
         value="0"
         link="http://<serverName>/<appName>/notes/alertMessage.html"/>
   </blox:grid>
   ```

   Note: Links used within `cellAlert` tags should be either absolute URLs, showing the entire path to the page, or a relative URL (including an initial forward slash) from the application context. When the link is an absolute URL, it must include the server name in the URL. The following two examples are both valid:

   ```
   link="http://<serverName>/<appName>/notes/alertMessage.html"
   link="/notes/alertMessage.html"
   ```

   In the URL above, `serverName` must be the server in which your DB2 Alphablox application is running.

3. Test your application page.

Note: When used in conjunction with cell links, you need to be aware that cell alerts take precedence over cell links when images are included in both. If a cell alert applies to a data cell, and there is also a defined cell link with an image or image alignment defined, the cell link will not appear. However, if you have a cell alert that does not include a link or an image, then both will be applied.

Information links

In analytic applications, there may be times when you may need to include links to more information about the data within your grids. Links can serve many different uses, including:

- links to more information about a heading
- links to information about a particular cell

- links to alert information on a cell, based on business logic

DB2 Alphablox offers three types of information links: header links, cell links, and cell alert links. The following lists the advantages and disadvantages of each of these link types.

Link Type
Summary of Uses

Header Links
- Links appear to the right of dimensional members when they appear in row or column headers
- Defined in the application definition page of the DB2 Alphablox Admin pages
- Always visible when member has an associated link
- Only one icon image available for all header links

Cell Links
- Links can be defined using scope.
- Defined using GridBlox `cellLink` property
- Different images can be defined based on `cellLink`
- Can result in opening of information window or can trigger execution of JavaScript function

Cell Alert Links
- Defined as part of the `cellAlert` property
- Can be used in conjunction with cell links, but if images appear in both, cell alert links take precedence
- Can be used to appear based on conditional logic or scoping
- Can result in opening of information window or can trigger execution of JavaScript functions

See details about the use of each of these information link types below.

Using header links

Header links are application-specific information links you can define to display web pages or trigger JavaScript functions when a user clicks on an information icon (represented as a blue circle with a white "i" within it) appearing next to a row or column member in a grid. Header links only appear in the headers for members which have been defined in the Header Links text box of the application definition page.

To add a header link for a specific application, open the application definition page in the DB2 Alphablox home page. Near the bottom of the page is the Header Links text box, where you can define header links using the following syntax:

```
memberName = URL
```

where the `memberName` is the unique member name defined in your data source and *URL* is either an absolute URL, showing the entire path to the page, or a relative URL (including an initial forward slash) from the application context. When the link is an absolute URL, it must include the server name in the URL. For example, to create an information link to a product page for Diet Cola, the following header link definition might be used:

```
Diet Cola = http://productServer/products/dietcola.html
```

or

```
Diet Cola = /<pathTo>/dietcola.html
```

Note: JavaScript protocol methods are not supported.

Using cell links

Just as header links can be used to place links in the row and column headers, the GridBlox `cellLink` property can be used to define hyperlinks on data cells. Unlike header links, cell links can also be used to invoke a JavaScript method using the JavaScript protocol method. Cell links, like other indexed properties, are evaluated according to their index values, which are either dynamically generated at runtime or defined by the developer.

The number of the cell link dictates the order in which it is evaluated, starting with the `cellLink` with an index value of 1. 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. Also, cell alert links take precedence over links created using `cellLink`. That is, if there is a cell alert link and a cell link defined for a particular data cell, the cell alert link will appear in the cell, but the cell link will not.

It is possible to have both cell alerts and cell links on the same data cell, but if both have image elements (`image`, `image_align`, or `link`) defined, the cell alert link will take precedence over the cell link— only one icon and link can be available on a cell and cell alerts (with a link) are assumed to be more important than a cell link.

Here is an example of a cell link property tag, which is nested within a GridBlox tag:

```
<blox:grid id="cellLinkGridBlox">
   <blox:cellLink
   scope="{Scenario:Variance %}"
   description="Opens information window"
   link="/<applicationDirectory>/links/Manhattan.html"/>
   <blox:data dataSourceName="QCC-Essbase"
      query='<ROW("All Products") "All Products"
         <COLUMN(Scenario) <CHILD Scenario
         <PAGE("All Locations") Manhattan Sales !'/>
</blox:grid>
```

The cell link defined above would appear only in the Manhattan data cell under Variance %. The page that would be opened is located in the links subdirectory in the application folder.

The specified URL can either be an absolute URL, showing the entire path to the page, or a relative URL (including an initial forward slash) from the application context. When the link is an absolute URL, it must include the server name in the URL. In the example above, `<serverName>` represents the server, and `<applicationDirectory>` represents the name of the application directory where the file is located.

See the *Developer's Reference* for details about the syntax and usage of the `cellLink` property and its associated methods `getCellLink()` and `setCellLink()`.

Using cell alert links

As described in "Cell alert links" on page 174, a cellAlert can also display a link. The combination of both links created with the `cellAlert` property and links created with the `cellLink` property provide you with different alternatives on how to highlight information your users may want to know.

Another option is to extend the user interface to get multiple images on a cell by using the UI Model's extensibility.

For details about both `cellAlert` links and links created with `cellLink`, see the GridBlox Reference section of the *Developer's Reference*.

Comments in grid data cells

The sharing of information within an organization frequently involves commentary on data, but often these comments get lost in e-mail messages or elsewhere. Incorporated into DB2 Alphablox is the ability to add these important comments to a commentary database and view them in the context of user analysis. Using this feature, users can view comments associated with particular data cells by retrieving comments on those cells, or by viewing comments in separate listings on an application page.

There are two types of comments supported using CommentsBlox components, cell-level comments and named comments. Cell level comments are comments attached to a specific data cells and displayed in a grid. They can be defined over a set of dimensions. Named comments are comments have string addresses that can be used to define the scope of the comments.

If comments have been added for a data cell in a grid and the grid has been comments-enabled, a comments indicator will appear. By default, the comments indicator is a small red triangle appearing in the upper right corner of data cells that have comments associated with them. When a user right-clicks on cells with comments indicators, a Comments option appears in the context (right-click) menu. Two submenu options are available, Add Comment for allowing users to add new comments to the selected cell and Display Comments for allowing users to view available comments on the selected cell.

Note: To use the Comments Management Dialog, you need to have rights for creating and dropping relational tables. For using the CommentsBlox API in developing custom commentary applications, you may need rights for selecting, inserting, updating, deleting, creating, and dropping tables.

Key Terms

Description

Comments Collection
A repository for a group of comments for a single multidimensional cube. Stored in a relational database.

CommentsBlox
Represents the comments collection on a page. Includes a set of tags which are nested within a DataBlox.

CommentsSet
A group of comments that exist for a single data cell or with the same address or name. Includes all comments that have one scope or address (e.g., comments for Product:100, Year:Qtr1, Scenario:Actual).

Elements of a comment

An individual comment has the following parts:

Comment Element
Description

Author
Required. The author of the comment. By default, this field is set by DB2 Alphablox at the time of comment creation to the currently logged in user.

Timestamp
Required. The time the comment was created. Automatically set by the server when the comment is first saved to the comment set.

Comment Text
Required. The text of the comment, which can include hyperlinks and could even formatted text (using HTML). No limit on text size.

Custom fields
To provide maximum flexibility, you can define additional custom fields for the comments in a comment set. Examples: subject, importance, cell value.

Address
Each comment has an address. For cell level comments, the address will be a list of <dimension, member> that uniquely identify the cell to which the comment is attached. For named comments, the address is simply a string whose meaning is defined by the developer. For example, a set of Blox-level comments might have an address that consists of the name of the Blox to which the comments are associated. For an application-level comment, the address might be the name of the application. Assemblers can use this string to define a namespace for comments as well as to assist in personalization capabilities.

Cell level comments may have an addressing scheme that incorporates a subset of the dimensions in a particular cube. Dimensions not included in the collection's definition are ignored. As an example, if your cube contains three dimensions, Time, Measures and Product, and you define that comments in your current collection are specified using values of Time and Measures, then any comments that are defined apply for any value of Product. Generally, you should include all dimensions in the comments definition that might be manipulated as part of the report.

This addressing scheme for cell level comments serves two purposes. First, it makes administration of comments easier, especially in larger outlines. Second, it makes it easier to share comments across cubes and data sources. A comment set defined over Product, Time and Measures may be applicable over a number of data sources, while a comment set defined over an every dimension in an outline runs the risk of becoming cube and data source specific.

Defining comments collections

To define a comments collection, you need to create a relational data source. This data source can hold multiple collections. Two steps are required to create a comments collection:

1. Create a data source and defined it on the DB2 Alphablox data source definition page.
2. Create the comments collection repository.

To define a comments collection, open the DB2 Alphablox Admin Pages, then click on Administration tab. In the menu on the left, under Runtime Management, click on Comments Management to open the Comments Management window.

A collection requires a collection name, selected dimensions from a cube and the creation of fields to be used. The author, timestamp, and comment text fields are automatically defined, but custom fields can also be created.

Help on configuring a comments collection is also available in the Comments Management window.

Enabling cell comments

To enable comments on data cells in a grid, you need to follow these steps:

1. In a standalone or nested GridBlox, add the `commentsEnabled` attribute and set it to `true`.
2. In the standalone or nested DataBlox for the grid above, add the CommentsBlox tag, specifying the `collectionName` and `dataSourceName` attributes for your comments collection. The data source and the collection names are defined using Comments Management under Administration tab of the DB2 Alphablox Admin Pages.

Here is an example of what a PresentBlox enabled to support comments would look like:

```
<blox:present id="myPresentBlox">
   <blox:grid commentsEnabled="true" />
   <blox:data dataSourceName="QCC-Essbase" query="<%=query%>">
       <blox:comments
         collectionName="sales_comments"
         dataSourceName="CommentsCollection" >
      </blox:comments>
   </blox:data>
</blox:present>
```

Once this has been done, users can right-click on data cells and add or view comments. No other steps are required by developers for basic comments support.

Adding custom comments support

The ability for users to add their own comments and view the comments of others is a powerful collaboration and information sharing mechanism. Out-of-the-box, enabling comments is easy to configure and use. But, the power and flexibility of the CommentsBlox capabilities allow developers to customize the use of commenting. Below are a couple of examples of potential customizations that can further enhance your applications using CommentsBlox.

Note: For details about the syntax and usage of CommentsBlox, see the CommentsBlox Reference section of the *Developer's Reference*.

Sometimes, users may prefer to be able to add comments about a particular topic or a particular analytic view without having to associate those comments with a particular data cell. This can easily be accomplished using the CommentsBlox tags and server-side Java API. Under the Commenting on Data section of the Blox Sampler application, the General Comments on a Page example shows an example of allowing general comments, appearing below a grid, to be added and viewed in a separate comments window.

Users may also want to be able to print out all of the comments associated with a particular grid. The Printing Comments in a Grid example, included in the Commenting on Data section of Blox Sampler , includes a button which will open a new browser window and display all of the comments in the grid.

Chapter 17. Interacting with data

This topic focuses on user behavior and interactivity with Blox. Primary issues discussed include how Blox behavior can be controlled and modified, basic techniques used to limit behavior, and how you can capture user actions and control interactivity and actions.

Interactivity considerations

The interactive, visual presentation Blox enable users to manipulate the views presented to them, drilling down or up in the data, changing chart types, and many other options. Depending on your applications, your audience, and their skill levels, you may decide you want to exclude or limit the control of the applications. The following subsections discuss issues to consider when limiting interaction with Blox.

Allowing limited or no interactivity

If your users only require a simple view of important data, and are not interested in manipulating the data for deeper analysis, a GridBlox, ChartBlox, or PresentBlox will be fine. You may even consider displaying a slice of data in a Blox view that offers no interactivity.

To prevent interactivity with a Blox, for example, you can add the Blox UI component tag (`<bloxui:component>`), setting the `clickable` tag attribute to `false`. For example, the following code shows the use of a nested `<bloxui:component>` tag to generate a PresentBlox with user interaction disabled:

```
<blox:present id="myPresentBlox"
   width="80%"
   height="70%"
   menubarVisible="false">
   <blox:toolbar
      visible="false" />
   <blox:data
      bloxRef="dataBlox" />
   <bloxui:component name="myPresentBlox"
      clickable="false" />
</blox:present>
```

Disabling interactivity may be the best solution for users who are either “too busy” to drill into data or are not interested in learning how to manipulate data. Upper management executives in your company, for example, may only be interested in seeing snapshot views of how the company is doing, leaving detailed analysis to business or financial analysts.

The following task shows how you can either disable an entire Blox or selected Blox nested within another Blox.

Disabling pivoting and drilling on columns

In the Blox UI Tags section of Blox Sampler, the butterfly report example includes an event filter to prevent users from pivoting or drilling on columns. Both of these user operations would result in the displayed asymmetric report no longer displaying properly. To prevent users from getting themselves into a situation that

is confusing and difficult to get back out of, you can add an event filter, using the UI Model, that traps a user's attempts to pivot or drill keeps the view usable.

The following code snippet shows an event handler used on the grid to prevent pivoting and drilling on columns:

```
<%
  GridBloxModel model =
    butterflyReportGridBlox.getGridBloxModel();
  model.populateDataNavigationButton();
  model.getGrid().getController().addEventHandler(
     new IEventHandler() {
  public boolean handleGridDataActionEvent(GridDataActionEvent
       event ) throws Exception {
   GridBrixCellModel cells[] = event.getGridCells();
// If any of the cells is a header cell, then ignore the data action

  for ( int i=0; i < cells.length; i++ )
  if ( cells[i].isColumnHeader() )
  return true;
  return false;
}
} );
%>
```

For the entire code example, see Blox Sampler.

For details about using event handlers with the DHTML extensibility capabilities of the Blox UI Model to customize application like this example, see "Blox UI Model events" on page 73.

Modifying interactivity using Blox properties

User interaction can also be controlled using Blox properties and methods. When the Toolbar, DataLayout, and Page panels are enabled on data presentation Blox, users can interact more with the data. You may find that while this helps some users, others will quickly become lost in the data, especially if they are not familiar with the structure of the data. Besides the techniques described above using the `<bloxui:component>` tag and Blox `visible` attributes, you can also use other Blox properties to tune the interactivity of your views, enabling some panels and not others, limiting the number of ways a user can get into a confusing situation. And, using personalization techniques, you can use server-side Java, JavaServer Pages, and JavaScript methods to customize interactivity based on the user login.

The following table lists some of the commonly used Blox properties that can affect user interactions with the data:

Blox	Property (or Associated Methods)	Description and Comments
GridBlox	`cellAlert`	Cell alert links can be used to open an information window or invoke a JavaScript function
	`cellLink`	Cell links can be used to open an information window or invoke a JavaScript function
	`expandCollapseMode`	Enables Windows Explorer-like use of plus and minus icons to drill up and down in grid
	`writebackEnabled`	Allows authorized users to enter data directly into grid based on scoped cells
DataBlox	`drillDownOption`	Determines whether drilling goes to next generation, all descendants, bottom generation, siblings, or same generation
	`drillKeepSelectedMember`	Keeps the selected member that is being drilled on
	`drillRemoveUnselectedMembers`	Removes members not selected when drilling
	`enableKeepRemove`	Specifies whether the Keep Only and Remove Only options are available
	`enableShowHide`	Specifies whether the Show Only, Show All, and Hide Only options are available
DataLayoutBlox	interfaceType	Specifies how users select dimensions, either using a drop lists or a drag-and-drop tree interface.
ReportBlox	See *Relational Reporting Developer's Guide*	

Note: Changes in interaction behavior resulting from property changes can result in users becoming confused or surprised when normally familiar behaviors don't act as expected. For example, setting the DataBlox `drillKeepSelectedMember` and `drillRemoveUnselectedMembers` to `true` can be useful at times, helping users effectively manage the amount of information visible on a grid or chart. An important consideration is that if all views in an application or across multiple applications do not behave the same way when the user drills, he may become confused when a particular Blox view behaves differently than all of the others encountered. One way to help users in situations like this is to clearly note on the page what the user should expect. Alternatively, radio buttons or check boxes can be used to allow users to toggle between the two drilling behaviors.

Grids

Grids are available as either a standalone GridBlox or nested within a PresentBlox. In either mode, users can drill, pivot, sort, and explore their data. A grid used in a PresentBlox, though, includes some additional functionality not available in the explicit GridBlox. The following table shows a summary of the key differences between standalone GridBlox and a nested GridBlox:

Functionality	Standalone GridBlox	Nested GridBlox (within PresentBlox)
DataLayoutBlox	Requires standalone DataLayoutBlox, using same DataBlox	Made available using PresentBlox `dataLayoutAvailable`
PageBlox	Requires standalone PageBlox, using same DataBlox	Available by setting `pageAvailable` to `true`
ChartBlox	Requires standalone ChartBlox, using same DataBlox	Grid automatically synchronizes with chart. Chart can be made available.

Since the functionality listed above is available by using a PresentBlox, the majority of the time you want to give access to users to this functionality.

Charts

Like grids, users can drill up or down in the data being displayed in charts. But, unlike grids, users may not realize they can interact with charts -- no visual cues, such as the grid's up/down arrows or plus/minus icons, exist to help users understand that they can directly interact with charts. The first time a user might realize they can drill on charts is when they see someone else doing it, or just happen to try it, or right-click on chart elements and see menu options. Most users discover that data values and labels will appear when they hover over chart bars and data points. If necessary, you can use ChartBlox properties, such as `pieFeelerTextDisplay` for pie charts and `dataTextDisplay` for bar charts, to display values or labels without requiring the users to move their cursors over a chart element.

Whether a user is accessing an application over the Internet from a remote location or using it while sitting in an office nearby you, they may not have received training or know much about your applications and how to use them. As a developer or application designer, you need to consider how to make analytic applications as easy to use as possible. If you present them with pages of charts

and no directions, you shouldn't be surprised to discover that many users will never interact with your charts, instead just viewing what you present to them. As you design, consider how you can increase the likelihood that your users will be successful and learn to use your charts more productively. Adding Help or Tips buttons to access DB2 Alphablox help or custom help pages will help them learn what they can do with applications. Alternatively, you could place some short hints directly on the pages. In some situations, you may find it useful to place user information directly in footnotes on a chart, using the ChartBlox `footnote` property.

Allowing user control of generations displayed

A simple way to add interactivity to a chart is to add the `totalsFilter` attribute to a ChartBlox, setting the property to 2. By setting the value to 2, you enable a totals filter slider panel to appear at the bottom of a chart. Depending on the query used, users may then be able to control which dimension levels are displayed. In the following ChartBlox example, the `totalsFilter` attribute is set to 2:

```
<blox:chart id="totalsFilterChartBlox"
   width="90%"
   height="90%"
   chartType="Bar"
   totalsFilter="2"
   title="Truffle Sales for 2001">
   <blox:data
      dataSourceName="QCC-Essbase"
      query='<ROW ("All Products")  <ICHILD "All Products"
         <COLUMN ("All Time Periods") <DESCENDANTS "2001" Sales !'/>
</blox:chart>
```

When rendered on a page, two generation selectors will appear on the panel below the chart. The selector on the left allows users to control the generation level of the All Products dimension, and the selector on the right allows them to select the generation level of several time periods in the year 2001.

Example: The "Chart totalsFilter Selector Enabled" example under the Interacting With Data section in Blox Sampler shows the use of the `totalsFilter` slider panel.

DataLayout interface

When a DataLayout panel (DataLayoutBlox) is available on a PresentBlox, users can access it to move dimensions between the Row, Column, and Other (page filter) axes in order to create the layout of data within grids and charts that they are interested in seeing. By default, the DataLayout panel displays the dimensions in drop lists (or selection lists) that allows users to click on a dimension name and select an option for the movement of the dimension. Alternatively, developers can set the DataLayout panel to use a drag-and-drop tree interface, more similar to Windows Explorer in behavior. To explicitly set the interface type for the DataLayout panel, set the DataLayoutBlox `interfaceType` attribute to one of two values, `dropLists` (default) or `tree`. The following example shows a DataLayoutBlox set to display a tree interface:

