IBM DB2 Alphablox

開発者用リファレンス

バージョン 8.3

SD88-6491-01
(英文原典：SC18-9435-01)

IBM DB2 Alphablox

開発者用リファレンス

バージョン 8.3

SD88-6491-01
(英文原典：SC18-9435-01)

ご注意！

本書および本書で紹介する製品をご使用になる前に、 565 ページの『特記事項』に記載されている情報をお読みください。

本書の内容は、IBM DB2 Alphablox for Linux, UNIX and Windows (製品番号 5724-L14) バージョン 8 リリース 3 および新版で特に指定のない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本マニュアルに関するご意見やご感想は、次の URL からお送りください。今後の参考にさせていただきます。

`http://www.ibm.com/jp/manuals/main/mail.html`

なお、日本 IBM 発行のマニュアルはインターネット経由でもご購入いただけます。詳しくは

`http://www.ibm.com/jp/manuals/` の「ご注文について」をご覧ください。

(URL は、変更になる場合があります)

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

原　典： SC18–9435–01
IBM DB2 Alphablox
Developer's Reference
Version 8.3

発　行： 日本アイ・ビー・エム株式会社

担　当： ナショナル・ランゲージ・サポート

第1刷 2005.10

この文書では、平成明朝体™W3、平成明朝体™W7、平成明朝体™W9、平成角ゴシック体™W3、平成角ゴシック体™W5、および平成角ゴシック体™W7を使用しています。この(書体*)は、(財）日本規格協会と使用契約を締結し使用しているものです。フォントとして無断複製することは禁止されています。

注* 平成明朝体™W3、平成明朝体™W7、平成明朝体™W9、平成角ゴシック体™W3、
平成角ゴシック体™W5、平成角ゴシック体™W7

目次

第 1 章 このリファレンスの使用法

この「*開発者用リファレンス*」は、アプリケーション開発のリファレンスとして使用するよう作成されています。DB2 Alphablox タグ・ライブラリーの使用に焦点を合わせています。

DB2 Alphablox タグ・ライブラリーには、6 つのタグ・ライブラリーがあります。

- Blox タグ・ライブラリー
- Blox フォーム・タグ・ライブラリー
- Blox ロジック・タグ・ライブラリー
- Blox Portlet タグ・ライブラリー
- Blox レポート・タグ・ライブラリー
- Blox UI タグ・ライブラリー

Blox タグ・ライブラリーは中核となるタグ・ライブラリーで、データへのアクセスおよび検索、データの表示、および JSP 内でのアプリケーション・インフラストラクチャーに関連した Blox の追加用タグが提供されます。 本書では、この Blox タグ・ライブラリー内の各 Blox に関する参照情報が提供されます。Blox タグ・ライブラリー内の各 Blox に関する以下のセクションは、他のタグ・ライブラリーの参照情報にもなります。 Blox レポート・タグ・ライブラリーが唯一の例外です。関係する Blox やアプリケーション開発方法が異なるため、このライブラリーについては *Relational Reporting 開発者用ガイド* で別個に説明されています。

Blox に関連する Java API および関連オブジェクトについては本書では説明されていませんが、提供されている Javadoc に記述されています。DB2 Alphablox Information Center には、4 セットの Javadoc が同梱されています。

Javadoc 名	**説明**
Blox API Javadoc	これは、Blox レポート・タグ・ライブラリーを除く、DB2 Alphablox タグ・ライブラリーのすべての Blox および関連オブジェクト用の Javadoc です。
Blox API Change List	この Javadoc セットは、Blox API Javadoc の今回のリリースと以前のリリースを比較することによって、新規の、変更された、使用すべきでない、または除去されたすべての API をリストします。
Relational Reporting API Javadoc	これは、リレーショナル・データ・ソースからのレポートの作成をサポートするための、Blox レポート・タグ・ライブラリー内のすべての Blox 用 Javadoc です。
FastForward API Javadoc	これは、FastForward アプリケーション・フレームワーク用の Javadoc です。

Blox プロパティー、メソッド、およびタグ属性

一般に、Blox 用のタグ属性は対応する Blox プロパティーにマップされます。たとえば、DataBlox には query タグ属性があり、データを取得するための照会を指定します。これは、DataBlox Java Bean で使用可能な query プロパティーに対応します。一般に各プロパティーには、setter メソッドおよび getter メソッドがあります。query プロパティーの場合、関連メソッドは getQuery() および setQuery(...) です。一部のタグ属性では、特定の状態または機能をオンまたはオフに指定するためにブール値を取ります。 たとえば、autoConnect および enableShowHide タグ属性とプロパティーの場合、対応するメソッドは isAutoConnect()、setAutoConnect(...)、isEnableShowHide() および setEnableShowHide(...) です。すべてのメソッドの参照情報は、Javadoc にあります。

タグ・リファレンス情報の見つけ方

Blox タグ・ライブラリー内のタグは、関連する Blox に基づいて編成されます。たとえば、DataBlox のタグは 179 ページの『第 10 章 DataBlox タグ・リファレンス』 セクション内にあります。Blox レポート・タグ・ライブラリーについては、*Relational Reporting 開発者用ガイド*を参照してください。その他の各タグ・ライブラリーについては、それぞれのセクションで説明されています。

「*開発者用リファレンス*」の Blox タグ・ライブラリーに関する各章は、適切な限り以下のセクションから構成されています。

セクション	説明
概説	その Blox についての簡単に説明し、そのユーザー・インターフェースに関する説明 (もしあれば) を行う。
JSP カスタム・タグ構文	Blox を作成するのに使用するカスタム・タグおよびタグ属性の構文を説明する。
カテゴリー別のタグ属性	カテゴリー別の Blox におけるタグ属性すべて、および関連するカスタム JSP タグ属性がリストされています。

目次にはこれらのセクションがそれぞれリストされており、索引には個々の API が名前でリストされています。

タグ属性の説明の使用法

各 Blox のタグおよびタグ属性は、対応する参照章で説明されています。このセクションでは、各プロパティーの説明がどのように編成されているかを説明します。

タグ属性

そのプロパティーが何をするのかの簡単な説明。

データ・ソース

このプロパティーが適用されるデータ・ソース・タイプ (たとえばマルチディメンション) をリストします。

構文

カスタム・タグ・ライブラリーの構文がリストされます。

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
この表で、タグ属性が受け取る引数が何かあればそれを説明します。		

使用法

このセクションで、プロパティーおよびタグ属性に関係のある使用上の注意が何かあれば、それを示します。

例

このセクションでは、タグ属性の使用例か、他の例へのリンクを示します。

関連項目

このセクションには、関連したプロパティーまたはトピックへの相互参照をリストします。

第 2 章 Blox、および Blox UI モデルの概説

この章では、Blox のカテゴリー、Blox のオブジェクト・モデル、および Blox の UI モデルについて説明します。 DB2 Alphablox に関する概念的な情報については、「*開発者用ガイド*」を参照してください。

Blox のカテゴリー

DB2 Alphablox には、強力な分析アプリケーションを作成するための Blox コンポーネントのセットがあります。このセクションでは、以下のように、これらの Blox コンポーネントの概説をカテゴリー別に行います。

- ユーザー・インターフェース Blox
- データ Blox
- 分析インフラストラクチャー Blox
- Blox UI コンポーネント
- ビジネス・ロジック Blox
- FormBlox
- Relational Reporting Blox

ユーザー・インターフェース Blox

これらの Blox は、ページ・フィルター、ツールバー、メニュー・バー、およびデータ・レイアウト・パネルによってサポートされる、グリッドおよびチャート形式のデータ・ナビゲーションのためのビジュアル・コンポーネントを提供します。このカテゴリーの Blox には、以下のものがあります。

- GridBlox: データを表形式で提示します。
- ChartBlox: データをチャートで提示します。
- DataLayoutBlox: ユーザーが望む行、列、またはページ・フィルターのいずれかの軸にディメンションを移動できるようにするデータ・レイアウト・パネルを提供します。
- PageBlox: グリッドおよびチャートで提示されるデータにページ・フィルターを提供します。
- ToolbarBlox: アイコンをクリックすることで、データ・ナビゲーションや分析の機能に簡単にアクセスできるようにします。
- PresentBlox: 上記のすべてのユーザー・インターフェース Blox を 1 つのコンテナーに結合し、レイアウトと Blox 相互の通信を良いものにします。

- ContainerBlox: すべてのユーザー・インターフェース Blox のための基礎となる Blox です。カスタム・ユーザー・インターフェースを作成したい場合、ContainerBlox から始めることができます。

これらの Blox のためのカスタム JSP タグは、Blox タグ・ライブラリー (`blox.tld`) にあり、使用可能です。

データ Blox

これらの Blox は、データ・ソースへのアクセスを提供します。 DataBlox は特に、ユーザー・インターフェース Blox が対象のデータ・ソースに接続してそこから結果セットを取得するのに必要です。このカテゴリーの Blox コンポーネントには、以下のものがあります。

- DataBlox: ユーザー・インターフェース Blox のために、データ・ソースにアクセスし結果セットを取得します。
- StoredProceduresBlox: リレーショナル・データベースへの接続を作成し、ストアード・プロシージャー・ステートメントを使用できるように準備することを可能にします。
- ResultSetBlox: ResultSet を、関連する DataBlox に任意にプッシュできるようにします。また、DataBlox への照会をインターセプトして任意の ResultSet を DataBlox に戻すメソッドを付加することにより、 JDBC データ・ソースに関連した通常の機能を拡張することもできます。
- JDBCConnection Bean: Alphablox リレーショナル・データ・ソースについての情報の取得、JDBC 呼び出しの実行、DB2 Alphablox で定義されている JDBC データ・ソースのプロパティーのオーバーライドができるようにします。

DataBlox と StoredProceduresBlox のためのカスタム JSP タグは、Blox タグ・ライブラリー (`blox.tld`) にあり、使用可能です。

分析インフラストラクチャー Blox

以下の Blox は分析インフラストラクチャーを構築するための手段を提供します。このカテゴリーの Blox には、以下のものがあります。

- AdminBlox: DB2 Alphablox の管理ページを通して、サーバー、ユーザー、グループ、役割、データ・ソース、およびアプリケーション・セットの情報へのプログラマチックなアクセスを提供します。
- BookmarksBlox: ブックマークをプログラマチックに作成および管理することができ、ブックマーク・プロパティーを動的に設定することが可能になります。
- CommentsBlox: セル・コメント (セル注釈とも言う) や、汎用のページ/アプリケーション・コメント機能をアプリケーションに提供します。
- RepositoryBlox: DB2 Alphablox Repository 内のユーザーおよびアプリケーション・プロパティーを保管および検索する手段を提供します。

これらの Blox のためのカスタム JSP タグも、Blox タグ・ライブラリー (`blox.tld`) にあり、使用可能です。

Blox UI コンポーネント

DHTML クライアントは、3 つの区別される部分からなる Blox UI モデル上に作成されます。それらは、ページ上のビジュアル要素 (コンポーネント)、コンポーネントからのイベントを処理するコントローラー、ユーザー・インターフェースおよび基礎となるアプリケーション・ロジックからの状態変更を通信するイベントです。これらの UI コンポーネントは拡張可能で、ユーザー・インターフェース Blox が提供するすぐに使用可能な機能を拡張することができます。

Blox UI タグ・ライブラリー (`bloxui.tld`) のタグによって、以下を行うことができます。

- メニュー・バーやツールバーへの項目の追加や独自のメニュー・バーやツールバーの作成など、ユーザー・インターフェースのコンポーネントのカスタマイズ
- 空の列や行を追加したり、行見出しを指定位置に追加 (バタフライ・レイアウト) するなどのレイアウトのカスタマイズ
- ユーザー・インターフェースに完全に統合できるカスタム・データ分析機能の追加

Blox UI タグ・ライブラリー (`bloxui.tld`) の説明は、417 ページの『第 23 章 Blox UI タグ・リファレンス』 にあります。すべてのコンポーネントとそのメソッドのリストについては、Javadoc の com.alphablox.blox.uimodel.* パッケージをご覧ください。

ビジネス・ロジック Blox

これらの Blox コンポーネントにより、アプリケーションにビジネス・ロジックを追加することができます。

- MDBQueryBlox: 基礎となるサーバーの照会言語に関係なく、OLAP 照会を 1 つの言語で構築できるようにします。
- MemberSecurityBlox: 無許可ユーザーからメンバーを隠すことが可能になります。
- TimeSchemaBlox:「最近 3 カ月」のデータの表示といった、動的な時系列をサポートします。

389 ページの『第 21 章 ビジネス・ロジック Blox および TimeSchema DTD リファレンス』 で説明されているように、ビジネス・ロジック Blox のためのカスタム JSP タグは、Blox ロジック・タグ・ライブラリー (`bloxlogic.tld`) にあり、使用可能です。

FormBlox

これらの Blox は、フォームに基づいたインターフェースを作成するために、Blox UI コンポーネントの上に作成されます。ユーザーが個別設定した照会を作成するために、提供されるデータ・ソース、ディメンション、メンバーその他のオプションを選択できるようにする使い慣れた HTML フォーム・インターフェースを提供する一群の FormBlox が使用可能です。

347 ページの『第 20 章 Blox Form タグ・リファレンス』 で説明されているように、FormBlox のためのカスタム JSP タグは、Blox フォーム・タグ・ライブラリー (`bloxform.tld`) にあり、使用可能です。

Relational Reporting Blox

リレーショナル・データ・ソースから対話式レポートを作成するように設計された一群の Blox です。詳細については、「*Relational Reporting 開発者用ガイド*」を参照してください。

Blox オブジェクト・モデル

Blox API は多数の Java™ オブジェクトで構成され、他のオブジェクトを使用して多くのオブジェクトにプログラマチックにアクセスすることができます。たとえば、PresentBlox にはメソッド `getDataBlox()`、`getPageBlox()`、`getGridBlox()`、`getChartBlox()`、`getToolbarBlox()`、および `getDataLayoutBlox()` があり、これらを使用して内部にネストされた DataBlox、 PageBlox、 GridBlox、 ChartBlox、 ToolbarBlox、および DataLayoutBlox にアクセスすることができます。このセクションでは、Blox API の全体的なオブジェクト・モデルを説明します。ほとんどのオブジェクト・モデルがそうであるように、Blox API をナビゲートするパスは多数あります。このセクションでは、読者にこれらの API について知っていただく目的で、最も基本的で一般的なアクセス・ポイントを説明します。

ヒント: Javadoc ツールによって生成された API 資料の使用に経験を積んだ開発者の場合、これはオブジェクト・モデルについて学習するための役に立つツールです。 DB2 Alphablox Blox API の Javadoc は、DB2 Alphablox 管理ページの「ヘルプ」リンクから使用可能です。Javadoc をローカルにインストールしている場合には、以下の場所にあります。

- *<alphablox_dir>/system/documentation/javadoc/blox/index.html* オンライン文書が DB2 Alphablox の既存インスタンスに関連付けられている場合。
- *<doc_install_dir>/javadoc/blox/index.html* オンライン文書がスタンドアロンでインストールされている場合。

このセクションでは、以下のトピックを扱います。

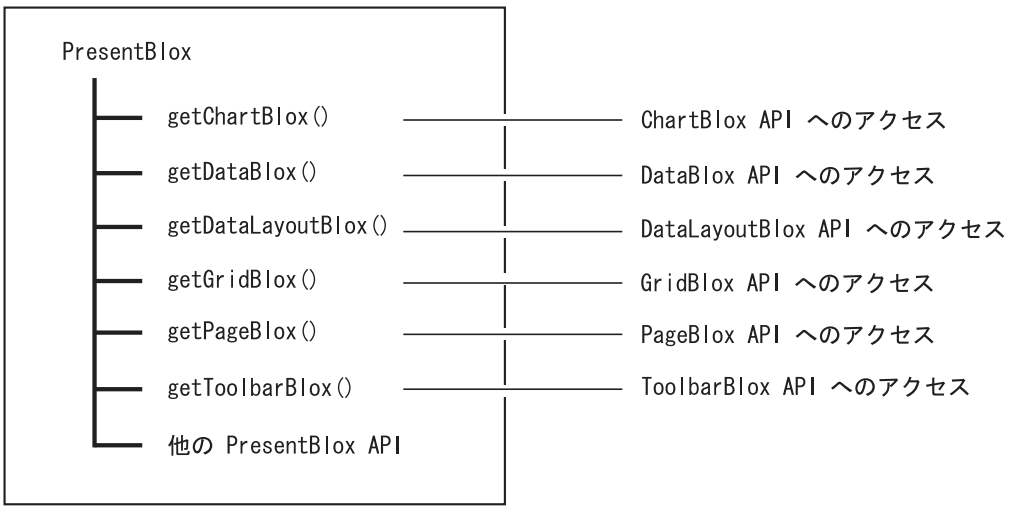
ContainerBlox - ユーザー・インターフェース Blox のためのコンテナー

ContainerBlox は、すべてのユーザー・インターフェース Blox のためのベース・クラスです。これらの Blox は ContainerBlox から `bloxModel` プロパティーを継承します。この Blox の `getBloxModel()` メソッドを使用して、結果的にこの Blox のための UI モデルにアクセスすることができます。 Blox モデルはそれぞれヘッダー・コンテナーとボディ・コンテナーから構成され、各コンテナーはいくつかの名前付き標準コンポーネントを含みます。これらの名前を使用して、コンポーネントを見つけてカスタマイズすることができます。このことは、13 ページの『Blox UI モデル』 で詳しく説明されています。

PresentBlox - ネストされたユーザー・インターフェース Blox を持つ単一の Blox

PresentBlox は、チャート、グリッド、データ・レイアウト・パネル、およびツールバーをすべて単一の Blox に表示する便利な方法です。ネストされた *Blox* を使用して、PresentBlox に表示される各部分の個々の要素をすべて制御することができます。こうした個々の要素のそれぞれには PresentBlox を通じてアクセスするので、PresentBlox をその他の Blox のコンテナーとみなすことができ、PresentBlox コンテナー内の他の Blox はネストされた Blox といいます。 Blox それぞれにはその Blox の状態を表すプロパティーがあり、ネストされた Blox のプロパティーにアクセスするには、その Blox を作成するのに使用するタグ・ライブラリーでプロパティーの値を指定するか、API を使用してプログラマチックにプロパティーの取得および設定を行います。

次の図は、PresentBlox を使用して他の Blox にアクセスする方法を示します。



ネストされた Blox

Blox には、他のネストされた Blox が含まれる場合があります。たとえば、ChartBlox および GridBlox (それぞれ独立型 Blox になる場合もあります) は、ネストする PresentBlox 内のネストされた Blox です。 DataBlox は、PresentBlox や ChartBlox などデータ・ソースを持つ Blox 内にネストすることができます。ネストされた Blox に Blox 固有のプロパティーを適用するには、ネストされたタグを追加します。ネストされた Blox は、オブジェクト・モデルを使用してアクセスします。外側の Blox から始め、内側の Blox オブジェクトへのアクセスを取得するための get メソッド (たとえば `getDataBlox()`) を呼び出して内側の Blox にアクセスします。

Blox タグ・ライブラリーを使用して、ネストされた Blox の作成とアクセスができます。 IBM DB2 OLAP Server や Hyperion Essbase データ・ソースのための次の例は、ChartBlox 内にネストされた DataBlox を示します。

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

通常 PresentBlox には、1 つの DataBlox を共有する複数の Blox が含まれます。

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

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

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

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

</blox:present>
```

複数のプレゼンテーション Blox が使用することになる明示的な DataBlox がある場合、bloxRef タグ属性を使用して、この DataBlox をネストされた Blox として使用することができます。

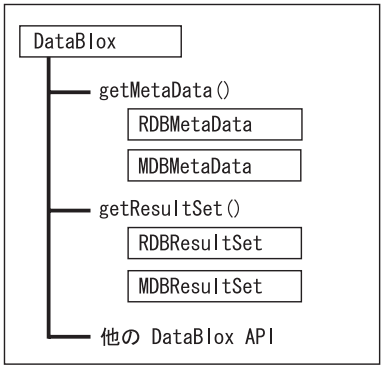
```
<blox:data id="FinancialCube"
   dataSourceName="FinancialCube">
   query="!" />

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

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

各 Blox カスタム・タグの構文については、その Blox の『JSP カスタム・タグ構文』のセクションを参照してください。

DataBlox - メタデータおよび結果セットへのアクセス

DataBlox は、照会の検索という意味ばかりでなく、データベース内のメタデータの検索および結果セットに取得されたデータの検索という意味でも、データ・ソースへのアクセスを提供します。メタデータというのは、特定のディメンションにどのメンバーが属するか、特定の表にどの列が属するか、ディメンションの名前は何か、表の名前は何かなどの、データに関する情報のことです。

次の図は、DataBlox を使用してメタデータや結果セット・オブジェクトにアクセスする方法を示します。メタデータや結果セット・オブジェクトにはそれぞれ複数のオブジェクトが含まれます。



RDBMetaData および RDBResultSet と関連した API を使用するには、次に示すように JSP ページに com.alphablox.blox.data.rdb パッケージをインポートする必要があります。

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

MDBMetaData および MDBResultSet と関連した API を使用するには、次に示すように JSP ページに com.alphablox.blox.data.mdb パッケージをインポートする必要があります。

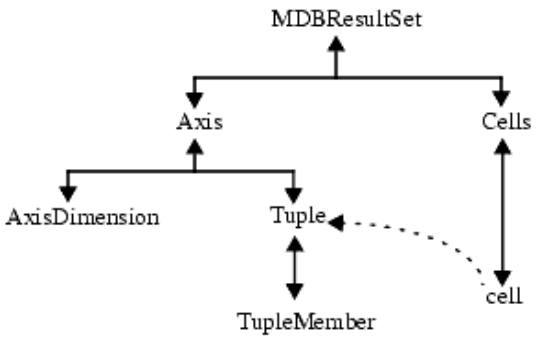
```
<%@ page import="com.alphablox.blox.data.mdb.*" %>
```

メタデータおよび結果セット

結果セットとメタデータは、データを階層的な仕方でたどる方法を提供します。これらは、豊富な API のセット (これによりデータの表示と対話を細かく制御することができます) を持つオブジェクトとして表現されます。これらのオブジェクトには、`getMetaData()` および `getResultSet()` DataBlox メソッドを使用してアクセスすることができます。

結果セット・オブジェクトには、照会からの実際のデータ値が入っています。このため、軸、タプル、ディメンション、メンバーの制限されたセットが入ります。メタデータ・オブジェクトは照会からの結果セットを必要とせず、データ・ソースのキューブ、ディメンション、およびメンバー (アウトライン) だけが関係します。一般的に言って、計算、書き戻し、カスタム・ビューといったデータ・ソースに固有の作業を行っている場合は、MDB や RDB の結果セットを使用します。作業がアウトラインのブラウズや照会の作成に関係する場合、メタデータ・オブジェクトを使用するべきです。

MDBResultSet: 以下の図に、MDBResultSet のオブジェクト階層を示します。矢印の向きは、あるオブジェクトの親または子を参照できるかどうかを示しています。点線の矢印は、いったん個々のセルに達したらそのタプルを見つけることができ、それによりそのタプルが存在している軸か特定のタプル・メンバーにアクセスできることを意味します。



通常、MDBResultSet オブジェクトには複数の軸と複数のセルがあり、それぞれの軸には複数のタプルかディメンションがあります。このため、軸、ディメンション、タプル、またはセルをすべて含む配列を戻すメソッドと、0 を基準とした指標を指定すれば特定の 1 つの軸、ディメンション、タプル、またはセルを戻すメソッドがあります。たとえば、`getAxes()` は結果セット内のすべての軸を含む配列を戻し、`getAxis(0)` は結果セット内の最初の軸を戻します。

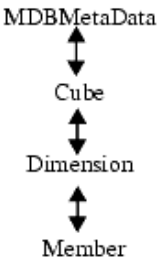
オブジェクトによっては、望む子オブジェクトへのアクセスを簡単にするタイプがあります。たとえば、軸オブジェクトには、`ROW_AXIS`、`COLUMN_AXIS`、`PAGE_AXIS`、および `SLICER_AXIS` というフィールドがあります。これにより、特定のタイプの軸に簡単にアクセスすることができます。同様に、AxisDimension には `ATTR_DIMENSION`、`MEASURES_DIMENSION`、および `TIME_DIMENSION` などのタイプがあり、それによって特定のタイプのディメンションに簡単にアクセスすることができます。

RDBResultSet: 以下の図に、RDBResultSet のオブジェクト階層を示します。矢印の向きは、あるオブジェクトの親または子を参照できるかどうかを示しています。

RDBResultSet オブジェクトには ResultColumn オブジェクトが含まれ、これにより結果セットの列タイプ、列名、および位置 (0 を基準とした指標) についての情報が得られます。行イテレーターとは、行ごとに反復してデータを取得するためのオブジェクトの配列 (`Object[]`) です。

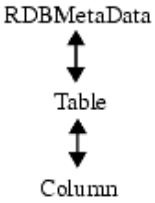
MDBResultSet オブジェクトと同様に、RDBResultSet オブジェクトには通常複数の列があります。このため、すべての ResultColumn オブジェクトを収容する配列を戻すメソッドと、特定の 1 つの列を戻すメソッドがあります。たとえば、`getColumns()` はこの結果セット内の結果列の配列を戻し、`getColumn(0)` はこの結果セット内の最初の結果列を戻します。

MDBMetaData: 以下の図に、MDBMetaData のオブジェクト階層を示します。矢印の向きは、あるオブジェクトの親または子を参照できるかどうかを示しています。



MDBMetaData オブジェクトにも、複数のキューブがある場合があります。(IBM DB2 OLAP Serverや Hyperion Essbase の場合は 1 つのキューブだけです。) 通常それぞれのキューブには複数のディメンションがあり、それぞれのディメンションには複数のメンバーがあります。結果として、多くの場合、キューブ、ディメンション、またはメンバーをすべて含む配列を戻すメソッドと、0 を基準とした指標を指定すれば特定の 1 つのキューブ、ディメンション、またはメンバーを戻すメソッドがあります。たとえば、`getCubes()` はキューブの配列を戻し、`getCube(0)` はこの MDBMetaData オブジェクトが記述するデータベース内の最初のキューブを戻します。

RDBMetaData: 以下の図に、RDBMetaData のオブジェクト階層を示します。



RDBMetaData オブジェクトには複数の表が含まれる場合があり、通常それぞれの表には複数の列があります。結果として、多くの場合、表や列をすべて含む配列を戻すメソッドと、0 を基準とした指標を指定すれば特定の 1 つの表か列を戻すメソッドがあります。たとえば、`getTables()` は表の配列を戻し、`getTable(0)` はこの RDBMetaData オブジェクトが記述するデータベース内の最初の表を戻します。

このセクションで説明した各オブジェクトのメソッドのリストは、182 ページの『カテゴリー別の DataBlox タグ属性』を参照してください。

Blox UI モデル

DHTML クライアントは、コンポーネント、コントローラー、およびイベントというフレームワークの 3 つの明確な概念を持つ Blox UI モデルに基づいています。コンポーネントは、ボタン、編集フィールド、イメージ、テキスト、ツールバー、およびメニューなどのページ上の視覚エレメントを作り上げます。コントローラーは、コンポーネントからのイベントを処理し、`ClickEvent`、`RightClickEvent`、`DoubleClickEvent`、または `SelectedEvent` などの一般的なコンポーネントの振る舞いをアプリケーション定義のアクションに変換します。イベントは、ユーザー・インターフェース、基礎となるアプリケーション・ロジック、およびモデル自体から、コンポーネントおよびコントローラーに状態変更を伝達します。

以下のセクションでは、コンポーネント、コントローラー、イベントについて概説します。これらは、Blox UI タグ・ライブラリーで作業したり、コンポーネントの振る舞いをカスタマイズしたいときに良く出てくる普通の概念および用語です。この一般情報は、DHTML クライアントの UI モデルを理解する助けになり、Blox UI モデルのカスタマイズや拡張を行うための基礎を提供します。 Blox UI モデルの拡張は上級トピックで、「*開発者用ガイド*」に説明されています。オブジェクトとそれに関連したメソッドについての詳細は、DB2 Alphablox Blox API の Javadoc の com.alphablox.blox.uimodel.* パッケージの下にあります。

コンポーネント

Blox UI モデルのビジュアル・ユーザー・インターフェース・オブジェクトはすべて、Component 基本クラスから生じます。このモデルは、 Button、 CheckBox、 RadioButton、 Edit (編集フィールド)、 ListBox、 DropDownList、 Menu、 Menubar、 Toolbar、 ToolbarButton、 DropDownToolbarButton、および ComponentContainer などの多数の核となる基本的なユーザー・インターフェース・コンポーネントを提供します。これらのコンポーネントはすべて、いくつもの共通プロパティーと振る舞いを共有し、基本的なコンポーネントのフォーマット制御と集中管理の両方を提供する階層に配列されます。こうしたコンポーネントのグループ分けは ComponentContainer によって可能になり、レイアウト、振る舞い、およびスタイルの目的でコンポーネントのグループ分けを行うことが可能です。コンポーネント、その階層、およびそのメソッドに関する詳細は、Javadoc の com.alphablox.blox.uimodel.core パッケージを参照してください。

これらの UI コンポーネントをさらに結合して、サーバーの Blox モデルに複合コンポーネントを作ります。 `BloxModel` は、`GridBloxModel`、`ChartBloxModel`、`DataLayoutBloxModel`、`PageBloxModel`、および `PresentBloxModel` の基本クラスです。 ViewBlox から派生した Blox の現在のビジュアル状態を表現するのに使用します。(8 ページの『ContainerBlox - ユーザー・インターフェース Blox のためのコンテナー』 にあるこの Blox のオブジェクト階層を参照してください。)

Blox モデルはそれぞれ以下の 2 つの名前付きコンテナーで構成されます。

- `ModelConstants.BLOXUI_HEADER` - ツールバーとメニュー・バーを含むコンテナー
- `ModelConstants.BLOXUI_BODY` - 固有の Blox 機能を提供するモデルを含むコンテナー

主なコンテナーそれぞれの内部には、いくつもの明確に名前を付けられた標準コンポーネントがあり、これらはカスタマイズしたり除去したりすることができます。これらの名前を使用して、カスタマイズのために簡単にコンポーネントを見つけることができます。コンポーネント名はすべて、インターフェース・クラス `ModelConstants` の定数として使用可能です。

次の表は、Blox UI モデル・コンポーネントのリストです。これらが Blox ユーザー・インターフェースを構成するコンポーネントです。

Blox UI モデル・コンポーネント

	説明
Button	プッシュボタン・コンポーネント。
CheckBox	チェック・ボックス・コンポーネント。

Component

すべての UI モデルのビジュアル・コンポーネントの抽象基本クラスです。このクラスは、すべてのビジュアル・コンポーネントを通じて共通なデフォルトの振る舞いとプロパティーを提供します。

ComponentContainer

UI モデル・オブジェクトのための汎用コンテナー。コンポーネント・コンテナーは、レイアウト、振る舞い、およびスタイルの目的でコンポーネントをグループ分けするのに使用します。たとえば、3 つのボタンを左から右に水平に並べたい場合、それらを ComponentContainer に入れ、そのレイアウトを水平に設定します。コンテナー内のコンポーネントの順番は重要で、表示順序に影響します。

1 つのコンポーネントは、厳密に 1 つのコンテナーに 1 回しか存在することができません。あるコンポーネントを別のコンテナーに追加すると、前のコンテナーからは自動的に削除されます。

Controlbar

ControlbarContainer に収容することが可能なコントロール・バー (メニューおよびツールバー) のための基本クラス。

ControlbarContainer

Controllbar のためのコンテナー。

DateChooser

このコンポーネントは、編集フィールドの隣にカレンダー・アイコンを追加して、 Edit コンポーネントを拡張します。アイコンをクリックすると、編集フィールドに取り込む日付を選択するためのカレンダー・ウィジェットが起動します。

Dialog

ダイアログとは、ユーザーから入力を収集したり、ユーザーに状況を表示したりするのに使用される浮動コンテナーです。ダイアログを作成してから、そのダイアログに (特に) Button、CheckBox、および RadioButtons のようなコンポーネントを追加して、ユーザーにオプション・リストを表示したり、決定を下してもらったりします。

DropDownList

DropDownList は、単一の表示されたオプションと他の選択項目のリストから成ります。一時に 1 つの選択項目しか選択することができません。DropDownList は、スペースが限られていて、考えられる選択項目を常時表示することが必要でない場合に使用します。

DropDownToolbarButton

DropDownToolbarButton には、選択項目のドロップダウン・リストと、現在表示されているドロップダウン・リストを呼び出すアクション・ボタンの両方があります。このコントロールは、アクション・ボタンがクリックされた場合と、選択項目が変更された場合に、ClickEvent を生成します。

Edit

編集フィールド・コンポーネント。 Edit コンポー

ネントにより、ユーザーは 1 行以上のテキストの入力および変更を行うことができます。テキストは、標準のユーザー UI メカニズムを使用して、編集フィールドにコピー、移動、挿入することができます。

GroupBox

GroupBox コンポーネントは、ダイアログおよび他のモデルのためのタイトル付きのコンテナーを提供します。 GroupBox は、基本的にダイアログ・ボックス内でコンポーネントをグループ分けするのに使用します。たとえば、チャートにオプションを設定するため専用のコンポーネントがいくつかある場合、これらを 1 つの GroupBox 内にまとめて、「チャート・オプション」とタイトルを付けることができます。

RadioButton コンポーネントが GroupBox の内部にある場合、名前付きグループ内の名前が付いていない RadioButton はすべて自動的にグループになります。 1 つのラジオ・ボタンを押すと、グループ内の他のラジオ・ボタンは選択解除されます。

Image

Image コンポーネントは、GIF、JPEG、および他の互換性のあるイメージを表示するのに使用します。イメージは、クリックされると ClickEvent を生成します。

ListBox

選択リストを作成する ListBox コンポーネント。

Menu

Menu コンポーネントは、MenuItem と他の Menu から成ります。 Menu 内部の Menu は、適当なサブメニュー動作をするサブメニューとして扱われます。 MenuItem は選択項目として表示され、選択されると ClickEvent を生成します。デフォルトでは、MenuItem の名前はコントローラー内のハンドラー・メソッドを構成するのに使用されます。たとえば、「drillDown」という名前を持つ MenuItem は、コントローラー内の「actionDrillDown」というメソッドにマップされます。新しいメニューにはすべて、デフォルトで垂直レイアウトが割り当てられます。

Menubar

メニュー・バー・コンポーネント。

MenuItem

メニュー項目コンポーネント。これは、メニューで選択可能な項目です。

MessageBox

メッセージ・ボックス・ダイアログ。 MessageBox ダイアログは、ユーザーに簡単な情報を表示するための便利な方法を提供します。また、ユーザーから YES/NO および OK/キャンセルの応答を収集するための簡単な手段も提供します。

RadioButton

ラジオ・ボタン・コンポーネント。通常、

RadioButton はグループにまとめられ、一時には名前付きグループ内の単一の RadioButton しか選択できないようにします。 RadioButton のグループ内のある RadioButton を選択すると、そのグループ内の現在選択されているボタンの選択は自動的に解除されます。

GroupBox を使用して、RadioButton の複数のセットを自動的にグループ化することができます。

Spacer

スペーサー・コンポーネント。コンポーネントどうしの間に固定の高さまたは幅 (あるいはその両方) のスペーシングを追加するのに使用します。

SpinnerButton

スピナー・コンポーネント。ユーザーからの整数の入力を受け付け、その値を増やしたり減らしたりするためのボタンを準備するのに使用します。初期値、増分、最小値、および最大値を設定することができます。

SplitterContainer

スプリッター・コンテナー・コンポーネント。ユーザーが間隔を調整できるスプリッター・バーを持つ 2 つのコンポーネントを表示するのに使用します。HorizontalLayout と VerticalLayout のいずれかを使用して、スプリッターの向きを制御します。

Static

静的コンポーネント。指示、ラベル、または値など対話の必要がない単純な静的テキストを表示するのに使用します。ユーザーの選択に応答するように、静的コンポーネントへの ClientLink をアタッチすることができます。

コンポーネントのタイトル・フィールドは、静的テキストの保管に使用されます。

StaticImage

ユーザー入力に応答しない静的イメージを表示するコンポーネント。 ClickEvent を生成するイメージのためには、Image コンポーネントを使用します。

TabbedContainer

子コンテナーすべてのためのタブを持つコンテナー・ウィンドウ。このコンテナーは、すべての子コンテナーに対応する一連のタブを表示するのに使用します。典型的な使用法は、タブ付きのダイアログ・ボックスをインプリメントすることです。このコンテナーに含めることができるのは、他のコンポーネント・コンテナーだけです。このコンテナーに付加されたスタイルはタブに適用されます。

子コンテナーのタイトルは、そのコンテナーのタブ・ラベルに使用されます。子コンテナーの選択状態が、選択されているタブを決定します。どの子コンテナーも選択されていなければ、最初のコンテナーが自動的に選択されます。複数の子コンテナーが

選択されているとマークされている場合、そのうちの最初のものが選択されていると見なされます。

子コンテナーがタブ付きコンテナーに追加された順番が、タブ順序を決定します。タブが上部および下部 (水平) の場合、最初のコンテナーは右側になります。タブが左および右 (垂直) の場合、最初のコンテナーは一番上になります。

Toolbar ControlbarContainer と関連してツールバーを表示するのに使用するツールバー・コンポーネント。

ToolbarButton ツールバー・ボタン・コンポーネント。ツールバー・ボタンはコンポーネント・モデルのどこにでも置くことができますが、基本的には ControlbarContainer 内で動作するように設計されています。コンポーネントの名前を使用して、`.gif` 拡張子を持つイメージ名が構成されます。

イベント

DHTML クライアントは、JavaScript™ コードによって作成、送信、インターセプトすることができる JavaScript オブジェクトとしてイベントを作成します。 DHTML クライアントは、イベントがデータのドリルアップなのかドリルダウンなのかなど、ドメインについては全く分かりませんし、ディメンションやメンバーのことも分かりません。ドメイン固有のロジックと情報はすべて、サーバーに保管されます。 DHTML クライアントに分かるのは、メニュー項目や「ヘルプ」ボタンがクリックされた時だけです。シングルクリック、ダブルクリック、右クリック、スクロール、ドラッグ・アンド・ドロップ、選択された、選択解除された、選択が変更された、内容が変更された、およびクローズされたなどの単純なイベントのセットしか認識できません。

コントローラー

ユーザーが `Button` を押したり `Menu` から `MenuItem` を選択すると、そのユーザー・イベントと関連付けられたアクションを呼び出して実行するのは、コントローラーの責任です。 `Controller` クラスは、すべてのモデル・コンポーネント・コントローラーの基本クラスです。コントローラーは、Component から派生したモデル・オブジェクトにはどれにでも接続することができます。

サーバー・サイド API とクライアント・サイド API

DB2 Alphablox で DHTML クライアントを使用してアプリケーションを作成するための API はサーバー・サイドで使用可能であり、開発者は (たとえば、JSP ページ上の Java スクリプトレットで) Java 呼び出しを行うことによりアクセスすることができます。 Java API がサーバー・サイド API と呼ばれるのは、ブラウザーに送信する前にコードがサーバー上で実行されるからです。

サーバー上でのコード実行は、多くの場合より効率的で、また複数のブラウザーで正しく機能する Web ページを作成するのがより簡単になります。 DHTML クライアントは、クライアントとサーバーが、ページ・リフレッシュなしで同期するよう

に設計されています。サーバー上でコードを実行すると、ページ全体ではなく、Blox UI の関係のあるところだけがリフレッシュされます。

クライアント上で行った方が良い作業を、DHTML クライアントの Client API を使用して行いたいという場合もあります。こうした API は、ブラウザーが解釈するので、クライアント・サイド API と言います。ユーザーがページ上のボタンやリンクをクリックした場合に、クライアント上の JavaScript コードによって、サーバー上の Blox プロパティーを変更するサーバー・サイド・コードを呼び出したいという場合が多くあります。

DHTML クライアントには、クライアント・サイドの比較的直接的な API があります。詳しくは、497 ページの『第 26 章 クライアント・サイド API』を参照してください。

第 3 章 一般 Blox リファレンス情報

このセクションでは、すべての Blox に当てはまる一般的なリファレンス情報を説明します。すべての Blox に共通なタグ属性についての情報は、35 ページの『第 4 章 共通 Blox タグ・リファレンス』を参照してください。

Blox の処理のヒント

Blox の処理を行う前に、以下のキー・ポイントに注意してください。

- API 変更リスト Javadoc には、推奨されない Blox メソッドとプロパティー、推奨されなくなったリリース、およびその代わりになるもののリストがあります。既存のアプリケーションを扱っている場合、この Javadoc を参照して、メソッドおよびプロパティーにどんな変更が必要か判断してください。
- Blox で無効なプロパティーを使用していると、JSP ファイルは正常にコンパイルできません。
- Blox プロパティー (と対応するタグ属性) には、大/小文字の区別があります。2、3 の例外はありますが、プロパティー名は Java Bean の命名規則に従います。つまり、名前の最初は小文字で、名前の中の新しい語や句はそれぞれ大文字で始めます (たとえば `dataSourceName`)。
- DB2 Alphablox を通じて実行される URL には、大/小文字の区別があります。
- 特定の Blox に関してデフォルト値や継承された値をオーバーライドするには、Blox を作成するのに使用するカスタム・タグにプロパティー・キーワードとローカル値を含めます。たとえば、`<blox:chart>` カスタム・タグ内の以下の属性は、ChartBlox が円グラフを表示するようにします。

  ```
  chartType="Pie"
  ```

さまざまデータ・ソースを扱う

DB2 Alphablox には、Data Manager と、以下をサポートする関連データ・アダプターがあります。

- アプリケーション・データ・ソースへの事前定義された名前付きの接続のコレクションのブラウズ

- 各データ・ソース内の使用可能なデータベースの公開
- 特定のデータ・ソースのための互換性のある照会タイプのパブリッシュ
- データ・ソースのメタデータの全探索を可能にする
- ユーザー・セッションのためのデータ・ソース接続の管理
- 照会オブジェクトの、基礎となるネイティブ照会言語への変換
- データ・ソースに対する照会の実行
- 照会の結果セットのデータとスキーマの問い合わせ
- ピボット、展開、ドリルによる結果セットの変更

DB2 Alphablox データ・アダプターは、リレーショナル・データベース、マルチディメンション・データベース、および DB2 Alphablox キューブにアクセスしてデータを取得することができます。 (DB2 Alphablox キューブは、リレーショナル・データベースからのデータをマルチディメンション形式にトランスフォームします。)

ほとんどの Blox プロパティーとメソッドは、すべてのタイプのデータ・ソースに適用できます。そうでないものについては、API の説明の中に特定の API がどのデータ・ソースを扱うことができるか (たとえば、すべて、マルチディメンション、リレーショナル、IBM® DB2 OLAP Server™ と Hyperion Essbase のみなど) を述べるセクションがあります。

JSP ファイル内の Blox

このセクションでは、Blox を含む JSP ページのコンポーネントを説明します。サンプル JSP ファイルを示してから、そのそれぞれの部分を説明します。

Blox を含むサンプル JSP ファイル

以下のコード・リストは、ページ上に 1 つの Blox を表示するのに必要なすべてのエレメントを含む JSP ファイルです。

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

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

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

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

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

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

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

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

<%-- Insert the Blox header --%>
<blox:header />

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

</head>
<body>

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

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

</body>
</html>
```

パッケージおよびタグ・ライブラリーのインポート

このセクションは通常 JSP ファイルの先頭にあります。パッケージ・インポート・ステートメントは、それらのパッケージの API を使用する場合にだけ必要です。Blox タグ・ライブラリーを使用するためには、タグ・ライブラリー・インポート・セクションが必須です。

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

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

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

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

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

<%-- Import the Blox Portlet Tag Library --%>
<%@ taglib uri="bloxportlettld" prefix="bloxportlet"%>
```

また、このセクションを使用して Java API 呼び出しで使用する Java パッケージをインポートします。 Java API は、JSP ページ上のスクリプトレットから呼び出すことも可能です。たとえば、次のようにします。

- MDBMetaData オブジェクトか MDBResultSet オブジェクトを使用する場合、次のインポート・ステートメントが必要になります。

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

- RDBMetData オブジェクトか RDBResultSet オブジェクトを使用する場合、次のインポート・ステートメントが必要になります。

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

- BookmarksBlox とその関連オブジェクトを使用する場合、次のインポート・ステートメントが必要になります。

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

- Comment オブジェクトを使用する場合、次のインポート・ステートメントが必要になります。

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

- サーバー・サイドの Event Filter オブジェクトを使用する場合、次のインポート・ステートメントが必要になります。

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

- StoredProcedure オブジェクトを使用する場合、次のインポート・ステートメントが必要になります。

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

- Blox を作成するのに、Blox タグ・ライブラリーを使用するのではなく、`<jsp:useBean>` タグを使用する場合、次のインポート・ステートメントが必要になります。

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

- Blox UI モデル API を使用する場合、次のインポート・ステートメントが必要になります。

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

API を使用するためにインポート・ステートメントが必要な場合、この情報はその API セクションそれぞれの先頭にリストされます。

コンテンツ・タイプ文字セット宣言を追加する

正しい文字セット・エンコードが行われるようにするには、次の行を JSP ファイルに追加して文字セットを設定します。

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

日本語や中国語などの 2 バイト文字を使用する言語システムで実行する場合に、このことは特に重要です。

Blox 作成タグ

Blox を作成するためのカスタム・タグは JSP ページ内のどこにでも置くことができますが、ページの HTML セクションより前に置くと、コードがよりきれいに見えて読みやすくなります。こうすると、アプリケーション・ロジックをページの表示エレメントから分離することができます。 Blox タグを HTML エレメントより前に置く場合には、`visible` プロパティーを `false` に設定してから、`<blox:display>`

タグを使用して実際にページ上に Blox を表示するようにもしなければなりません。 <blox:display> タグについて詳しくは、29 ページの『<blox:display> タグ』を参照してください。

HTML <head> 内の <blox:header> タグ

<blox:header> タグは、ページの HTML <head> セクションに置かなければなりません。これは、DB2 Alphablox がこのヘッダー・タグのヘッダーのテーマとスタイル情報を正しく置換して、DHTML モードでレンダリングされるページが正しく表示できるようにするために必要です。さらに、ページのファイル・キャッシングを管理する数行のコードの追加も行います。さらに重要なこととして、これはクライアントとサーバーの間の通信サービスのための基礎となり、クライアント上の JavaScript オブジェクトからサーバー・サイド・コードを実行することができるようになります。

ページ URL およびポートレット統合やクライアント Bean 登録用のコンテキスト・パスを指定するための追加のタグ属性について詳しくは、30 ページの『<blox:header> タグ』を参照してください。

ヒント: <blox:header> タグは、JSP ファイル内で Blox を表示のために設定するより前に置かなければなりません。つまり、visible プロパティーが true (デフォルト) である Blox 作成タグより前に置くか、Blox をページ上で表示する <blox:display> タグより前に置かなければなりません。

Blox API を含むスクリプトレット

JSP ページ内のどこにでも望む Java コードを置くことができ、そうしたコードは、ページがユーザーに送信される前にサーバー上で実行されます。 Java コードはスクリプトレット内で Blox API を使用することができ、何であれ、Java や Web アプリケーション・サーバーが稼働している環境で使用可能なものを使用できます。 Java コードが Blox API を使用する場合、そのコードがスクリプト記述する Blox 定義はスクリプト記述コードより前に置かなければなりません。 JSP ページに Java コードを置くには、有効な Java コードを次の文字セットの間に置きます。

```
<%
%>
```

JSP エンジンはこれを Java コードと認識し、コンパイルし実行します (コンパイルは、最初にページがロードされたときか、前回のコンパイル以降にページが変更された場合だけ行われます)。たとえば、次のスクリプトレットでは、サーバー・サイドの ChartBlox API と、rowSelections および columnSelections プロパティーの値を Java コンソールに表示するための標準の out.write() Java メソッドが使用されています。

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

注: アプリケーション・サーバーを実行しており、DB2 Alphablox コンソールが利用可能でない場合、開発中に出力を表示するのに、出力をログ・ファイルに書き込むとか、UI モデルの MessageBox を使用するなどの手法を使用することができます。

注: JSP テクノロジーにはスクリプト記述のための技法が多数あります。 JSP ファイル内でスクリプト記述するさまざまな方法に関する詳細については、JSP リファレンス・ブックを参照してください。

スクリプトレットの評価方法 - タグの内側と外側の比較

Blox タグはページがユーザー・セッションのために最初にロードされるときだけ評価されますが、タグの外側にあるものはすべてページがロードされるたびに評価されます。 Blox タグが評価されるときに、その時点におけるすべてのプロパティーの状態がページにレンダリングされます。スクリプトレットが Blox タグの内側にあると、そのスクリプトレット内のコードは Blox がレンダリングされる前に評価されるので、そのコードによってプロパティーに加えられる変更はページにレンダリングされる Blox に反映されます。