```
<blox:present ...>
   ...
   <blox:dataLayout
      interfaceType="tree" />
   ...
</blox:present>
```

Note: The interface type can only be set by the developer -- there is no user interface option for users to select this interface option. By default, users will see the drop list interface.

Interactions between grids and charts

Grids and charts can appear individually using GridBlox and ChartBlox components, or nested together within a PresentBlox component. When occurring as standalone Blox, each Blox can use implicit data sources (defined in the nested DataBlox) or explicit independent data sources (using standalone DataBlox components). Grid and chart views can also share a common standalone DataBlox as their data source. This is always the case with the GridBlox and ChartBlox nested within a PresentBlox.

When GridBlox and ChartBlox components share a common data source (using a standalone DataBlox), operations on a GridBlox are reflected in the ChartBlox. Thus, when a user drills down on a member in a GridBlox, a ChartBlox sharing the same DataBlox, will also perform and display the same drill operation. This synchronization between grids and charts occurs within a PresentBlox since they share the same data source.

Header links, cell alerts, cell links, and other GridBlox features are not available in charts. In order to have both charts and these features, you will probably want to use a PresentBlox. Also, if a grid is not visible in the PresentBlox view, users will not see alerts or be able to access grid-based links unless they access them through the grid component.

Another important point to realize is that the formatting of data is set independently in grids and charts. Thus, if you want both grids and charts to use the same formatting of values, you'll need to remember to set all of the following Blox properties:

Blox **Property**

ChartBlox
`y1FormatMask y2FormatMask`

GridBlox
`defaultCellFormat cellFormat`

Setting the "No data available" message in grids and charts

When a data source is not available or a result set has not yet been retrieved, DB2 Alphablox grids and charts will display the following default message: "No data available."

If the retrieval of a result set takes longer than a user might expect, the default "No data available" message can be deceiving. While it is true that no data is available at that moment, if the user waits a while longer, the data will usually appear. If the retrieval takes more than a few seconds, users may think that the application is not working properly and try reloading the application or the page, without waiting long enough for the data to appear. When this could be an issue, many DB2 Alphablox developers set the `noDataMessage` to a message like one of the following: `Please wait for data...` or `Waiting for data...`.

This is usually a good solution, except when there really is no data available. In these cases, the message does not change to indicate that no data is available.

Consider the implications of your noDataMessage before deciding to modify it, but typically the benefit of a clearer message outweighs the likelihood that the data source will actually not be available.

To modify the message that appears in grids and charts, add the noDataMessage attribute to a PresentBlox, GridBlox, or ChartBlox. If the noDataMessage attribute is added to a PresentBlox, the new message will appear in both the nested GridBlox and ChartBlox displays. If you would prefer to set the values separately for the nested GridBlox and ChartBlox, you can set the noDataMessage attributes on each nested Blox separately. The following settings on the nested GridBlox and ChartBlox in a PresentBlox will result in the different messages appearing in a grid and a chart:

```
<blox:present ...>
  <blox:grid
     noDataMessage="Grid not available"/>
  <blox:chart
     noDataMessage="Chart not available"/>
</blox:present>
```

If used cautiously, changing the noDataMessage attribute can result in a better application. For more information on the noDataMessage, see the Common Blox section of the *Developer's Reference*.

HTML form elements and FormBlox components

Grid and chart views created using PresentBlox, GridBlox, and ChartBlox can include built-in features such as toolbars, page filters, and contextual (right-click) menus. If a toolbar is available, the user has options menus available for modifying charts, grids, and data appearance and behavior. Sometimes, though, having many options available can be overwhelming for novices and casual users. Depending on your specific needs, the best option may be to offer a limited number of choices instead. Using a combination of HTML form elements, JavaScript, Java, and the a rich Blox API, you can create custom analytic applications with options tuned to the needs and skills of your users, or to customize interaction. And, in Blox Sampler, you will find some examples using HTML form elements (buttons, checkboxes, etc.) to offer controlled interactivity or options for changing data views.

Most often, though, a more compelling option may be to use the FormBlox components, available when using the Blox Form Tag Library, to manage HTML form elements. Details about the FormBlox components and how to use them are discussed more thoroughly in "Using the Blox Form Tag Library" on page 43 of this guide and in the Blox Sampler examples.

Also, Blox Sampler includes many examples of interactivity with Blox controlled using HTML elements and FormBlox components. In particular, take a look at the examples in the Using FormBlox and Logic Blox section or the Interacting with Data section.

In the following subsections, some of the standard HTML form elements and their FormBlox equivalents will be highlighted.

Selection lists

Using Blox API properties and methods, PageBlox page filters can be customized with fixed choice lists (using fixedChoiceLists), virtual roots (using

dimensionRoot), and the Member Filter (using moreChoicesEnabled and moreChoicesEnabledDefault). Sometimes, though, a PageBlox page filter can't solve all of your requirements.

If you turn off the toolbar on a grid and chart views, you can create drop-down menu items to replace ones no longer available. Hardcoded selection lists let you offer controlled options for end users while making your analytic views easier to use. For example, the chart types list available with the Charts button can be overkill for a particular view, so you could offer a limited subset of chart types in a selection list. This way users can have some choices in how data is displayed, while not being offered choices that may not make sense on a particular view.

Cascading selection menus are useful for letting end users select an option from one list, then offering secondary menu choices based on their selections on other lists. Creating your own cascading menus using HTML form elements and JavaScript can be a major undertaking. But, using the MemberSelectFormBlox available in the Blox Form Tag Library, and with much less coding, you can quickly create a cascading menu and tie it to a data view. The FormBlox components used will also handle persistence. That is, during a user's session, the dynamically-generated HTML form elements will maintain their selections when users leave the page and return later. To see this in action, take a look in the Using FormBlox and Logic Blox section of Blox Sampler, where you can find a MemberSelectFormBlox example that has three selection lists that change dynamically based on user selections.

Page filters within a Blox will typically display only members of a dimension in a prescribed order. But, by moving page filter selections to HTML form elements, either custom-coded or using the MemberSelectFormBlox, you can create dynamically-generated page filters that are based on customized queries against your data sources.

Check boxes and radio buttons

When you do not include a toolbar on a grid or a chart view, you can use check boxes and radio buttons, either custom-coded or created using FormBlox components (CheckBoxFormBlox and RadioButtonFormBlox) to give users choices that are not accessible when the menu bar or toolbar is not available. Since users frequently don't make their way to various dialog boxes available in the toolbar, they may not even know that some options are available. Another advantage of not using the toolbar and menu bar on a view is that you can offer a limited set of clear options more visibly on the page. For example, you might include Suppress Missing and Suppress Zeros check boxes that toggle states depending on whether the check box is checked or not. Radio buttons are great for offering options that are not compatible with each other. Blox Sampler has examples throughout using FormBlox components to manage check boxes and radio buttons.

Standard HTML buttons

Standard HTML buttons are useful in applications for executing queries, resetting queries, and generating views. But these buttons are less desirable for showing different views, since unless the title changes on the page, a user may not be able to tell which button they used to get the current view. Using radio buttons can be a good alternative, since the particular item selected always has the radio button highlighted.

Text fields

Text entry fields are not used as often as other HTML form elements in analytic application, but they can be useful in special situations. Most of the time, fixed choice options (such as those available with radio buttons, check boxes, and selection lists) are preferable, since they help prevent misspellings and issues you can encounter with expected values (e.g. users entering characters instead of numbers). In some situations, your best or only practical option is to allow users to enter values on their own. For example, in analytic applications text fields can be used to allow users to input data for writeback to a data source or to personalize applications by setting their own threshold levels on cell alerts. Text fields and text areas can be used in allowing users to add comments in an application. In DB2 Alphablox applications, you have the option of either using custom-coded HTML text fields or to use the EditFormBlox component to create text fields for use in your analytic applications. The EditFormBlox has built-in capabilities of automatically changing properties in other Blox components.

Using Toolbar buttons

Each Blox component which users interact with can include a toolbar for accessing Blox functionality. DB2 Alphablox provides several ways to create tailored Blox toolbars, which may enhance the user experience.

By default, each Blox displays its toolbar. To make a toolbar invisible on a specific Blox, set the Blox's `visible` property to `false`.

Used in combination with the `clickable` property of the Blox set to false (see "Allowing limited or no interactivity" on page 183), the result is a static data presentation, rather than an interactive UI. This may be appropriate for quick snapshots and executive reports. Making Blox toolbars invisible may also be appropriate where an application uses a custom HTML user interface to replace toolbar functionality.

By default, a text-based menu bar appears above the Blox toolbar. To make it appear, set the Blox `menubarVisible` property to `true`.

Each button on the toolbar does not display a descriptive text label by default. To turn on this text (thus increasing the display space required for some buttons), set the `textVisible` property to `true` Note that turning on toolbar text results in a bigger toolbar, which will take more of the Blox area.

You can specify the buttons to be removed from the toolbar using the `removeButtons` property of the nested ToolbarBlox. For details on these and other ToolbarBlox functionality, see the ToolbarBlox section of the *Developer's Reference*.

Events

DB2 Alphablox provides properties and methods for handling events. An *event* is a normal action that you can use to trigger further processing.

Blox can capture the following user actions and treat them as events:

- drill down or up
- pivot
- select header or cell menu item
- change the page filter

- load or saving a bookmark
- change the data value in a grid cell
- keep or remove only
- hide or show only

Descriptions on Blox UI Model events are described beginning in “Blox UI Model events” on page 73. Also, the UI Model exposes a number of events that can be issued by the client, such as a `ClickEvent`. For each of these events, the DHTML Client API defines JavaScript objects. As a result, JavaScript can be used to create event objects and sent the event to the server. For more information, see “Sending events” on page 95, “Intercepting events” on page 95, and “Invoking JavaScript directly from the user interface” on page 96.

Chapter 18. Inputting and modifying data

This topic explains how to input, or writeback, data to data sources using DB2 Alphablox. Also discussed is the use of calculated members to create new data derived from data retrieved from your data source.

Writeback to multidimensional data sources

DB2 Alphablox Java methods can be used by developers to modify a result set and update its underlying data source. In addition, users can review, edit or input new values into a grid's data cells and update the underlying database.

To give users the ability to update values in data cells and then write those values back to the underlying data source, developers need to include a set of properties and associated methods that:

- enable the grid to be edited
- define the cells available for editing
- specify the style (usually a foreground or background color) for displaying an editable cell
- specify the style for displaying an edited cell
- optionally, specify the processing to occur when a user edits a cell

GridBlox properties and associated writeback methods

The following GridBlox properties and associated Java methods are required or available for designing and managing writeback applications:

Property And Methods	Description
`writebackEnabled isWritebackEnabled() setWritebackEnabled()`	Required to enable writeback; permits users to edit cells in the grid
`cellEditor getCellEditor() setCellEditor()`	Required to enable writeback; specifies a rule for defining and highlighting an editable area of data cells
`CSS Themes`	Can be used to specify the appearance of cells the user has edited or can edit

GridBlox Java writeback methods

The following table lists all GridBlox Java methods that do not have associated properties:

Methods
: **Descriptions**

`getWritebackValue()`; `setWritebackValue()`
: Sets or returns the value of a specific data cell changed in the grid

`listCellEditorIds()`
: Returns a list of IDs of all the cell editors defined as an array of integers

getChangedCellList()
Returns a String of edited cells

getChangedCellValues()
Returns a String of edited cell values

For complete information on each of the server-side GridBlox Java methods available for writeback, see the GridBlox sections of the *Developer's Reference*.

Enabling GridBlox components with writeback

The following example includes the minimum properties required to enable writeback on a GridBlox component and other commonly used properties.

```
<blox:grid id="Grid1"
  width="800"
  height="500"
  writebackEnabled="true">
  <blox:data
     bloxRef="Data1"/>
  <blox:cellEditor
     scope="{Market:New_York}"/>
</blox:grid>
```

For complete information on each of these properties, see the GridBlox section of the *Developer's Reference*.

The GridBlox properties above are for enabling writeback, defining editable cells, and changing the appearance of writable data cells. To writeback to a data source, DataBlox writeback methods must be used.

DataBlox methods for writeback

After you have configured a GridBlox for writeback, you need to use server-side DataBlox methods to perform the writeback operations. The write back methods are designed for applications that write back data to a DB2 OLAP Server, Essbase, or Microsoft Analysis Services 2000 data source. Some of the methods are specific to DB2 OLAP Server or Essbase only. The following table lists all available DataBlox Java writeback methods:

Method
Description

writeback()
A convenience writeback method that takes 3 arguments, and uses these methods:
- lockCurrentDataSet()
- setDataValues()
- commitData()
- unlockAll()
- executeCustomCalc()
- refresh()

lockCurrentDataSet()
Locks the called-upon result set; does not lock the entire database

setDataValues()
Changes data values in the result set at the coordinates specified

commitData()
Writes the current data set back to the database

unlockAll()
Unlocks any data that was previously locked in a DB2 OLAP Server or Essbase database

executeCustomCalc()
Executes a calculation script on a DB2 OLAP Server or Essbase database

refresh()
Refreshes the current data set

executeNamedDBCalcScript()
Executes the named DB2 OLAP Server or Essbase calc script

For complete details about these writeback methods, see the DataBlox Reference in the *Developer's Reference*.

Enabling writeback to multidimensional databases

Using the server-side Java APIs, you can create application pages that allow users to writeback to multidimensional databases. Included in Blox Sampler are three examples, one using a custom Java class (recommended approach) and two using Blox UI controllers (generic and custom).

Note: DB2 OLAP Server and Hyperion Essbase queries do not support the use of attribute dimensions in writeback operations.

The following steps go through code for using the custom Writeback class available in Blox Sampler:

1. Add a JSP `page` directive importing the required classes:
```
<%@ page import="bloxsampler.writeback.Writeback,
                  com.alphablox.internal.PresentBlox" %>
```
2. Add a JSP `taglib` directive for the Blox tag libraries that to be used on the page:
```
<%@ taglib uri="bloxtld" prefix="blox" %>
<%@ taglib uri="bloxuitld" prefix="bloxui" %>
```
3. In the head section of the page, don't forget to add the required `<blox:header/>` tag which automatically adds required CSS and JavaScript methods:
```
<head>
   <blox:header />
</head>
```
4. Add a JavaScript function that uses the Blox UI Model to perform a simulated click event on the server, as if the user had clicked the Writeback menu option under Data in the PresentBlox menu bar:
```
<script language="JavaScript">
  function wb() {
    bloxAPI.sendEvent(new ClickEvent('wb3PresentBlox',
      <%= wb3PresentBlox.getBloxModel().searchForComponent(
      "dataWriteback").getUID() %>));
   }
</script>
```
5. Add a PresentBlox with the grid `writebackEnabled` attribute set to `true` and include a scriptlet that references the custom Writeback class (with two arguments, the Blox name and the scope:
```
<blox:present id="wb3PresentBlox"
   height="400"
   width="600"
```

```
   pageAvailable="true"
   chartAvailable="false"
   dataLayoutAvailable="false">

   <blox:data
      dataSourceName="QCC-Essbase"
      query='<SYM <ROW("All Products") <CHILD Truffles
        <COLUMN(Scenario) "Initial Budget" Manhattan
        <ICHILD "Jan 01" "Units Sold" !' />
<%
  new Writeback(wb3PresentBlox,"{Scenario:Initial Budget}");
%>
</blox:present>
```

6. Add a button on the page for users to click in order to invoke writeback to the data source.

```
<form>
   <input type="button" value="Submit Changes"
      onclick='setTimeout( "wb();", 1 );'>
</form>
```

   The JavaScript `wb` function is called after a brief timeout in order to allow the data to be updated on the server before this function is called. If the JavaScript `setTimeout` function is not used, the correct data may not be written back to the data source.

In this example, writeback has now been incorporated into the PresentBlox. A working example of this writeback technique is included in Blox Sampler in the section on Inputting and Changing Data. The source code for the Java class is available in the application's `WEB-INF/src/` directory. You can modify this source code to make any further changes you may want, then compile it for use in your applications.

Updating relational data sources

DB2 Alphablox supports standard SQL statements for updating relational data sources. These statements include, but are not limited to `INSERT`, `UPDATE`, `CREATE`, and `DELETE`. You can use Java methods to construct the appropriate SQL statement, then pass the statement to the application's DataBlox.

Note: Writing data back to a relational data source does not affect the user's view of the data, but before the user can see the effect of the changed data, the query must be re-executed.

Updating relational data sources Using writeback

The following steps illustrate how to update a relational data source using Java methods. The example inserts a new column containing current date into a table.

1. Create SQL query string named `query1`. It will insert the current date into a table named "`review_data`."

```
String query1 = "insert into review_data values(TO_CHAR(SYSDATE,
   'HH:MM:SS-MMDD'))";
```

2. Call the appropriate `setQuery` and `connect` methods on the data source for a PresentBlox named `Present1`. Pass the SQL query that inserts the new column.