Blox タグの外側のコードは、Blox が評価されてページにレンダリングされた後で評価されるので、Blox タグの外側のスクリプトレットでのプロパティーの変更は、ページが再ロードされるまでページ上の Blox に現れません。それで、Blox プロパティーの値を変更するスクリプトレットを Blox タグの内側に置けば、それは Blox がページにレンダリングされる前に評価されて、変更は最初にレンダリングされるページに表示されます。Blox プロパティーの値を変更するスクリプトレットを Blox タグの外側に置けば、それは Blox がページにレンダリングされた後で評価されて、プロパティーの変更はページが再ロードされるまで Blox に反映されません。

プロパティーの設定方法を決定するためにスクリプトレット内であるロジックを実行しなければならないものの、そのロジックを (ページが 1 回目にロードされたときだけでなく) ページがロードされるたびに実行したいという場合があります。そのコードを Blox タグの内側に置くと、ユーザー・セッションでそのページが 1 回目にロードされた時には実行されますが、セッション内のそれ以降のページ・リフレッシュでは Blox タグ内のコードは実行されません。この場合、Blox タグの `visible` プロパティーに `false` を設定して、プロパティーを設定するコードを Blox タグの外側のスクリプトレットに置くことができます。それから、ページ内の後の方で `<blox:display>` タグを使用してページ上に Blox を表示します。この技法を使用すれば、Blox タグの外側で設定したプロパティーが、ユーザー・ページに表示される Blox に反映されることになります。次の疑似コードで、この技法を示します。

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

<%
   // this scriptlet executes some code to set a property
```

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

Blox API を含む JavaScript コード

JSP ページの HTML セクション内のどこでも、HTML <script> タグを使用して JavaScript エレメントを置くことができます。

ヒント: HTML ページに <script> タグを置く最良の方法は、 <head> タグや <body> タグの間に置くか、または <head> タグや <body> タグがない場合には <html> タグの間に置くことです。例外は、<script> タグが <head> タグや <body> タグを書き出す場合です。

HTML および JavaScript エレメント

もちろん、HTML エレメントや JavaScript エレメントは何でも JSP ページに置くことができ、それらはそのままブラウザーに渡されます。 JSP エンジンは、HTML エレメントと JavaScript エレメントはすべて無視します。

URL 属性

DB2 Alphablox には、アプリケーションのためのレンダリング・モード、保管された状態、およびテーマを変更する便利な方法として、複数の URL 属性があります。 URL 属性は、ランタイム処理を定義するためにアプリケーションの URL に追加することができます。 URL 属性の形式は以下の通りです。

```
attribute=value
```

たとえば、render 属性で DB2 Alphablox がページをクライアント・ブラウザー配信する前にページをレンダリングする形式を指定します。次の属性は、ページを DHTML で配信することを指定します。

```
render=dhtml
```

単一の属性を URL に追加するには、次の例に示すように URL の最後にまず「?」記号を付けてから属性を付加します。

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

他の URL 属性を追加するには、次のように & 文字を付けて付加します。

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

このセクションでは、以下の有効な URL 属性を説明します。

- 28 ページの『render』
- 28 ページの『theme』

ヒント: URL 属性には大/小文字の区別があり、すべて小文字です。

URL 属性と RepositoryBlox を使用して、保管されたアプリケーション状態をロードすることの例については、Javadoc 内の Repository restoreApplicationState() メソッドを参照してください。

render

render=*string*

このアプリケーション・ページ上のすべての Blox のための配信フォーマットを指定します。なお、Blox のレンダリング・プロパティーはこの属性に優先します。

使用できる値には、以下のものがあります。

dhtml 完全に対話式の DHTML フォーマットでレンダリングします (デフォルト)。 JSP ページに <blox:header /> タグが必要です。

printer 印刷に適したフォーマットでレンダリングします。JSP ページに <blox:header /> タグが必要です。

xls Microsoft® Excel へのエクスポートに適したフォーマットでレンダリングし、MIME タイプを XLS に設定します。

ページを Excel で開けるように MIME タイプを設定するには、JSP ページに <blox:header /> タグを置かなければなりません。 <blox:header /> タグについての情報は、25 ページの『HTML <head> 内の <blox:header> タグ』を参照してください。

xml XML フォーマットでレンダリングします。

配信フォーマットについて詳しくは、「*開発者用ガイド*」の『DHTML クライアントで使用可能なレンダリング・フォーマット』を参照してください。

重要: theme 属性なしでレンダリング属性を使用すると、DB2 Alphablox はデフォルトのテーマと自動ブラウザー検出を使用します。

theme

theme=*themeName*

dhtml レンダリング・モードで使用されます。このページをレンダリングする時に使用するテーマを指定します。テーマ名がデフォルトであるか、この属性が使用されないと、DB2 Alphablox はブラウザー・タイプ、ブラウザー・バージョン、クライアント・オペレーティング・システム、およびレンダリング・フォーマットに基づいて自動的に最も適したテーマを選択します。

注: theme 属性を認識できるようにするには、JSP ページの HTML 部分の <head> タグと </head> タグの間に次の行を追加しなければなりません。

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

theme 属性の有効な値は、coleman (デフォルト) と financial です。 coleman テーマはグレーと青のトーンで、financial テーマは薄い緑のトーンです。

データ・タイプ・マッピング

次の表は、Alphablox のデータ・タイプがどのように JDBC および Java のデータ・タイプにマップされるかを示します。 Java タイプのデータ範囲は、特定のデータベースがサポートする範囲と違う場合があります。

JDBC タイプ	Alphablox タイプ	Java タイプ	Java 範囲
TINYINT	IntegerOperand	int	32 ビット符号付きの 2 の補数の整数
SMALLINT	IntegerOperand	int	32 ビット符号付きの 2 の補数の整数
INTEGER	IntegerOperand	int	32 ビット符号付きの 2 の補数の整数
BIT	boolean	boolean	true または false
BIGINT	LongOperand	long	64 ビット符号付きの 2 の補数の整数
REAL	FloatOperand	Float	32 ビット IEEE 754
FLOAT	DoubleOperand	double	64 ビット IEEE 754
DOUBLE	DoubleOperand	double	64 ビット IEEE 754
NUMERIC	CurrencyOperand	double	64 ビット IEEE 754
DECIMAL	CurrencyOperand	double	64 ビット IEEE 754
CHAR	String	String	
VARCHAR	String	String	
LONGVARCHAR	String	String	

注: データ・タイプ・マッピングに関しては、以下の点に気を付けてください。

- 特定の Java データ・タイプの内容も、データベースのものと違っている場合があります。たとえば、Java の日付値は 1900 年以降のものです。しかし、ほとんどのデータベースはそれより前の日付値を保持することができます。
- JDBC データ・タイプは RDBMS に直接マップできない場合があります。この点での助けを得るには、JDBC ドライバーのベンダーにお問い合わせください。

<blox:display> タグ

`<blox:display>` タグは、既に作成されていてそのタグを置いたところに表示されている Blox を参照します。このタグの最も一般的な使い方は、処理ロジックの実行後に、既に存在していて `visible` 属性を `false` に設定したプレゼンテーション Blox を表示することです。 JSP ページの HTML 部分の Blox を表示したい場所に、`<blox:display>` タグを置いてください。

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

または、次のようにします。

```
<blox:display
    blox="<%=myBlox %>" />
```

- これはエレメントのない空のタグなので、上の例で示すように省略表現を使用して終了する必要があります。アプリケーション・サーバーによっては、`</blox:display>` 終了タグを使用して終了すると、エラーになります。

- <blox:display> タグを使用する前に、それが参照する Blox (bloxRef 属性または blox 属性の値) は既にインスタンス化されていなければなりません。インスタンス化が起きるのは、Blox が別のページで前に作成されているか、Blox が JSP ファイル内で <blox:display> タグより前にある場合です。
- bloxRef 属性によって、Blox の名前を参照できます。 blox 属性を使用すると、実際の Blox オブジェクトを参照できます。たとえば、以下の GridBlox がある場合

  ```
  <blox:grid id="mygrid" visible="false" />
  ```

  以下のいずれかの属性を使用して、後から表示できます。

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

  または、次のようにします。

  ```
  <blox:display blox="<%= mygrid %>" />
  ```

  以下の GridBlox がある場合

  ```
  <blox:grid id="mygrid" bloxName="grid1" visible="false" />
  ```

  以下のコードのいずれかを使用して、後ほど表示できます。

  ```
  <blox:display bloxRef="grid1" />
  ```

  または、次のようにします。

  ```
  <blox:display blox="<%= mygrid %>" />
  ```
- bloxRef と blox 属性の両方を指定すると、 bloxRef の値は無視されます。

<blox:display> で使用可能なタグ属性は以下の通りです。

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

<blox:display> タグで設定した属性は、それが参照するタグで設定されているかもしれない属性をオーバーライドします。上記の例は、id が myBlox である以前に定義された Blox を表示する <blox:display> タグを示していて、オリジナルの render、width、および height 設定を display タグで指定されたものでオーバーライドします。

注: <blox:display> タグが Relational Reporting Blox を参照することはできません。 Relational Reporting Blox は、リレーショナル・データ・ソースからの対話式レポートを生成する ReportBlox をサポートする Blox です。これらの Blox の使用法と参照資料は、「*Relational Reporting 開発者用ガイド*」に文書化されています。

<blox:header> タグ

<blox:header> タグが dhtml モードで適切にレンダリングされるには、Blox のページの HTML <head> セクションに置かれる必要があります。このタグは以下を行います。

- DB2 Alphablox がこのヘッダー・タグのヘッダーのテーマとスタイル情報を正しく含むようにする
- 指定のコンテナー要求および応答 (たとえば、HTTP 要求と応答、またはポートレット要求と応答) に対して適切な BloxContext、BloxRequest、さらに BloxResponse オブジェクトを作成する
- クライアントとサーバーの間の通信サービスのための基礎となり、クライアント上の JavaScript オブジェクトからサーバー・サイド・コードを実行できるようする

サーブレット要求 URI が DB2 Alphablox やアプリケーションのコンテキスト・パスを参照しないポータルやプロキシー・フロントエンドなどの場合、<blox:header> タグには pageURL および contextPath という 2 つの属性があり、ページ URL とアプリケーション・コンテキスト・パスを明示的に指定することができます。

<blox:header> タグにより、<blox:clientBean> 内部タグを使用してカスタム Bean を登録することもできます。これにより、DB2 Alphablox プログラミング・フレームワークにサーバー・サイド Bean を登録して、Bean のメソッドをクライアント上で使用可能にします。このようにして、クライアント・サイド JavaScript から、Bean のサーバー・サイド・メソッドを呼び出すことができます。次の例は、myBean という名前のカスタム Bean と changeColor という名前のそのメソッドの登録を示します。

```
<blox:header>
   <blox:clientBean name="myBean" bean="<%= myBean %>">
      <blox:method name="changeColor" />
   </blox:clientBean>
</blox:header>
```

Bean 名は、BloxContext に属性として追加されます。これは、BloxContext オブジェクトでの setAttribute() メソッドの呼び出しに類似しています。

```
bloxContext.setAttribute("myBean", myBean);
```

こうすると、次のようにクライアント API の callBean() メソッドを使用して、サーバー上の changeColor メソッドを呼び出すことができます。

```
myBean.changeColor("red");
```

Bean を持っておらず参照するだけの場合であっても、bean 属性を指定しないで Bean を登録し、クライアント・サイドの JavaScript から Bean のサーバー・サイドのメソッドを起動できます。以下の例は、bean 属性なしでの Bean 登録を示しています。

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

<blox:header> タグは、theme タグ属性も提供します。 この属性で、このページで使用するテーマを指定できます。 DB2 Alphablox は、以下の優先順位に基づいて、ページに使用するテーマを判別します。

1. URL パラメーター。 ページが呼び出されたときに、theme=*themeName* URL 属性を付加することで指定されたテーマは、他のテーマ設定をオーバーライドします。

2. <blox:header> タグ。 このページ・レベルのテーマ設定は、アプリケーション・レベルまたはシステム・レベルの設定をオーバーライドします。
3. DB2 Alphablox の管理ページから、アプリケーションの定義に指定されたアプリケーションのテーマ設定。
4. DB2 Alphablox の管理ページから、「汎用プロパティー (General Properties)」セクションの「システム (System)」リンクの下で指定された、この DB2 Alphablox インスタンスのデフォルトのテーマ。

ヒント: <blox:header> タグは、JSP ファイル内で Blox を表示のために設定するより前に置かなければなりません。つまり、visible 属性が true (デフォルト) に設定されている Blox 作成タグより前に置くか、Blox をページ上で表示する <blox:display> タグより前に置かなければなりません。

注: テーマやクライアント・サイドの JavaScript コードなしで、適切な BloxContext、 BloxRequest、および BloxResponse オブジェクトを作成するだけの場合には、32 ページの『<blox:bloxContext> タグ』 を参照してください。

<blox:bloxContext> タグ

<blox:bloxContext> タグは、指定の要求と応答のタイプに基づいて適切な BloxContext、BloxRequest、および BloxResponse オブジェクトを作成するという点において <blox:header> タグと似ています。 しかしこのタグは、ページ上にレンダリング JavaScript または不必要なテーマは配置しません。このタグの使用例には、特定の JSP ページに Blox がないものの、別の JSP ページが作成した Blox にアクセスして属性を動的に変更する必要がある場合があります。
<blox:bloxContext> タグと <blox:header> タグはどちらも同じ変数を宣言しようとするため、これらのタグは 1 つの JSP ページでは共存できません。

<blox:session> タグ

このタグにより、DB2 Alphablox セッションの作成を同期することができます。ブラウザーに複数のセッション cookie を送りたくない、またはフレーム内の各 JSP で <blox:header/> タグを使用して Blox を持たないページ上に不要な JavaScript やスタイルを置きたくなくて、フレームセット、または Tomcat を用いる iframe を使用しようとしている場合に役に立ちます。このタグにはオプションの key 属性があり、これはアプリケーションまたはブラウザー・セッション (つまりフレームセット・セッション ID) に固有でなければなりません。 key 属性が指定されない場合、DB2 Alphablox セッションが作成され、セッション ID cookie が戻されます。固有 ID が渡された場合、そのキーを持つ DB2 Alphablox セッションが既に作成されているかどうかを DB2 Alphablox にチェックさせます。たとえば、次のようにします。

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

または、次のようにします。

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

ここで、syncKey はこのページが呼び出されるときに渡されます。

固有でない、または無効なキーが指定された場合、間違ったセッションにデータが表示されるとかセッションの有効期限が切れたというメッセージなどの予期しない結果や望ましくない結果になります。

注: IBM WebSphere または BEA WebLogic の場合、キーはデフォルトの J2EE セッション ID になります。ですから、このタグを使用する必要はありません。

<blox:logo> タグ

このタグは、 DB2 Alphablox 製品 Web サイトへのハイパーリンクを持つ「DB2 Alphablox」ロゴを追加します。必要なのは、次の行だけです。

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

例外

Blox Java API は、エラー状態に達すると例外をスローし、望むならこうした例外をキャッチして何か処理を行うことができます。たとえば、例外をキャッチして、どうやって継続するかについてを指示する特定のエラー・メッセージをユーザーに送信するようにしたいという場合があります。各 API がスローする例外は、構文記述の API シグニチャーに文書化されています。

例外を使用したい場合、DB2 Alphablox に付属の Javadoc で文書化されています。この Javadoc は、DB2 Alphablox ドキュメンテーション CD にあります。また、DB2 Alphablox 管理ページの「ヘルプ」リンクからも使用可能です。この資料を Windows システム上にローカルにインストールすると、「IBM DB2 Alphablox オンライン資料 (IBM DB2 Alphablox Online Documentation)」プログラム・グループの下に Windows ショートカットが作成されます。

例外をキャッチする一般的なやり方は、次の疑似コードのような `try...catch` 構文を使用することです。

```
<%
try {
%>

<% original JSP Code %>

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

例外をキャッチするにしてもしなくても、カスタム・エラー・ページを追加するのは良い習慣です。詳しくは、JSP または Java のリファレンス・ブック、あるいは、「*開発者用ガイド*」の『エラー処理』セクションを参照してください。

第 4 章 共通 Blox タグ・リファレンス

この章には、複数の Blox に共通のタグ属性およびプロパティーの参照資料が含まれています。Blox についての一般的な参照情報は、21 ページの『第 3 章 一般 Blox リファレンス情報』を参照してください。このリファレンスの使用方法については、1 ページの『第 1 章 このリファレンスの使用法』を参照してください。

カテゴリー別の共通の Blox タグ属性

以下の表には、複数の Blox に共通のタグ属性およびプロパティーがリストされています。属性とプロパティーは以下のように編成されています。

アプリケーションおよびセッションに関連するタグ属性

これらのタグ属性は、アプリケーションのインスタンス化およびユーザー・セッションに影響します。

- `helpTargetFrame`
- `localeCode`

振る舞いおよび外観に関連するタグ属性

このセクションのタグ属性は、Blox の振る舞いおよび外観に影響します。

- `bloxEnabled`
- `bloxName`
- `applyPropertiesAfterBookmark`
- `bookmarkFilter`
- `maximumUndoSteps`

ブックマークおよびアプリケーション状態に関連するタグ属性

以下の表には、ブックマークに関連するタグ属性がリストされています。

- `bookmarkFilter`
- `applyPropertiesAfterBookmark`

レンダリングに関連するタグ属性

これらのタグ属性は、Blox の配信に影響します。

- removeAction
- render
- rightClickMenuEnabled

メニュー・バーに関連するタグ属性

以下のタグ属性は PresentBlox、GridBlox、または ChartBlox でメニュー・バーを表示するかどうかを決定します。

- menubarVisible

ポップアウトに関連するタグ属性

以下の表では、PresentBlox、独立型 GridBlox、または独立型 ChartBlox をポップアウトした別のブラウザー・ウィンドウに表示することに関するタグ属性をリストします。

- enablePoppedOut
- poppedOut
- poppedOutHeight
- poppedOutTitle
- poppedOutWidth

複数の Blox に共通するタグ属性

このセクションでは、複数の Blox によってサポートされているタグ属性について説明します。属性は、アルファベット順にリストされています。一般に各タグ属性は、関連する setter および getter Java メソッドによって 1 つの Blox プロパティーにマップされます。複数の Blox に共通のメソッドに関しては、Javadoc 内の com.alphablox.blox.Blox、com.alphablox.blox.ViewBlox および com.alphablox.blox.DataViewBlox を参照してください。

applyPropertiesAfterBookmark

ブックマークの取得後、ブックマーク中のプロパティーを Blox プロパティーがオーバーライドするかどうかを指定します。

データ・ソース

すべて

構文

applyPropertiesAfterBookmark="*applyAfterBookmark*"

引数	デフォルト	説明
applyAfterBookmark	false	true を指定すると、ブックマークがロードされた後に Blox カスタム・タグで指定されているプロパティーを適用します。

使用法

値 true は、ブックマーク・プロパティー値をアプリケーション・ページのプロパティー値で上書きします。

注: DataBlox の dataSourceName プロパティーは applyPropertiesAfterBookmark 設定を無視します。データ・ソース A が PresentBlox によってページ上で現在使用されており、ユーザーがデータ・ソース B に保管されたブックマークをロードした場合には、applyPropertiesAfterBookmark が true に設定されていたとしても、データ・ソース B が使用され、ロードされます。

例

```
applyPropertiesAfterBookmark="true"
```

bookmarkFilter

ブックマークを保管し、そこからロードするデフォルトの場所を指定します。この場所を使用してブックマークのグループ化や可視性を提供できるため、アプリケーション開発者は、エンド・ユーザーに使用可能なブックマークのセットを制御することができます。指定するフィルターにより、ブックマークは DB2 Alphablox リポジトリー中の通常のブックマーク・ディレクトリー (public、private、または group) の下の filterName というサブディレクトリーに保管されます。

さらに、bookmarkFilter プロパティーによって、複数の Blox や複数のアプリケーションを通してブックマークを共有できるようになります。

データ・ソース

すべて

構文

bookmarkFilter="*filterName*"

引数	デフォルト	説明
filterName	なし	この引数には 2 つの形式があります。1 つはサブディレクトリー名を指定するストリングです。 2 番目は以下の形式のコンマで区切られたリストのあるストリングです。 "*subDirectory*, name=*BloxID*, application=*AppContext*, user=*UserName*" name、application、および user 文節はすべてオプショナルで、subDirectory 文節は必要です。

使用法

bookmarkFilter プロパティーにより、各 Blox のブックマークをカテゴリー化することができ、ブックマークにさらに柔軟性を加えることができます。

例えば、1 つは marketing、もう 1 つは sales ための 2 つのエントリー・ポイントのある単一の PresentBlox (例えば、異なるデータ・ソースから異なる照会を実行

する 2 つのリンクがあり、結果を同じ PresentBlox で表示する) を考慮します。bookmarkFilter プロパティーを最初のリンクに設定し、marketing のすべてのユーザーが、アプリケーションのマーケティングの部分から作成されたブックマークを見ることができるようにし、bookmarkFilter プロパティーを 2 番目のリンクに設定して sales のすべてのユーザーが、アプリケーションの販売の部分から作成されたブックマークを見ることができるようにします。最終的な結果として、sales のユーザーと marketing のユーザーは、両方とも同じ PresentBlox でデータを表示しているのに、別々のブックマークのセットを見ることになります。このスキームにより、ページ上の Blox の数を最小限にとどめられます。Blox 数の最小化は資源の最適化に役立つ方法です。ページ上で、複数のページが複数のアプレットで最新表示することでブラウザーが予期しない仕方で振る舞うことのある Netscape ブラウザーで、これは特に有用です。

構文セクションで説明されている filterName 引数の 2 番目の形式は、Blox がプログラマチックな詳細 (例えば、ユーザー・プロファイル、四半期など) に基づいて自動的に作成されるアプリケーションで有用です。そのような動的な Blox を作成すると、Blox が作成される度に異なる名前を持つことになり、異なる Blox インスタンスを通して一貫したブックマークのセットを持つことが困難になります。bookmarkFilter をある基準 (BloxID、アプリケーション、ユーザー) に基づいて設定することにより、ブックマークの望まれるセットをそれに合ったユーザーが使用できるようにすることができます。

例

以下の例では、DB2 Alphablox リポジトリーの marketing アプリケーションのサブディレクトリー、marketingBookmarks にブックマークが保管され、検索されるように設定します。

```
bookmarkFilter="marketingBookmarks, name="myPresentBlox", application=marketing"
```

bloxEnabled

Blox インターフェースが対話式であるか、グレー表示されるかを指定します。

データ・ソース

すべて

構文

```
bloxEnabled="enable"
```

引数	デフォルト	説明
enable	true	対話性を使用可能にし、Blox を表示するには true を指定し、対話性を使用不可に設定し、 Blox をグレー表示するには false を指定します。

使用法

値 false は、対話式でないグレー表示された Blox を表します。値 true は、ユーザーがドリルアップ、ドリルダウンしたり、チャート・タイプを変更したりすることのできる対話式インターフェースを表します。Blox を表示したい (グレー表示で

なく) が、ユーザーがデータと対話することを望まない場合には、
bloxui:component タグの clickable 属性を使用してください。 423 ページの『例 3: PresentBlox をクリックできないように設定する』を参照。

例

```
bloxEnabled="false"
```

bloxName

Blox の名前を指定します。これはオプショナル属性で、Blox の名前とそれに対応する JavaScript 名の動的設定を可能にする拡張機能です。

データ・ソース

すべて

構文

```
bloxName="bloxName"
```

使用法

Blox タグを使用して Blox を定義する場合、最も外側の Blox を id 属性を使用して固有に識別する必要があります (ネストされた Blox は別の id を持つことができません)。この id は必須であり、動的に設定することはできません。以下の 2 つの目的で使用されます。

- JSP ページ内のスクリプト記述の変数名として。
- また、Blox の名前およびそれに対応して DB2 Alphablox の下で作成される JavaScript オブジェクトとして (ブラウザー内での Blox の JavaScript 作成に使用される)。

オプションの bloxName 属性が指定されていない場合、Java スクリプト変数およびサーバー上の Blox オブジェクトの名前の両方として id が使用されます。

1 つの id を Blox 名および Java スクリプト変数名として使用することで、開発の要件がすべて満たされるはずです。幾つかのケースでのみ、この 2 つを分離して Blox 名を動的にタグで作成する必要のある場合があるかもしれません。また、これはサーバー・サイド・コードを再利用する (クライアント・サイド call() メソッドを使用して、Blox でサーバー・サイド・コードを実行する、または以下の例で示されるようにするなど) のにも役立ちます。Blox に bloxName の値を指定する場合、以下のようになります。

- この bloxName が、Blox DB2 Alphablox をこのオブジェクトの名前として識別する名前になる
- この bloxName が、レンダリングされた JavaScript オブジェクトの名前 (この Blox を参照する際に JavaScript コード中に使用される) になる
- id は、JSP ページで使用する Java スクリプト変数としてのみ使用されることになる

スクリプト変数名と Blox 名の分離: 以下の例では、bloxName が指定された場合の、スクリプト変数名と Blox 名の間の違いを示します。コードは、salesDataGrid と呼ばれる GridBlox を作成します。これは最初は表示されません (visible="false")。

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

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

このグリッドには、id の値である、そのスクリプト変数名を使用してスクリプト記述することに注意してください。処理ロジックがなされた後、この GridBlox は以下のように <blox:display> タグを使用して表示されます。

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

bloxName 属性の値の動的な設定: setAlerts.js という名前の JSP ページがある場合、これは以下のように渡される任意の GridBlox で指定されるしきい値に対するセル・アラート・フォーマットを設定します。

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

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

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

あるいは、汎用 JSP コードを再利用する場合には、以下のようにします。

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

ここで、gridDefaults.jsp は以下を行います。

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

要約:

- id は必須で、bloxName はオプショナルです。

- Blox の名前がタグで指定されている場合、それが bloxName 属性の値です。提供されていない場合は、id 属性の値が Blox 名および Java スクリプト変数名の両方として使用されます。
- bloxName 属性の値が与えられていない限り、この文書セットを通じて、「Blox 名」という語句は id 属性の値を指します。
- ほとんどの場合、id のみを指定する必要があり、bloxName を気にする必要はありません。スクリプト変数名と Blox 名を区別する必要のある場合にのみ、bloxName を指定する必要があります。
- bloxName 属性の値をあえて指定する場合には:
  - id は、JSP ページでスクリプト記述する、Java スクリプト変数名です。
  - bloxName は、それを参照する際に使用する Blox 名です。

注: bloxName は数値であってはならず、数値で開始することもなく、次のような特殊文字も含めないでください。~、!、@、#、$、%、^、&、*、-、+、=、(、)、?、<、>、/、:、;、'、または "。

注: getBloxName() メソッドをネストされた Blox で呼び出すと、その Blox に生成された名前を戻します。

例

以下のコードは、myGrid という名前のローカル・スクリプト変数と、salesGrid という名前のグリッド・ピアを作成します。

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

このグリッドにスクリプト記述する場合、グリッドのスクリプト変数名 (id) を使用します。以下のコードは getBloxName() メソッドの結果を示します。コメントは戻り値を示します。

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

関連項目

43 ページの『id』

bloxRef

使用する別の Blox の名前を指定します。bloxRef 属性は DataBlox (blox:data) および display (blox:display) カスタム・タグ・ライブラリーを通して使用できます。

データ・ソース

すべて

構文

bloxRef="*bloxName*"

使用法

ネストされた Blox で bloxRef タグ属性を使用し、別の Blox として作成された Blox を参照します。

例

以下の DataBlox は、HTML ページの <head> セクションで作成されました。

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

その後この DataBlox を他の Blox (例えば、GridBlox) でネストされた Blox として参照できます。以下のように bloxRef 属性で参照します。

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

enablePoppedOut

これは、ContainerBlox から継承されたプロパティーです。詳しい説明は、175 ページの『enablePoppedOut』を参照してください。

height

ページ上の Blox の高さを指定します。

データ・ソース

すべて

構文

height="*height*"

使用法

Blox の表示域の高さを指定します。値はピクセルで表す (height="300") か、またはブラウザー表示域に対する比率で表す (height="40%") ことができます。

helpTargetFrame

ユーザー・ヘルプが表示されるターゲット・ブラウザー・ウィンドウまたはフレーム・セット・フレームを示します。

データ・ソース

すべて

構文

helpTargetFrame="*helpTargetFrame*"

引数	デフォルト	説明
helpTargetFrame	"AlphabloxHelp"	ブラウザー・ウィンドウまたはフレーム・セットを識別するストリング。

使用法

デフォルトは AlphabloxHelp で、これは別のブラウザー・ウィンドウです。

例

```
helpTargetFrame ="Browser Window Name"
```

id

Blox の名前を指定します。その後この名前は他の Blox、または Java、あるいは JSP ページ上の JavaScript コードから参照することができます。

データ・ソース

すべて

構文

id="*idString*"

引数	デフォルト	説明
idString	なし	Blox の名前を表す識別ストリング。 idString は、他の Blox タグから、bloxRef 属性を使用して参照できます。

使用法

id 属性は外部 Blox でのみ有効です。ネストされた Blox は、id 属性を持つことができません。オプショナル bloxName 属性を指定する場合、id は JSP ページ上の Java スクリプト変数名としてのみ使用されます。値 bloxName がサーバー上に作成された Blox ピアの名前、およびレンダリングされた JavaScript オブジェクトの名前になります。

注: id は数値であってはならず、数値で開始することもなく、次のような特殊文字も含めないでください。~、!、@、#、$、%、^、&、*、-、+、=、(、)、?、<、>、/、:、;、'、または "。

関連項目

39 ページの『bloxName』

localeCode

数値のフォーマット設定用のロケールを設定します。このプロパティーを使用して、DB2 Alphablox が稼働している地域とは異なる地域での数値のフォーマットを表示することができます。例えば、コードをアプリケーションに追加して localeCode プロパティーをユーザーによって設定し、フランスのユーザーがその地域用にフォーマット設定された数字を見ることができ、ドイツのユーザーはその地域用にフォーマット設定された数値を見ることができるようにする必要があります。

データ・ソース

すべて

構文

localeCode="*locale*"

引数	デフォルト	説明
locale	なし	これは 2 つの部分から成るパラメーターです。最初の部分は 2 桁の言語コードで 2 番目の部分は 2 桁の国別コードです。これらは下線 (_) で区切られています。言語コードは、以下から検索することができます。 http://lcweb.loc.gov/standards/iso639-2/langcodes.html 国別コードは、以下で検索することができます。 ftp://ftp.ripe.net/iso3166-countrycodes.txt 両方とも、2 桁のコードが使用されます。

使用法

localeCode プロパティーは外部 Blox に (例えば、PresentBlox 上に) 設定してください。このプロパティーを内部 Blox に設定しても、外部 Blox に影響がありません。

外部 Blox に (例えば、PresentBlox 上に) localeCode を設定するために setLocaleCode() メソッドを使用する場合、内部 Blox (例えば、GridBlox) は localeCode プロパティーにその値を使用します。ただし、内部 Blox (例えば、GridBlox) で getLocaleCode() を呼び出すと、使用されている値ではなく、元の値が戻されます。

ユーザー・プロファイルに基づいて localeCode プロパティーを個別設定するには、異なる国のユーザーに対して有効な値でカスタム・ユーザー・プロパティーを定義します。その後、アプリケーション・ページで RepositoryBlox を作成し、カスタム・ユーザー・プロパティーの値 (RepositoryBlox getUserProperty() メソッドで) を取得し、それから localeCode プロパティーをそれに応じて設定 (setLocaleCode() メソッドを使って) します。

localeCode プロパティーを設定しない場合、デフォルト値は DB2 Alphablox が稼働している地域です。localeCode プロパティーを同じページ上で、異なる Blox に対して異なる値に設定しないでください。

例

ロケール・コードを英語、米国に設定するには以下のようにします。

```
localeCode="en_US"
```

ロケール・コードを英語、英国に設定するには以下のようにします。

```
localeCode="en_GB"
```

maximumUndoSteps

メニュー・バーの取り消しボタンで追跡するステップの最大数を指定します。

データ・ソース

すべて

構文

```
maximumUndoSteps="steps"
```

引数	デフォルト	説明
steps	50	追跡するステップの最大数。

使用法

このプロパティーは、PresentBlox、GridBlox、ChartBlox、DataLayoutBlox、および PageBlox に適用されます。このプロパティーを 0 に設定した場合、「取り消し」および「再実行」ボタン、そしてメニュー項目は、ツールバーおよびメニュー・バーから除去されます。

menubarVisible

Blox の上部にテキスト・ベースのメニュー・バーが表示されるかどうかを指定します。

データ・ソース

すべて

構文

```
menubarVisible="visible"
```

引数	デフォルト	説明
visible	true	メニュー・バーを表示するには true に設定し、メニュー・バーを非表示にするには false に設定します。デフォルトは true です (デフォルト・アプリケーションのレンダリング・モードが DHTML に設定されている場合)。

使用法

メニュー・バーとそのドロップダウン・メニューの内容は自動的に Blox の内容と一致します。

例

```
menubarVisible="true"
```

noDataMessage

Blox にデータがない場合に Blox にストリングが表示されるように設定します。

データ・ソース

マルチディメンション

構文

noDataMessage="*message*"

引数	デフォルト	説明
message	"Data not available"	任意のストリング。

使用法

新しいメッセージが設定された場合、次回に結果セットがデータなしで戻ってくる (updateResultSet() または connect() への呼び出しなど) までは、画面は更新されません。DataLayoutBlox では、このプロパティーは無視され、使用可能なデータがない場合にはメッセージは表示されません。

例

```
noDataMessage="No data at this time"
```

poppedOut

これは、ContainerBlox から継承されたプロパティーです。詳しい説明は、176 ページの『poppedOut』を参照してください。

poppedOutHeight

これは、ContainerBlox から継承されたプロパティーです。詳しい説明は、176 ページの『poppedOutHeight』を参照してください。

poppedOutTitle

これは、ContainerBlox から継承されたプロパティーです。詳しい説明は、177 ページの『poppedOutTitle』を参照してください。

poppedOutWidth

これは、ContainerBlox から継承されたプロパティーです。詳しい説明は、178 ページの『poppedOutWidth』を参照してください。

removeAction

どのデータ分析アクション (ある場合) を右クリック・メニューおよび「データ」メニューから除去するかを指定します。

データ・ソース

すべて

構文

removeAction="*dataActions*"

引数	デフォルト	説明
dataActions	空ストリング	削除するアクションの、コンマで区切られたリスト。

使用法

リスト内の有効な項目は以下のとおりです。

- Find
- Drill Up
- Drill Down
- Pivot
- Data Sort
- Remove Only
- Keep Only
- Hide Only
- Show Only
- Show All
- Expand All
- Show Bottom Level
- Show Siblings
- Swap
- Drill Through
- Member Filter
- Comments
- Traffic Lights

例

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

render

アプリケーション・ページ上の特定の Blox に配信フォーマットを指定します。

データ・ソース

すべて

構文

render="*renderMode*"

引数	デフォルト	説明
renderMode	dhtml	ページがレンダリングされるモード。可能な値は、以下の表を参照してください。

使用法

このプロパティーを使用することで、同じページ上で Blox の異なる配信フォーマットが使用可能になります。このプロパティーを個々の Blox に設定すると、アプリケーションの URL のレンダリング属性がオーバーライドされます。render 属性がページ上のすべての Blox に適用されるのに対し、このプロパティーは特定の Blox に適用されます。したがって、特定のフォーマットでのみ Blox が配信されるようにするには、Blox で render プロパティーを使用してください。

可能な値は以下のとおりです。

値

dhtml	完全に対話式の DHTML フォーマットでレンダリングします (デフォルト)。 JSP ページに <blox:header> タグが必要です。
html	静的な HTML フォーマットにレンダリングします。JSP ページに <blox:header> タグが必要です。
printer	印刷に適したフォーマットでレンダリングします (多くのブラウザーは、対話式 Java アプレットの出力の印刷をサポートしません)。 JSP ページに <blox:header> タグが必要です。
xls	Microsoft Excel へのエクスポートに適したフォーマットでレンダリングし、MIME タイプを XLS に設定します。 **注:** ページを Excel で開けるように MIME タイプを設定するには、JSP ページに <blox:header> タグを置かなければなりません。 <blox:header> タグについての情報は、25 ページの『HTML <head> 内の <blox:header> タグ』を参照してください。
xml	XML フォーマットでレンダリングします。このフォーマットは DataBlox にのみ適用されます。詳しくは、545 ページの『付録 B. Alphablox XML Cube の使用』を参照してください。 **注:** Microsoft Excel フォーマットにレンダリングするには、URL render=xls 属性を使用しなければなりません。

rightClickMenuEnabled

Blox ユーザー・インターフェースで右クリック・メニューをオンにするかオフにするかを指定します。

データ・ソース

すべて

構文

rightClickMenuEnabled="*enabled*"

引数	デフォルト	説明
enabled	true	true に設定された場合、右クリック・メニューは使用可能にされ、ユーザーはデータ分析または操作タスクを実行することができます。false に設定された場合、右クリック・メニューは使用不可になります。デフォルトは true です。

使用法

GridBlox および ChartBlox のみに、さまざまなデータ・ナビゲーション・オプションのある右クリック・メニューがあります。

visible

ページ上で Blox を表示するかどうかを指定します。

データ・ソース

すべて

構文

```
visible="boolean"
```

引数	デフォルト	説明
boolean	true	Blox がページにレンダリングされるようにするには true に設定し、Blox の作成は望むもののページに表示しない場合には、false に設定します。

使用法

Blox を作成しても表示しないためには、visible プロパティーを false に設定します。<blox:display> タグを使用して、後に Blox を表示できます。デフォルト値は true です。

visible プロパティーを ToolbarBlox で使用する場合には、ツールバーをオフにすることでユーザーに及ぶ影響を慎重に考慮してください。ほとんどのアプリケーションでは、Blox ツールバーまたはメニュー・バーを通して提供される機能を提供することが必要です。メニュー・バーが Blox でオフにされた場合、「取り消し」/「再実行」ボタン、PDF/Excel へのエクスポートなどのオプション、またグリッドのオン/オフ、チャート、ページ・フィルター、およびデータ・レイアウト・パネルがツールバーを通してのみ使用可能になります。

width

ページ上の Blox の幅を指定します。

データ・ソース

すべて

構文

```
width="width"
```

引数	デフォルト	説明
width	なし	有効な HTML 幅値を表すストリング。

使用法

Blox の表示域の幅を指定します。値はピクセルで表す (`width="500"`) か、またはブラウザー表示域に対する比率で表す (`width="50%"`) ことができます。

第 5 章 AdminBlox タグ・リファレンス

この章には AdminBlox の参照資料が含まれています。Blox についての一般的な参照情報は、21 ページの『第 3 章 一般 Blox リファレンス情報』を参照してください。このリファレンスの使用方法については、1 ページの『第 1 章 このリファレンスの使用法』を参照してください。

AdminBlox の概説

AdminBlox では、DB2 Alphablox ホーム・ページにある管理タブを通して設定される、サーバー、ユーザー、グループ、役割、データ・ソース、Alphablox ログ・システム、およびアプリケーションに関する情報へのプログラマチックなアクセスが提供されています。

DB2 Alphablox ホーム・ページにある管理タブにより、サーバー・ログ・ファイル名、メッセージ・レベル、Telnet コンソール・ポート、クラスタリング・オプション、および Telnet ユーザー名とパスワードといったプロパティーをサーバー管理者が指定する手段が提供されています。管理タブ下のデータ・ソース、ユーザー、グループ、役割、キューブ、およびアプリケーション・リンクによって、管理者は DB2 Alphablox によって処理されるようデータ・ソース、ユーザー、グループ、役割、キューブ、およびアプリケーションを定義できます。いったん指定すると、この情報はリポジトリーに保管されます。アプリケーション開発者は AdminBlox とその関連オブジェクトおよびメソッドを通してその情報にアクセスできます。

	メソッド		戻されるオブジェクト
AdminBlox ->	`getApplication(...)` `getApplications()`	->	アプリケーション
	`getCube(...)` `getCubes()`	->	Cube
	`getDataSource(...)` `getDataSources()`	->	データ・ソース
	`getGroup(...)` `getGroups()`	->	グループ
	`getLog(...)`	->	ログ
	`getRole(...)` `getRoles()`	->	役割
	`getServer(...)`	->	サーバー
	`getUser(...)` `getUsers()`	->	ユーザー

AdminBlox と RepositoryBlox は両方ともリポジトリーに保管された情報へのアクセスを可能にしますが、AdminBlox は特に、サーバー、アプリケーション、ユーザ

ー、およびデータ・ソース上の一般の管理詳細用です。例えば、サーバーに使用可能なすべてのデータ・ソースに関する情報を得たり、アプリケーションのセッションのタイムアウト設定を検索したり、あるいは特定の SMTP サーバーを識別したりすることができます。他方、RepositoryBlox では現行のユーザー、アプリケーション、およびカスタム・プロパティーに関する情報へのアクセスが得られます。

AdminBlox では、特定のサーバー・モニターや管理の必要に合わせて、独自の管理アプリケーションを構築することができます。この機能のため、アプリケーション・レベルで適切なアクセス制御がなされるように配慮してください。AdminBlox には組み込まれたセキュリティーはありません。

Application オブジェクト

Application オブジェクトはリポジトリー内の Alphablox アプリケーションを表します。これは、管理タブの下でアプリケーション・リンクを通して指定された情報を入手するメソッドを提供します。提供される getter メソッドで、アプリケーションのデフォルト保管状態、表示名、定義済みヘッダー・リンク、およびセッションがタイムアウトになる非アクティブの時間といった情報を検索することができます。

Cube オブジェクト

Cube オブジェクトは、リポジトリー内の Alphablox キューブを表します。 これは、管理タブの下で Cube リンクを通して指定されたキューブ定義にアクセスするメソッドを提供します。また、 `start()`、`stop()`、`refreshData()` などのキューブを管理するメソッドも提供します。 キューブの定義 (CubeDefinition オブジェクト) を取得すると、ディメンション、レベル、ディメンションの算出メンバーとデフォルトのメンバーなど、キューブに関する概要のメタデータにアクセスできます。 また、最新表示の間隔やキューブ開始時のキューブ・キャッシュ・シード照会など、キューブに関する管理情報の一部も含まれています。

DataSource オブジェクト

DataSource オブジェクトはリポジトリー内の Alphablox データ・ソースを表します。これは、管理タブの下でデータ・ソース・リンクを通して指定された情報を入手するメソッドを提供します。提供される getter メソッドで、データ・ソースのアダプター・タイプ、アプリケーション/カタログ/データベース/スキーマ名、データ・ソースへログインするためのデフォルト・ユーザー名とパスワード、および戻す最大列と行といった情報を検索できます。

User オブジェクト

User オブジェクトはリポジトリー内の Alphablox ユーザーを表します。これは、管理タブの下でユーザー・リンクを通して指定された情報へアクセスするメソッドを提供します。提供される getter および setter メソッドで、ユーザー名、E メール・アドレス、および 1 次グループ関連といった情報を検索、変更することができます。

Group オブジェクト

Group オブジェクトはリポジトリー内の Alphablox グループを表します。これは、管理タブの下でグループ・リンクを通して指定された情報へアクセスするメソッド

を提供します。提供される getter および setter メソッドで、どのユーザーまたはサブグループが特定のグループに属するかなどの情報を検索、変更することができます。

Role オブジェクト

Role オブジェクトはリポジトリー内の Alphablox 役割を表します。これは、管理タブの下で役割リンクを通して指定された情報へアクセスするメソッドを提供します。提供されるメソッドで、どのユーザーまたはグループが特定の役割に属するかなどの情報を検索、変更することができます。

Log オブジェクト

Log オブジェクトは、メッセージを Alphablox ログ・システムへ配置するために使用されます。メッセージ・レベルは、軽微なものから最も重大なものまでの重大度順で、DEBUG、VERBOSE、INFO、SYSTEM、WARNING、および ERROR です。これらのメッセージは、ログ・ファイルおよび登録済みコンソールに、それらのメッセージ・レベル設定に応じてログされます。ログ・ファイルは、DB2 Alphablox リポジトリー中の `<alphablox_dir>/repository/logs/<instance_name>/logs` の下にあります。

Server オブジェクト

Server オブジェクトは、DB2 Alphablox によってリポジトリーに保管された、サーバーに関連した情報を表します。これは、管理タブの下でサーバー・リンクを通して指定された情報を入手するメソッドを提供します。

サーバー・メッセージ・レベル

DB2 Alphablox では、サーバーのモニターおよびデバッグの目的でメッセージをログするための 7 段階のメッセージ・レベルが提供されています。管理者は、「管理」タブ下のシステム・ページで「新規ログ開始メッセージ・レベル」および「新規ログ終了メッセージ・レベル」を設定できます。これらの 2 つのプロパティー値を設定すると、指定のレベルの範囲内のメッセージを含むログが作成されます。

以下の表では、メッセージ・レベル定数およびそれぞれのレベルでログされるメッセージの種類の説明をリストします。また、com.alphablox.blox.repository パッケージの Server オブジェクトの `levelIntToString()` および `levelStringToInt()` メソッドによって使用されるストリングおよび整数値も示します。Server オブジェクトにアクセスするには、`AdminBlox.getServer()` メソッドを使用します。

メッセージ・レベル定数	ストリング	値	説明
MESSAGE_LEVEL_DEBUG	DEBUG	1	システムのデバッグを援助するメッセージ。
MESSAGE_LEVEL_VERBOSE	VERBOSE	2	すべてのシステム・メッセージ。
MESSAGE_LEVEL_INFO	INFO	3	管理者のアクションは必要とされない小さなシステム・イベント。
MESSAGE_LEVEL_SYSTEM	SYSTEM	4	メジャーの、通常のシステム・イベントのメッセージ (新規セッションなど)。

メッセージ・レベル定数	ストリング	値	説明
MESSAGE_LEVEL_WARNING	WARNING	5	リカバリー可能な小さなエラーが生じたことを示し、調査する問題を管理者に示唆するメッセージ。
MESSAGE_LEVEL_ERROR	ERROR	6	操作を完了できず、リカバリー不能なエラーが生じたことを示すメッセージ。
MESSAGE_LEVEL_FATAL	FATAL	7	サーバーの終了の原因となり得る致命的エラーが生じたことを示すメッセージ。

AdminBlox JSP カスタム・タグ構文

DB2 Alphablox タグ・ライブラリーは、それぞれの Blox を作成するために JSP ページで使用するカスタム・タグを提供します。このセクションでは、AdminBlox を作成するためのカスタム・タグの作成方法を説明します。すべての属性を含むタグのコピー・アンド・ペースト・バージョンについては、519 ページの『AdminBlox JSP カスタム・タグ』を参照してください。

構文

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

ここで、それぞれ以下のとおりです。

attribute は属性表にリストされている属性の 1 つです。

value は、属性の有効値です。

属性は以下のいずれかになります。

属性
56 ページの『id』
56 ページの『bloxName』

使用法

各カスタム・タグには 1 つ以上の属性を含めることができ、それぞれを 1 つ以上のスペースまたは改行文字で区切ります。余分のスペースまたは改行文字は無視されます。読みやすくするため、同じインデントでそれぞれ別々の行に属性を並べることができます。

属性リストの終わりのタグを以下のようにして、終了タグ </blox:admin> を省略表現で置換できます。

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

例

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

AdminBlox の例

この例では、AdminBlox を通してメッセージを Alphablox ログ・システムへログする方法を例示します。これは特に、モニター、ロギング、およびデバッグ問題に役立ちます。これにより、メッセージと例外の両方をログすることが可能です。

ロギング・メカニズムはマルチスレッドであるため、メッセージの順序が予想と少々異なる場合があることに注意してください。

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

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

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

1. com.alphablox.blox.repository.Log クラスをインポートする。
2. `<blox:admin>` タグを使用して AdminBlox を追加する。
3. AdminBlox の `getLog()` メソッドを通して Log オブジェクトにアクセスする。
4. `sendMessage(...)` を使用して、ログにメッセージを送信する。
5. `sendException(...)` を使用して、ログに例外を送信する。

これで、以下の項目がログ・ファイルに生成されます。

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

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

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

```
at
com.ibm.ws.webcontainer.servlet.StrictLifecycleServlet._service(StrictLifecycleServlet.ja
va:174)
[ more stack traces below omitted... ]
```