```
Present1.getDataBlox().setQuery(query1);
Present1.getDataBlox().connect();
```

Enabling writeback to Microsoft Analysis Services

For Microsoft Analysis Services, you can writeback to the leaf-level members only using the techniques described above. To update data in non-leaf members requires the use of the MDX `UPDATE CUBE` command in a DataBlox `setQuery` or `executeQuery` method. For more information on the `UPDATE CUBE` command, see the Microsoft Analysis Services documentation.

Creating a calendar control

The Blox UI model provides a DateChooser component that lets you create a graphical calendar control which allows users to select dates for use in generating analytic views or other application uses. This component adds a text field on the Web page with a small calendar icon next to it. When users click the icon, a small calender pops up, allowing users to select a date to populate the text field with an appropriately-formatted date.

The DateChooser component includes the following subcomponents:

Subcomponent	Description
A graphical user interface of a text field	The DateChooser component extends the Edit component and supports most of the Edit API, such as `insertText()`, `setValue()`, `getValue()`, and `clear()`.
An image for launching the calendar	The DateChooser component includes an Image subcomponent that is displayed next to the text field for launching the calendar. The `getIcon()` method provides access to this Image component. For example, `getIcon().setImageURL("myCalendar.gif")` sets the image file to use for the icon. By default the image is theme-based. The specified image should be placed in the theme's image directory in the DB2 Alphablox repository. To put the image in a different location, call the Image's `setThemeBasedImage(boolean)` method and set it to `false`. See the Blox API Javadoc documentation for details.
A calendar adapter	This CalendarAdapter object is a wrapper that implements the com.alphablox.blox.uimodel.ICalendar interface. This interface conforms to a subset of the java.util.Calendar API, such as `getTimeInMillis()` and `getFirstDayofWeek()`. By default, the DateChooser component creates a Gregorian calendar with an English locale. With the ICalendar interface, you can provide your own calendar object and use it to instantiate a CalendarAdapter object. Your calendar object needs to implement all the methods in ICalendar in order to work. The International Components for Unicode (ICU) libraries provide a whole set of calendar objects and API that allow you to create non-Gregorian calendars easily. See the topic on "General steps to create a non-Gregorian calendar" on page 201 for details.

Subcomponent	Description
A date format adapter	This adapter is a wrapper that implements the com.alphablox.blox.uimodel.IDateFormat interface. The IDateFormat interface conforms to a subset of the java.text.DateFormat API such as `getCalendar()`, `setCalendar()`, and `format()`. Date formats supported are: • FULL (default) • LONG • MEDIUM • SHORT See http://java.sun.com/j2se/1.4.2/docs/api/java/text/DateFormat.html for examples of the different date formats. With the IDateFormat interface, you can create a non-Gregorian calendar with locale-specific date format. The date formatter object you create needs to implement all the methods in the IDateFormat interface. See the topic on "General steps to create a non-Gregorian calendar" on page 201 for details.

Creating a Gregorian calendar

The following steps create a Gregorian calendar with the date format in English. When users click the calendar icon and choose a date from the calendar that pops up, the selected date is used to populate the text field using the DateFormat.SHORT date format.

1. Import the following packages and classes:
```
<%@ page import="com.alphablox.blox.uimodel.core.*,
          java.text.DateFormat,
          com.alphablox.blox.uimodel.*"%>
```
2. Import the Blox Tag Library:
```
<%@ taglib uri="bloxtld" prefix="blox"%>
```
3. Add a ContainerBlox to contain your DateChooser component:
```
<blox:container id="dateChooserContainer">
```
4. Get the BloxModel of the container:
```
<%
   BloxModel model = dateChooserContainer.getBloxModel();
   ...
%>
```
5. Create your DateChooser object and specify the date format:
```
 <%
   ...
   DateChooser datechooser1 = new DateChooser(DateFormat.SHORT);
   datechooser1.setName("dateChooser1");
%>
```

   Note: If you call the `DateChooser()` constructor without any argument, the default data format is FULL.

6. Add the DateChooser to the model of the container:
```
<%
   ...
    model.add(datechooser1);
%>
```

Here is the complete example:

```
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ page import="com.alphablox.blox.uimodel.core.*,
         java.text.DateFormat,
         com.alphablox.blox.uimodel.*"%>
<%@ taglib uri="bloxtld" prefix="blox"%>

<html>
<head>
<blox:header/>
</head>
<body>
<blox:container id="dateChooserContainer">
<%
    BloxModel model = dateChooserContainer.getBloxModel();
    DateChooser datechooser1 = new DateChooser( DateFormat.SHORT );
   datechooser1.setName("dateChooser1");
   model.add(datechooser1);
%>
</blox:container>
</body>
</html>
```

See the DateChooser Component example in Blox Sampler, under the UI Extensibility section.

Specifying the selected date when the calendar is launched

By default, when a calendar is launched, the calendar for the current month is displayed with the current date selected. To specify a different initial date for selection, use a DateChooser constructor that supports the specification of a selected date. The following example creates a Gregorian calendar with the selected date set to January 01, 2005.

Below is the complete example:

```
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ page import="com.alphablox.blox.uimodel.core.*,
                 com.alphablox.blox.uimodel.*,
                 java.text.DateFormat,
                 java.util.Date,
                 java.util.Calendar,
                 java.util.GregorianCalendar"%>
<%@ taglib uri="bloxtld" prefix="blox"%>

<html>
<head>
<blox:header/>
</head>
<body>
<blox:container id="dateChooserContainer">
<%
    BloxModel model = dateChooserContainer.getBloxModel();
    GregorianCalendar mydate = new GregorianCalendar(2005, Calendar.JANUARY, 01);
    Date d = mydate.getTime();
    DateChooser datechooserJan05 = new DateChooser( d );
    datechooserJan05.setName("datechooserJan05");
    datechooserJan05.setWidth(300);
    model.add(datechooserJan05);
%>
</blox:container>
</body>
</html>
```

Creating a non-English Gregorian calendar

Before creating a non-English Gregorian calendar, you should follow the steps described in Creating a Gregorian calendar to add a basic calendar control on your page.

The following steps demonstrate how to create a French calendar control.

1. Import the java.text.DateFormat, java.util.Date, and java.util.Calendar classes. You import statement now looks as follows:

```
<%@ page import="com.alphablox.blox.uimodel.core.*,
                 com.alphablox.blox.uimodel.*,
                 java.text.DateFormat,
                 java.util.Date,
                 java.util.Calendar"%>
```

2. Create the Calendar object and set its locale to FRANCE.

```
Calendar frenchCalendar = Calendar.getInstance( Locale.FRANCE );
ICalendar frenchAdapter = new CalendarAdapter( frenchCalendar );
```

3. Apply the same calendar to the date formatter.

```
DateFormat frenchFormatter =
    DateFormat.getDateInstance( DateFormat.FULL, Locale.FRANCE );
frenchFormatter.setCalendar( frenchCalendar );
IDateFormat frenchFormat = new DateFormatAdapter( frenchFormatter );
```

   If the same calendar is not applied to the formatter, the text field will default to the English format.

4. Apply the same locale to the DateChooser:

```
DateChooser datechooserFRENCH =
   new DateChooser( frenchAdapter, frenchFormat, Locale.FRANCE );
```

Below is the complete example:

```
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ page import="com.alphablox.blox.uimodel.core.*,
                 com.alphablox.blox.uimodel.*,
                 java.text.DateFormat,
                 java.util.Locale,
                 java.util.Calendar"%>
<%@ taglib uri="bloxtld" prefix="blox"%>

<html>
<head>
<blox:header/>
</head>
<body>
<blox:container id="dateChooserContainer">
<%
    BloxModel model = dateChooserContainer.getBloxModel();

    Calendar frenchCalendar = Calendar.getInstance( Locale.FRANCE );
    ICalendar frenchAdapter = new CalendarAdapter( frenchCalendar );
    DateFormat frenchFormatter =
        DateFormat.getDateInstance( DateFormat.FULL, Locale.FRANCE );

    frenchFormatter.setCalendar( frenchCalendar );
    IDateFormat frenchFormat = new DateFormatAdapter( frenchFormatter );

    DateChooser datechooserFRENCH =
        new DateChooser( frenchAdapter, frenchFormat, Locale.FRANCE );
    datechooserFRENCH.setName("datechooserFRENCH");
    model.add(datechooserFRENCH);
```

```
%>
</blox:container>
</body>
</html>
```

General steps to create a non-Gregorian calendar

For calendars other than the Gregorian, such as the Chinese, Islamic, Japanese, and Hebrew calendars, you need to provide your own calendar object and date formatter. There are four general steps involved:

- With your own calendar object, instantiate a CalendarAdapter object that implements the com.alphablox.blox.uimodel.core.ICalendar interface.
- With your own date formatter, instantiate a DateFormatAdapter object implements the com.alphablox.blox.uimodel.core.IDateFormat interface.
- Apply your calendar to your DateFormatAdapter instance.
- Apply the appropriate locale to the DateChooser component.

The International Components for Unicode (ICU) is a widely used set of Java libraries for Unicode support. Among the features and extensibility it offers is the support for non-Gregorian calendars and different date formats. With the ICU package, you can easily create your own calendar object and date formatter. See the topic on"Installing ICU" and the steps detailed in "Creating a non-Gregorian calendar."

Installing ICU

The International Components for Unicode (ICU) is a widely used set of Java libraries for Unicode support. Among the features and extensibility it offers is the support for non-Gregorian calendars and different date formats.

1. Go to http://www-306.ibm.com/software/globalization/icu/downloads.jsp.
2. Download the JAR file for ICU4J (ICU for Java) class files.
3. Place the downloaded `icu4j.jar` file in your Java class path. The location is different depending on your application server. For details on where to put the JAR files, see the topic on Setting class path.

For steps to create a non-Gregorian calendar control, see the topic on "Creating a non-Gregorian calendar."

Creating a non-Gregorian calendar

Before creating a non-Gregorian calendar, you should:

- Follow the steps described in "Creating a Gregorian calendar" on page 198 to add a basic calendar control on your page.
- Follow the steps described in "Installing ICU" to ensure the needed ICU JAR file is installed properly.

The following steps demonstrate how to create a Japanese calendar control. The same approach can be used to create an Islamic, Hebrew, or Chinese calendar control.

1. Add the following packages to your import statement:
   - com.ibm.icu.util.JapaneseCalendar
   - java.util.Locale

   You import statement now looks as follows:

```
<%@ page import="com.alphablox.blox.uimodel.core.*,
                 com.alphablox.blox.uimodel.*,
                 com.ibm.icu.util.JapaneseCalendar,
                 com.ibm.icu.text.DateFormat,
                 java.util.Locale"%>
```

2. Create a CalendarAdapter object with your calendar object. The CalendarAdapter object needs to implement the ICalendar interface.

```
com.ibm.icu.util.Calendar japaneseCalendar =
  new com.ibm.icu.util.JapaneseCalendar();
ICalendar japaneseCalendarAdapter =
  new CalendarAdapter( japaneseCalendar );
%>
```

3. Create a DateFormatAdapter object with your date formatter. The same calendar must also be applied to your DateFormatAdapter.

```
com.ibm.icu.text.DateFormat japaneseDateFormat =
   com.ibm.icu.text.DateFormat.getDateInstance(
     com.ibm.icu.text.DateFormat.FULL, Locale.JAPAN );
japaneseDateFormat.setCalendar( japaneseCalendar );
IDateFormat japaneseDateFormatter =
  new DateFormatAdapter( japaneseDateFormat );
```

   If the same calendar is not applied to the formatter, the text field will default to the English format.

4. Apply the Japanese locale to the DateChooser.

```
DateChooser datechooserJAPAN =
   new DateChooser( japaneseCalendarAdapter,
     japaneseDateFormatter, Locale.JAPAN );
```

Below is the complete example:

```
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ page import="com.alphablox.blox.uimodel.core.*,
                 com.alphablox.blox.uimodel.*,
                 java.text.DateFormat,
                 java.util.Date"%>
<%@ taglib uri="bloxtld" prefix="blox"%>

<html>
<head>
<blox:header/>
</head>
<body>
<blox:container id="dateChooserContainer">
<%
   BloxModel model = dateChooserContainer.getBloxModel();

   // Create your Japanese CalendarAdapter
   com.ibm.icu.util.Calendar japaneseCalendar =
     new com.ibm.icu.util.JapaneseCalendar();
   ICalendar japaneseCalendarAdapter =
     new CalendarAdapter( japaneseCalendar );

   // Apply the Japanese date format to your DateFormatAdapter
   com.ibm.icu.text.DateFormat japaneseDateFormat =
      com.ibm.icu.text.DateFormat.getDateInstance(
        com.ibm.icu.text.DateFormat.FULL, Locale.JAPAN );
   japaneseDateFormat.setCalendar( japaneseCalendar );
   IDateFormat japaneseDateFormatter =
     new DateFormatAdapter( japaneseDateFormat );

   // Apply the same locale to the chooser
   DateChooser datechooserJAPAN =
      new DateChooser(japaneseCalendarAdapter,
        japaneseDateFormatter, Locale.JAPAN );
```

```
   datechooserJAPAN.setName("datechooserJAPAN");
   model.add(datechooserJAPAN);
%>
</blox:container>
</body>
</html>
```

Important:

- If you use the ICU package, for Chinese calendars, you must set the milliseconds explicitly. Otherwise the fields will not evaluate properly.

  ```
  com.ibm.icu.util.Calendar chineseCalendar =
    new com.ibm.icu.util.ChineseCalendar();
  chineseCalendar.setTimeInMillis((new Date()).getTime());
  ICalendar chineseCalendarAdapter =
    new CalendarAdapter(chineseCalendar);
  ```

- By default, the display direction is set to go from left to right. For bidirectional languages such as Hebrew or Arabic, you must set the direction of your DateChooser as follows:

  ```
  myDateChooser.setBidiDirecton(DateChooser.RTL);
  ```

Calculated members

Calculated members are data members that include dynamically generated data derived from calculations performed against members that actually exist in your result set, and then displayed in newly created rows or columns. Some data sources, such as DB2 OLAP Server, Hyperion Essbase, and Microsoft Analysis Services, can generate calculated members using their query languages, the Essbase Report Specification Language and the Microsoft Multidimensional Expression (MDX) Language. These calculated members, however, cannot typically be interacted with. For example, drilling up or down in cubes using calculated members derived from queries result in the calculated member names being resubmitted to the database. Since these members do not exist natively in the data sources, these queries will fail.

DB2 Alphablox provides a built-in capability to create calculated members after the result sets have been returned from the data sources and manages the interactions so that users can interact with the data and use these calculated members as if they were real members.

Creating calculated members in DB2 Alphablox

Calculated members are created in DB2 Alphablox applications using the `calculatedMembers` property of the DataBlox. One important advantage of using DB2 Alphablox calculated members is that they allow you to add new data to your result sets without having to modify the actual data sources. This can be especially useful when you cannot wait for database changes or just want to experiment with new measures that can be derived from existing data. Here are some examples where calculated members might be useful to your users:

- the variance and variance percentage between the values of two members (such as Budget and Actual)
- the average for all members on a designated dimension (such as Dollar Sales)
- percentage of total sales

For details on the syntax and usage of the DataBlox `calculatedMembers` property, see the *Developer's Reference*. In the following sections, a few important topics about calculated members are presented.

Custom calculation guidelines

You should be aware of the following guidelines and restrictions when working with custom calculations:

Defining custom calculations

The following guidelines apply to defining a custom calculation:

- The definition of a calculated member is an arithmetic expression that is evaluated according to the normal rules of mathematical precedence.
- To position a calculated member, you can specify a reference member that the calculated member should come before in the grid. Otherwise, the calculated member will appear at the end of the existing dimension.
- When more than one calculated member is defined to be positioned before the same reference member, the calculated members are ordered with the last calculated member in the definition closest to the reference member.
- A calculated member must be defined in terms of existing, displayed members of that dimension.
- When more than one calculated member is defined on a dimension without positioning, the members are added to the dimension in the order of definition.
- The definition expression for a calculated member can use a previously defined calculated member (backward reference), but not an as-yet-undefined calculated member (forward reference).

Custom calculation restrictions

- In order for calculated members to be charted, the Generation Filter must be set to show all generations. This can be done by setting the `totalsFilter` property to 0 or by setting the Generation Filter to "Show all generations" in the DHTML client's Chart Options dialog box.
- Calculated members can be saved and restored through normal bookmark operations.
- Member names cannot contain the equal sign (=), curly brackets ({}), or double quotes (").
- Unique members names are required when using Microsoft Analysis Services or DB2 Alphablox Cube Server data sources.
- With DB2 OLAP Server or Essbase, you are strongly advised to use unique member names in calculated member expressions -- if the DataBlox `useAliases` property is set to false or users disable alias names in the UI, calculations may fail.
- Use the `ifNotNumber` function in an expression to provide special case logic if you want missing or null values for a given member to be treated as a specified number.

  Tip: Make sure you understand what the resulting values for the calculations will be when specifying values with the `ifNotNumber` function. Substituting a value for some missing or null values might not make sense when used in a calculation.

Conditions preventing proper data display

The following table lists conditions that prevent the display of meaningful data, and the consequences in grid displays when the conditions occur.

Condition	The Grid Cell Displays:
Divide by zero	The calculated member does not appear in the grid
Reference to members not in the result set	Empty string (or the value specified in the `missingValueString` property)
Invalid calculation expression	`#Error` (and an error appears on the console or log)
Reference to a missing or non-numeric value	Empty string (or the value specified in the `missingValueString` property)

Calculated member property syntax

To specify one or more custom calculations on a member, use the DataBlox `calculatedMembers` property (or the `setCalculatedMembers` method). Note that the tag attribute syntax can include multiple custom calculations in a single statement:

```
calculatedMembers="dim1:calc1{refMember1:gen:missingIsZero}=
   expr1{scopeDim:scopeMember}, dim2:calc2{refMember2} =
   expr2{scopeDim:scopeMember},..., dimn:calcN
      {refMemberN:gen:missingIsZero} =
   exprN{scopeDim:scopeMember}"
```

where:

- The `dimN` value is the name of the dimension on which to create a calculated member.
- The `calcN` value is the name of calculated member.
- [Optional] The `refMemberN` value is the name of an existing member which the calculated member calculation will come before in the grid. The `refMemberN` cannot be another calculated member.
- [Optional] The `missingIsZero` component of the `definitionString` can be used if you want all missing values for members involved in the calculation to be treated as zero. By default, all missing values in calculations are treated as missing. [Note: This keyword only affects calculations using member variables. It has no effect on calculation functions.]
- The `exprN` value is the arithmetical expression involving members of `dim`. 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.

Functions available for calculated members

In the calculation expressions you use, the following tables summarize the arithmetic, search, special calculation, and missing value related functions available. For details on the syntax and usage, see the DataBlox section of the *Developer's Reference*.

Arithmetic Function
Description

Abs Returns the absolute value of a member. This can only be used on a single number item such as the result of another calculation or a single member.

Average
Returns the average of all the numbers in the definition, which is the sum divided by count.

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.

Min Returns the lowest value in all the numbers in the definition.

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; can only be used on a single number item such as the result of another calculation or a single member.

Sqrt Returns the square root of a number; can only be used on a single number item, such as the result of another calculation or a single member.

Stdev Returns the standard deviation of all the numbers in the definition.

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 is the average squared deviation of each number in the set from the average.

Search Function
Description

Child Returns all children of a specified member.

Descendants
Returns all descendants of a specified member.

Leaf Returns all leaf-level descendants of the specified member.

Special Calculation Function
Description

Rank Returns the values from the specified dimensions in ascending or descending order for the specified member.

RunningTotal
Returns the cumulative sum of values from the specified dimension for the specified member.

Missing Value Related Function
Description

ifNotNumber

Can be used to provide special case logic to handle missing or null values in the result set used in the calculation.

Details on the syntax and usage of these arithmetic, search, special calculation, and missing value related functions can be found in the `calculatedMembers` property description in the DataBlox section of the *Developer's Reference*.

Calculated member examples

The following examples illustrate some common uses for calculated members:

- Define a custom calculation with cell values showing the variance percentage between actual and budget values:

```
calculatedMembers = "Scenario:Variance % = (Actual-Budget)/Budget*100"
```

- Define a custom calculation with cell values showing the variance percentage between actual and budget values, and provide logic to substitute a value of 1,000,000 for Actual and a value of 5,000 for Budget (do not use commas when specifying the number):

```
calculatedMembers="Scenario:Variance %=(ifNotNumber(Actual,1000000)-
   ifNotNumber(Budget,5000))/ifNotNumber(Budget,5000) * 100"
```

- Define a custom calculation that displays the sales to date for the first two quarters of the year:

```
calculatedMembers = "Year: YTD = \"Q1,2000\" + \"Q2,2000\""
```

- Combine two custom calculations in a single attribute (where Scenario is on one dimension and Year on another):

```
calculatedMembers="Scenario:Variance % =(Actual-Budget)/Budget*100,
   Year: YTD = \"Q1,2000\" + \"Q2,2000\""
```

- Combine two custom calculations in a single attribute, and substitute different values for the same member used in different expressions:

```
calculatedMembers ="Scenario:Variance % =
   (Actual-ifNotNumber(Budget, 10000))/ifNotNumber(Budget, 10000) * 100,
   Scenario: Difference = Actual-ifNotNumber(Budget, 0)"
```

- Define a custom calculation with cell values showing the variance percentage between actual and budget values. Position the calculated member `Variance %` to come before the member `Actual` on the grid.