AdminBlox タグ属性

このセクションでは、AdminBlox によってサポートされているタグ属性について説明します。AdminBlox を使用してサーバー・リソースをモニターして管理するには、AdminBlox API を使用する必要があります。AdminBlox メソッドの詳細については、Javadoc 内の com.alphablox.blox.AdminBlox を参照してください。

id

これは共通の Blox タグ属性およびプロパティーです。詳しい説明は、43 ページの『id』を参照してください。

bloxName

これは共通の Blox タグ属性およびプロパティーです。詳しい説明は、39 ページの『bloxName』を参照してください。

第 6 章 BookmarksBlox タグ・リファレンス

この章では、ブックマークの全般的な説明を行い、BookmarksBlox タグ属性の参照資料を提供します。 Blox についての一般的な参照情報は、21 ページの『第 3 章 一般 Blox リファレンス情報』を参照してください。このリファレンスの使用方法については、1 ページの『第 1 章 このリファレンスの使用法』を参照してください。

BookmarksBlox の概説

Blox ユーザー・インターフェースを使用すれば、エンド・ユーザーは、後で検索する時にアクセス可能なプライベート、パブリック、またはグループでデータ・ビューにブックマークを付けることができます。ビューのブックマークの設定は、ツールバーの「ブックマーク」ボタンまたは右クリック・メニューから「ブックマーク」オプションを使用して行えます。ユーザーは、自分にとって可視である既存のブックマークをロード、削除、または名前変更することもできます。

ブックマークというのは、実質的にはプロパティー・セットの集合です。各ブックマークには、以下の情報が含まれます。

- 状態が保管されている Blox の名前
- ブックマーク追加時の、Blox の初期アプリケーション状態から現在の状態へのプロパティーの変更
- ブックマークを所有するユーザーの名前
- ブックマークの可視性
- ブックマークに関する説明

ブックマークの保管時には、Blox の現在の (データとのユーザー対話が行われた後の) 状態と初期状態 (デフォルトのプロパティー値か Blox 作成時に指定された値) の差だけがリポジトリーに保管されます。ブックマークのロード時には、リポジトリーに保管された Blox プロパティーの情報に基づいて、データ・ソースからライブ・データを取得します。

さまざまな API を持つ BookmarksBlox を使用すると、ブックマークをプログラマチックに作成および管理することができ、ブックマーク・プロパティーを動的に設定することが可能になります。たとえば、ブックマークに保管されているデータ照会を動的に変更して、時系列レポートや現在の四半期のデータを常時取り出すレポートを作成することができます。カスタム・ブックマーク・プロパティーを使用して、レポート・レイアウトの選択をユーザーごとに保管したり、独自のセキュリティーをインプリメントしたりすることができます。データ・ソース内のメンバー名

つまりアウトラインに変更があった場合、ブックマークに保管された照会を変更することができます。独自のブックマーク管理ユーザー・インターフェースを作成することさえできます。

BookmarksBlox API を使用するには、BookmarksBlox をページに追加します。これにより、それぞれのブックマークに Bookmark オブジェクトとしてアクセスできます。

ブックマークの概念と機能

ブックマーク付けは豊富な API を持った強力な機能で、この API を使用することによりさまざまなカスタム・アクションを実行することができます。このセクションでは、ブックマークおよびそれに関連した Bookmark オブジェクトに関して、以下のかぎとなる概念と機能を説明します。

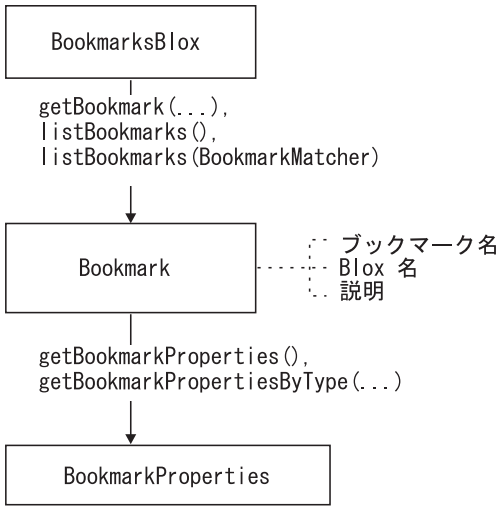
- 58 ページの『ブックマークとは ?』
- 59 ページの『Blox のデフォルト状態と初期アプリケーション状態と Blox の現在状態』
- 59 ページの『カスタム・ブックマーク・プロパティー』
- 60 ページの『ブックマークの可視性』
- 61 ページの『Blox タイプおよびバインディング』
- 62 ページの『ブックマーク・マッチャーとブックマーク・フィルター』
- 63 ページの『ブックマーク・イベントとイベント・フィルター』
- 64 ページの『逐次化照会とテキスト形式の照会』
- 65 ページの『Bookmark オブジェクトのための静的フィールド』
- 66 ページの『ブックマーク名に関する制約事項』

ブックマークとは ?

ブックマークとは、プロパティー・セットの集合です。ブックマークにはそれ自体に、アプリケーション、説明、名前、および可視性などのプロパティーがあります。また、個々の Blox に関する情報も保管します。この Blox は、ネストされた Blox を持たない独立型 Blox (DataBlox など) の場合も、ネストされた Blox を持つ Blox (PresentBlox など) の場合もあります。たとえば、PresentBlox で Bookmark を追加する場合、ネストされた個々の Blox についての情報も保管されます。

ブックマークの名前、Blox 名、所有者の名前、および可視性などの検索条件を指定することにより、BookmarksBlox API を使用して、特定のブックマークにアクセスすることができます。さらに、BookmarksBlox API を使用して、プロパティーを変更したり、ブックマークを別の Blox に適用することさえ可能です。

以下の図に、BookmarksBlox のオブジェクト階層を示します。



注: Bookmark および BookmarkProperties オブジェクトにアクセスするには、JSP に次のインポート・ディレクティブを追加する必要があります。

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

注: ブックマーク名に使用できる文字は、A-Z、a-z、0-9、およびアンダースコアー (_) だけです。

Blox のデフォルト状態と初期アプリケーション状態と Blox の現在状態

BookmarkProperties オブジェクトは、初期 Blox 状態と同じではないプロパティーだけを保持します。たとえば、ChartBlox の chartType がタグで設定されていない場合、デフォルトのチャート・タイプは「垂直バー、横並び、立体効果」です。ブックマークが保管されて、現在表示されているチャート・タイプが「垂直バー、横並び、立体効果」である場合、Blox のために保管されるプロパティーのリストにチャート・タイプ・プロパティーは存在しません。

Blox のデフォルト状態以外に、ChartBlox タグを使用してチャート・タイプを「円グラフ」に設定することが可能です。このように指定されたプロパティーは、他の指定されていない Blox プロパティーのデフォルト値と共に、Blox がどのようにインスタンス化されてレンダリングされるかを決定します。これが、初期アプリケーション状態です。チャート・タイプを変える、ドリルダウンする、メンバーを非表示にする、軸を交換する、他のセル・バンド・スタイルを変更するなど、ユーザーがデータと対話を行うと、この状態が変化します。現在のビューでブックマークを保管すると、そのブックマークに保管されるのは、初期アプリケーション状態と現在の Blox 状態の差です。

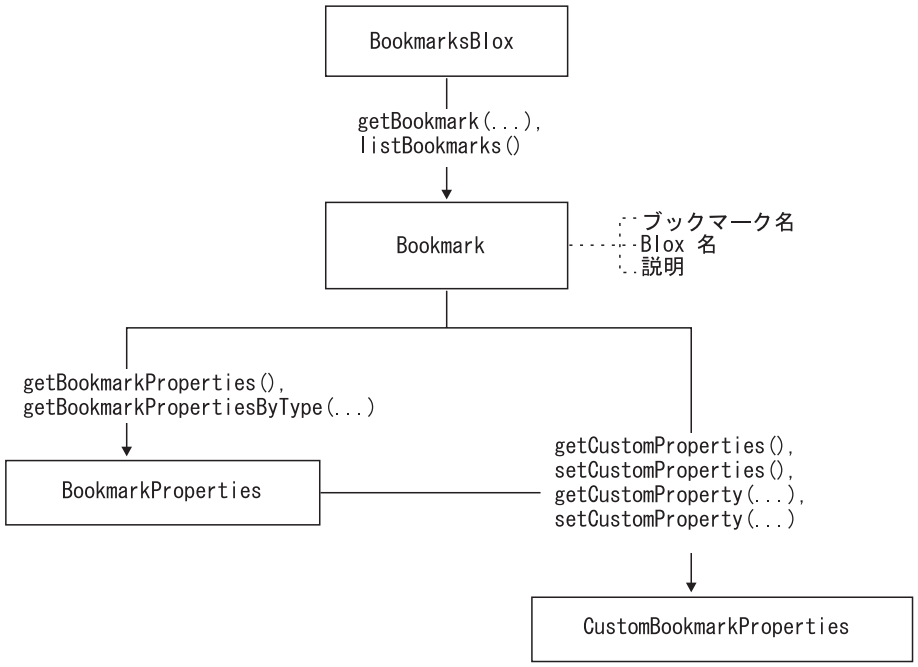
カスタム・ブックマーク・プロパティー

デフォルトのブックマーク・プロパティー以外に、ブックマークにカスタム・プロパティーを追加することもできます。 RepositoryBlox で使用できるカスタム・ユーザー・プロパティーやカスタム・アプリケーション・プロパティーと同様、カスタム・ブックマーク・プロパティーを使用して、アプリケーションで必要などんな情報でも、名前と値のペアにして保管することができます。たとえば、ブックマークにナビゲーション・ツリー・メニューを作成したいという場合があります。カスタ

ム・ブックマーク・プロパティーを使用して、ツリー・メニューを動的に作成するためのフォルダー名を保管することができます。または、特定のユーザーやグループだけがナビゲーション・ツリーに特定のフォルダーが表示されるように、アクセス制御をインプリメントすることができます。こうしたプロパティーはブックマークの振る舞いに全く影響を与えませんが、カスタム・プロパティーの保管と取得が可能になります。

カスタム・ブックマーク・プロパティーは、「DB2 Alphablox 管理ページ (DB2 Alphablox Admin Pages)」を使用して定義しないという点と、RepositoryBlox 経由でアクセスしないという点で、カスタム・ユーザー/アプリケーション・プロパティーとは違います。カスタム・ブックマーク・プロパティーの作成とアクセスには、まず JSP ファイルに BookmarksBlox を追加してください。そうすれば、以下を行うことができます。

- `BookmarksBlox.getBookmark(...).setCustomProperties()` メソッドを使用して、カスタム・プロパティーを設定する
- `BookmarksBlox.getBookmark(...).getCustomProperties()` メソッドを使用して、1 つのブックマークと関連したすべてのカスタム・プロパティーを取得する
- `getCustomProperty(`*key*`)` メソッドを使用して、個々のカスタム・プロパティーにそのキーでアクセスする



ブックマークの可視性

ブックマークは、プライベートやパブリックにすることも、グループだけに表示されるようにすることもできます。デフォルトでは、(Blox ユーザー・インターフェースを使用して) ユーザーが、または (BookmarksBlox API を使用して) 開発者が別の指定をしない限り、ブックマークはプライベート・ブックマークとして追加されます。ブックマークの可視性は、以下の静的フィールドを使用してマークされます。

- PRIVATE_VISIBILITY
- PUBLIC_VISIBILITY

グループ可視性に関しては、グループの名前を使用して、グループ・ブックマークを取得します。

Blox タイプおよびバインディング

Blox ユーザー・インターフェースを使用して Blox でブックマークを保管する場合、ネストされた Blox すべてのプロパティーも初期状態と同じでないものは保管されます。 PresentBlox でブックマークが保管される場合、Blox が初期状態とは違う状態であれば、そのブックマークのフォルダーで、以下のようにネストされた Blox のために別個のフォルダーがある場合があります。

- <blox name>_data (プレゼンテーション Blox の外側で id を使用して明示的に定義しない暗黙の DataBlox を使用する場合)
- <blox name> (PresentBlox 用)
- <blox name>_chart
- <blox name>_datalayout
- <blox name>_grid
- <blox name>_page
- <blox name>_toolbar

BookmarksBlox API を使用して、ネストされた Blox のプロパティー・セットにその Blox タイプを指定してアクセスすることができます。 Blox タイプは静的フィールドを使用してマークされます。

- CHART_BLOX_TYPE
- DATA_BLOX_TYPE
- DATALAYOUT_BLOX_TYPE
- GRID_BLOX_TYPE
- PAGE_BLOX_TYPE
- PRESENT_BLOX_TYPE
- TOOLBAR_BLOX_TYPE

これにより、特定の Blox タイプのプロパティー・セットに直接アクセスして変更することができます。

ブックマークの物理位置は、バインディングと呼ばれます。バインディングは、論理名とコンテキストとのオブジェクトの関連です。これは Java Naming and Directory Interface (JNDI) に基づいており、統一されたインターフェースを持つ Java テクノロジーのアプリケーションが、データベース、ファイル、ディレクトリー、オブジェクト、およびネットワークをシームレスにナビゲートできるようにします。 J2EE コンテナーはこの情報を使用して、必要なリソースを見付けます。Bookmark オブジェクトで getContainer() および getBinding() メソッドを使用して、ブックマークの物理位置を取得することができます。

ブックマーク・マッチャーとブックマーク・フィルター

ある基準にかなうブックマークのリストを検索したり、アプリケーション、ユーザーの特定のグループ、または特定のユーザーのためのブックマークのリストを取得することがあります。アプリケーション、ユーザー、およびグループに特定の情報はリポジトリーに保管されるので、ブックマーク・フィルタリングをサポートするオブジェクトは、`com.alphablox.blox.repository` パッケージにあります。こうしたオブジェクトには、BookmarkMatcherApplications、 BookmarkMatcherGroups、BookmarkMatcherUsers、および BookmarkMatcherAll が含まれます。

BookmarkMatcherApplications オブジェクトは、どのアプリケーションがブックマークを所有するかに基づいてブックマークを検索するのに使用します。アプリケーション・ブックマークは、パブリック・ブックマークと同じことです。BookmarkMatcherGroups オブジェクトは、どのグループがブックマークを所有するかに基づいてブックマークを検索するのに使用します。 BookmarkMatcherUsers オブジェクトは、どのユーザーがブックマークを所有するかに基づいてブックマークを検索するのに使用します。ユーザー・ブックマークは、プライベート・ブックマークと同じことです。 BookmarkMatcherAll オブジェクトにより、特定のアプリケーション、ユーザー、可視性、または Blox 名のためのすべてのブックマークを検索することができます。

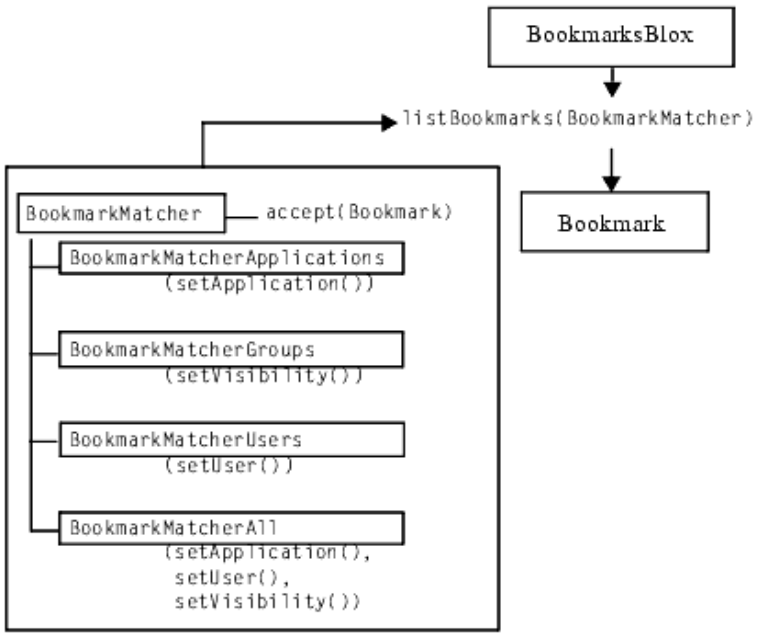
これらの BookmarkMatcher オブジェクトはすべて、以下の点を除いて、拡張 Java SDK File Filter クラスと同じように機能します。BookmarkMatcherUsers には、ユーザーのための特定のブックマークを検索するのにオプションで呼び出すことができる `setUser()` メソッドがあります。BookmarkMatcherApplications には、アプリケーションのための特定のブックマークを検索するのにオプションで呼び出すことができる `setApplication()` メソッドがあります。そして BookmarkMatcherGroups には、グループのための特定のブックマークを検索するのにオプションで呼び出すことができる `setVisibility()` メソッドがあります。これらのオブジェクトのそれぞれには、`accept()` メソッドがあります。このメソッドは、返却されるブックマークのリストに個々の Bookmark オブジェクトを含めるかどうかを調べるため、すべてのオブジェクトに関して呼び出されます。

実行したいブックマーク・マッチングのタイプに応じて、これらのオブジェクトを使用することも、それらを拡張することも可能です。何らかのタイプのカスタムのアプリケーション / グループ / ユーザー・ブックマーク・マッチングを行う場合、BookmarkMatcherApplications、BookmarkMatcherGroups、BookmarkMatcherUsers、または BookmarkMatcherAll を使用するか拡張することを推奨します。 DB2 Alphablox では、こうしたクラスはアプリケーション、グループ、およびユーザーの検索を高速に行うように最適化されているからです。

注: これらの BookmarkMatcher オブジェクトにアクセスするには、JSP に次のインポート・ディレクティブを追加する必要があります。

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

次の図では、これらのオブジェクトがどのように Bookmark オブジェクトに関連しているかを示します。



ブックマーク・イベントとイベント・フィルター

ブックマークを削除、編集、追加、または保管するためにユーザーがクリックしたときに、イベントをインターセプトすることができます。サーバー・サイドのイベント・フィルターを使用して、サーバーがイベントを処理する*前に* インターセプトすることが可能です。サーバー・サイドのイベント・フィルターを使用することには、通常 2 つのステップが関係します。

1. まず、共通 Blox メソッド addEventFilter() を使用して、特定のイベント・フィルター・オブジェクトを追加します。たとえば、以下のようにします。

```
<blox:present id="myPresent">
...
<%
  myPresent.addEventFilter(new LoadFilter() );
%>
</blox:present>
```

2. 次に、イベントがトリガーされたときに呼び出される、対応するイベント・フィルター・オブジェクト (BookmarkDeleteFilter、BookmarkLoadFilter、BookmarkRenameFilter、および BookmarkSaveFilter) と対応するメソッド (bookmarkDelete(BookmarkDeleteEvent)、bookmarkLoad(BookmarkLoadEvent)、bookmarkRename(BookmarkRenameEvent) 、または bookmarkSave(BookmarkSaveEvent)) をインプリメントする独自のクラスを作成します。たとえば、以下のようにします。

```
public class LoadFilter implements BookmarkLoadFilter
{
   public void bookmarkLoad( BookmarkLoadEvent bre )
   {
      //actions to take when the event is triggered
   }
}
```

ブックマーク・イベントとイベント・フィルターについて詳しくは、 72 ページの『例 5: サーバー・サイドの bookmarkLoad イベント・フィルターの使用』、35 ページの『ブックマークおよびアプリケーション状態に関連するタグ属性』 を参照してください。

逐次化照会とテキスト形式の照会

ブックマークの最初の作成時に、基礎となる DataBlox に設定されたオリジナルの照会と、現在のデータ・ビューを生成する関連した照会の間の差分 も保管されます。ファイル・リポジトリーの場合、2 つのファイル、*bookmarkName*.data と *bookmarkName*.query が、リポジトリーのブックマークの <blox name>_data フォルダーに保管されます。 .data ファイルは、テキスト・ファイルで、アプリケーション名、データ・ソース名、最後に実行した照会、およびページ軸メンバーなどのデータ・ソースに再接続するための基本的なプロパティーを保持します。このファイルの内容は、次のようになります。

```
Associated.query = q2report
ResultSet.Market = East,West,South,Central,Market
applicationName = SalesApp
connectOnStartup = true
dataSourceName = TBC
dimensionsOnPageAxis = {[null]}
parentFirst = {[null]}
query = <Row(Market) <ICHILD Market <Column(Year) Year !
```

テキスト形式の照会

.data テキスト・ファイルは、ブックマークが最初に追加されたときに作成されます。ブックマークが作成された方法に応じて、query 項目はあったりなかったりします。ブックマークが API を通じてブックマーク・オブジェクトの query プロパティーを設定することにより作成された場合は、テキスト・ファイル内に照会ストリングがあるはずです。ただし、ブックマークが保管し直されるときに、このファイルは更新されません。ユーザーが異なるデータ・ビューを使用してブックマークを保管し直すときに、このテキスト形式の照会が逐次化照会と同期しているようにするには、以下のようにします。

1. 最初に、DataBlox の generateQuery() メソッドを使用して、現在のデータ・ビューのためのテキスト形式の照会を取得します。
2. addEventFilter() 共通 Blox メソッドを使用して、ブックマークが保管し直されるたびにブックマークに保管されている照会を更新する BookmarkSaveFilter インターフェースをインプリメントするメソッドを追加します。

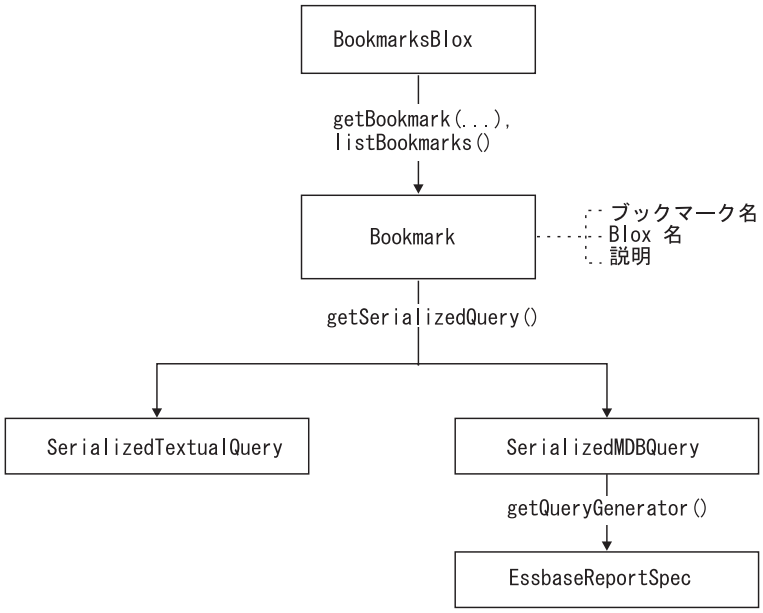
テキスト形式の照会を同期させておくと、データ・アウトラインが変わったときなど、後でテキスト形式の照会を変更することができます。 DB2 Alphablox は逐次化オブジェクトに一致するように結果セットを操作する必要がないので、テキスト形式の照会の方が効率的である場合があります。

ブックマークがロードされるときに、デフォルトでは、逐次化照会が使用されます。テキスト形式の照会を使用してブックマークをロードするには、DataBlox の textualQueryEnabled プロパティーに true を設定します。ブックマークがロードされる前に照会を変更する方法の例については、73 ページの『例 6: ブックマークの照会をロード時に取得する』を参照してください。

逐次化照会

.query ファイルには、データが入っていないという点を除いて、GridBlox 結果セットととてもよく似た逐次化オブジェクトが入っています。それは、軸、タプル、ディメンション、およびメンバーについての情報を保管します。軸、タプル、ディメンション、およびメンバーにプログラマチックにアクセスし、ブックマークをロードする前に照会を変更することができます。あるいは、メンバー名やデータ・アウトラインが変わった場合に、リポジトリー内に保管されているすべてのブックマークを変更することができます。

次の図は、どのように BookmarksBlox を通して SerializedMDBQuery オブジェクト (マルチディメンション・データ・ソース用) と SerializedTextualQuery オブジェクト (リレーショナル・データ・ソース用) にアクセスできるかを示します。
SerializedMDBQuery オブジェクトによって、関係する軸、ディメンション、タプル、およびメンバーに関する情報を取得し、古いメンバーを新しいメンバーで置き換えることができます。また、特定の Essbase レポート・スクリプトを取得するために EssbaseReportSpec オブジェクトにアクセスすることもできます。
SerializedTextualQuery オブジェクトは、保管されている照会を取得し、新規の照会を設定するのに使用します。



Bookmark オブジェクトのための静的フィールド

Bookmark オブジェクトには、Blox のタイプ、ブックマーク可視性、およびヌル・ディメンションを示す以下の静的フィールドがあります。

カテゴリー別の静的フィールド

Blox タイプ

CHART_BLOX_TYPE

DATA_BLOX_TYPE

DATALAYOUT_BLOX_TYPE

GRID_BLOX_TYPE

PAGE_BLOX_TYPE

PRESENT_BLOX_TYPE

TOOLBAR_BLOX_TYPE

UNKNOWN_BLOX_TYPE

ブックマークの可視性

PRIVATE_VISIBILITY

PUBLIC_VISIBILITY

ヌル・ディメンション

NULL_DIMENSION

これらの静的フィールドにより、定数を使用して Blox タイプとブックマークの可視性を指定したり識別したりすることができます。

ブックマーク名に関する制約事項

ブックマークの名前に関して幾つかの制約事項があります。

- 名前をヌルまたはブランクにはできません。
- 大/小文字に関係なく、名前に以下の予約語を使用することはできません。
  - properties
  - public
  - private
- 使用できる文字は、A から Z まで、a から z まで、0 から 9 まで、ダッシュ (″-″)、下線 (″_″)、およびスペースです。
- 名前の先頭または末尾にスペースを含めることはできません。

BookmarksBlox の JSP カスタム・タグ構文

Alphablox タグ・ライブラリーは、各 Blox を作成するために JSP ページで使用するカスタム・タグを提供します。このセクションでは、BookmarksBlox を作成するためにカスタム・タグを作成する方法を説明します。すべての属性を含むタグのコピー・アンド・ペースト・バージョンについては、520 ページの『BookmarksBlox JSP カスタム・タグ』を参照してください。

パラメーター

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

ここで、それぞれ以下のとおりです。

attribute 属性表にリストされている属性の 1 つです。

value 属性の有効な値です。

有効な属性を次の表にリストします。

属性
id
bloxName

使用法

各カスタム・タグには 1 つ以上の属性を含めることができ、それぞれを 1 つ以上のスペースまたは改行文字で区切ります。余分のスペースまたは改行文字は無視されます。読みやすくするため、同じインデントでそれぞれ別々の行に属性を並べることができます。

</blox:bookmarks> という終了タグの代わりに、次のように、省略表現を使用して属性リストの後ろでタグを終了することができます。

```
id="myBookmarksBlox" />
```

例

```
<blox:bookmarks
   id = "myBookmarksBlox" />
```

BookmarksBlox の例

このセクションには、BookmarksBlox の使用法、その関連オブジェクト、および関連したメソッドを示す例があります。

- 67 ページの『例 1: すべてのブックマークのカウント数を取得する』
- 68 ページの『例 2: ブックマークのプロパティー・セットの取得』
- 70 ページの『例 3: 指定した基準と一致するブックマークのリストの取得』
- 70 ページの『例 4: BookmarksBlox API を使用したブックマークの作成』
- 72 ページの『例 5: サーバー・サイドの bookmarkLoad イベント・フィルターの使用』
- 73 ページの『例 6: ブックマークの照会をロード時に取得する』

例 1: すべてのブックマークのカウント数を取得する

この例では、以下の点を例示します。

- リポジトリーに保管されているすべてのブックマークにアクセスするための BookmarksBlox およびその `listBookmarks()` メソッドの使用法。`listBookmarks()` メソッドはブックマーク・オブジェクトの配列を戻します。
- 配列の長さを取得することにより、ブックマークの総数のカウント数を取得する方法。

```
<%@ 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).
```

例 2: ブックマークのプロパティー・セットの取得

この例では、ブックマーク名、アプリケーション名、ユーザー名、Blox 名、およびブックマークの可視性を基にしてブックマークにアクセスする方法と、そのプロパティー・セットにある情報を取得する方法を示します。特に、以下の点を例示します。

- 個々のブックマーク (Bookmark オブジェクト) にアクセスするための BookmarksBlox の使用法。
- Bookmark オブジェクトの `getName()`、`getVisibility()`、`getDescription()`、`getBloxType()`、および `getBinding()` メソッドの使用法。
- 個々のプロパティーにアクセスするための Bookmark オブジェクトの `getBookmarkProperties()` メソッド (ネストされた Blox ごとに 1 つ) の使用法。

生成される出力は、次のようになります。

The bookmark you are looking for exists.

1. The Repository JNDI binding for this bookmark is:
   `users/admin/salesapp/mygrid/bookmark/q2fy02WestSales/properties`
2. The bookmark name is: q2fy02WestSales
3. The type of Blox this bookmark was saved for is: grid
4. The bookmark description is: The Q2 West Sales
5. The bookmark visibility is: private
6. The bookmark contains Blox properties in the repository
7. Types of Blox properties saved in the bookmark:
   - grid
   - data

コードは次のようになります。

```
<%@ 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>
```

例 3: 指定した基準と一致するブックマークのリストの取得

この例では、以下の点を例示します。

- BookmarkMatcher オブジェクトを使用することによる、指定されたユーザー、およびこの例では、ユーザー「admin」の取得。
- Bookmark オブジェクトの `getBinding()` メソッドおよび `getBloxType()` メソッドの使用とそれらの出力。

生成される出力は次のようになります。

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)

コードは次のようになります。

```
<%@ 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>
```

例 4: BookmarksBlox API を使用したブックマークの作成

この例では、BookmarksBlox、Bookmark、および BookmarkProperties クラスを使用して新規ブックマークを作成する方法を示します。プログラマチックにブックマークを作成する 2 つの方法があります。

- すべてのブックマーク・オプションを BookmarksBlox.createBookmark(...) に提供する
- Blox を他の必要な情報と共に BookmarksBlox.createBookmark(...) に提供する

次の例では、後者の方法を例示します。

1. ブックマーク名、アプリケーション名、ユーザー名、Blox 名、可視性、およびブックマークに関連した説明を指定します。
2. 次に createBookmark() メソッドを使用して「bk」という Bookmark オブジェクトを作成し、その Blox タイプを GRID_BLOX_TYPE に指定します。
3. 「bk」オブジェクトのために、GridBlox 固有のプロパティーを保管する「gridBloxProp」という BookmarkProperties オブジェクトのインスタンスおよび DataBlox 固有のプロパティーを保管する「dataBloxProp」という別のオブジェクトのインスタンスを作成します。 gridBloxProp については、cellBandingEnabled を true に設定し、dataBloxProp については、照会を「!」に設定し、データ・ソースに再接続するように指定します。
4. saveAll() メソッドを呼び出し、今リポジトリーに作成したブックマークを保管します。

生成される出力は、次のようになります。

(ここに GridBlox が来ます)

```
We've got a Bookmark object from BookmarksBlox.createBookmark()!
Created a bookmark: q2fy02WestSales
At binding: users/jdoe/salesapp/mygrid/bookmark/q2fy02westsales/properties
```

コードは次のようになります。

```
<%@ taglib uri="bloxtld" prefix="blox" %>
<!--import the following package in order to access the
    com.alphablox.blox.repository.BookmarkMatcherUsers class-->
<%@ page import="com.alphablox.blox.repository.*" %>
<blox:header />
<blox:bookmarks id="myBookmarksBlox" />

<blox:grid id="myGrid" width="500" height="320">
   <blox:data dataSourceName="qcc-essbase" query="!"/>
</blox:grid>
<%
// (1) Specify the bookmark properties
   String bookmarkName = "q2fy02WestSales";
   String applicationName = "SalesApp";
   String userName = "jdoe";
   String bloxName = "myGrid";
   String visibility = myBookmarksBlox.PRIVATE_VISIBILITY;
   String description = "Bookmark for Q2FY02 West Region Sales";
   Bookmark bk = null;

// (2) Create a Bookmark object called "bk"
   bk = myBookmarksBlox.createBookmark(bookmarkName,
      applicationName, userName, bloxName, visibility,
      myBookmarksBlox.GRID_BLOX_TYPE);

%>
<p>We've got a Bookmark object from BookmarksBlox.createBookmark()!<p/>

<%
// (3) Set the bookmark's description and its GridBlox and DataBlox
//     properties
```

```
   bk.setDescription(description);
   bk.setCustomProperty("Report", "West Region Sales Report");

   BookmarkProperties gridBloxProp =
      bk.createBookmarkProperties(myBookmarksBlox.GRID_BLOX_TYPE);
   gridBloxProp.setProperty("bandingEnabled", true);
   BookmarkProperties dataBloxProp =
      bk.createBookmarkProperties(myBookmarksBlox.DATA_BLOX_TYPE);
   dataBloxProp.setProperty("connectOnStartup", true);
   dataBloxProp.setProperty("query", "!");

// (4) Save the bookmarks to the repository. Must call save() or
//     saveAll() to save the bookmark to the repository.
   bk.saveAll();

%>
Created a bookmark: <%= bookmarkName %><br>
      At binding: <%= bk.getBinding() %>
<%
   bk = null;
%>
```

例 5: サーバー・サイドの bookmarkLoad イベント・フィルターの使用

この例では、bookmarkLoad イベントがトリガーされたときに、サーバー・サイドのイベント・フィルターを使用してカスタム・タスクを実行する (この例では、ロードされたブックマークの名前を通知する MessageBox をポップアップする) 方法を示します。

1. サーバー・サイドのイベント・フィルターを使用するには、共通 Blox メソッド addEventFilter() を使用して、最初に特定のイベント・フィルター・オブジェクトを追加します。

   ```
   <blox:present id="myPresent">
   ...
   <%
     myPresent.addEventFilter(new LoadFilter() );
   %>
   </blox:present>
   ```

2. 次に、イベントがトリガーされたときに呼び出される、対応するイベント・フィルター・オブジェクト (BookmarkLoadFilter) および対応するメソッド (bookmarkLoad(BookmarkLoadEvent)) をインプリメントするクラスを自分で作成します。このためには、JSP に com.alphablox.blox.filter.* パッケージのインポート・ステートメントを追加する必要があります。

   ```
   public class LoadFilter implements BookmarkLoadFilter
   {
      public void bookmarkLoad( BookmarkLoadEvent bre )
      {
         //actions to take when the event is triggered
      }
   }
   ```

コードは次のようになります。

```
<%@ page import="com.alphablox.blox.filter.*" %>
<%@ page import="com.alphablox.blox.*" %>
<%@ page import="com.alphablox.blox.repository.Bookmark,
                 com.alphablox.blox.uimodel.core.MessageBox,
                 com.alphablox.blox.uimodel.BloxModel" %>
<%@ taglib uri="bloxtld" prefix="blox"%>
```

```
<html>
<head>
  <title>Bookmarks Filter Events</title>
  <!-- Blox header tag -->
  <blox:header/>
</head>

<body>
<blox:present id="myPresent" >
   <blox:data dataSourceName="QCC-Essbase" query="!"/>
<%
   myPresent.addEventFilter(new LoadFilter(myPresent.getBloxModel()));
%>
</blox:present>

</body>
</html>

<%!
public class LoadFilter implements BookmarkLoadFilter {
   BloxModel model;
  public LoadFilter (BloxModel model) {
      this.model = model;
   }
   public void bookmarkLoad( BookmarkLoadEvent ble ) throws Exception {
      Bookmark bookmark = ble.getBookmark();
      String name = bookmark.getName();
      StringBuffer msg = new StringBuffer("A bookmark called " + name + " is
loaded.");

      MessageBox msgBox = new MessageBox(msg.toString(), "Bookmark Loaded",
MessageBox.MESSAGE_OK, null);
     model.getDispatcher().showDialog(msgBox);
      }
   }
%>
```

例 6: ブックマークの照会をロード時に取得する

この例では、bookmarkLoad イベントがトリガーされたときに、ブックマーク付きで保管されたテキスト形式の照会を取得する方法を示します。

1. サーバー・サイドのイベント・フィルター BookmarkLoadFilter を使用して、ブックマークのロード時にカスタム・アクションをトリガーします。サーバー・サイドのイベント・フィルターの例については、72 ページの『例 5: サーバー・サイドの bookmarkLoad イベント・フィルターの使用』を参照してください。次のように、イベント・フィルターは PresentBlox タグ内部に追加すれば、ページの再ロードのたびに追加する必要はなく、1 回だけで済みます。

   ```
   <blox:present id="myPresent" ...>

     <% myPresent.addBookmarkLoadFilter(new LoadFilter()); %>

   </blox:present>
   ```

2. 次のように、DataBlox の textualQueryEnabled プロパティーを true に設定して、ブックマークがロードされたときにテキスト形式の照会が適用されるようにします。

   ```
   <blox:present id="myPresent" ...>

     <blox:data
       ...
       textualQueryEnabled="true" />
   ```

```
    <% myPresent.addBookmarkLoadFilter(new LoadFilter()); %>

  </blox:present>
```

3. ブックマークのロード時には、そのブックマークの SerializedMDBQuery オブジェクト (マルチディメンション・データ・ソース用) か SerializedTextualQuery オブジェクト (リレーショナル・データ・ソース用) からテキスト形式の照会を取得します。 SerializedMDBQuery には `generateQuery()` メソッド、SerializedTextualQuery には `getQuery()` メソッドがあり、これらはテキスト形式の照会を戻します。なお、`generateQuery()` メソッドは、IBM DB2 OLAP Server と Hyperion Essbase でだけ機能します。

完全なコードは次のようになります。

```
<%@ page import="com.alphablox.blox.filter.*,
                 com.alphablox.blox.repository.BookmarkProperties,
                 com.alphablox.blox.repository.SerializedQuery,
                 com.alphablox.blox.repository.SerializedTextualQuery,
                 com.alphablox.blox.repository.SerializedMDBQuery,
                 com.alphablox.blox.repository.Bookmark,
                 com.alphablox.blox.uimodel.core.MessageBox,
                 com.alphablox.blox.uimodel.BloxModel" %>

<%@ taglib uri="bloxtld" prefix="blox"%>
<html>
<head> <title>Bookmarks Filter Events</title>
<blox:header/>

</head>
<body>
<blox:present id="myPresent" width="800" height="600">
  <blox:data dataSourceName="QCC-Essbase"
    query="<ROW (¥"All Locations¥") Central East West <COLUMN (¥"All Time
Periods¥") 2001 !"
    useAliases="true"
    textualQueryEnabled="true" />

  <% myPresent.addEventFilter(new LoadFilter(myPresent.getBloxModel())); %>

</blox:present>

</body>
</html>


<%! public class LoadFilter implements BookmarkLoadFilter
{
  BloxModel model;
  public LoadFilter (BloxModel model) {
    this.model = model;
  }

  public void bookmarkLoad( BookmarkLoadEvent ble ) throws Exception
  {
    Bookmark bookmark = ble.getBookmark();
    SerializedQuery sq = bookmark.getSerializedQuery();
    SerializedTextualQuery stq = null;
    SerializedMDBQuery smq = null;
    String query = null;
    if( sq instanceof SerializedTextualQuery )
    {
       stq = (SerializedTextualQuery)sq;
       query = stq.getQuery();
    }
```

```
    else if( sq instanceof SerializedMDBQuery )
      {
        smq = (SerializedMDBQuery)sq;
        query  = smq.generateQuery();
      }
     StringBuffer msg = new StringBuffer("query=" + query);

     MessageBox msgBox = new MessageBox(msg.toString(), "Bookmark Event
Filter Message", MessageBox.MESSAGE_OK, null);
     model.getDispatcher().showDialog(msgBox);

  }
}
%>
```

BookmarksBlox タグ属性

このセクションでは、BookmarksBlox によってサポートされているタグ属性について説明します。一般に、ブックマークの管理に関連するタスクには、BookmarksBlox API を使用する必要があります。 BookmarksBlox メソッドの詳細については、Javadoc 内の com.alphablox.blox.BookmarkBlox を参照してください。

id

これは共通の Blox タグ属性およびプロパティーです。詳しい説明は、43 ページの『id』を参照してください。

bloxName

これは共通の Blox タグ属性およびプロパティーです。詳しい説明は、39 ページの『bloxName』を参照してください。

第 7 章 ChartBlox タグ・リファレンス

この章には ChartBlox の参照資料が含まれています。Blox についての一般的な参照情報は、21 ページの『第 3 章 一般 Blox リファレンス情報』を参照してください。このリファレンスの使用方法については、1 ページの『第 1 章 このリファレンスの使用法』を参照してください。

ChartBlox の概説

ChartBlox では、円グラフ、棒グラフ、および折れ線グラフなどの幅広いフォーマットでデータが表示されます。ユーザーは、ChartBlox グラフィカル・ユーザー・インターフェースを通して、チャート・タイプや方向などのチャート属性を変更できます。

グラフィカル・ユーザー・インターフェース

ChartBlox グラフィカル・ユーザー・インターフェース (GUI) は、チャートの表示域とオプショナルのチャート・コントロールから成ります。ユーザーがメンバー上 (凡例、ラベル上またはチャート自体の中) を右クリックすると右クリック・メニューが表示され、このメニューから、ドリルアップ、ドリルダウン、ピボット、および非表示/表示といったデータ・ナビゲーション・オプションが使用できます。チャート・タイプ、軸の配置の変更、またはデータの構成を行うには、ユーザーはメニュー・バーの「チャート」>「オプション...」メニューから「チャート・オプション」ダイアログにアクセスできます。

使用可能なチャート・タイプ

以下の表で、ChartBlox が DHTML クライアントでレンダリングされた場合に使用可能なすべてのチャート・タイプの有効な名前をリストします。これらの名前の 1 つを `chartType` プロパティーの値として使用する場合には、括弧付きのコメントを省略してください。

注: チャート・タイプに関しては、以下の点に気を付けてください。

- `chartType` プロパティーには、値としてテキスト・ストリングのみを使用できます。この表に表示されているとおりのスペルにするようにしてください。
- `get/setChartTypeAsInt()` メソッドには、各チャート名の左に表示されている整数を使用します。

整数値	チャート
200	垂直バー、横並び、立体効果 (デフォルト)
201	縦線、絶対、立体効果
202	垂直エリア、絶対、立体効果
0	3D 棒グラフ
17	垂直バー、横並び、(または単に「バー」)
18	垂直バー、積み重ね
19	垂直バー、横並び、双軸
20	垂直バー、積み重ね、双軸
21	垂直バー、横並び、バイポーラー
22	垂直バー、積み重ね、バイポーラー
24	水平バー、横並び
25	水平バー、積み重ね
26	水平バー、横並び、双軸
27	水平バー、積み重ね、双軸
28	水平バー、横並び、バイポーラー
29	水平バー、積み重ね、バイポーラー
31	垂直エリア、絶対
32	垂直エリア、積み重ね
33	垂直エリア、絶対、バイポーラー
34	垂直エリア、積み重ね、バイポーラー
35	垂直エリア、百分率
41	縦線、絶対 (または単に「線」)
42	縦線、積み重ね
43	縦線、絶対、双軸
44	縦線、積み重ね、双軸
45	縦線、絶対、バイポーラー
46	縦線、積み重ね、バイポーラー
47	縦線、百分率
55	円グラフ
61	散布図
67	レーダー、線
68	レーダー、エリア
85	ヒストグラム、垂直
86	ヒストグラム、水平
89	バブル・チャート

502	ダイヤル・チャート (ダイヤル・チャートの使用法について詳しくは、79 ページの『ダイヤル・チャート』を参照してください。)
510	ウォーターフォール
511	累積

DHTML Client のチャート作成エンジンでは、本当の 3 次元チャート・タイプはサポートされません。したがって、単一の序数軸 (O1) のみがあります。立体効果のあるチャート・タイプには次のものがあります。

- 3D 棒グラフ (これは基本的には垂直バー、横並びのチャートで、深さが最適化されます)。
- 垂直バー、横並び、立体効果 (デフォルト)
- 縦線、絶対、立体効果
- 垂直エリア、絶対、立体効果

幾つかのチャート・タイプでは、チャート化するエレメントごとに複数のデータ値を指定する必要があります。以下の表では、それらのチャートを、エレメントごとに必要なデータ値数および順序の要件と共にリストします。

整数	タイプ	データ値と順序
61	散布図	マーカーごとに 2 つの値: X と Y、この順で。
89	バブル・チャート	マーカーごとに 3 つの値: X、Y、および Z、この順で。

ダイヤル・チャート

ダイヤル・チャートには、描画の前に、幾つかのパラメーターの指定が必要です。この指定には、ダイヤル盤の開始、終了番号およびステップ・サイズが含まれます。ダイヤル・チャートの指定には、幾つかのネストされたタグが使用可能です。詳細は、77 ページの『ChartBlox の概説』および 153 ページの『ダイヤル・チャートのタグ・リファレンス』を参照してください。

チャートの軸

チャート・タイプによって、チャートには、序数軸 (O1) 1 本と、数値軸 (X1、Y1、および Y2) 3 本までが含まれます。序数軸にはグループまたはカテゴリーが含まれており、O1 軸は、バブル・チャートおよび散布図を除くすべてのチャート・タイプに含まれています。X1 軸はバブル・チャートおよび散布図にのみ含まれます。 Y1 軸は、円グラフ以外のすべてのチャート・タイプに含まれます。Y2 軸は、双軸チャートにのみ含まれます。

スタイルの指定

フォントおよび前景色を指定して、チャートのタイトル、軸タイトル、脚注、およびラベルのスタイルを設定できます。例えば、以下のタグ属性では、タイトルのスタイルに太字の 24 ポイントの Arial フォントで #990099 の色が使用され、脚注スタイルに赤のイタリックの 14 ポイントのモノスペース・フォントが使用されるよう設定されます。

```
<blox:chart id="myChart"
   titleStyle="font=Arial:Bold:24, foreground=#990099"
   footnoteStyle="font=Monospace:Italic:14, foreground=red"
/>
```

以下のように、関連したネストされたタグを使用してスタイルを指定することもできます。

```
<blox:chart id="myChart">
   <blox:titleStyle
      font="Arial:Bold:24"
      foreground="#990099"
   />
   <blox:footnoteStyle
      font="Monospace:Italic:14"
      foreground="red"
   />
</blox:chart>
```

タイトル、脚注、ラベル、または軸タイトルのスタイルを設定すると、基礎となるテーマがオーバーライドされます。

フォント

フォント属性は、コロンで区切られたフォント名、スタイル、およびポイント・サイズを取ります。

font name: style: *point*

ここで、それぞれ以下のとおりです。

- *font name*: これらはオペレーティング・システムに従って定義されます。一般に受け入れられるのは以下のフォント名です。
  - Arial
  - Courier
  - Helvetica
  - TimesRoman
  - SansSerif
  - Serif
  - Monospace

  受け入れ可能なフォントは、ブラウザーおよびクライアントのマシンによってかなり異なります。したがって、総称名が提供されています (Monospace、SansSerif、および Serif)。総称名の代わりに実際に使用するフォントは、各ブラウザーによって定義されます。

  フォントが指定されない場合、デフォルトは SansSerif です。サーバーが非西洋の言語システムで稼働している場合、そのフォントの文字セットに見つからない文字が正しく表示されない場合があります。この問題を避けるために、ご使用のロケールで正しく表示されるフォントを常に指定してください。
- *Style*: フォント・スタイルは以下のいずれかです。
  - `plain`
  - `italic`
  - `bold`
  - `bolditalic`
- *Point*: ポイント・サイズの整数 (通常は 8 から 36)。

ヒント: フォントが指定されない場合、デフォルトは SansSerif です。サーバーが非西洋の言語システムで稼働している場合、そのフォントの文字セットに見つからない文字が正しく表示されない場合があります。この問題を避けるために、ご使用のロケールで正しく表示されるフォントを常に指定してください。フォント仕様に関係するプロパティーには、axisTitleStyle、labelStyle、footnoteStyle、および titleStyle が含まれます。

ヒント: 3 つの属性のうちのいずれかが指定されていない場合、デフォルトの、または現行の継承されたフォント値が適用されます。ただし、以下の表で示すように属性を分離するコロンは組み込む必要があります。

属性値	結果
font=:Bold:12	現行のフォントが使用されるが、スタイルが太字に、サイズが 12 ポイントに変更される。
font=Helvetica	フォントが Helvetica に変更されるが、スタイルとサイズは変更されない。
font=::14	現行のフォントとスタイルが使用されるが、サイズが 14 ポイントに変更される。
font=:Plain:	現行のフォントとサイズが使用されるが、スタイルが Plain に変更される。

前景

カラー名は大/小文字を区別しません。認識されるカラー名は以下のとおりです。

```
Black Blue Cyan DarkGray Gray Green          Magenta Orange Pink Red White Yellow
LightGray
```