```
calculatedMembers = "Scenario: Variance % {Actual} =
   (Actual-Budget)/Budget * 100"
```

- To add a separate ranking within each group, you can use the Rank function, as shown in this example:

```
calculatedMembers="All Products:Rank =
   Rank(All Products,All Locations,2,DESC,GROUPDIM)
```

Note: To clear calculated members, pass an empty string to `setCalculatedMembers`.

You can also perform calculations with nested dimensions on the same axis or calculations within calculations. Specifying the scope of the calculated members or assign a generation number to your calculation which aids in positioning the calculated member. For more details and examples showing the use of the `calculatedMembers` property and associated methods, see the DataBlox section of the *Developer's Reference*.

Calculated members using Essbase report script commands

Calculated members can also be created using Essbase report script commands. While this can be useful in some situations, the data displayed as a result cannot be interacted with without generating DB2 OLAP Server or Essbase error messages indicating that calculated members (which do not actually exist in the DB2 OLAP Server or Essbase cube) do not exist. For example, when you drill on a grid in which calculated members exist, drilling and other operations are disabled. As a preferred alternative, whenever possible, you should create calculated members using DB2 Alphablox.

For details about creating calculated members using Essbase report script commands, see your DB2 OLAP Server or Essbase documentation. Also, be sure to check the "Essbase report script commands supported by DB2 Alphablox" on page 124.

Chapter 19. Filtering data

This topic discusses tips and techniques for filtering data for the users, either to help more effectively work with large result sets, limit access to information, or personalize the information they see.

Hiding dimensions and members

Access to data allows users to creatively ask questions and compare data in creative ways, but sometimes the amount of available information can be overwhelming. For naive end users, too much data to look at can leave them at a loss for where to begin. For savvy users, lots of data can actually become "noise" to them, distracting them from being able to focus on the essential data. As a developer, you need to keep your intended audience in mind, and develop analytic applications which give users just the right amount of information. Users with full access to the DataLayout panel, the Member Filter dialogs, and complete listings of data can end up spending a lot of their time using Hide/Show and Keep/Remove functionality to winnow down information to the information they need. But, as a developer, you can help them out by using DataBlox properties to filter out data which is not relevant to the task at hand, or is data that is seldom used.

With attribute dimensions in DB2 OLAP Server and Hyperion Essbase as well as virtual dimensions in Microsoft Analysis Services, businesses slice their data in many ways unavailable before. For example, in the QCC sample data sources, you can analyze chocolate sales by groupings of pieces per package or ounces per package, whether they have nuts or, list products by their introduction dates, or analyze stores considering their square footage. This is great, when you want to use this information in your analysis. But, if you don't, their presence in views can just become a nuisance. Luckily, DB2 Alphablox has two properties which make it easy to hide dimensions and members, the DataLayout `hiddenDimensionsOnOtherAxis` property and the DataBlox `hiddenMembers` property.

The DataLayoutBlox `hiddenDimensionsOnOtherAxis` property can be used to list dimensions which you do not want to appear in the DataLayout panel. For an example, see the Filtering Data section of Blox Sampler. For details about the syntax and usage of `hiddenDimensionsOnOtherAxis`, see the DataLayoutBlox Reference section of the *Developer's Reference*.

The DataBlox `hiddenMembers` property can be used sometimes to hide members that don't make sense to be displayed. In the Scenario dimension, for example, the top-level member is Scenario, but this member is really just a bucket for holding the members grouped as children under it. And, in DB2 OLAP Server and Hyperion Essbase, Scenario actually displays the data from the first child under it -- but there is no data for the member Scenario. In the Filtering Data section of Blox Sampler, an example of hiding the Scenario member (in DB2 OLAP Server, Essbase, and Microsoft Analysis Services) shows the children of Scenario. Note that when you drill up on one of the children, Scenario will actually appear (even though it is listed as a hidden member). But, when you drill back down into Scenario, it disappears once again. This behavior is unavoidable, due to limitations of how drilling operations need to perform. For details about the syntax and usage of `hiddenMembers`, see the DataBlox Reference section of the *Developer's Reference*.

Using the dimensionRoot property

One of the simplest ways to filter information, or control access to it, is to use the DataBlox `dimensionRoot` property specify specific members on dimensions to be used as virtual roots for your users. Once a particular dimension root member is defined as a new "virtual" root, users will be prevented from drilling up into members above the defined root. This setting applies to page filters, rows and columns, and lists of dimension members that appear in the Member Filter. A virtual root may be useful for limiting access to areas of your data that you do not want others to access. This property can be used for limited security use or just to help prevent users from getting lost in the data.

You are probably using database security to prevent users from seeing data values that they should not have access to, however, sometimes it is possible that just seeing the member names may be too much information to be shared. In limited cases like this, the `dimensionRoot` may be useful. For example, a database listing prospective customers in all regions of the country may provide useful information to a disgruntled employee who is just about to leave your company for a competitor. If you minimizing access to information like this is important to you, then the `dimensionRoot` property may present a useful option.

Note: You should not use the `dimensionRoot` property as your only means to prevent users from accessing private information. It is always possible that other database tools may be used against the same data source to access information that you have blocked using a virtual root defined in a DataBlox.

Your primary goal could also be to improve usability by blocking paths to information that are not relevant to the needs of particular users, thus preventing them from drilling around in information paths that may cause them to become lost in the data. If some information is not relevant to the task they need to perform, then the use of `dimensionRoot` could help minimize this "lost in navigation space" problem.

Setting virtual roots for users

In the following example, users are restricted in their ability to access information about particular regions in the country. These simple steps will allow you to configure a DataBlox to create "virtual" roots for your users tied to the region of the country they work in:

1. Add the `dimensionRoot` attribute to a standalone or nested DataBlox.

   ```
   <blox:data id="myDataBlox"
      dimensionRoot=""
      ...
   </blox:data>
   ```

2. Add a dimension and the single member in that dimension that you want to use as a "virtual" root for your users.

   For example, if you wanted to limit access to the East region in the QCC database, here's what the `dimensionRoot` setting would look like:

   ```
   dimensionRoot="All Locations:East"
   ```

3. Save your page and test it.

This is a very simple example, with a hardcoded `dimensionRoot` setting. The `dimensionRoot` could also be dynamically set as a page loads, by basing the

dimensionRoot setting on a value stored in the DB2 Alphablox Repository. A custom user property could have as its value the name of the region that should appear in the data source.

Note: For security reasons, you will probably want to disallow user editing in situations like this, to prevent users from changing their own regions and gaining access to other regions.

Note: See the *Administrator's Guide* for instructions on how to create custom user properties for users and applications.

Once a custom user property is configured using DB2 Alphablox, you can access that value using server-side Java methods. On a JSP page with a DataBlox, your dimensionRoot setting could be configured dynamically in the DataBlox tag or using the associated setDimensionRoot() method. If you included a RepositoryBlox above the DataBlox tag on a JSP page, you can dynamically include the value within the DataBlox tag using a JSP expression statement. The code would look similar this example:

```
dimensionRoot="<%= myRB.getUserProperty("userRegion") %>"
```

where myRB stands for the name of the RepositoryBlox that is defined above this setting that can get the value from the DB2 Alphablox Repository. In this example, the custom user property is userRegion. This very simple technique can also be applied to other properties as well.

Fixed choice lists

Typically, page filters in DB2 Alphablox applications let users roam up and down in information within a dimension without any restrictions, and limit access to the Member Filter. The dimensionRoot property allows you to restrict information access by defining a root that a user cannot get below. But sometimes the better option is to limit the number of choices a user can select from by setting the PageBlox fixedChoiceLists, moreChoicesEnabledDefault, and moreChoicesEnabled properties to create fixed choice lists.

Example: A "Fixed Choice List" example showing the use two of these PageBlox properties is available in the Filtering Data section of the Blox Sampler application.

Using the fixedChoiceLists property

The PageBlox fixedChoiceLists property places named dimensions and members on a page filter's drop list for users to pick from. Unlike normal page filters, fixed choice lists limit users' options to the ones specified by you. The default value of fixedChoiceLists is an empty string, giving users full access to dimensions and their members. When dimensions and specific members are specified using this property, users can access only the members you have defined. For example, to limit a user to see only two regions, Central and East, the PageBlox fixedChoiceLists attribute would look similar to this example:

```
fixedChoiceLists="All Locations:Central,East"
```

If your initial query does not include one of the fixed choice list members in it, the top-level member for a dimension specified in the fixed choice list will also initially appear in the list. After a user selects one of the fixed choice list members, the top-level member will then disappear from the list.

In order for a fixed choice list to appear in the PageBlox, you must also remember to specify the dimensions from that list in the `selectableSlicerDimensions` property of the DataBlox. In our example, the DataBlox (not PageBlox) attribute would appear like this:

```
selectableSlicerDimensions="All Locations"
```

Note: If your initial query has more than one member from a dimension in the fixed choice, then the fixed choice list page filter will not appear in the PageBlox. For example, if your initial query was:

```
query='<SYM <ROW ("All Products") <ICHILD "All Products"
  <COL(Scenario) <CHILD Scenario "2001" Central East !'/>
```

then the fixed choice list would not appear in the PageBlox. To make this query work, you should only include the member in the query that you want to appear by default in the fixed choice list, either Central or East.

Using the moreChoicesEnabledDefault and moreChoicesEnabled properties

The DataBlox `moreChoicesEnabledDefault` property, by default, allows users to select the More Choices option on a page filter. To disable this default feature, set the property to false, as shown here:

```
moreChoicesEnabledDefault="false"
```

Alternatively, you can use the more selective version of this property, the `moreChoicesEnabled` property. This option requires you to specify individual dimensions for which you do not want the More Choices menu option to appear.

See the *Developer's Reference* for details about the two variants of the `moreChoicesEnabled` property.

Using MemberSecurityBlox to filter members

MemberSecurityBlox, included in the Blox Logic Tag Library, can be used to filter lists of dimension members based on access permissions. MemberSecurityBlox suppresses access to members using the DataBlox `suppressNoAccess` property based on specified `MemberSecurityFilter` values. This property can take multiple root members and allows specifying multiple dimension:member pairings for filtering.

For an example of the use of MemberSecurityBlox, see "Listing cube members using MemberSecurityBlox" on page 52. For details about the syntax and usage of the MemberSecurityBlox, see the Business Logic Blox and TimeSchema DTD Reference section of the *Developer's Reference*.

Using HTML form elements and FormBlox components

Even though DB2 Alphablox provides many properties for configuring and filtering information delivery within the presentation Blox components, you can also move some of this functionality out of these Blox and onto web pages. HTML form elements, including selection lists, check boxes, and radio buttons, as well as FormBlox can be used to access information and make selections. You can create powerful, yet easy-to-use applications that any web user can use by removing Blox

menu bar and toolbars, incorporating HTML form elements that invoke server-side logic using the DHTML Client API's JavaScript methods.

Besides using standard HTML form elements and the Blox Client API, you can also use the Blox Form Tag Library, including FormBlox components, to build analytic applications. FormBlox generate customized form elements with commonly used functionality required for building analytic applications. For example, the DimensionSelectFormBlox and MemberSelectFormBlox generate HTML selection lists automatically populated with dimension names and member names, and can be used to create simple lists for users to select from. An additional benefit of FormBlox components is that they persist and maintain state even when users leave a page and return to it later in their browser session.

Using both standard HTML form elements and the FormBlox components, you can restrict the interactivity of user interactions and filter out options you do not want to expose to users. For details about the available FormBlox components, see Chapter 6, "Blox Form Tag Library," on page 43 and the Blox Form Tags Reference section of the *Developer's Reference*.

Using queries

Perhaps the most effective way to filter data before it gets to the user is to construct good query statements. Using queries that only return data that is relevant to the immediate task, you can avoid large result sets, which take longer to execute on the database server and impact network traffic.

It is outside the scope of this guide to explain how to optimize queries for your particular data source. Check your database documentation and other resources for details about filtering data using query statements. Also, Chapter 14, "Retrieving data," on page 121 in this guide provides limited information about query techniques which might be useful for filtering information from supported multidimensional and relational databases.

Data suppression using Blox properties

DB2 Alphablox offers several properties that can enhance the usability and performance of your analytic applications. The following sections briefly describe how `suppressMissingOnRows`, `suppressMissingOnColunns`, `suppressZeros`, `suppressDuplicates`, and `suppressNoAccess` properties of the DataBlox can be used. See the DataBlox Reference section of the *Developer's Reference* for more details on the syntax and usage of these properties.

While suppressing data makes sense in many situations where there are lots of rows or columns of data that would be filled entirely of zeros or missing data, suppressing this information can mislead a business user who, for example, would actually need to know that data is missing in order to take action on this. If users have access to menu bars on Blox, they can change this setting manually in the Data Options dialog but they may not know about the setting or may not think of unsuppressing data that they didn't know was missing. If menu bars are not available and you want to suppress data, consider placing form elements (such as a checkbox or radio buttons) on pages, allow users to control this setting and realize that a suppression of data is in use.

Using the suppressMissingOnRows and suppressMissingOnColumns properties

The suppressMissingOnRows and suppressMissingOnColumns properties removes rows or columns from your grids when there is no data at all in the returned rows or columns. If any cells in a row or column in your data set has a value in it, the entire row or column is visible.

To enable this feature, add the suppressMissingOnRows and suppressMissingOnColumns attribute to a DataBlox and set the value to true, like this:

```
suppressMissingOnRows="true"
suppressMissingOnColumns="true"
```

You can also programmatically control this feature using the associated Java methods listed in the DataBlox Reference section of the *Developer's Reference*.

For DB2 OLAP Server and Essbase data sources, when suppressMissingOnRows or suppressMissingOnColumns is enabled by setting the property to true in your DataBlox, suppression is performed on both the database server and within DB2 Alphablox. Here is a summary of the behavior you should expect when using DB2 OLAP Server or Essbase:

- If the initial query is a report script (instead of a bookmark), DB2 Alphablox does the suppression of the missing data.
- If the query is the result of a bookmark, a drill, or a pivot, the DB2 OLAP Server or Essbase server is asked to suppress rows with missing values.

Relying on the Essbase report script command <SUPPRESSMISSING alone is not generally the best solution, since DB2 Alphablox will not then remove missing data that results from drilling or other operations.

When end users are using the DHTML client, the addition of the Essbase <SUPPRESSMISSING command to your initial report script command most likely will not noticeably affect performance, even when queries return large result sets (more than 1000 rows). The DHTML client optimizes the result sets it retrieves, limiting it to what can be viewed in a particular instance.

Note: Use the GridBlox missingValueString property or its associated methods to specify what should be displayed in cells that have no value. This property is useful when entire rows or columns are not suppressed with the DataBlox suppressMissing property or the Essbase <SUPPRESSMISSING report script command.

See the DataBlox Reference section of the *Developer's Reference* for details about the missingValueString property.

Using the suppressZeros property

When the DataBlox suppressZeros property is set to true (default is false), all rows or columns containing only zeros will be suppressed. If any cells in a row or column in your data set have values other than zero in it, the entire row or column will be displayed.

To enable this feature, add the `suppressZeros` attribute to a DataBlox and set the value to `true`, like this:

```
suppressZeros="true"
```

You can also programmatically control this feature using the associated Java methods listed in the DataBlox section of the *Developer's Reference*.

Using the suppressDuplicates property

The `suppressDuplicates` property of a DataBlox, when set to true (the default setting), removes all duplicate header values from rows or columns in your grids.

If you do not want to suppress duplicate header values, add the `suppressDuplicates` attribute to a DataBlox and set the value to `false`, like this:

```
suppressDuplicates="false"
```

You can also programmatically control this feature using the associated Java methods listed in the DataBlox Reference section of the *Developer's Reference*.

Note: To suppress duplicate DB2 OLAP Server or Essbase shared members in the initial query, use the `<SUPSHARE` report script command in your query statement. For more information about the suppression of DB2 OLAP Server or Essbase shared members, see the DB2 OLAP Server or Hyperion Essbase documentation.

Chapter 20. Persisting and bookmarking data

Persistence of data and views is an important consideration for analytic applications. While most of the data accessed is stored in databases, DB2 Alphablox can be used to manage data persistence for application states, bookmarks, and custom properties. A brief discussion and example of using JavaServer Pages techniques for persisting data values during a session is included.

Data persistence in DB2 Alphablox

An DB2 Alphablox application is a collection of resources. DB2 Alphablox provides built-in features that can be used to allow users to save and restore a variety of bookmarks and application states. For example, after drilling and pivoting, or selecting a preferred chart layout, a user can bookmark the current view for later recall. And, depending on application settings you have defined, application states can be maintained and automatically stored when a session is over, either when a user closes a browser window or when the current session times out.

Bookmarks and saved application states can be saved publicly or privately. Public bookmarks and application states can be shared among all users with access to that application. For information on using this feature through the Blox user interface, see the user help page. The following sections discuss further details about bookmarks and application states, and list the available Blox properties and methods that developers can use for managing bookmarks and application states.

Application states

Application states are another way for DB2 Alphablox to save information in your application. When a user starts a session that accesses an application, DB2 Alphablox creates an instance of the application. As long as the session is alive, this instance maintains the state of a user's current application session, including the status of application resources, such as query result sets, grid and chart appearance, sorts and other changes made by the user.

An **application state** in a DB2 Alphablox application is a representation of the state of all of the Blox in that application at a particular moment in time. During the process of using a DB2 Alphablox application, a user's current application state is tracked and maintained by DB2 Alphablox. This state is defined as the **current application state**. Alternatively, a **saved application state** is a representation of all of the Blox within an application at the particular time the application state was saved.

While bookmarks save the state of an individual Blox, application states save the state of the entire application. Application states can be saved and restored later as needed. Also, application states can be saved publicly for sharing between all users with access to an application, or privately for each individual user.

Application state management is typically handled automatically by DB2 Alphablox based on settings made in the application definitions. If the application definition has the `Automatic Save Enabled` property set to `yes`, DB2 Alphablox automatically saves the current state of an application in the Repository when:

- the user exits the application (by closing the browser)
- the user session times out (by default, after 15 minutes of inactivity)

When the user next accesses the application, DB2 Alphablox restores the most recent saved state of the application if the Restore Application State setting in the application definition is set to `yes` (default is `no`). If both Automatic Save Enabled and Restore Application State are set to `yes`, then the user can access the original, default application state through the Applications page on the DB2 Alphablox home page. Also, a developer could include a custom button on a page within an application to allow users to restore the last saved state.

Note: After a session times out, if the user attempts to work with it in the browser window, a message appears instructing the user to press the browser's Refresh (or Reload) button to reconnect to the DB2 Alphablox.

Note: DB2 Alphablox does not save the data in an application state. When a saved application state is restored, the application retrieves fresh data from the database.

The following table lists RepositoryBlox methods relevant to managing application states:

Java Methods

`delete()`
`deleteApplicationState()`
`exists()`
`getApplicationStateNameAndDescription()`
`list()`
`load()`
`rename()`
`renameApplicationState()`
`restoreApplicationState()`
`save()`
`saveApplicationState()`
`search()`

Custom properties in the DB2 Alphablox Repository

Using the DB2 Alphablox Repository, you can create custom user, group, and application properties that can be retrieved or modified using standard JSP methods. After being created in the user, group, and application definitions, these custom properties can be stored and retrieved using a RepositoryBlox. Using JavaServer Pages technology, you can substitute the values from these properties into your Java code and within your Blox tag attributes as runtime expression values. Custom properties are available according to the DB2 Alphablox property inheritance hierarchy.

Note: If you are adding Blox to a portal application, keep in mind that portlets rely on the portal infrastructure to access user profile information. Therefore you should use the Portlet API for user information rather than through DB2 Alphablox.

Creating custom user properties

To understand how to use a custom property, consider the ChartBlox `chartType` property. The default value is `3D Bar`, but for a set of financial applications, a CFO may prefer line charts. A developer could define a custom property to specify a different default for this user. The custom property name would be the same as the Blox property (`chartType`) and would have a value of `Line`.

Based on this example, the following steps would create the necessary custom user property:

1. From the DB2 Alphablox home page, choose the Administration tab.
2. Click the Server link and then User Definitions under Custom Properties. The User Definitions page opens.
3. Click the Create button at the bottom of the page. The Create User Custom Property page appears.
4. Complete the following entries:
   - Property Name: `chartType`
   - Default Value: `Line`
   - Value List: `Line, 3D Bar`
5. Click Save to save the new property. Next you will assign this custom property to a user's definition.
6. From the DB2 Alphablox home page, choose the Administration tab.
7. Click the Users link. The User Definition page opens, displaying a list of existing user definitions.
8. To define a new user, click the Create button. The Create User page appears, displaying the General Properties panel. (To assign a value for a custom property to an existing user definition, select the user name from the list and click the Edit button.)
9. Provide (or edit) entries for Username, Password, and Confirm Password.
10. The `chartType` property appears at the bottom of the General Properties panel. Ensure that the property value is set to `Line`.
11. Click Save.
12. Now test your property by logging in as the user you just created.

JavaServer Pages technology and data persistence

JavaServer Pages technology offers several methods for managing data values throughout user sessions and between sessions. JavaServer Pages technology offers several different ways that data values can be stored and retrieved, including URL rewriting, hidden form values, `request` object methods, and `session` object methods. See a JavaServer Pages book or other JSP resources for descriptions of the techniques available within JSP-based applications.

In the following task, you will learn an example of using one of these techniques—the `request` object `getParameter` method.

Using request parameters to retrieve a URL attribute values

When a web page is submitted, the URL address can pass information that can be retrieved within a linked page. A common use in an DB2 Alphablox application is to create a custom print page using the Blox view on a page. Using the DB2 Alphablox URL render attribute, you could simply open a new print page a line like this:

```
window.open("view-print.jsp?render=printer","_blank");
```

Using this JavaScript method, a new browser window would open displaying the current Blox view rendered into HTML for printing. But, if you use HTML form elements (buttons, check boxes, etc.) and text on the page being rendered, all of those elements and text would be included on the printable page. This would not be an ideal solution.

Instead of this, you can create a custom print page that will retrieve elements from the Blox view and incorporate them into a custom print page. The following steps show you how you can pass a value between JSP pages using the `getParameter` method:

1. On the page with a Blox view that you want to print, create a JavaScript function that will construct a URL that will pass information to the custom print page.

   Here is an example of a JavaScript function that creates a URL address, passing the time, region (from the selected value in an HTML selection list), render mode, and HTML theme:

   ```
   function printPreview() {
     var   region=document.RegionForm.RegionSelectionList.
             options[document.RegionForm.RegionSelectionList,
             selectedIndex].text;

     var timestamp=new Date();

     var URL="passingValues-print.jsp?Region="+escape(region)+
              "&TimeStamp="+escape(timestamp.toString())+
              "&render=printer"+
              "&theme=printer";

     window.open(URL,"PrintPreviewWindow");
   }
   ```

2. In your custom print page, capture the values from the URL query string, incorporating them into your page as needed.

   In the body of the custom print page, the following example shows the time and region being placed on the page using the request object `getParameter` method:

   ```
   <h1>Sales for <%= request.getParameter("Region") %></h1>
   <p>
   <blox:display bloxRef="RegionPresentBlox"/>
   </p>
   <h3><%= request.getParameter("timestamp") %></h3>
   ```

Note: The "Passing Values Between Pages" example under the Persisting and Bookmarking section in the Blox Sampler example set demonstrates the use of request parameters.

Bookmarks - developer details

As a developer, there are some important details to be aware of when working with bookmarking:

- A bookmark is a collection of property sets (name-value pairs) used to restore the state of a Blox. Here's an example of a property set for a Blox property:
  - `dividerLocation = 0.25`
- A bookmark includes not only Blox properties, but also bookmark properties. These bookmark properties include:
  - application
  - Blox type (Present, Chart, Grid, etc.)
  - Description
  - Bookmark name
  - Hidden (boolean)
  - Reference to Blox properties
- When a bookmark is saved, it is the **difference** between the initial Blox state and the current state of the Blox when the bookmark is saved. The **initial state**

includes the defaults plus the defined tag and tag attribute properties. In the DB2 Alphablox Repository, bookmarks with property sets only exist for Blox whose properties that have changed since their initial state.

- When a bookmark is saved, it is saved to and restored from a specific location in the DB2 Alphablox Repository based on the application name, Blox name, bookmark type (public, group, private), and the group or user name.
- The bookmarks query file, includes a serialized Query object, which is much like a grid result set with no data (that is, tuples of member objects). This is not the textual query, which represents the query defined in the initial Blox state.
- Using bookmark filters, you can created context filters for bookmarks, which are useful in creating applications with a single Blox on a page, setting views based on bookmark menu items. And, using shared bookmarks, you can create "published" and self-service applications.

Details about common Blox properties, BookmarksBlox tags and tag attributes, and the available server-side APIs can be found in the Common Blox Reference and BookmarksBlox Reference sections of the *Developer's Reference*.

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).
```

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:

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

```
<%@ 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>
```

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 add a message to the console) 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()`.

   ```
   <% myPresent.addEventFilter(new LoadFilter()); %>
   ```

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.

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

<html>
<head>
  <title>Bookmarks Filter Events</title>
  <!-- Blox header tag -->
  <blox:header/>
</head>

<%!
public class LoadFilter implements BookmarkLoadFilter {
   public void bookmarkLoad( BookmarkLoadEvent ble )
        throws Exception {
      Bookmark bookmark = ble.getBookmark();
      String name = bookmark.getName();
      System.out.println("A bookmark called " + name + " is
        loaded.");
      }
   }
%>

<body>
<blox:present id="myPresent" >
   <blox:data dataSourceName="TBC"
```

```
      query="<Row(Market) <ICHILD Market <Column(Year) Year !"/>
<%
   myPresent.addEventFilter(new LoadFilter());
%>
</blox:present>
</body>
</html>
```

Customizing applications using the BookmarksBlox API

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.

Here are a couple of interesting examples of bookmark customization examples that are included in Blox Sampler.

Using bookmark events

Four bookmark events are available to be used within DB2 Alphablox applications: `load`, `save`, `rename`, and `delete`. Any combination of these events can be registered with a Blox, including multiple events of a similar type. When registered, these events will be called before the actual process is started, allowing you a chance to customize bookmark behavior.

A typical event looks like:

```
public class LoadFilter implements BookmarkLoadFilter {
    public void bookmarkLoad( BookmarkLoadEvent ble ) throws Exception {
        Bookmark bookmark = ble.getBookmark();
        String name = bookmark.getName();
       System.out.println("Bookmark " + name + " applied");
    }
}
```

In this example, an event gets the name of the bookmark being loaded, then displays it in the console.

Using registered events

Registering events is done within Blox tags, like this:

```
<blox:present id="myPresent3" >
   <blox:data
      dataSourceName="QCC-Essbase"
      query="!"/>
<%
   myPresent3.addEventFilter(new LoadFilter());
   myPresent3.addEventFilter(new SaveFilter());
```

```
   myPresent3.addEventFilter(new RenameFilter());
   myPresent3.addEventFilter(new DeleteFilter());
%>
</blox:present>
```

The Blox tag above registers all four bookmark events.

Using dynamic queries with bookmarks

With the new bookmark APIs, you can tell the server to execute a query different than the one that originally saved with the bookmark and to use the result as the result set.

The following example, using a Microsoft Analysis Services MDX query statement, shows the modification of a bookmark to store a parameterized textual query that will be used when the bookmark is loaded. Forcing a bookmark to use a different textual query involves saving the query with the bookmark properties and setting the `textualQueryEnabled` property to `true`.

In a bookmark `save` event, you can do the following to save a parameterized query:

```
// Parameterized query (NOTE: :year and :quarter)
final String PARAM_QUERY = "SELECT {[Products].[Category].[All Products],
   [Products].[Category].[All Products].children} ON ROWS,
   {[Time].[Calendar].[All Time Periods].[:year],
   [Time].[Calendar].[All Time Periods].[:year].[:quarter]}
   ON COLUMNS FROM [QCC]";

// get the Bookmark Object from the BookmarkSaveEvent

Bookmark bookmark = bse.getBookmark();

// Find DataBlox properties for this bookmark

BookmarkProperties data =
   bookmark.getBookmarkPropertiesByType(Bookmark.DATA_BLOX_TYPE);

// If DataBlox properties not found in existing property set, create

if (data == null) {
  data = bookmark.createBookmarkProperties(Bookmark.DATA_BLOX_TYPE);
  }

// Set textualQueryEnabled to true, saving the query above to bookmark

  data.setProperty("textualQueryEnabled", true);
  data.setProperty("query", PARAM_QUERY);
```

When the bookmark is loaded, you can use a bookmark `load` event to replace the parameters with relevant information given by a user, for example:

```
// get the Bookmark Object from the BookmarkLoadEvent
Bookmark bookmark = ble.getBookmark();

// find a DataBlox properties for this bookmark

BookmarkProperties data =
   bookmark.getBookmarkPropertiesByType(Bookmark.DATA_BLOX_TYPE);

if (data != null) {
```

```
    // Get the parameterized query from the bookmark
     String query = data.getProperty("query");

// Replace the parameters with real information

// NOTE: replaceText simply replaces any references to the 2nd argument
// with the contents of the third argument.

     query = replaceText(query, ":year", "2002");
     query = replaceText(query, ":quarter", "Qtr2");

// set the new un-parameterized query

data.setProperty("query", query);
}
```

When the bookmark is loaded, the parameters will be exchanged for `2002` and `Qtr2` and the query will be executed.

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>
```

Getting DB2 OLAP Server or Essbase serialized queries in text form when a bookmark is loaded

This example demonstrates how to get a serialized query in text form from the bookmark (which is not the same as the query that is in the DataBlox). Note that this example only works with DB2 OLAP Server and Essbase data sources. To reference Microsoft Analysis Services, you need to save the query yourself.

1. Set the DataBlox's `textualQueryEnabled` property to `true`:

   ```
   <blox:data...
         textualQueryEnabled="true" />
   ```

2. The server-side event filter, `BookmarkLoadFilter`, is used to trigger the custom action when the bookmark is loaded. See "Using a remote PDF processor" on page 244 for an example of the server-side event filter.
3. When a bookmark is loaded, a serialized query in text form is retrieved.

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" %>

<%@ taglib uri="bloxtld" prefix="blox"%>
<html>
<head>
<blox:header/>
<%!
  public class LoadFilter implements BookmarkLoadFilter
  {
   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();
           }

     System.out.println("query=" + query);
     }
```

```
    }
%>
<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());
%>
</blox:present>
</body>
</html>
```

Using custom properties to restrict access

Custom properties are an enhancement to existing bookmarks. You can now place additional key/value information into a bookmark to be used when the bookmark is loaded during a bookmark `load` event.

In a bookmark `save` event, you can add custom properties to the bookmark, for example:

```
// get the Bookmark Object from the BookmarkSaveEvent
Bookmark bookmark = bse.getBookmark();

// add username of bookmark owner as a custom property
bookmark.setCustomProperty("Owner", "Admin");
```

Note: You can create a constructor for your `BookmarkSaveEvent` class, or any other bookmark event for that matter, to take a parameter such as an owner name

When the bookmark is loaded, you can use a bookmark `load` event to get the custom property and see if the owners match:

```
// get the Bookmark Object from the BookmarkLoadEvent
Bookmark bookmark = ble.getBookmark();

// get the owner custom property
String owner = bookmark.getCustomProperty("Owner");

// compare this user and the owner
if (!owner.equalsIgnoreCase(currentUser)) {

    // if user and owner do not match, stop bookmark load
    ble.cancelEvent();
}
```

```
You can also do the same to stop the deletion of a bookmark
  that the current user does not own:
```

```
// get the Bookmark Object from the BookmarkDeleteEvent
Bookmark bookmark = bde.getBookmark();

// get the owner custom property
String owner = bookmark.getCustomProperty("Owner");

// compare this user and the owner
if (!owner.equalsIgnoreCase(currentUser)) {
    // if user and owner don't match, stop bookmark delete
    bde.cancelEvent();
}
```

Chapter 21. Distributing views

You can share analytic views by e-mail and bookmarking.

Even though most analysis happens by individuals sitting alone in either an office or cubicle, the information and results of analysis are often shared with others in a business, including executives, colleagues, and customers. DB2 Alphablox applications, thanks to the ubiquity of the Internet and web browsers, can be shared with others in your office or company to distances around the world. In the following topic, e-mail, bookmarking, and printing methods of distributing and sharing information are discussed.

Creating mail links using an e-mail bean

An e-mail bean is available for e-mailing a static view of the data to one or more e-mail recipients. This bean and a set of support JSP files and images are provided in the e-mail example under the Application Studio. The core file in this example is `EmailBean.class` file that needs to be included with your application.

To use this bean, an SMTP server needs to be specified to DB2 Alphablox. This can be done in the DB2 Alphablox Admin Pages, under the System link of the Administration tab.

Below is a general overview of the steps involved. For detailed step-by-step instructions on configuring and customizing the files for your application, please see the live example.

1. The first step involves copying Java class files into your application. In particular, the `EmailBean.class` file and two other support class files (`HTMLFileParser.class` and `HTMLFile.class`) need to be copied into your application's `WEB-INF\classes\alphablox\` directory. All Java classes, servlets, beans, or other utility classes need to reside in `WEB-INF\classes\`. In this case, we create a subdirectory called `alphablox\` under `classes\`.
2. The next step involves copying following files into your application directory:
   - `emailSend.jsp`
   - `emailError.jsp`
   - `emailTemplate.jsp`
   - `emailDialog.html`

   The purpose of each file is listed in the table at the end of this section. You may wish to modify or customize `emailError.jsp`, `emailTemplate.jsp` and `emailDialog.html`. Suggested modifications are included in the table.
3. There are a number of images that are part of the example implementation, as well as a style sheet. You may wish to modify and/or use these files as well. If you use the images, copy them into your application directory in an images subdirectory. The stylesheet (`styles1.css`) can be copied directly into your application directory. See the table at the end of this section.
4. There is a file included with the example called `emailExample.jsp`. You can use this file as an example of how to incorporate the e-mail functionality into your application. In `emailExample.jsp`, a JavaScript function is defined called

`openEmailDialog()`. This function invokes the email dialog. Code is also added so that when the button in the example is clicked the `openEmailDialog` function is invoked.

File	Description	Modification
`emailExample.jsp`	The JSP file that contains: • the user interface Blox to be emailed • an email link or button that triggers the email functionality	Copy the block of JavaScript code in this file that brings up `emailDialog.html` in a separate, sized browser window into your JSP file containing Blox. In your e-mail link or button, specify to call the JavaScript function copied.
`emailDialog.html`	The HTML file called by `emailExample.jsp`; contains a form for filling in sender, recipient, subject, and body of the email message. Upon form submission, the `emailSend.jsp` file is invoked with all the parameters passed via form post.	You can use it as it is or modify the title, logo, or style sheet reference for your application.
`emailSend.jsp`	The JSP file that interfaces with the Email bean to send the email.	Do not modify this file.
`emailTemplate.jsp`	The returned page informing the user the email has been sent.	You can use it as it is or modify the title or text for your application.
`emailError.jsp`	The error page for `emailSend.jsp`. If something goes wrong trying to send the email, the error information will be displayed in this page.	You can use it as it is or customize it for your environment.
`emailBlox.gif`	The “mailbox” image used in `emailExample.jsp` as the email icon.	You can use it as it is or modify.
`required.gif`	The small red arrow that indicates a required field in the email dialog.	You can use it as it is or modify.
`gridlogo-sm.gif`	Alphablox logo shown to the left of the “send e-mail” button in the email dialog.	You can use it or replace it with an image of your own.
`grid-bg.gif`	The image that is tiled to form the background of the email dialog.	You can use it or replace it with an image of your own.

File	Description	Modification
style1.css	The style sheet used by `emailDialog.html`.	You can use it as it is or modify.

Bookmarks

Bookmarks can be used to share instances of Blox views with fresh data among others within defined groups or publicly (to others with application access rights). Bookmarks allow analysts and managers to quickly share customized views of data without requiring them to wait for developers to become freed up to create custom application views. Instead, by bookmarking views and sharing them with others, all members of a group can share information.

Bookmarks can also be used for groups to share saved views that will potentially be added to applications as fully customized views.

Note: The use of bookmarks for long-term use of views is not recommended since bookmarked views maintain the use of member names that may not be valid after a few months. Also, the addition of new members to a data source may not be reflected in the bookmark view without modifications to the bookmarks.

Note: Sometimes users become concerned about bookmarks, misunderstanding what is actually being saved. When a bookmark is saved, there is no data stored. Every time a bookmark view is opened, an appropriate query is resubmitted to the server, and fresh data is retrieved. Also, a bookmarked view does not give others access to information which database security would keep from them.

For more information about bookmark functionality that you can use in developing analytic applications, see “Custom PDF report properties using <blox:pdfReport> tags” on page 240. For details about the available bookmark-related properties and methods, see Common Blox Reference and BookmarksBlox Reference sections of the *Developer's Reference*.

Printing

One of the advantages of DB2 Alphablox applications is that the data presented to users is available immediately, it is updated when the data source is updated, and can be shared without having to be printed and distributed though company mail. But, inevitably, users want to print copies for sharing with others. See other sections in this guide to learn more about how to effectively deliver analytic view through print and PDF renderings:

- “Printer format (render=printer)” on page 153
- “PDF format” on page 153
- “Printing Blox output” on page 154
- Chapter 23, “Converting to PDF,” on page 237

Chapter 22. Exporting data

Exporting is a way to output data to a spreadsheet or other formats. This topic discusses how you can create applications that support exporting grid views to Microsoft Excel or in an XML format.

Exporting to spreadsheets

By default, the toolbar includes an **Export to Excel** button and the menu bar includes a **File** > **Export to Excel** option that allows users to export the data in the grid to Microsoft Excel. To copy only selected cells into a spreadsheet, users can also select a range of cells within a grid, copy this data (via the **File** > **Copy** menu option or the **Copy** button in the toolbar), and then paste it into a spreadsheet, such as Microsoft Excel or Lotus® 1-2-3®.

Exporting to XML

Data that is exported to XML format can be used by application developers to deliver information to other applications or can be used with Java, JavaScript, and JavaServer Pages technologies to create custom views of data. The following task explains how a query result can be exported into an XML format.

Rendering result sets into XML format

Rendering a query result set from an application data source into XML format involves the following steps:

1. Define an HTML page with a standard DataBlox.

   Note: The DB2 Alphablox XML Cube can only access the result set of an explicitly-defined DataBlox. It cannot access the result set of the implicitly-defined DataBlox that underlies a PresentBlox, GridBlox, or ChartBlox.
2. Use DataBlox properties or methods to specify its data source and query string.
3. Define both the application and data source to DB2 Alphablox.
4. Invoke the application, being sure to add the render attribute to the application's URL:

   `.../AppName.jsp?`**`render=XML`**

   The value of XML for the render attribute triggers DB2 Alphablox to perform the following processing:
   - access the DB2 Alphablox XML Cube DTD (Document Type Definition)
   - render the DataBlox result set in XML (replacing the DataBlox on the page)
   - make the XML document available for further processing

Note: When using an standalone DataBlox for rendering to XML, the Blox header tag (`<blox:header/>`) is not required on your application page, and may result in the page not being displayed properly. Alternatively, instead of using the `render=xml` URL attribute, you may want to use a DataBlox with the common `render` property, setting its value to `xml`.

The next section shows the example result set from the previous page rendered into XML.

Rendering result sets into XML Format: Sample DB2 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 '/AlphabloxServer/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" span="1"
          spanIndex="0" leaf="false">East
        </member>
      </tuple>
      <tuple index="1">
        <member name="West" index="0" gen="2" span="1"
          spanIndex="0" leaf="false">West
        </member>
      </tuple>
      <tuple index="2">
        <member name="South" index="0" gen="2" span="1"
          spanIndex="0" leaf="false">South
        </member>
      </tuple>
      <tuple index="3">
        <member name="Market" index="0" gen="1" span="1"
          spanIndex="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" span="1"
          spanIndex="0" leaf="false">Audio
        </member>
      </tuple>
      <tuple index="1">
        <member name="Visual" index="0" gen="2" span="1"
          spanIndex="0" leaf="false">Visual
        </member>
      </tuple>
      <tuple index="2">
        <member name="Product" index="0" gen="1" span="1"
          spanIndex="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>
```

Chapter 23. Converting to PDF

DB2 Alphablox includes out-of-the box support for exporting analytic views within Blox to Adobe Acrobat PDF files suitable for printing, saving for later reference, or sharing with others. Users of web-based applications frequently need to save their current work, either printing or saving it for later reference or sharing with others. The Convert to PDF option provided by DB2 Alphablox offers advantages that address many of the problems frequently found when using standard web-based printing. These problems include:

- **Web browser printing.** By default, Microsoft Internet Explorer does not print background colors and images. To print a file the way it appears on the screen, Microsoft Internet Explorer users must be familiar with and check the "Print background colors and images" option on the Advanced dialog window of the Internet Options under the Tools menu.
- **Page size issues.** When a table or chart is wider than the width of the selected paper, information on the right side of the page will be lost during printing.
- **Saving for later use.** Saving web pages for later reference can be a challenge. Microsoft Internet Explorer offers the option of saving a file as a single MIME HTML (`.mht`) file, with web page images embedded within the file.
- **E-mailing reports to others.** If DB2 Alphablox users e-mail web pages to others using the browser options, recipients usually receive files with missing images, which includes any chart images. The e-mailed pages include links to temporary chart image files on the server, but these files do not exist after the sender's browser session has ended. Also, recipients may be prompted to log into systems they do not have access to

Using the DB2 Alphablox Convert to PDF option, assemblers and users have more control over their files and printing. Some of the benefits of using this option include:

- **Better page layout control.** With the assembler or user page layout options available with the Convert to PDF functionality, the layout of a saved or printed view can be more finely controlled than using standard browser controls.
- **Single file format.** A generated PDF file is a single file that can be easily be printed, saved to a hard disk, or e-mailed to other people.