色は、その色の赤、緑、青それぞれの彩度を指定する RGB 値で表すこともできます。RGB 値で色を指定するには、それぞれの 3 つの数値をそれに相当する 16 進数または 10 進数に変換します。それからその結果得られるストリングを、16 進数ストリングであれば番号記号 (#) で始めて入力します。例えば、#00FF00 は 100% 緑色の 16 進数ストリングです。認識されるカラー名の RGB、16 進数、および 10 進数を以下の表にリストします。

ヒント: ブラウザー・セーフ・カラーのパレットについては、以下のような Web サイトを確認してください。

- http://www.visibone.com/colorlab/

カラー名	RGB 値	16 進数値	10 進数値
Black	000 000 000	000000	0
Blue	000 000 255	0000FF	255
Cyan	000 255 255	00FFFF	65535
DarkGray	064 064 064	404040	4210752
Gray	128 128 128	808080	8421504
Green	000 255 000	00FF00	62580
LightGray	192 192 192	C0C0C0	12632256

カラー名	RGB 値	16 進数値	10 進数値
Magenta	255 000 255	FF00FF	16711935
Orange	255 200 000	FFC800	16762880
Pink	255 175 175	FFAFAF	16756655
Red	255 000 000	FF0000	16711680
White	255 255 255	FFFFFF	16777215
Yellow	255 255 000	FFFF00	16776960

ChartBlox の JSP カスタム・タグ構文

DB2 Alphablox タグ・ライブラリーは、それぞれの Blox を作成するために JSP ページで使用するカスタム・タグを提供します。このセクションでは、ChartBlox を作成するためのカスタム・タグの作成方法を説明します。すべての属性を含むタグのコピー・アンド・ペースト・バージョンについては、520 ページの『ChartBlox JSP カスタム・タグ』を参照してください。

パラメーター

```
<blox:chart
    [attribute="value"] >
       [<blox:axisTitleStyle
              [attribute="value"] />]
       [<blox:dial
              [attribute="value"] />
       [<blox:footnoteStyle
              [attribute="value"] />]
       [<blox:labelStyle
              [attribute="value"] />]
       [<blox:seriesFill
              [attribute="value"] />]
       [<blox:titleStyle
              [attribute="value"] />]
</blox:chart>
```

ここで、それぞれ以下のとおりです。

attribute 属性表にリストされている属性の 1 つです。

value 属性の有効な値です。

属性は以下のいずれかになります。

属性
id
absoluteWarning
aggregateIdenticalInstances
applyPropertiesAfterBookmark
areaSeries
autoAxesPlacement
axisTitleStyle
backgroundFill

属性
barSeries
bloxEnabled
bloxName
bookmarkFilter
chartAbsolute
chartCurrentDimensions
chartFill
chartType
columnLevel
columnSelections
comboLineDepth
dataTextDisplay
dataValueLocation
depthRadius
dwellLabelsEnabled
enablePoppedOut
filter
footnote
footnoteStyle
formatProperties
gridLineColor
gridLinesVisible
groupSmallValues
height
helpTargetFrame
histogramOptions
labelStyle
legend
legendPosition
lineSeries
lineWidth
localeCode
logScaleBubbles
markerShape
markerSizeDefault
maxChartItems
maximumUndoSteps
menubarVisible
mustIncludeZero
noDataMessage
o1AxisTitle

属性
pieFeelerTextDisplay
poppedOut
poppedOutHeight
poppedOutTitle
poppedOutWidth
quadrantLineCountX
quadrantLineCountY
quadrantLineDisplay
removeAction
render
rightClickMenuEnabled
riserWidth
rowHeaderColumn
rowLevel
rowSelections
rowsOnXAxis
seriesColorList
showSeriesBorder
smallValuePercentage
title
titleStyle
toolbarVisible
totalsFilter
useSeriesShapes
visible
width
x1AxisTitle
x1FormatMask
x1LogScale
x1ScaleMax
x1ScaleMaxAuto
x1ScaleMin
x1ScaleMinAuto
XAxis
XAxisTextRotation
y1Axis
y1AxisTitle
y1FormatMask
y1LogScale
y1ScaleMax
y1ScaleMaxAuto

属性
y1ScaleMin
y1ScaleMinAuto
y2Axis
y2AxisTitle
y2FormatMask
y2LogScale
y2ScaleMax
y2ScaleMaxAuto
y2ScaleMin
y2ScaleMinAuto

<blox:axisTitleStyle> ネストされたタグ

91 ページの『axisTitleStyle』を参照。

属性
font
foreground

<blox:footnoteStyle> ネストされたタグ

103 ページの『footnoteStyle』を参照。

属性
font
foreground

<blox:labelStyle> ネストされたタグ

108 ページの『labelStyle』を参照。

属性
font
foreground

<blox:seriesFill> ネストされたタグ

122 ページの『seriesFill』を参照。

属性
index
value

<blox:titleStyle> ネストされたタグ **125 ページの『titleStyle』を参照。**
属性
font
foreground

使用法

各カスタム・タグには 1 つ以上の属性を含めることができ、それぞれを 1 つ以上のスペースまたは改行文字で区切ります。余分のスペースまたは改行文字は無視されます。読みやすくするため、同じインデントでそれぞれ別々の行に属性を並べることができます。

ネストされたタグがない場合 (<blox:titleStyle> または <blox:footnoteStyle> タグなど)、属性リストの終わりのタグを以下のようにして、終了タグ </blox:chart> を省略表現で置換できます。

```
width="650" />
```

ネストされたタグがある場合は、省略表現は無効となり、終了タグが必要となります。

例

```
<blox:chart
     height="400"
     width="400"
     chartType="bar" />
```

カテゴリー別の ChartBlox タグ属性

以下の表は、固有の ChartBlox タグ属性をリストしています。複数の Blox に共通するタグ属性のリストについては、35 ページの『カテゴリー別の共通の Blox タグ属性』を参照してください。ChartBlox によってサポートされるプロパティーおよびメソッドは、以下のように相互参照として編成されています。

- 86 ページの『チャートの外観に関連するタグ属性』
- 87 ページの『チャート・データに関連するタグ属性』
- 88 ページの『チャート・ラベルのタグ属性』
- 89 ページの『チャート・ポップアウトのタグ属性』
- ダイヤル・チャートについては、147 ページの『ダイヤル・チャートの概説』を参照してください。

チャートの外観に関連するタグ属性

以下に、チャートの外観に関連するタグ属性をリストします。

チャートの外観

- areaSeries
- autoAxesPlacement

- backgroundFill
- barSeries
- chartFill
- chartType
- comboLineDepth
- dataTextDisplay
- depthRadius
- footnoteStyle
- formatProperties
- gridLineColor
- gridLinesVisible
- histogramOptions
- labelStyle
- lineSeries
- lineWidth
- markerShape
- markerSizeDefault
- quadrantLineCountX
- quadrantLineCountY
- quadrantLineDisplay
- removeAction
- rightClickMenuEnabled
- riserWidth
- seriesColorList
- seriesFill
- showSeriesBorder
- titleStyle
- trendLines
- useSeriesShapes

チャート・データに関連するタグ属性

以下のタグ属性は、チャート内のデータに関連しています。

- absoluteWarning
- aggregateIdenticalInstances
- chartAbsolute
- chartCurrentDimensions
- columnLevel
- rowLevel
- columnSelections
- rowSelections

- dataValueLocation
- filter
- groupSmallValues
- legend
- localeCode
- logScaleBubbles
- maxChartItems
- mustIncludeZero
- rowsOnXAxis
- smallValuePercentage
- totalsFilter
- x1FormatMask
- x1LogScale
- x1ScaleMax
- x1ScaleMaxAuto
- x1ScaleMin
- x1ScaleMinAuto
- XAxis
- y1Axis
- y2Axis
- y1FormatMask
- y2FormatMask
- y1LogScale
- y2LogScale
- y1ScaleMax
- y1ScaleMaxAuto
- y2ScaleMax
- y2ScaleMaxAuto
- y1ScaleMin
- y1ScaleMinAuto
- y2ScaleMin
- y2ScaleMinAuto

チャート・ラベルのタグ属性

以下のタグ属性は、チャートに表示されるラベルに関連しています。

- axisTitleStyle
- dwellLabelsEnabled
- footnote
- footnoteStyle
- labelStyle

- legendPosition
- o1AxisTitle
- pieFeelerTextDisplay
- rowHeaderColumn
- title
- titleStyle
- XAxisTextRotation
- x1AxisTitle
- y1AxisTitle
- y2AxisTitle

チャート・ポップアウトのタグ属性

以下は、ChartBlox を別のポップアップ・ブラウザー・ウィンドウに表示することに関連したタグ属性です。

- enablePoppedOut
- poppedOut
- poppedOutHeight
- poppedOutTitle
- poppedOutWidth

ChartBlox タグ属性

このセクションでは、ChartBlox タグ属性について説明します。属性は、アルファベット順にリストされています。ChartBlox メソッドについては、 Javadoc 内の com.alphablox.blox.ChartBlox クラスを参照してください。

id

これは共通の Blox プロパティーおよびタグ属性です。詳細記述は、43 ページの『id』を参照してください。

absoluteWarning

ユーザーがチャート値を設定して絶対値を表示する際に少なくとも 1 つのデータ値が負の値である場合、この警告がチャートの脚注に追加されます。

データ・ソース

すべて

構文

absoluteWarning="*warning*"

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
warning	"Warning: Values are absolute"	警告メッセージ・ストリング。

aggregateIdenticalInstances

等しい系列を、チャート内の別系列として各行をグリッドに表示するのではなく、グループ化して 1 つの系列としてその集合データをチャートに表すかどうかを指定します。

データ・ソース

リレーショナル・データのみ

構文

```
aggregateIdenticalInstances="aggregate"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
aggregate	true	このプロパティーを true に設定すると、系列が等しいインスタンスが存在する場合には、その集合データがチャート化されます。 false に設定すると、グリッド上の各行は、別の系列として個別に表示されます。

applyPropertiesAfterBookmark

これは共通の Blox プロパティーおよびタグ属性です。詳細記述は、36 ページの『applyPropertiesAfterBookmark』を参照してください。

areaSeries

組み合わせチャート中のどのデータ系列がエリア系列であるかを指定します。

データ・ソース

すべて

構文

```
areaSeries="series"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
series	空ストリング	コンマで区切られたストリングで、エリア系列で表示されるメンバー名を定義する

面グラフ、棒グラフ、および折れ線グラフを表示する場合、これら 3 つのチャート・タイプを組み合わせたチャートを表示することができます。データ系列を折れ線に、別のデータ系列を棒に、3 番目を面にすることができます。

このプロパティーは、組み合わせチャートの一部をなす面グラフ・タイプで表されるメンバーを識別します。表示されるメンバー名は、コンマで区切られたストリングとして定義されます。凡例項目を成す複数のディメンションがある場合 (「チャート軸の配置 (Chart Axes Placement)」で定義)、タブ ("¥t") を使用してディメンションを区切る必要があります。

例

```
myPresent.getChartBlox().setAreaSeries("Qtr1¥tAudio, Qtr2¥tAudio,
Qtr3¥tAudio");
```

関連項目

94 ページの『barSeries』、110 ページの『lineSeries』

autoAxesPlacement

チャート軸に情報がどのように配置されるかを定義します。

データ・ソース

すべて

構文

```
autoAxesPlacement="auto"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
auto	true	有効な値は true または false です。

使用法

デフォルトでは ChartBlox は、X 軸、凡例、フィルター、y1 軸、および y2 軸上にデータを配置する通常のデフォルトを使用するように指示されています。これらの軸、凡例、またはフィルターのいずれかを特別に設定したい場合には、このプロパティーを false に設定する必要があります。

例

```
autoAxesPlacement="false"
```

関連項目

102 ページの『filter』、109 ページの『legend』、135 ページの『XAxis』、136 ページの『y1Axis』、141 ページの『y2Axis』

axisTitleStyle

チャートの軸タイトルのスタイル (前景色およびテキスト書式) を指定します。

データ・ソース

すべて

構文

```
axisTitleStyle="style"
```

または

```
<blox:axisTitleStyle
    font=""
    foreground="">
</blox:axisTitleStyle>
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
style	空ストリング	スタイル属性を定義するストリング。

使用法

スタイル・ストリングの指定方法について詳しくは、79 ページの『スタイルの指定』を参照してください。

例

```
axisTitleStyle = "foreground=white, font=Courier:Bold:10"
```

関連項目

103 ページの『footnoteStyle』、108 ページの『labelStyle』、125 ページの『titleStyle』、92 ページの『backgroundFill』

backgroundFill

チャート・フレームの外側の領域を埋める、単一カラー、色のグラジエント、またはイメージを指定できるようにします。

データ・ソース

すべて

構文

```
backgroundFill="fill"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
fill	ヌル	チャートの背景の領域に色、グラジエント、またはイメージを定義するストリング。

使用法

ストリング fill は、背景に表示する単一カラー、カラー・グラジエントのための 2 色、またはイメージへの URL のいずれかです。 2 色の指定には、標準 Java カラー名または RGB 値を使用します。RGB 値を使用する場合には、0xffffff の形式で入力してください。色のグラジエントを使用する場合、ストリングはコンマで区切られた 2 色のリストである必要があります。グラジエントの方向は、ストリングの最後の項目として、以下の表から適切なグラジエント修飾子を追加することにより指定できます。

グラジエント方向	修飾子
右方	1 (2 色のリストで方向が指定されていない場合のデフォルト)
左方	2
下	3
上	4
下/左	5
上/左	6
下/右	7
上/右	8

イメージの使用を指定する場合には、以下のいずれかである必要があります。

- アプリケーション・コンテキストからイメージへの相対 URL。例えば、 JSP が「salesApp」というアプリケーションにあり、salesApp/images/ ディレクトリー中のイメージ・ファイル logo.gif を背景に使用する場合には、次のようにします。

  ```
  backgroundFill = "images/logo.gif"
  ```

- “http:” で始まる絶対 URL。

  ```
  backgroundFill = "http://serverName/path/to/image.gif"
  ```

  参照されるイメージ・ファイルがあるサーバーに認証が必要ないことを確認してください。認証が必要な場合、イメージはロードされず、デフォルト系列の色が使用されます。これは、チャート作成エンジンに、認証に必要なユーザー名およびパスワードがないためです。

- 以下のファイル・プロトコルを使用する “file:” で始まる URL。

  ```
  backgroundFill = "file:///C|/Alphablox5/webapps/salesApp/images/logo.gif"
  ```

  これは、DB2 Alphablox が稼働しているサーバー上のイメージへのファイル・パスです。

イメージはデフォルトでタイル表示されます。イメージを広げて領域を埋めるようにする場合には、

```
, stretch"
```

を URL の最後に追加します。

例

以下の例では、背景が単一カラーで塗りつぶされます。

```
backgroundFill = "red"
```

以下の例では、背景が青から緑色の右下方向のグラジエントで塗りつぶされます。

```
backgroundFill = "blue, green, 7"
```

以下の例では、背景が黄色から緑色のグラジエントで塗りつぶされます。方向が指定されていないため、デフォルトで左から右になります。

```
backgroundFill = "yellow, green"
```

以下の例では、イメージを広げて背景を埋めます。

```
backgroundFill = "images/logo.gif, stretch"
```

関連項目

96 ページの『chartFill』、122 ページの『seriesFill』

barSeries

組み合わせチャート中のどのデータ系列がバー系列であるかを指定します。

データ・ソース

すべて

構文

```
barSeries="series"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
series	空ストリング	コンマで区切られたストリングで、バー系列で表示されるメンバー名を定義する

使用法

面グラフ、棒グラフ、および折れ線グラフを表示する場合、これら 3 つのチャート・タイプを組み合わせたチャートを表示することができます。データ系列を折れ線に、別のデータ系列を棒に、3 番目を面にすることができます。

このプロパティーは、組み合わせチャートの一部をなす面グラフ・タイプで表されるメンバーを識別します。表示されるメンバー名は、コンマで区切られたストリングとして定義されます。凡例のアイテムを成す複数のディメンションがある場合 (「チャート軸の配置 (Chart Axes Placement)」で定義)、タブ (“¥t”) を使用してディメンションを区切る必要があります。

例

```
barSeries = "Qtr1¥tVideo, Qtr2¥t Video, Qtr3¥tVideo"
```

関連項目

90 ページの『areaSeries』、110 ページの『lineSeries』

bloxEnabled

これは共通の Blox プロパティーおよびタグ属性です。詳しい説明は、38 ページの『bloxEnabled』を参照してください。

bloxName

これは共通の Blox プロパティーおよびタグ属性です。詳しい説明は、39 ページの『bloxName』を参照してください。

bookmarkFilter

これは共通の Blox プロパティーおよびタグ属性です。詳しい説明は、37 ページの『bookmarkFilter』を参照してください。

chartAbsolute

負の値を正の値として扱うかどうかを指定します。

データ・ソース

すべて

構文

```
chartAbsolute="chartAbsolute"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
chartAbsolute	false	ブール値。負の値が正の値として扱われるようにするには true を指定し、そうでない場合は false を指定します。

使用法

例えば、円グラフでは負の値は表示されません。このプロパティーを true に設定すると、値はチャート中で正の値として表示されます。

ヒント: 1 つ以上のチャート値が負の値である場合、ChartBlox は警告メッセージを表示します。メッセージのテキストを変更するには、absoluteWarning プロパティーを使用してください。

例

```
chartAbsolute ="true"
```

関連項目

89 ページの『absoluteWarning』

chartCurrentDimensions

現行のメンバーがチャート・フィルターに使用されるよう指定します。

データ・ソース

すべて

構文

```
chartCurrentDimensions="members"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
members	ヌル	現在選択されているチャート・フィルター項目であるストリングの配列。メンバーの設定時には、メンバーはチャートのページ・フィルター中のディメンションと同じ順序である必要があります。例えば、製品およびロケーションがチャートのページ・フィルターにある場合、“Coke, East” を選択メンバーとして指定できます。

chartFill

チャート・フレームの内側の、データの表示ではない領域を埋める、単一カラー、またはイメージを指定できるようにします。

データ・ソース

すべて

構文

```
chartFill="fill"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
fill	ヌル	チャートの背景の領域に色またはイメージを定義するストリング。デフォルトは #F0F0F0 (とても薄いグレー) です。

使用法

ストリング fill は、背景に表示する単一カラーまたはイメージへの URL のいずれかです。色の指定には、標準 Java カラー名または RGB 値を使用します。RGB 値を使用する場合には、0xffffff の形式で入力してください。

イメージの使用を指定する場合には、以下のいずれかである必要があります。

- アプリケーション・コンテキストからイメージへの相対 URL。例えば、 JSP が「salesApp」というアプリケーション・コンテキストにあり、 salesApp/images/ ディレクトリー中のイメージ・ファイル logo.gif を使用する場合には、以下の相対 URL を指定します。

  ```
  chartFill = "images/logo.gif"
  ```

- “http:” で始まる絶対 URL。

  ```
  chartFill ="http://serverName/path/to/image.gif"
  ```

参照されるイメージ・ファイルがあるサーバーに認証が必要ないことを確認してください。認証が必要な場合、イメージはロードされず、デフォルトの色が使用されます。これは、チャート作成エンジンに、認証に必要なユーザー名およびパスワードがないためです。

- 以下のファイル・プロトコルを使用する “file:” で始まる URL。

  ```
  chartFill ="file:///C|/DB2Alphablox/webapps/salesApp/images/logo.gif"
  ```

  これは、DB2 Alphablox が稼働しているサーバー上のイメージへのファイル・パスです。

イメージはデフォルトでタイル表示されます。イメージを広げて領域を埋めるようにする場合には、

```
, stretch"
```

を URL の最後に追加します。

例

```
chartFill = "red"
chartFill = "http://someServer/images/mypicture.gif"
chartFill = "file:///C|/Alphablox5/webapps/salesApp/images/logo.gif, stretch"
```

関連項目

103 ページの『footnoteStyle』、108 ページの『labelStyle』、125 ページの『titleStyle』、122 ページの『seriesFill』

chartType

表示するチャートのタイプを識別します。

データ・ソース

すべて

構文

```
chartType="type"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
type	"Vertical Bar, Side-by-Side, 3D Effect"	77 ページの『使用可能なチャート・タイプ』を参照。

使用法

頻繁に使用される値には、“3D Bar”、“Bar”、“Pie”、および “Line” が含まれます。値は、77 ページの『使用可能なチャート・タイプ』の表にある項目の 1 つと完全に一致する必要があります。

注: さまざまなタイプを表示するのに一番良い方法は、ChartBlox のある簡単なアプリケーション・ページを作成することです。それからそのアプリケーションを呼び出し、「**チャート・タイプ**」ダイアログ・ボックスを使用してチャート・タイプのプレビューを表示します。

例

```
chartType="Vertical Bar, Stacked"
```

columnLevel

チャートが使用するデータ世代を指定します。

データ・ソース

すべて

構文

```
columnLevel="levels"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
levels	なし	ディメンション・レベルのセットを指定する整数の、コンマで区切られたリスト。レベル 0 がすべてのレベルの親です。最初の整数は列軸の最初のディメンションのレベルで、2 番目の整数は 2 番目のディメンションのレベルを示すという具合です。

使用法

このプロパティーの設定では、totalsFilter プロパティーを 2 に設定する必要があります。

例

以下の例では、列軸に配置する最初のディメンションの世代レベルを 2 に、そして 2 番目のディメンションに関しては 4 に設定します。

```
columnLevel="2, 4"
```

関連項目

119 ページの『rowLevel』、126 ページの『totalsFilter』

columnSelections

チャート化するデータのサブセットを指定します。

データ・ソース

すべて

構文

```
columnSelections="selections"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
selections	ヌル	セミコロンで区切られたタプルで成るストリング。タプルのメンバーはコンマで分離されています。

使用法

値は、セミコロンで区切られたタプルのリストから成るストリングで、各タプルのメンバーがコンマで分離されています。columnSelections および rowSelections は両方とも、ユーザーがグリッド中のデータを選択し、選択したデータのチャート化を選択する際に自動的に設定されます。しかし、DHTML クライアントへのロード時にチャートが指定されたデータを表示するように、Blox でこれらを定義できます。チャートにデータが表示されるようにするには、rowSelections および columnSelections プロパティーの両方を設定する必要があります。どちらかが設定されていない場合、チャートは空になります。

デフォルト値 null は、すべてのデータがチャート化されることを指示します。

注: メンバー名中にコンマまたはセミコロンがある場合には、以下のように各メンバー名を二重引用符で囲み、二重引用符をエスケープする必要があります。

```
columnSelections="¥"East¥", ¥"Qtr1¥"; ¥"East¥", ¥"Qtr2¥""
```

例

```
columnSelections="East, Qtr1; East, Qtr2"
rowSelections="Actual, Audio; Actual, Visual"
```

関連項目

120 ページの『rowSelections』 追加の例は、25 ページの『Blox API を含むスクリプトレット』を参照してください。

comboLineDepth

コンボ・チャートの線の深さを指定します。

データ・ソース

すべて

構文

```
comboLineDepth="depth"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
depth	0	コンボ・チャートの線の深さ (ピクセル)。

dataTextDisplay

棒グラフまたはウォーターフォール図で、データ値を各バーの上に表示するかどうかを制御します。

データ・ソース

すべて

構文

```
dataTextDisplay="display"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
display	false	ブール引数。 true の値は、棒グラフまたはウォーターフォール図のバーの上にデータ値が表示されるように設定します。 false の値は、データ値が表示されないよう指示します。

例

```
dataTextDisplay="true"
```

dataValueLocation

チャートで使用されるディメンション名およびメンバー名のリストを指定します。

データ・ソース

すべて

構文

```
dataValueLocation="data"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
data	ヌル	"Dimension:Member1, Member2..." という形式のストリング。 軸に複数のディメンションがある場合、“¥t” を使用して、ディメンションとディメンション上のメンバーを区切ってください。 "Dim1¥tDim2...:Member1ofDim1¥tMember1ofDim2, Member2ofDim1¥tMember2ofDim2..."

使用法

マルチディメンションのデータをチャート化する際に、このプロパティーは、エレメントごとに複数のデータ値を必要とするチャート・タイプにどのデータを使用するかを定義します。リレーショナル・データをチャート化するときには、常にこのプロパティーを使用してどのデータをチャート化するかを定義する必要がありま

す。数値データの列のみを使用します。列に他のデータが含まれていると、チャートに NULL 値が使用され、意味のないチャートになります。

データの構文は、ディメンション名の後にコロンおよびコンマで区切られたメンバー名のリストが続いたものです。軸に複数のディメンションがある場合 (例えばバブル・チャート)、“¥t” を使用してディメンションおよびメンバーを区切ることができます。