Convert to PDF options

Using Convert to PDF, developers can offer users customizable reports based on analytic views presented in either a single Blox or with multiple Blox on a web page. In this section, the default user interface options are discussed, then several ways that the Convert to PDF process can be customized by you.

Default user interface options

By default, Export to PDF options are available on the menu bar and toolbar of PresentBlox, GridBlox, and ChartBlox components. When a user either selects `File > Export to PDF` on the menu bar or presses the Export to PDF button on the Blox toolbar, the user will see the default Create PDF Report Dialog window. Within this dialog, users can modify the following settings:

- Orientation: `Landscape` (default) or `Portrait`

- Page Size: Letter (default), `Legal`, `A3`, `A4`
- Theme: a selection list of themes available on the server, with a default value the same as the server's Default HTML Theme (default server theme is `coleman`)
- Header Text: blank text entry field
- Footer Text: blank text entry field

You can customize this dialog or even choose to not have the dialog appear.

Creating global default PDF report properties

Custom global default PDF report properties can defined in an optional PDF Report properties file (`pdfreport.properties`) placed in the following directory:

```
<alphabloxAnalytics_dir>/repository/theme/
```

Any settings in this file will be used by default in all DB2 Alphablox applications. An example PDF report properties file (`example_pdfreport.properties`) is available in the same directory. This example file uses the same properties that are hardcoded into DB2 Alphablox. When you add a `pdfreport.properties` file in the directory specified above, it will override the hardcoded values and use your new global default settings.

To create a default PDF report properties file, make a copy of the example file, renaming it to `pdfreport.properties` and modify the properties in that file to meet your needs. The following properties can be specified in this file:

Property
Description

`header` Header, including text and layout. Defined using XHTML (see note below table) and macros.

Available macros: Date: `<date/>` Time: `<time/>` Page count: `<totalpages/>` Current page: `<pagenumber/>` PDF Dialog Input: `<pdfDialogInputN/>` (where N is integer from 1 to 5). By default, `<pdfDialogInput1/>` defines the header and `<pdfDialogInput2/>` defines the footer.

Example:

```
header=<table border-bottom='1px' width = '100%'><tr><td valign =
'middle'><img src='/AlphabloxServer/theme/i/brand.gif'/> </td> <td
align = 'center' style='font: bold 30px Helvetica; color: #333333;'
valign='middle'> <span><pdfDialogInput1/></span></td><td align =
'right' style = 'font: 8px Helvetica; color: black;' valign =
'top'><span/></td></tr></table>
```

`footer` Footer, including text and layout. Defined using XHTML tags (see note below table) and macros.

Available macros: Date: `<date/>` Time: `<time/>` Page count: `<totalpages/>` Current page: `<pagenumber/>` PDF Dialog Input: `<pdfDialogInputN/>` (where N is integer from 1 to 5). By default, `<pdfDialogInput1/>` defines the header and `<pdfDialogInput2/>` defines the footer.

Example:

```
footer = <table border-top='1px' width='100%'><tr> <td align='left'
style='font: 8px Helvetica; color: black;' valign='bottom'
width='33%'> <span><date/> <time/></span></td> <td align = 'center'
```

```
style = 'font: bold 10px Helvetica; color: #333333;' valign =
'bottom' width = '33%'><span> <pdfDialogInput2/> </span> </td> <td
valign = 'bottom' width='34%'> <p style = 'font-
size:10;align:right;valign:bottom;'> <pagenumber/> of
<totalpages/></p></td></tr></table>
```

headerHeight
Header height. Valid units include: pixels (`px`), points (`pt`), inches (`in`), millimeters (`mm`), and centimeters (`cm`). If not specified, pixels (`px`) will be used.

Example:

```
headerHeight=50
```

footerHeight
Footer height. Valid units include: pixels (`px`), points (`pt`), inches (`in`), millimeters (`mm`), and centimeters (`cm`). If not specified, pixels (`px`) will be used.

Example:

```
footerHeight=10
```

```
footerHeight=0.5in
```

margin Margin. Valid units include: pixels (`px`), points (`pt`), inches (`in`), millimeters (`mm`), and centimeters (`cm`). If not specified, pixels (`px`) will be used.

Example:

```
margin=18
```

size Paper size, used to define paper size (`A3`, `A4`, `Letter`, `Legal`) and orientation (`landscape`, `portrait`). Valid attributes are: `[ A3 | A4 | Letter | Legal | Custom [ [Portrait | Landscape] | [width | [ height]]].` Default page size is locale-specific: In US or Canada, default is `Letter`, otherwise the page size default will be `A4`. Default orientation is `Landscape`.

Examples:

```
size=Letter Portrait
```

```
size=A4 Landscape
```

```
size=Legal
```

```
size=Custom 15in 100mm
```

`size=Custom 8in` (in this case, the default height is used)

themeListEnabled
Theme list enabled. Value can be `true` (default) or `false`.

```
themeListEnabled=true
```

pdfDialogInput1

```
pdfDialogInput1=Header Text
```

pdfDialogInput2

```
pdfDialogInput2=Footer Text
```

repeatPageFilters
Repeat page filter on pages after the first page.

```
repeatPageFilters=true
```

theme Theme name, same as theme name used in DB2 Alphablox Repository.

```
theme=my_own_theme
```

Note: XHTML tag and CSS limitations:

1. `<center>` is not supported.
2. For a non-breaking space, use the Unicode character (` `) instead of the XHTML character (` `).
3. CSS shorthand attributes should follow the W3C CSS specifications.

Using JSP tags to customize PDF reports

The DB2 Alphablox Blox Tag Library offers two custom JSP tags, `<blox:pdfReport>` and `<blox:pdfDialogInput>`, that can be used to customize PDF properties on your JSP pages.

Custom PDF report properties using <blox:pdfReport> tags

The `<blox:pdfReport>` tag can be used by developers to specify custom PDF report properties either at the Blox-level or the session-level (overriding the hardcoded PDF report properties). To set PDF properties that only affect a single Blox, add a nested `<blox:pdfReport>` tag to the Blox for which you want the properties to be applied when rendering to PDF.

To specify PDF properties that will apply to all Blox on the same JSP page, placing the `<blox:pdfReport>` tag outside of a Blox on a JSP page will result in the PDF properties being applied to all PDF dialogs for Blox on that page.

The following table describes the tag attributes that can be used in defining PDF properties with the `<blox:pdfReport>` tag:

Property
Description

footer Footer. Defined using XHTML tags (see note below table) and macros.

Available macros: Date: `<date/>` Time: `<time/>` Page count: `<totalpages/>` Current page: `<pagenumber/>`

Examples:

```
footer="<table border-top='1px' width='100%'> <tr> <td align='left'
style='font: 8px Helvetica; color: black;' valign='bottom'
width='33%'> <span><date/> <time/></span></td> <td align='center'
style='font: bold 10px Helvetica; color: #333333;' valign='bottom'
width='33%'> <span> <pdfDialogInput2/> </span> </td> <td
valign='bottom' width='34%'> <p style='font-
size:10;align:right;valign:bottom;'> <pagenumber/> of <totalpages/>
</p> </td> </tr> </table>"
```

footerHeight
Footer height. Valid units include: pixels (`px`), points (`pt`), inches (`in`), millimeters (`mm`), and centimeters (`cm`). If not specified, pixels (`px`) will be used.

Examples:

```
footerHeight="10"
```

```
footerHeight="0.5in"
```

header Header. Defined using XHTML tags (see note below table) and macros.

Available macros: Date: <date/> Time: <time/> Page count: <totalpages/> Current page: <pagenumber/>

Examples: header="<table border-bottom='1px' width = '100%'> <tr> <td valign = 'middle'> </td> <td align='center' style='font: bold 30px Helvetica; color: #333333;' valign='middle'> <pdfDialogInput1/> </td> <td align='right' style='font: 8px Helvetica; color: black;' valign='top'> </td> </tr> </table>"

headerHeight
Header height. Valid units include: pixels (px), points (pt), inches (in), millimeters (mm), and centimeters (cm). If not specified, pixels (px) is used.

Examples:

headerHeight="10"

headerHeight="1in"

margin Margin. Valid units are: pixels (px), points (pt), inches (in), millimeters (mm), and centimeters (cm). If not specified, pixels (px) will be used. Value of 1in results in pages with one-inch margins.

Examples:

margin="1in"

margin="40"

size Paper size, used to define paper size (A3, A4, Letter, Legal) and orientation (landscape or portrait). Valid attributes are: [A3 | A4 | Letter | Legal | Custom [[Portrait | Landscape] | [width | [height]]]. Default page size is locale-specific: In US or Canada, default is Letter, otherwise the page size default will be A4. Default orientation is Landscape.

Examples:

size="Letter Portrait"

size="A4 Landscape"

size="Legal"

size="Custom 15in 100mm"

size="Custom 8in" (in this case, the default page size height is used)

theme Server HTML theme defining layout styles. Value can be any predefined or custom DB2 Alphablox theme.

Example:

theme="coleman"

themeListEnabled
Theme list enabled. Value can be true (default) or false.

Example:

themeListEnabled="false"

Note: XHTML tag and CSS limitations:

1. is not supported.

2. For a non-breaking space, use the Unicode character ( ) instead of the XHTML character ( ).
3. CSS shorthand attributes should follow CSS specification.

Examples:

```
<blox:pdfReport
   size="A3 portrait"
   margin="30mm" />

---

<blox:pdfReport
   size="Letter portrait"
   margin="0"
   theme="myTheme"
   themeListEnabled="false"/>

---

<%
  String header="<span style='color:red'>This report has
     <totalpages> pages </span>";
%>

<blox:pdfReport
   header="<%=header%>"
   headerHeight"50px"
   footer="<%=some_xhtml_variable%>"
   footerHeight"1in"
```

Customizing PDF Report Dialog options using the <blox:pdfDialogInput> tag

The <blox:pdfDialogInput> tag is used to specify the input field labels and text fields to be added to the Create PDF Report dialog. It can only be used as a nested tag within a <blox:pdfReport> tag.

The following table describes the tag attributes and brief descriptions available on the <blox:pdfDialogInput> tag:

Tag Attribute
Description

index An integer from 1 to 5 that defines which of five fields to be defined.

Example:

index="5"

displayName
The label for the text.

Example:

displayName="Report Header"

defaultValue
[Optional] Default string appearing within the text field defined by the displayName attribute.

Example:

defaultValue="2004 Revenue Report"

Examples:

```
<blox:pdfReport>
   <blox:pdfDialogInput index="1"
   displayName="Report Title"
   defaultValue="My Application Name" />
</blox:pdfReport>

<blox:pdfReport>
   <blox:pdfDialogInput index="1"
      displayName="Report Title"
      defaultValue="My Application Report" />
   <blox:pdfDialogInput index="2"
      displayName="Footer"
      defaultValue="My Application Report" />
</blox:pdfReport>
```

Creating a PDF file displaying multiple Blox components

On some web pages, where you are displaying multiple Blox on a page, you may want to offer users the option of exporting all of the Blox to a single PDF file.

The following steps can be used to create a single button that users can press to generate a single PDF file from multiple Blox on the page:

1. Add any required page directives and taglib directives at the top of the page.

   ```
   <%@ taglib uri="bloxtld" prefix="blox" %>
   <%@ taglib uri="bloxuitld" prefix="bloxui" %>
   <%@ page import="com.alphablox.blox.Blox,
                    com.alphablox.blox.pdfreport.PDFReport" %>
   ```

   In this example, the standard Blox tag library and the Blox UI tag library are used to define the presentation Blox and nested titles of those Blox. The page directive gives access to the Java classes required for creating the button to be used to generate the multiple Blox on a single PDF file.

2. [Optional] Add a title to the individual Blox using a `<bloxui:title>` tag nested within the presentation Blox.

   ```
   <blox:grid id="myGridBlox"> ...
      <bloxui:title title="PresentBlox View"
         style="padding:10;font-weight:bold;"
         alignment="left" />
      ...
   </blox:grid>
   ```

   In this example, the `<bloxui:title>` tag creates a title appearing just above this PresentBlox on the JSP page. And, since this tag is nested within the PresentBlox tag, it will also appear on the PDF file. Note that any titles or other text placed on the JSP page will not appear in the PDF file.

3. Add the button for rendering multiple Blox to a single PDF file.

   ```
   <blox:container id="containerName" visible='true'>
     <%
        String bloxNames="myGridBlox,myChartBlox,myPresentBlox";
        PDFReport.addButton(containerName,"buttonName",
           "Create PDF Report",bloxRequest,bloxNames);
     %>
   </blox:container>
   ```

   In this example, the `bloxName` string defines the list of Blox that will be rendered to PDF, in the order in which they should appear in the PDF file.

4. With these additions to the JSP file, users will be able to generate a single PDF file displaying all of the presentation Blox defined above.

Specifying PDF storage locations and file names

By default, PDF files generated on your application server by DB2 Alphablox are stored only temporarily on the server. In some instances, you may find it useful to be able to specify a permanent storage location and a unique file name, allowing users or yourself to create HTML pages with links to those particular files or to mail hyperlinks to these stored documents. To specific a storage location and file name, you need to create two JSP session attributes, one for the PDF file (using PDF_FILE_NAME) and one for the directory storing your PDF files (using PDF_DIRECTORY_NAME). These attributes are optional and you can use either one or both of these session attribute settings depending on your particular application.

Using a remote PDF processor

For performance, memory management, or to share PDF processing for multiple DB2 Alphablox hosts, you may decide to run your PDF engine on an remote dedicated server. For details about configuring a remote PDF server, see the Using a Remote PDF Processor in the *Administrator's Guide*.

Chapter 24. Error handling

Unfortunately, errors happen in software problems. As a developer, though, you have some ability to manage how errors are handled. This topic includes information on how you can use Blox exceptions, properties, and methods to handle errors that may occur.

Exceptions

Although errors occur in software (hopefully, seldom), what matters to the end user is what happens after the error happens. Does the program just stop working? Or, does it recover gracefully? An exception is an event that disrupts the normal execution of a program. The Java language allows you to catch, or try to catch, exceptions that occur in order to handle them gracefully in a controlled manner. For more information about exceptions in general and how to handle them, see a good Java or JavaServer Pages reference.

Many of the Blox Java classes will throw an exception, allowing you to use the error-handling capabilities of the Java language. The Blox exceptions that are available can be found in the Javadoc documentation included in the following directory on DB2 Alphablox:

```
<db2alphablox_dir>/system/documentation/javadoc/index.html
```

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

Custom Error Pages

When the JSP Engine attempts to compile a page but fails, an error message and stack trace are generated and displayed to the end user. Most of these messages mean little to the end users and tell them little to understand what happened. As a developer, you have the option of creating custom error messages to be displayed to your users instead of the default standard JSP error page. Two page directive attributes, `errorPage` and `isErrorPage`, allow you to define where a JSP page should look for the custom error page and to define particular JSP pages as custom error pages. Brief descriptions of these attributes and the steps for using them to create your own custom error pages are included below. For more information on the use of the error page directives, see a basic JSP book.

errorPage Attribute

The `errorPage` attribute of a page directive specifies an alternate page to use as an error page, and is defined as follows:

```
<% page errorPage="/errorPage.jsp" %>
```

The `errorPage` value specifies the relative URL where a JSP page can be found within the same web application. In the example above, the value includes a forward slash (“/”) in front of the specified page. The forward slash, although not required, informs the application server that the URL that follows is relative to the root directory of your web application. By using the forward slash in this page directive, you can place a custom error page in one location in your application directory and use this same page directive in all of the application’s JSP files, even if they are located within subdirectories.

isErrorPage Attribute

A custom error page must include a page directive with an `isErrorPage` attribute set to `true`. The page directive, with its boolean `isErrorPage` value set to `true`, appears as follows:

```
<%@ page isErrorPage="true" %>
```

This directive gives the page access to information from the `exception` implicit object, and allows you to control the display of information the user sees.

Creating simple custom error pages

The following steps will guide you through the process of creating a custom error page:

1. Create a basic JSP file that will be used as your custom error page and save it as `errorPage.jsp`.
2. Add a page directive, with an `isErrorPage` attribute set to true, at the top of the page. For example:

   ```
   <%@ page isErrorPage="true" %>
   <html>
   ...
   </html>
   ```
3. Create the layout for the body of the error page, displaying what you want your end users to see if an error occurs.

   For example, you might want to display an error page heading and include the URL of the page that the error occurred on. Also, you may decide to display the top-level error message and not display the stack trace of the exception, since your users would not be likely to know how to interpret it.

   Here is a simple example:

   ```
   <%@ page isErrorPage="true" %>
   <html>
   <head>
      <title>Error Page</title>
   </head>
   <body>
   <h2>Your application has generated an error</h2>
   <h3>Please notify your help desk.</h3>
   <b>Exception:</b><br>
   <%= exception.toString() %>
   </body>
   </html>
   ```
4. To test your custom error page, add the following page directive, with the `errorPage` attribute value pointing to the location of your custom page:

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

   In this example, the custom error page resides within the same directory as the test JSP page.
5. Test your error page by generating an error.

   One way to generate an error is to include the following scriptlet, which will cause a “divide by zero” runtime error:

   ```
   <%
      int i = 10;
      i = i / 0;
   %>
   ```

Example: This custom error page example is included in the Error Handling section of the Blox Sampler example set.

Note: The various examples and the Basic Template in the Application Studio all include an error page that you can examine and copy for your own use.

Blox properties and error handling methods

The following Blox properties and methods are available to be used in customizing your applications to handle error conditions. For details on these properties and methods, see the *Developer's Reference*.

noDataMessage

The `noDataMessage` property, which defines a string that is displayed in Blox when no data is available, is one way to notify users of possible application errors. This common Blox property, applicable to the ChartBlox, GridBlox, and PresentBlox, is displayed when one of these Blox has been instantiated, but the data is not available, either because it has not yet been received or because an error has occurred. The default message is "No data available" and appears prominently in grids and charts.

If the default "No data available" message is displayed in a grid or chart for more than a few seconds, users may perceive this as an error message indicating that no data will become available. Most of the time this is a reasonable assumption, and the message should not be modified.

Sometimes data retrieval can take longer than anticipated. This could be caused by a complex query, a large data set returned, or a slow connection. In instances like these, some developers modify the `noDataMessage` string to "Please wait" or some other alternate message. Even though this is a reasonable use of this property, you should be aware that changing the displayed message can sometimes cause confusion to end users when data is actually not available. When data is actually not available, the message may still show "Please wait." If you consider changing this message, the benefit of having an initial message that more often than not is accurate may outweigh the small risk that a user will actually be told to wait when there is a real data availability issue.

Another alternative is to use the associated `setNoDataMessage` method programmatically to return a different message depending on the events that occur. While more complicated to create than just using the `noDataMessage` attribute, you may want to explore this option.

onErrorClearResultset

The DataBlox `onErrorClearResultSet` boolean property specifies whether the existing result set should be cleared from a DataBlox if a subsequent database operation fails. For more information on this property and its associated methods, see the DataBlox section of the *Developer's Reference*.

Chapter 25. Adding user help

This section discusses some of the issues involved in supplying user help in applications created with DB2 Alphablox.

In an ideal world, applications are intuitive and users don't need any help figuring out how to use them. Unfortunately, this is rare, and the more complex the application, the more likely it is that you will need to offer user help.

It is a common oversight in the design and development of applications to forget to consider adding help to your applications. If the users of your application will be frequent, skilled users, and receive training, adding help throughout an application may not be critical, but can be useful. If your users will be untrained or casual (infrequent) users, however, offering user help can be and important determinant in your application's success. Your design group should consider user help and, if necessary, schedule time and resources for help development into your application development cycle.

The following sections discuss the availability and behavior of user help in DB2 Alphablox applications and the implications of your design decisions.

Using existing DB2 Alphablox user help

When your applications include a GridBlox, ChartBlox, or PresentBlox, a toolbar can be made available for users. Depending on which Blox is being used, the Help button on the toolbar opens up the DB2 Alphablox user help system with a help page about that particular Blox. For example, clicking the Help button on the PresentBlox toolbar opens up a page titled, "Using PresentBlox," which describes PresentBlox and has additional links for further help.

Alternatively, if a toolbar is not available, selecting **Help** from the Help menu in the menu bar brings up a help page for the user interface Blox, depending on if it is a PresentBlox or a standalone GridBlox or ChartBlox. For example, on a standalone GridBlox, selecting the **Help** menu option brings up a help page, titled "Using the Grid."

Frequently, you may decide not to include a toolbar on analytic views in your applications. In these cases, you should make sure the menu bar is available. By default, the menu bar is available in these user interface Blox. If for any reason you need to turn off both the menu bar and the toolbar, you may want to consider offering custom user help.

Creating custom user help

If you decide not to make the toolbar available on your analytic views, or you decide you want to provide target custom user help for your users, you can add appropriate help links or buttons. In particular, you may want to consider offering custom user help in the following situations:

- neither the toolbar or the menu bar will be unavailable
- the toolbar or menu bar are customized with custom buttons and menu options

- your pages make use of custom HTML form elements instead of built-in Blox user interface elements (such as page filters and toolbar buttons) to manage the interaction and analysis of Blox views
- your views include members or other labels that would benefit from a glossary or other help information

If the toolbar and menu bar are customized, you may need to customize the existing user help as well. The help files are located in the documentation directory:

```
<db2alphablox_dir>/system/documentation/help/dhtml
```

Before modifying the files, you should first make a copy of the directory. Also keep in mind that files in this directory will be removed and replaced with DB2 Alphablox user help files when you upgrade the server.

If you turn off the toolbar and the menu bar entirely, besides using standard HTML technology to provide custom user help, you may also want to consider using DB2 Alphablox information links, discussed in the next section, as a way to provide targeted and visible help information.

Using information links for help

As discussed in "Information links" on page 175, there are three types of information links available on Blox: header links, cell links, and cell alert links. These links (by default, represented with a white "i" within a blue circle) can be used for many purposes, including linking to information relevant to the row or column headers, or on specified data cells. Keep in mind that these links can also be used for targeted user help, perhaps defining what a particular member represents, or how a particular data cell should be evaluated. One of the benefits of information links is that they are highly visible and hard to ignore. Of course, that can also be a reason to not include them, or at least to use them judiciously.

Chapter 26. Working with DB2 Alphablox FastForward

The DB2 Alphablox FastForward application framework allows application administrators (OLAP administrators) to copy the framework, configure report templates, and quickly deploy analytic applications to line of business users. This section includes an overview of the FastForward framework and new report templates can be added to customize the application framework.

Note: DB2 Alphablox FastForward is not available when DB2 Alphablox is installed to run on a WebSphere Portal server.

DB2 Alphablox FastForward overview

DB2 Alphablox FastForward is a sample application framework, preinstalled on DB2 Alphablox, for quickly developing, deploying, and sharing custom analytic views throughout business organizations. Out-of-the-box, the FastForward framework delivers common application services, including security, collaboration, customization, and personalization. Application administrators, typically OLAP administrators, can create new versions of an FastForward application, publish reports by selecting report templates and configuring report parameters, and then deploy the new application without ever looking at code. And, because of its flexibility and extensibility, JSP developers can modify or extend the application framework, and add new custom report templates for application administrators to configure and deploy.

Built into the FastForward application framework are features commonly found in reporting and analytic applications, including:

- exporting to Microsoft Excel
- generation of printable views
- easy saving and sharing of personal views of data
- e-mailing views to others
- easy navigation between different views

By including these features, DB2 Alphablox makes it easy for you to expedite the development of this class of commonly-used reporting and analytic applications. You don't need to create navigation systems, toolbars, security, and mechanisms for the saving and sharing of reports as these have already been pre-coded for you. By separating out the development and configuration tasks, application administrators can focus on configuring and deploying existing report templates, letting you focus your attention on the more challenging requirements.

Roles of FastForward users

The three major roles of DB2 Alphablox FastForward users include those of application administrators, template developers, and end users. A good synergy between these three groups will help ensure the success of FastForward-based applications. More about these three roles are briefly described below.

Application administrators

Application administrators, typically OLAP administrators, should be able to create new versions of FastForward applications by defining a few settings, create reports

based on the available report templates, then quickly deploy solutions to end users. If end user requirements cannot be met using an existing report template, the application administrator works together with template developers to create new report templates. An application administrator should be able to accomplish their work using their OLAP database experience, the documentation on administering Alphablox FastForward applications (see the *Administrator's Guide*), and the online Administration Help (available in the Admin Tasks mode of a FastForward application).

Template developers

Template developers are typically JSP developers primarily responsible for creating custom report templates when existing ones cannot be used by an application administrator to configure requested reports. In consultation with application administrators and end users, template developers should be able to create new report templates by modifying existing report templates or creating new ones as necessary.

Using the Blox tag libraries, server-side Java API, and DHTML Client API, as well as your web programming experience, template developers should be able to create templates for almost every conceivable need. Besides being familiar with building DB2 Alphablox applications and views, developers should also be familiar with the FastForward User Help (available from the Help button in user mode), the Administrator Help (available from the Help button in Admin Tasks mode, when logged in as an administrator), and Administering FastForward Applications in the *Administrator's Guide*.

End users

End users, typically business analysts and other line of business users in your organization, should be able to log into a FastForward application and use published reports to analyze business issues. Depending on the interactivity available in a particular FastForward-based application, end users can manipulate data, drill around data hierarchies, change chart types, add comments, and more. After modifying views to answer particular business questions, users can preserve their current views, creating saved reports under the Private tab for later use or by sharing them under the Groups tab to defined groups of application users.

For each report, users typically have a few other options available from the application toolbar, located above the reports. Besides saving reports for online analysis, the export to Excel option allows users to export views to Microsoft Excel spreadsheets for offline analysis at a later time. Users can also print a copy of a particular view using the Print Preview option. And, if desired, they can open an email message containing a link to the current view, add comments, and send it to other application users.

If necessary reports are not available in their applications, end users typically request new reports directly from application administrators.

Customizing Alphablox FastForward

Although DB2 Alphablox FastForward includes sample report templates that can be used out-of-the-box to start creating applications immediately, most likely your reporting and analytic applications will require you to create custom report templates for application administrators. To help you get started in working with DB2 Alphablox FastForward applications, this section includes overviews of the FastForward architecture and report templates. You'll then learn how to create new report templates.

FastForward application architecture

A FastForward application consists of two major components, the framework and the report templates used within that framework. The FastForward framework includes JSP pages, JavaBeans components, and many other files that define an application framework, including common application services such as application configuration, navigation, and security.

The sample DB2 Alphablox FastForward application is located in the following directory:

```
<alphabloxDirectory>/system/ApplicationStudio/FastForward/
```

Important: Do not delete or modify this directory - it contains the files used in the sample application. When upgrading DB2 Alphablox, this directory will be overwritten.

When logged in as a DB2 Alphablox administrator (with an `AlphabloxAdministrator` or other defined administrator role), you can create new FastForward applications by clicking on the Admin Tasks button, then create a new version of the sample application by clicking on Create in the dialog box. When you click on the Create option, a dialog window will prompt you to define your new application's context name (directory name), display name (which appears on the Applications page), a brief application description, and the Administrator role allowed to edit the application.

Note: When a new FastForward application is created, an administrator role is assigned. Only users who are members of that assigned role (default is `AlphabloxAdministrator`) can administer that particular application.

After clicking OK, a new J2EE application is automatically created and a copy of the FastForward application framework and sample report templates are copied to that application's directory, typically located here:

```
<alphabloxDirectory>/webapps/<applicationDirectory>
```

Note: For details about configuring FastForward applications, see Administering Alphablox FastForward Applications in the *Administrator's Guide*.

When you create a copy of a FastForward application, the following directories and files are added to the web application you created. Following is a summary of the directory structure and files that included in the FastForward framework

Directory
Description

admin Includes most of the files used when a FastForward application is used in Admin mode, including the admin help directory.

admin/help
Help files specific to application administration

admin/images
Contains images used in Admin mode.

help Contains user help files. Can be customized by application developers as needed.

images Contains image files used in User mode.

templates
Includes the collection of report templates available to the application. Subcomponents of this directory are described below in the Report Templates section.

WEB-INF
The standard J2EE application directory that contains files required to run the application.

WEB-INF/classes
Includes compiled Java class files for the JavaBeans components used in the application.

WEB-INF/src
Source files for the JavaBeans components, allowing you to customize or extend as needed.

WEB-INF/tlds
Copies of the Blox tag library descriptor files, including `blox.tld`, `bloxform.tld`, `bloxlogic.tld`, `bloxreport.tld`, and `bloxui.tld`.

WEB-INF/ui
Includes XML files used to define the look-and-feel of the application, including `buttons.xml`, `toolbar.xml`, and `toolbarhelp.xml`. These files can be edited to add or remove buttons, change button images, and tooltip descriptions as needed. The remaining XML files should not be changed by application developers.

For most applications, you will be primarily interested in the `templates` directory, where the report templates are stored. The next section describes report templates and the templates directory in more detail.

Report templates

Report templates are the heart of a FastForward application, supplying the application administrator with the edit pages used to quickly configure reports for end users. A report template consists minimally of four files: the edit page, the template parameters file, the report page, and the help page. The **edit page** is the page used by application administrators to create a new report by selecting from selection lists, radio buttons, and checkboxes representing to define report and data options. The options, or parameters, available in the edit page are defined in the **template parameters file**. What the end user sees is determined by the layout of the **report page**. A **help page**, tied to a particular report, is also included. Optionally, print and Excel pages may be included in report templates. New report templates can be created by template developers, adding to the collection of report templates available and reducing the need for custom report development of frequently used classes of reports. And, as a result, end users can access, modify, and save copies of reports for later reuse and sharing.

For each FastForward application, the collection of report templates available to that application are located in the following directory:

```
<applicationDirectory>/templates/
```

Inside of the `templates` directory are a collection of report template subdirectories, each of which makes up a self-contained report template. When a new version of a FastForward application is created, sample report templates are copied and included in this directory. New report templates can also be added, on-the-fly, by simply dropping the template into the directory containing the set of available

report templates. Besides the sample report templates included with DB2 Alphablox, others will be downloadable from Alphablox, shared among template developers in your organization, or created by third-party developers. When a new report template is added to the `templates` directory, it becomes immediately available to application administrators in their list of available report templates. Also, zipped copies of report templates dropped into the `templates` directory are automatically uncompressed and made available in the menu. You can continue to add new report templates as needed, making them quickly available to application administrators without having to stop and start the server.

Inside of each report template directory are at least the following three required files, summarized here:

File Name
Description

`edit.jsp`
The **edit page**, viewed and configured by administrators to create new reports or to edit existing reports.

`template.xml`
The **template parameters file** that includes a list of all of the parameters (properties) that can be set by the application administrator.

`report.jsp`
The **report page**, which when combined with the values defined by administrators, generates the report that users can view and manipulate.

The following template files are optional:

File Name
Description

help.jsp
The **help page** provides useful information to users about this particular report. The sample templates include a help page prototype.

excel.jsp
The **Excel page** providing customization of the HTML page sent to Microsoft Excel.

print.jsp
The **print page** provides customization of the HTML page that is made available for printing.

The complexity of a report template is determined by what users expect to see (displayed in the `report.jsp` file), by the number and types of parameters (specified in the `template.xml` file), and the user interface for configuring those parameters (generated using the `edit.jsp`). With simple report templates, a limited number of parameter settings may be possible, with many of the setting predefined. More flexible templates offer more configuration options, but also require a larger number of parameters to be made available.

Following the description of the sample report templates that ship with DB2 Alphablox, the section, “Creating custom report templates” on page 256, will cover more details about these three files in the context of describing the steps to create new report templates.

Sample report templates

The sample report templates that ship with DB2 Alphablox include a variety of commonly used report types, typically encountered in most businesses. Although some of these samples may be useful as they exist, these templates can also be used to help you learn how to develop your own custom report templates.

The sample templates included with DB2 Alphablox target different types of reporting needs and are summarized here:

Sample Template	Directory Name	Description
Interactive Present Blox	`InteractiveBlox`	Sample reports: Interactive Analysis
Sample Allocation	`SampleAllocation`	Sample reports: Sales by Store (Sales Analysis)
Sample Report	`SampleReport`	Sample reports: Sample Report (Sales Analysis)
Sample Trending	`SampleTrending`	Sample reports: Sales Trend by Region (Sales Analysis)
Sales Variance	`SampleVariance`	Sample reports: Sales Variance (Variance Analysis)
Interactive Variance	`VarianceQCC`	Sample reports: Ad-Hoc Variance Analysis (Variance Analysis)

The sample templates will not address all of your particular user needs, but are useful for giving a jump start in learning about the power and flexibility of report templates as well as serving as great examples for learning how to code solutions for particular problems. While it is possible for you to deploy a copy of the sample Alphablox FastForward application with little modification, the power of the application framework and report templates is realized as you begin adding custom report templates tailored to your unique user needs. The next section explains how to create a simple report template.

Creating custom report templates

To help you get started in developing your own report templates, this section walks you through the most important steps you need to know to begin creating custom report templates. As described earlier, each report template contains three important files, the report page (`report.jsp`), the template parameters file (`template.xml`), and the edit page (`edit.jsp`). The following steps describe how to create each of these required files for a simple allocation report template.

Creating or modifying the report page (report.jsp)

The report page (`report.jsp`) generates the view that FastForward application users will see, after the report has been configured by an application administrator. This page typically includes the following information: report title, data views, and user controls. Report pages range from static viewing pages to guided analysis reports and at the high-end may include many user controls.

To create a report page, start by creating a representative JSP page that includes the functionality that your end users will be using.

In this example, the report loads with a title and basic pie chart view showing percentages of subcategories of specified product grouping.

To create a report template that can be used to generate this view, you need to create parameters that will be read in to generate the report. These parameters, which will be defined later in the template parameters file (`template.xml`), include the properties that an application administrator will be able to set when using the edit page (`edit.jsp`) to configure an application. You'll learn how to create the edit page after we've finished creating the report page and template parameters file.

Taking the report prototype file, we add the following line to the top of your file:

```
<%@ include file="../../reportdata.jsp" %>
```

This JSP `include` directive results in the contents of the `reportdata.jsp`, located in the template's root directory, being added to the report page when it is compiled. The `reportdata.jsp` file imports necessary class files, adds `taglib` directives for `bloxtld` and `bloxformtld`, and makes the following objects available for use inside the `report.jsp`:

Object Purpose

report Provides access to report parameters (or properties)

user Provides access to user parameters

appContext
Provides access to application parameters

template
Provides access to the template and its parameters

savedState
Provides access to the saved private and group reports

These objects implement the following methods:

Method
`String getParameter(String name)` - Returns the parameter value of the named parameter or null. Returns null if the named parameter is not defined.
`String getParameter(String name, String default)` - Returns the parameter value of the named parameter or the `defaultValue` argument. Returns the given `defaultValue` if the named parameter is not defined.
`String[] getParameterValues(String name)` - Returns the parameter values defined for the named parameter. Returns an empty array if no parameter values are defined.
`String[] getParameterNames()` - Returns an array containing all of the named parameters defined for this object.

To parameterize your report prototype, you now need to define a JSP scriptlet at the top of the report page specifying the parameters you want to use. You need to

specify the pie slice parent member and the measure to be used, then generate a query string that reads in these two parameters, like this for a DB2 OLAP Server or Essbase data source:

```
<%
  String pieSliceParent =
     report.getParameter("pieSliceParent","Specialties");
  String measure = user.getParameter("preferredMeasure", "Sales");
  String query = "&lt;SYM &lt;COLUMN (\"Measures\") \""+measure+"\"
     &lt;ROW (\"All Products\") &lt;CHILD \""+pieSliceParent+"\" !";
%>
```

If you are using Microsoft Analysis Services, the third line, specifying the query string, would look like this:

```
String query = "SELECT DISTINCT( {[Measures].["+measure+"]} ) ON COLUMNS,
{"+pieSliceParent+".children} ON ROWS FROM [qcc]";
```

Note that the `getParameter` method used in this example allows you to set default values for both properties using an optional second argument, following the parameter name.

Next, substitute a JSP expression in each place on the page that you want to have a parameter value appear. In the DataBlox tag, you need to read in a user's query using `<%= query %>`:

```
<blox:data id="dataBlox"
  dataSourceName="QCC-Essbase"
  useAliases="true"
  query="<%= query %>"/>
```

You also need to add two more JSP expressions, `<%= measure %>` and `<%= pieSliceParent %>`, to substitute the `measure` and `pieSliceParent` values into the report title, like this:

```
<h3>Comparing <%=measure%> for subcategories of <%= pieSliceParent %> </h3>
```

The result will be a title that is appropriate for the pie chart displayed.

After you've added the JSP `include` directive at the top of the page and the JSP expressions that will read in the saved values, the complete `report.jsp` file should look similar to this:

```
<%@ include file="../../reportdata.jsp" %>
<%@ taglib uri="bloxlogictld" prefix="bloxlogic"%>
<%@ taglib uri="bloxuitld" prefix="bloxui"%>

<%
  String pieSliceParent =
     report.getParameter("pieSliceParent","Specialties");
  String measure = user.getParameter("preferredMeasure", "Sales");
  String query = "<SYM <COLUMN (\"Measures\") \""+measure+"\"
     <ROW (\"All Products\") <CHILD \""+pieSliceParent+"\" !";
%>

<blox:header/>
<body>
<blox:data id="dataBlox"
  dataSourceName="QCC-Essbase"
  useAliases="true"
  query="<%=query%>"/>

<h3>Comparing <%=measure%> for product code subcategories:
```

```
   <%=pieSliceParent%></h3>

<blox:chart id="chartBlox"
  bloxName="chart"
  height="100%"
  width="100%"
  chartType="Pie"
  totalsFilter="0" >
  <blox:data bloxRef="dataBlox" />
</blox:chart>
</body>
```

Creating or modifying the template parameters file (template.xml)

Next, you need to define the parameters, or properties, that a FastForward application administrator will be configuring. To do this, you need to create or modify an example `template.xml` file, including only the parameters required for the report template. This should be a relatively quick process.

At the top of the `template.xml` file is the required DTD specification for this XML file:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE template PUBLIC "-//Sun Microsystems, Inc.//DTD
   Web Application 2.2//EN" "../template.dtd">
```

Following this specification, the `template` element is included, incorporating the parameters for this particular template. The following table lists the available nested elements of the `template` element:

Element
: **Description**

`display-name`
: Contains the name to be displayed in the selection list of report template choices in the edit page. An optional **`lang`** attribute can be added defined using the standard language codes and country subcodes (for example, `en-GB` or `fr`).

`description`
: [Optional] Brief report description that will appear in the edit page. An optional **`lang`** attribute can be added defined using the standard language codes and country subcodes (for example, `en-GB` or `fr`).

`report-page`
: Specifies the report page that will be used to generate the report

`edit-page`
: Specifies the file name for the edit page that creates the view used by application administrators to define the report

`report-params`
: [Optional] Defines the collection of report parameters that can be set by the administrator. Each `param` element is nested within this element.

`param` [Optional] Specifies that the contents of this element define a parameter.

`param-name`
: [Optional] Nested within the `param` element, this tag specifies the name of the parameter used in coding templates.

param-label
[Optional] Nested within the param element, this tag specifies the display name for a parameter, and is seen by application administrators in the edit page of a report template. An optional **lang** attribute can be added defined using the standard language codes and country subcodes (for example, en-GB or fr).

default-value
[Optional] Default value for the parameter. When a report doesn't supply a value, this value will be used.

print-page
[Optional] Specifies the print page used in a report

excel-page
[Optional] Specifies the export to Excel page used in a report

help-page
[Optional] Specifies the help page used in a report

Note: The template parameters must be defined in the order should appear in the edit page. Also, the edit page and report page file names can be anything reasonable. The file names used in the example below follow the practice used in the sample report templates included with FastForward, and make a reasonable naming practice you may want to continue following.

In this example, you need to modify the display-name, description, and report-params elements. The report-page and edit-page elements define the actual file names for those pages.

Following with this example, here is the contents of the entire template.xml file, with the defined parameters:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE template PUBLIC "-//Sun Microsystems, Inc.//DTD
    Web Application 2.2//EN" "../template.dtd">

<template>
   <display-name>Allocation (Essbase Version)</display-name>
   <description>First Allocation Template (Essbase Version)
   </description>
   <report-page>report.jsp</report-page>
   <edit-page>edit.jsp</edit-page>
   <report-params>
      <param>
         <param-name>pieSliceParent</param-name>
         <param-label>Parent Member of Pie Slices:</param-label>
      </param>
   </report-params>
</template>
```

Note that the report-params element defines the collection of report parameters that will be used in this template. The nested param element above, pieSliceParent, defines the parameter for specifying the parent member of the pie slices to be displayed. After the report page and the template parameters file have been created, your final task is to create the edit page, which displays the selectable options for application administrators to configure.

Creating or modifying the edit page (edit.jsp)

The edit page is the page that application administrators use to define the report parameters, or properties, that typically appear in selection lists, radio buttons, and checkboxes. The edit pages in the sample report templates included in the sample FastForward application are more complex than the edit page required for this example, but the same essential steps are involved.

Let's begin creating the edit page. At the top of the `edit.jsp` file, add a JSP `page` directive that specifies any of the required classes that need to be imported:

```
<%@ page import="com.alphablox.blox.form.FormEventListener,
     com.alphablox.blox.DataBlox,
     com.alphablox.blox.form.TimePeriodSelectFormBlox,
     com.alphablox.blox.logic.timeschema.TimeSchemaBlox,
     com.alphablox.blox.form.FormEvent,
     com.alphablox.blox.ServerBloxException,
     fastforward.*,
     com.alphablox.blox.form.MemberSelectFormBlox,
     com.alphablox.blox.data.mdb.Member" %>
```

Next, add JSP `taglib` directives to access Blox tag libraries used on the page:

```
<%@ taglib uri="bloxtld" prefix="blox" %>
<%@ taglib uri="bloxformtld" prefix="bloxform" %>
<%@ taglib uri="bloxlogictld" prefix="bloxlogic" %>
```

And, required on all JSP pages that access the Blox tags is the `<blox:header>` tag:

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

Next, specify the DataBlox that the edit page will use to generate the selection list options. The following `<blox:data>` tag defines a DataBlox which is preconfigured to use the `QCC-Essbase` data source (which should already be specified in the DB2 Alphablox applications definition page):

```
<blox:data id="dataBlox"
     useAliases="true"
     dataSourceName="QCC-Essbase" />
```

Additionally, specify any tag attributes you might need. In this example, the `useAliases` tag attribute value of `true` tells the server that you want to see the display member names, not the unique member names, from the defined DB2 OLAP Server or Hyperion Essbase data source. If you are using Microsoft Analysis Services, the data source name for this example would be `QCC-MSAS`, and you wouldn't add the `useAliases` tag attribute since it applies only to DB2 OLAP Server and Essbase data sources.

Next, specify the MemberSelectFormBlox with an id of `pieSliceParent` to generate the selection list with the pie slice parent options returned from the data source:

```
<formblox:memberSelect id="pieSliceParent"
     visible="false"
     dataBloxRef="dataBlox"
     dimensionName="All Products"/>
```

Note: A couple of important points about using FormBlox here:

- FormBlox defined in the `edit.jsp` page must have `id` attribute names identically matching parameter names used in the `template.xml` file. In the example here, note that the MemberSelectFormBlox `id` is `pieSliceParent`, matching the `pieSliceParent` parameter defined in the `template.xml` file.

- Remember to set the `visible` tag attribute to `false` in order to prevent the Blox from rendering before any processing logic is done. After the processing logic is finished, in this example, the `renderControls` method of the `TemplateHelper` class below will render the Blox on the page. If you forget to add this visible attribute (`visible="false"`), or if you accidentally set it to `true`, you will unexpectedly see duplicate Blox on a page.

Now you are finished defining the selection list that will appear in the edit page and can build the page.

The following JSP scriptlet generates the page to be displayed, applying previously saved parameter values and rendering the page controls (previously set to not be visible). It also establishes a validator that will be used to ensure that administrators enter the expected information. The validation step is optional, but enhances the robustness of your application, helping to ensure that users enter expected values:

```
<%
   TemplateHelper.applySavedParameters(pageContext);
   TemplateHelper.renderControls(pageContext);
   Template template=(Template)request.getAttribute("template");
   template.setValidator(new Validator());
%>
```

Next, the `Validator` is defined, which implements `ReportValidator`. Implementing `ReportValidator` requires the class to define one function, `validate(ReportData data)`, which does the critical work. The validator gives access to the defined parameters the same way the report gets access to the parameters -- by calling `getParameterValue()` on the data object for any parameters that need to be checked.

In this example, a check verifies that administrators do not select a `pieSliceParent` that is a leaf member (i.e., it has no children). Also, an appropriate error message is added, appearing if a user attempts to use a disallowed value.

```
<%!
  public class Validator implements ReportValidator {
    public void validate(ReportData data) throws ServerBloxException {
      String pieSliceParent=data.getParameterValue("pieSliceParent");
        if (pieSliceParent == null){
            data.addError("Please select a member from the products.");
               return;
            }
// There is a FormBlox associated with pieSliceParent
// Use this FormBlox to get the selected member object and validate it
     MemberSelectFormBlox select =
       (MemberSelectFormBlox)data.getFormBlox("pieSliceParent");
     Member members[] = select.getSelectedMembers();
// there is only one selected member -- it cannot be a leaf
     if (members[0].isLeaf() == true) {
          data.addError("The selected member must have some children");
     }
   }
  }
%>
```

The edit page is now complete. Here is a complete copy of the entire `edit.jsp` file for reference:

```
<%@ page import="com.alphablox.blox.form.FormEventListener,
                 com.alphablox.blox.DataBlox,
                 com.alphablox.blox.form.TimePeriodSelectFormBlox,
                 com.alphablox.blox.logic.timeschema.TimeSchemaBlox,
```

```
                com.alphablox.blox.form.FormEvent,
                com.alphablox.blox.ServerBloxException, fastforward.*,
                com.alphablox.blox.form.MemberSelectFormBlox,
                com.alphablox.blox.data.mdb.Member"%>

<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxformtld" prefix="formblox"%>
<%@ taglib uri="bloxlogictld" prefix="bloxlogic"%>

<blox:header />

<blox:data id="dataBlox"
   useAliases="true"
   dataSourceName="QCC-Essbase" />

<formblox:memberSelect id="pieSliceParent"
   visible="false"
   dataBloxRef="dataBlox"
   dimensionName="All Products" />

<%
    TemplateHelper.applySavedParameters(pageContext);
    TemplateHelper.renderControls(pageContext);
    Template template=(Template)request.getAttribute("template");
    template.setValidator(new Validator());
%>

<%!
    public class Validator implements ReportValidator {
      public void validate(ReportData data) throws ServerBloxException {
          String pieSliceParent=data.getParameterValue("pieSliceParent");
          if (pieSliceParent == null){
             data.addError("Please select a member from the products.");
             return;
          }
          // There is a FormBlox associated with pieSliceParent
          // You can use this FormBlox to get the selected member
          // object and validate it,

          MemberSelectFormBlox select =
             (MemberSelectFormBlox)data.getFormBlox("pieSliceParent");
          Member members[] = select.getSelectedMembers();
          // there is only one selected member -- it should not be a leaf
          if (members[0].isLeaf() == true) {
             data.addError("The selected member must have some children");
          }
       }
    }
%>
```

The edit page was the final page that you needed to create to have a working template. After you've finished this simple template, you can use the edit pages in the sample report templates to help you when you begin building more complex templates. Before moving on to more ambitious report templates, though, you should test the report template you just created.

Creating optional template pages

As described earlier, a template can include a help page, a print page, and an Excel page. These can be customized to meet your specific application requirements.

For each report template created, it is recommended that you include a help page tied to the usage of the report made available to users. The sample report

templates include a link, called "Report Help," pointing to an example report help page. The file name of the help page is specified in the `help-page` parameter of the `template.xml` file.

In a FastForward application, the Print Preview button located on the application toolbar results in the current report being rendered with the `render` URL attribute set to `printer`. For details about the Printer format, see "Printer format (render=printer)" on page 153. Other details about creating custom print pages can be found in "Printing with HTML-based printing" on page 155.

Also on the applications toolbar is the Export to Excel option for FastForward reports. This option generates results in the current report being rendered with the `render` URL attribute set to `xls`. For details about the Excel format, see "Export To Excel format (render=xls)" on page 154.

Localizing FastForward applications

DB2 Alphablox support localization globally at the server level. As a result, Blox are localized according to the locale of the server and not based on incoming user requests. All strings in JSP files or in Java classes are extracted from a message bundle file, `FastForwardBundle_<lang>.properties`, where `<lang>` language code, including any country subcodes (for example, `en-GB` or `fr`). All JSP example files included with FastForward applications use the standard internationalization (i18n) tag library, available from the Apache project. As described earlier in the Template Parameters File section, an optional `lang` attribute can be applied to the `display-name`, `description`, and `param-label` elements. Multiple instances of these elements can be used to support multiple languages.

Testing report templates

Assuming you've correctly created the report page (`report.jsp`), parameters definition page (`template.xml`), and the edit page (`edit.jsp`), you now have a simple report template that can be used in your FastForward application's administration pages. To test the template, place this entire template in the `templates` directory of a FastForward test application. Next, open your application in Microsoft Internet Explorer, then click on the Admin Tasks button. Go ahead and create a new report, select your new template from the list of available report templates, and see how it works. You can preview the report while on edit page by clicking the Preview button, or click Return to Application to test it as an end user.

Saving report templates

To use a report template, all of the template files you have created (including `report.jsp`, `template.xml`, and `edit.jsp`) should be stored in a subdirectory of the application's `template` directory. The name of the template displayed in the edit page is read from the `display-name` element defined in the `template.xml` file.

Sharing report templates

Now that you have a working template, keep in mind that you may want to share your handiwork with other template developers. An exchange of templates can be useful for meeting the needs of other application users, or may become good learning examples for others. Remember that if you zip up your template directory and pass it on to others, they can just drop the zip file in their FastForward application's template directory and begin allowing the application administrator

to begin using it immediately. New report templates will appear after a new browser session opens the FastForward application.

Saving state using the savedState Object

When a reports are created by an administrator and made available to users, they become ″published″ reports, which are simply JSP pages with defined parameters. When a report is saved by a user for either private or group access, FastForward attempts to save the report so that it can be restored later with the user's current changes. Here is a summary of how this works:

1. When the user clicks the Save Report button, FastForward stores the following information:
   a. the name of the template and report template that the new saved report is based on
   b. the parameters associated with that report
   c. the `FormValues` property of every FormBlox on the page
   d. a bookmark for every bookmarkable Blox (GridBlox, ChartBlox, PresentBlox, and DataBlox) on the page
2. When the saved report is reloaded again later by the user, the report becomes reconstituted in essentially the same order:
   a. the page is loaded and compiled based on the saved parameters
   b. call `setFormValues()` with the saved `FormValues` property
   c. call the `restoreBookmark()` method on each Blox with a saved bookmark

This is usually acceptable for most situations. Sometimes, though, you may want to deviate from this standard restore procedure. For example, you may want to change the data shown in the report so that it is based on the current calendar day/quarter/month or, in the case of group access reports, you may want to load a personalized report, with certain values being set based on which user loads a report.

In order to make this easier, FastForward provides the `savedState` object, which is available from the JSP page whenever the report has been restored from private or group access. If a report has been "published," the `savedState` object is unavailable and all references to it will return `null`.

The `savedState` object provides the following capabilities:

- the ability to turn off the default restore behavior
- the ability to get any FormBlox or other standard Blox on the page
- the ability to get a bookmark associated with any bookmarkable Blox
- the ability to restore the state of the various Blox in any order desired.

For details on these and other capabilities of the `savedState` object, refer to the FastForward Javadoc documentation, available from the Help menu in the DB2 Alphablox Admin Pages.

In order to use this object in creating new reports, you need to add your custom report restore logic to the end of the JSP file, thus applying your logic after the affected Blox are instantiated but before they are rendered on the page.

In the following example, a GridBlox is modified on the restored report to have row banding disabled on the grid:

```
<blox:present id="myBlox" visible="false"
   width="100%" height="100%"
   dataLayoutAvailable="<%=dataLayoutVisible%>"
   menubarVisible="<%= menubarVisible %>">
   <blox:grid visible="<%=gridVisible%>"
      bandingEnabled="<%=gridBanding%>"/>
   <blox:chart visible="<%=chartVisible%>"
      chartType="<%=chartType%>"
      totalsFilter="0"/>
   <blox:data bloxRef="dataBlox" />
   <blox:toolbar visible="<%=toolbarVisible%>"
      removeButton="Save,Load" />
   combineToolbars(myBlox.getBloxModel()); %>
</blox:present>
<%
   if (savedState != null) { Bookmark bookmark =
   savedState.getBloxBookmark("myBlox");
   BookmarkProperties gridProps =
     bookmark.getBookmarkPropertiesByType(Bookmark.GRID_BLOX_TYPE);
     gridProps.setProperty("bandingEnabled", "false");   }
%>
<blox:display bloxRef="myBlox"/>
```

Next steps

Once you have mastered some simple report templates, you can move onto more challenging ones. The sample report templates included in the sample Alphablox FastForward application are a fertile source of ideas and code for you to use in constructing your own templates.

Also, use all of the available developer resources, including the *Developer's Reference*, this guide, the *FastForward API Javadoc* documentation, and the *Blox API Javadoc* documentation. These are also available from the Help menu on the DB2 Alphablox Admin Pages.

Chapter 27. DHTML client DOM API

Developers should not write client-side code that manipulates or traverses the DOM generated by the DHTML DB2 Alphablox client unless using the published DHTML Client DOM API described in the flowing sections, as the implementation will likely change going forward.

GridBlox Client API

The following topics describe the client APIs available on GridBlox components when displayed on JSP pages.

Blox definition

```
<blox:grid id="myBlox"
  width="80%"
  height="30%"
  bandingEnabled="true"
  visible="false">
  <blox:data bloxRef="dataBlox" />
</blox:grid>
```

The DHTML client will return a JavaScript object in the document namespace.

To get a reference to the JavaScript object for a Blox, use:

```
var gridBlox = document.myBlox;
```

or

```
var gridBlox = myBlox;
```

Grids

The GridBlox provides access to the grids contained within it using a zero-based grids array.

To get a reference to a grid contained in a GridBlox defined earlier, use:

```
var myGrid = myBlox.grids[n];
```

where `n` is the zero-based number of the grid.

This returns the element that represents the grid. In addition, the grid contains two attributes for the total number of rows and columns.

To return the number of scrollable rows or collumns in a grid:

```
var numRows=myGrid.getRowCount(); var numCols=myGrid.getColumnCount();
```

These values represent the total number of scrollable elements and do not distinguish between header and data rows or columns, if they are scrollable. If the number of scrollable elements does not fill the available area, then scrolling is not required. If the size of the grid area changes, scroll bars will be automatically added or removed as needed.

To scroll the grid to the indicated row and column, use the `scrollTo` method:

```
myGrid.scrollTo(row,column)
```

The `scrollTo` method scrolls the grid to the indicated row and column.

To determine if scrolling is enabled for the grid:

```
var enabled = myGrid.isScrollingEnabled()
```

The `isScrollingEnabled` method returns `true` if scrolling is enabled for the grid.

Selection

The grid provides end users with the ability to select one or more cells. They may then perform an action on the selected cells. The DHTML client provide programmatic access to those selected cells using the following methods.

Selection object

To access the selection object representing the cells that are currently selected in that grid, use:

```
var myGrid=myBlox.grids[0]; var select = myGrid.selection;
```

Retrieving visible selected cell ID values

The selection object provides a method for retrieving a zero based array of strings where each string is the ID of a cell currently selected in the associated grid.

```
var selectedCellIds = select.getCellIds();
```

This array will only contain the IDs of the cells that are both selected and currently visible. If you require the full set of selected cells you should access the model instead.

To determine if a cell is selected, use:

```
var selected = selection.isSelected( cellID)
```

Returns `true` if the cell is selected. Selected cells may or may not be visible on the client, but their selected state is preserved by the client.

To control the selection of the cell, use:

```
selection.selectCell(cellID,selected)
```

The `cellID` must be a valid grid cell ID. Set `selected` to `true` to select the cell or `false` to deselect the cell.

To clear all selected cells:

```
selection.clearSelection()
```

All cell selections will be cleared.

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:

DB2	DB2 OLAP Server	DB2 Universal Database™
IBM	WebSphere	

Alphablox and Blox are trademarks or registered trademarks of Alphablox 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

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X

Program Number: 5724-L14

Printed in USA

SC18-9434-01

Spine information:

IBM DB2 Alphablox

DB2 Alphablox Developer's Guide

Version 8.3