```
dataValueLocation="Scenario¥tMeasures: Var% LY¥tFS Sales,
   Act¥tPromo %, Act¥tFS Sales"
```

上記の例では、2 つのディメンションが軸 Scenario および Measures を構成します。これらのディメンションはその間の “¥t” で区切られ、使用するメンバーもその間の “¥t” で指定されています。

リレーショナル・データの場合、列ディメンションの名前は常に “Columns” です。例えば、以下のようにします。

```
dataValueLocation="Columns: Product1, Product2"
```

特定のチャートが正しく表示されるためには、特定の順序でデータ値を定義する必要があります。その順序は使用するチャートのタイプによります。チャート・タイプおよびデータ値要件のリストは、77 ページの『使用可能なチャート・タイプ』を参照してください。

depthRadius

2 次元チャートにおける立体効果の深さを設定します。

データ・ソース

すべて

構文

```
depthRadius="radius"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
radius	0	立体効果の度合いを示す 0 から 100 までの整数。

使用法

デフォルト値 0 は、立体効果を除去します。値が高ければ高いほど、それだけ立体効果が著しくなります。

例

```
depthRadius="45"
```

dwellLabelsEnabled

ユーザーがチャート・エレメント上でマウスを動かした時に、ポップアップ・ラベル (データ値のテキスト記述) が表示されるかどうかを指定します。

データ・ソース

すべて

構文

```
dwellLabelsEnabled="enabled"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
enabled	true	ブール引数。 true の値はマウスオーバー・ラベルがチャートに表示されることを示し、false の値はラベルが表示されないよう指示します。

例

```
isDwellLabelsEnabled();
setDwellLabelsEnabled(false);
```

enablePoppedOut

これは、ContainerBlox から継承されたプロパティーおよびタグ属性です。ChartBlox が PresentBlox 内にネストされている場合、次のようになります。

- poppedOut プロパティーおよびそれに関連したプロパティーが PresentBlox で指定されている場合、PresentBlox の設定が使用されます。
- poppedOut プロパティーおよびそれに関連したプロパティーが PresentBlox で指定されていない場合、ネストされた ChartBlox のポップアウト設定が PresentBlox に適用されます。

詳しい説明は、175 ページの『enablePoppedOut』を参照してください。

filter

チャート・ディメンション・フィルターに表示されるディメンションを指定します。

データ・ソース

すべて

構文

```
filter="filter"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
filter	空ストリング	フィルター・ディメンションを定義するコンマで区切られたストリング。

使用法

デフォルトを使用すると、ChartBlox がディメンション配置を決定します。setFilter() メソッドはチャートを自動的にリフレッシュします。

例

```
filter="Product"
```

footnote

チャートの脚注 (チャートの右下) にテキストが表示されるよう指定します。

データ・ソース

すべて

構文

```
footnote="text"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
text	空ストリング	任意のストリング。そのテキストがチャートの脚注に表示されます。

例

```
footnote="Company Confidential"
```

関連項目

103 ページの『footnoteStyle』

footnoteStyle

脚注のスタイル (前景色およびテキスト書式) を指定します。

データ・ソース

すべて

構文

```
footnoteStyle="style"
```

または

```
<blox:footnoteStyle
    font=""
    foreground="">
</blox:footnoteStyle>
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
style	空ストリング	スタイル属性を定義するストリング。

使用法

スタイル・ストリングの指定方法について詳しくは、79 ページの『スタイルの指定』を参照してください。

例

```
footnoteStyle="foreground=white, font=Courier:Bold:10"
```

関連項目

91 ページの『axisTitleStyle』、103 ページの『footnote』、108 ページの『labelStyle』、125 ページの『titleStyle』

formatProperties

デフォルトをオーバーライドするチャート形式プロパティー・ストリングを指定します。これらの書式プロパティーは、色、スタイル、およびデータ系列の色や X 軸テキスト回転のカスタム角度などの他のチャートの属性を設定するために DHML クライアント・ユーザー・インターフェースによって使用されます。

データ・ソース

すべて

構文

```
formatProperties="formatProperties"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
formatProperties	空ストリング	引数は、オブジェクト・プロパティー/値ストリングのコンマで区切られたストリングの形式である必要があります。各オブジェクトのプロパティー/値ストリングにはそれぞれ、 セミコロンで区切られ、中括弧で囲まれた *プロパティー :値* の組がなければなりません。たとえば、以下のようにします。 *ObjectName={property1:value1; property2:value2;}, ObjectName2={property4:value4; property5:value5;}"* 複数の値が関係するプロパティーでは、値をコンマで区切ります。 *ObjectName1={property1:value1; property2:value2a,value2b;}, ObjectName2={property3:value3a, value3b; property4:value4a,value4b;}"* 代わりに、有効範囲ストリングをキーとして使用することもできます。たとえば、以下のようにします。 *{Dim0:Mem0}{Dim1:Mem1}= {property0:value0,value1;property1:value2;}, {Dim2:Mem2,Mem3}={property2:value3;}* 有効範囲ストリングの構文は、DataBlox の calculatedMembersセクションにある197 ページの『有効範囲』を参照してください。

使用法

このプロパティーは現在、個々のデータ系列の色、x 軸テキスト回転のカスタム角度、およびウォーターフォール型カラー配列の設定のみに使用されています。他のすべては、通常の名前のチャート・プロパティーを通して設定してください。

例

```
formatProperties="colorSeries_default_0 = {foreground:yellow;},
colorSeries_default_1 = {foreground:red;},
colorSeries_default_4 = {foreground:#FF9900;},
chart={XAxisTextRotation:45;}"
```

gridLineColor

グリッド線の色を設定します。

データ・ソース

すべて

構文

```
gridLineColor="color"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
color	black	カラーの名前または 16 進値。

使用法

アプリケーションが DHTML クライアントにレンダリングされた際のグリッド線のデフォルト・カラーは #D0D0D0 (薄いグレー) です。Java カラーについて詳しくは、http://java.sun.com/j2se/1.4.2/docs/api/java/awt/Color.html を参照してください。

例

```
gridLineColor="red"
gridLineColor="#00ffff"
```

gridLinesVisible

2 次元のチャートで、線が背景に表示されるかどうかを指定します。

データ・ソース

すべて

構文

```
gridLinesVisible="enabled"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
visible	true	グリッド線を表示する場合は true、グリッド線を隠す場合は false を指定します。

使用法

3D 棒グラフなどの幾つかのチャートでは、gridLinesVisible が true に設定されていてもグリッド線は表示されません。

例

```
gridLinesVisible="false"
```

groupSmallValues

比較的小さい値を円グラフの「その他」の項目にグループ化します。このプロパティーは円グラフのみに影響します。

データ・ソース

すべて

構文

```
groupSmallValues="groupSmall"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
groupSmall	true	ブール引数。 true の値はチャートに小さすぎる値は「その他」のカテゴリーにグループ化されることを示し、false の値はそれらがチャート化されることを示します。

使用法

小さい値の多い円グラフは読みにくいため、項目をグループ化するとチャートの読みやすさが向上します。

このグループ化の最小パーセンテージは、smallValuePercentage プロパティーによって設定されます。

例

```
groupSmallValues="false"
```

関連項目

124 ページの『smallValuePercentage』

height

これは共通の Blox プロパティーおよびタグ属性です。詳しい説明は、42 ページの『height』を参照してください。

helpTargetFrame

これは共通の Blox プロパティーおよびタグ属性です。詳しい説明は、42 ページの『helpTargetFrame』を参照してください。

histogramOptions

ヒストグラム・チャートのオプションを設定します。

データ・ソース

すべて

構文

```
histogramOptions="options"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
options	binMode=basic; binSize=false; binCount=6; addCumm=false; sort=false	セミコロンで区切られたオプションと値の組のストリング。有効なオプションは以下のとおりです。 • addCumm: true または false。チャートに累積比率線を追加する (Pareto グラフでのように) にはtrue。デフォルトは false です。 • binCount: チャートに含めるビン数。 • binList: コンマで区切られた数値のリスト。明示的にビンの範囲を設定するために使用されるカスタム・ビン作成 (binMode=custom) 用。リスト中のそれぞれの数値は、ビンの上限値でその値も含まれます。 • binMode: basic (デフォルト) または custom。basic モードでは、ビンは binCount または binSize のいずれかを通して設定されます。binCount の設定値はビン範囲を決定します。binSize の設定値はチャート中のビン数を決定します。 • binSize: 値のソートに使用されるビンのサイズ。正の数値でなければなりません。 • maxBin: 最後の (最高の) ビンに保管する最大値。binCount または binSize のいずれかと組み合わせて使用され、ビン範囲またはビン数を決定します。これが設定されない場合、データ中の最大値が使用されます。 • minBin: 最後の (最低の) ビンに保管する最小値。binCount または binSize のいずれかと組み合わせて使用されます。これが設定されない場合、データ中の最小値が使用されます。 • useSize: true または false。基本ビン作成 (binMode=basic) では、useSize が true の場合、binSize を使用してビンが作成されます。そうでない場合は、binCount に基づいてビンが作成されます。 • sort: true または false。値 true では降順ソートになります。このオプションが addCumm (sort=true;addCumm=true) と組み合わされると、Pareto グラフが作成されます。

例

以下の例では、10 ビンのあるヒストグラム・チャートが作成され、このチャートには累積比率線が含まれます。

```
<blox:chart histogramOptions="addCumm=true;sort=true;binCount=10" .../>
```

labelStyle

チャート・ラベルのスタイル (前景色およびフォント) を指定します。

データ・ソース
すべて

構文
```
labelStyle="style"
```

または

```
<blox:labelStyle
    font=""
    foreground="">
</blox:labelStyle>
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
style	空ストリング	スタイル属性を定義するストリング。

使用法
スタイル・ストリングの指定方法について詳しくは、79 ページの『スタイルの指定』を参照してください。

例
```
labelStyle="foreground=white, font=Courier:Bold:10"
```

関連項目
91 ページの『axisTitleStyle』、103 ページの『footnoteStyle』、125 ページの『titleStyle』

legend
凡例に表示されるディメンションを指定します。

データ・ソース
すべて

構文
```
legend="legend"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
legend	空ストリング	コンマで区切られた、ディメンションのストリング。

使用法
デフォルトを使用すると、ChartBlox がディメンション配置を決定します。
setLegend() メソッドはチャートを自動的にリフレッシュします。

例

```
legend="Measures, Market"
```

関連項目

110 ページの『legendPosition』

legendPosition

チャート凡例の表示/非表示、およびどこに表示するかを指定します。

データ・ソース

すべて

構文

```
legendPosition="position"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
position	bottom	値として none、bottom、right のいずれかを含むストリング。アプリケーションのレンダリング・モードが DHTML である場合、デフォルトは bottom です。アプリケーションのレンダリング・モードが Java に設定してある場合、デフォルトは right です。 ダイヤル・チャートでは、凡例がユーザーに意味がないため、ネストされた <blox:dial> タグを使用してダイヤル盤を指定した場合、legendPosition は自動的に none に設定されます。詳しくは、148 ページの『ダイヤル・チャートの作成』を参照してください。

使用法

有効な値は以下のとおりです。

- none - では凡例を表示しません。
- bottom - では凡例をチャートの下部に表示します。
- right - では凡例をチャートの右に表示します。

アプリケーションのデフォルト・レンダリング・モードが DHTML に設定されている場合、デフォルトは bottom です。

例

```
legendPosition="none"
```

関連項目

109 ページの『legend』

lineSeries

組み合わせチャート中のどのデータ系列が線系列であるかを指定します。

データ・ソース
すべて

構文
```
lineSeries="series"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
series	空ストリング	コンマで区切られたストリングで、線系列で表示されるメンバー名を定義する

使用法
面グラフ、棒グラフ、および折れ線グラフを表示する場合、これら 3 つのチャート・タイプを組み合わせたチャートを表示することができます。データ系列を折れ線に、別のデータ系列を棒に、3 番目を面にすることができます。

このプロパティーは、組み合わせチャートの一部をなす面グラフ・タイプで表されるメンバーを識別します。表示されるメンバー名は、コンマで区切られたストリングとして定義されます。軸を成す複数のディメンションがある場合 (「チャート軸の配置 (Chart Axes Placement)」で定義)、タブ (¥t) を使用してディメンションを区切る必要があります。

例
```
lineSeries="Qtr1¥tAll Products, Qtr2¥tAll Products, Qtr3¥tAll Products"
```

関連項目
90 ページの『areaSeries』、94 ページの『barSeries』

lineWidth
折れ線グラフの線の幅を制御します。

データ・ソース
すべて

構文
```
lineWidth="width"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
width	3	線幅をピクセルで定義する正の整数。

例
```
lineWidth="7"
```

localeCode

これは共通の Blox プロパティーおよびタグ属性です。詳しい説明は、43 ページの『localeCode』を参照してください。

logScaleBubbles

対数スケールを使用してバブル・サイズをバブル・チャートに設定するかどうかを指定します。

データ・ソース

すべて

構文

```
logScaleBubbles="useLogScale"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
useLogScale	false	対数スケールを使用してバブル・サイズを設定するときは、true です。

markerShape

マーカーの形状を設定します。

データ・ソース

すべて

構文

```
markerShape="shape"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
shape		数の、コンマで区切られたリスト。有効な値は以下のとおりです。 • 0 = ヌル • 1 = 正方形 • 2 = 円 • 3 = ひし形 • 4 = プラス記号 • 5 = 三角形/下向き • 6 = 三角形/上向き

使用法

形状は繰り返します。プロパティーを ″1,3,4″ に設定すると、最初の系列のマーカーが正方形、次がひし形、3 番目がプラス記号になり、その後 4 番目は正方形になり、という結果になります。

例

```
markerShape="1, 3,4"
```

markerSizeDefault

折れ線グラフおよびバブル・チャートに表示されるマーカーのサイズを設定します。

データ・ソース

すべて

構文

```
markerSizeDefault="size"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
size	30	有効値は 0 から 100 です。値 30 は約 10 ピクセルです。値 100 は約 30 ピクセルです。

例

```
markerSizeDefault="10"
```

maxChartItems

チャートの結果セットで許可する項目の最大数を設定します。結果セットがこの数を超過した場合、チャート生成は停止し、エラー・メッセージが出ます。

データ・ソース

すべて

構文

```
maxChartItems="items"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
items	256	チャート化できる項目の最大数を示す、正の整数。

使用法

チャートによっては、一定の数の項目がチャート化された後に読みにくくなるものがあります。このプロパティーは、チャート化する項目数を制限するのに役立ちます。チャート化してもまだ読むのが可能な項目の実際の数は、チャートのサイズが大きいほど大きくなります。

例

```
maxChartItems="10"
```

maximumUndoSteps

これは共通の Blox プロパティーおよびタグ属性です。詳しい説明は、44 ページの『maximumUndoSteps』を参照してください。

menubarVisible

これは共通の Blox プロパティーおよびタグ属性です。詳しい説明は、45 ページの『menubarVisible』を参照してください。

mustIncludeZero

チャートの軸上にゼロを含めるかどうかを指定します。

データ・ソース

すべて

構文

```
mustIncludeZero="includeZero"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
includeZero	true	ブール引数。 true の値はゼロがチャート化されることを示し、false の値はゼロがチャート化されないことを示します。

使用法

true に設定された場合、チャートの軸に常にゼロがあるようになります。チャートは、測定の実際の開始点に関わりなく、ゼロからカウントを始めます。false に設定した場合、測定はチャート上の最小値に最も近い点から始まります。ログ・スケール ([axis]LogScale) または [axis]ScaleMin を使用するには、mustIncludeZero は false に設定する必要があります。

例

```
mustIncludeZero="false"
```

noDataMessage

これは共通の Blox プロパティーおよびタグ属性です。詳しい説明は、45 ページの『noDataMessage』を参照してください。

o1AxisTitle

O1 軸のタイトルを明示的に定義します。

データ・ソース

すべて

構文

```
o1AxisTitle="title"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
title	ヌル	軸タイトルのテキストを示す任意のストリング。

使用法

O1 軸は、グループまたはカテゴリーを含む、チャート中の最初の序数軸です。チャート軸について詳しくは、79 ページの『チャートの軸』を参照してください。リレーショナル・データをチャート化する場合、チャートは軸タイトルを自動的には表示しません。チャートに表示するタイトルはすべて定義する必要があります。

マルチディメンションのデータ・ソースでタイトルを指定することもできますが、必要ではありません。この場合のデフォルト値、ヌルが自動的に軸タイトルを設定し、空ストリングはタイトルを表示しません。getter メソッドの戻り値 `null` は、チャートがマルチディメンション・データ・ソースから自動的に軸タイトルを判別したことを示します。

例

```
o1AxisTitle="This is the O1 Axis"
```

関連項目

130 ページの『x1AxisTitle』、137 ページの『y1AxisTitle』、142 ページの『y2AxisTitle』

pieFeelerTextDisplay

このプロパティーは円グラフの扇形スライスにラベルを付けるか、またどのように付けるかを定義します。

データ・ソース

すべて

構文

```
pieFeelerTextDisplay="type"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
type	1	0 から 3 の整数で、それらの値も含む。

使用法

有効な値とその意味は以下のとおりです。

0 = 扇形スライスにラベルを付けない。
1 = 「引き出し線」(扇形スライスからテキストへ伸びる線) の最後にそれぞれのテキスト・ラベルを表示する。
2 = ラベルのみを、引き出し線なしで表示する。ラベルはスライスのすぐ外側に配置される。
3 = ラベルを直接、扇形スライス上に配置する。

例

```
pieFeelerTextDisplay="3"
```

poppedOut

これは、ContainerBlox から継承されたプロパティーです。ChartBlox が PresentBlox 内にネストされている場合、次のようになります。

- poppedOut プロパティーおよびそれに関連したプロパティーが PresentBlox で指定されている場合、PresentBlox の設定が使用されます。
- poppedOut プロパティーおよびそれに関連したプロパティーが PresentBlox で指定されていない場合、ChartBlox のポップアウト設定が PresentBlox に適用されます。

詳しい説明は、176 ページの『poppedOut』を参照してください。

poppedOutHeight

これは、ContainerBlox から継承されたプロパティーです。詳しい説明は、176 ページの『poppedOutHeight』を参照してください。

poppedOutTitle

これは、ContainerBlox から継承されたプロパティーです。詳しい説明は、177 ページの『poppedOutTitle』を参照してください。

poppedOutWidth

これは、ContainerBlox から継承されたプロパティーです。詳しい説明は、178 ページの『poppedOutWidth』を参照してください。

quadrantLineCountX

バブル・チャートに表示される縦線の数を設定します。これは他のすべてのチャート・タイプでは無視されます。

データ・ソース
すべて

構文
```
quadrantLineCountX="count"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
count	1	値は 1 以上でなければなりません。

使用法
四分区間線をすべて除去するには、quadrantLineDisplay プロパティーを使用します。

例
```
quadrantLineCountX="2"
```

関連項目
117 ページの『quadrantLineCountY』、118 ページの『quadrantLineDisplay』

quadrantLineCountY
バブル・チャートに表示される水平線の数を設定します。これは他のすべてのチャート・タイプでは無視されます。

データ・ソース
すべて

構文
```
quadrantLineCountY="count"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
count	1	値は 1 以上でなければなりません。

使用法
四分区間線をすべて除去するには、quadrantLineDisplay プロパティーを使用します。

例
```
quadrantLineCountY="2"
```

関連項目
116 ページの『quadrantLineCountX』、118 ページの『quadrantLineDisplay』

quadrantLineDisplay

バブル・チャートで四分区間線を表示するかどうかを設定します。

データ・ソース

すべて

構文

```
quadrantLineDisplay="display"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
display	true	四分区間線を表示しない場合には、このプロパティーを false に設定します。

例

```
quadrantLineDisplay="false"
```

関連項目

116 ページの『quadrantLineCountX』、117 ページの『quadrantLineCountY』

removeAction

これは共通の Blox プロパティーです。詳しい説明は、46 ページの『removeAction』を参照してください。

render

これは共通の Blox プロパティーです。詳しい説明は、47 ページの『render』を参照してください。

rightClickMenuEnabled

これは共通の Blox プロパティーです。詳しい説明は、48 ページの『rightClickMenuEnabled』を参照してください。

riserWidth

棒グラフの棒の幅を設定します。この値は使用可能なスペースに対する比率です。

データ・ソース

すべて

構文

```
riserWidth="width"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
width	75	使用可能なスペースに対するパーセンテージ。値 100 ではグループ間のスペースがなくなります。

例

```
riserWidth="85"
```

rowHeaderColumn

どの列が行ラベルの名前を含み、チャートがどの列を使用して軸ラベルを作成するかを指定します。

データ・ソース

リレーショナル

構文

```
rowHeaderColumn="name"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
name	ヌル	列名を含むストリング。

使用法

リレーショナル・データ専用です。

例

```
rowHeaderColumn="Column header name"
```

rowLevel

チャートが使用するデータ世代を戻します。

データ・ソース

すべて

構文

```
rowLevel="levels"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
levels	なし	ディメンション・レベルのセットを指定する整数の、コンマで区切られたリスト。レベル 0 がすべてのレベルの親です。

使用法

このメソッドでは、totalsFilter プロパティーを 2 に設定する必要があります。

例

```
rowLevel="3, 1"
```

関連項目

98 ページの『columnLevel』、126 ページの『totalsFilter』

rowsOnXAxis

使用可能に設定された場合、チャートの軸を交換し、データが反対の軸に表示されます。

データ・ソース

リレーショナル

構文

```
rowsOnXAxis="rowsOnXAxis"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
rowsOnXAxis	false	ブール引数。 true の値は行軸が X 軸にチャート化されることを示し、false の値は行軸が Y 軸にチャート化されることを示します。

例

```
rowsOnXAxis="true"
```

rowSelections

チャート化するデータのサブセットを指定します。

データ・ソース

すべて

構文

```
rowSelections="selections"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
selections	なし	セミコロンで区切られたタプルで成るストリング。タプルのメンバーはコンマで分離されています。

使用法

値は、セミコロンで区切られたタプルのリストから成るストリングで、各タプルのメンバーがコンマで分離されています。これらのプロパティー は、ユーザーがグリッド中のデータを選択し、選択したデータのチャート化を選択する際に自動的に設定されます。しかし、DHTML モードでのロード時にチャートが指定されたデータを表示するように、Blox でこれらを定義できます。チャートにデータが表示されるようにするには、rowSelections および columnSelections プロパティーの両方を設定する必要があります。どちらかが設定されていない場合、チャートは空になります。

デフォルト値 null は、すべてのデータがチャート化されることを指示します。

注: メンバー名中にコンマまたはセミコロンがある場合には、以下のように各メンバー名を二重引用符で囲み、二重引用符をエスケープする必要があります。

```
rowSelections="¥"Actual¥", ¥"Audio¥"; ¥"Actual¥", ¥"Visual¥""
```

例

```
columnSelections="East, Qtr1; East, Qtr2"
rowSelections="Actual, Audio; Actual, Visual"
```

関連項目

98 ページの『columnSelections』

seriesColorList

現行の系列をチャート化するときに使用される色のリストを設定します。

データ・ソース

すべて

構文

```
seriesColorList="list"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
list	ヌル	コンマで区切られた、色のリスト。

使用法

現行の系列をチャート化するときに使用される色のリストを設定します。色はコンマで区切られたストリングで指定されており、標準 Java カラー名または 16 進値です。アプリケーションが DHTML でレンダリングされるときのデフォルト・カラーは、"#9691C7", "#00B09B", "#68AEE0", "#008B87", "#99CCCC", "#005699", "#C2C4C6", "#998300", "#CCAE99", "#A76100", "#E0CB68", "#B03400". です。

チャート化するデータに十分な色をリストするようにしてください。十分な色を定義しない場合、指定した色が順々に残りの系列で反復されます。

例

```
seriesColorList = "red, green, blue, #FFCCFF"
```

関連項目

122 ページの『seriesFill』

seriesFill

チャート内でデータを表すバー、線、または面を埋める色のグラジエントまたはイメージを指定できるようにします。この API は、色またはイメージの充てん先であるチャート中のデータの系列を指示する、index 引数を取ります。

データ・ソース

すべて

構文

```
<blox:seriesFill
   index=""
   value="" >
</blox:seriesFill>
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
index	ヌル	存在するデータ系列の数を表す整数。
value	ヌル	指定のデータ系列に、色のグラジエントまたはイメージを定義するストリング。

使用法

value タグ属性/メソッド引数は、ストリングで、ある領域に表示する、色のグラジエントのための 2 色のリストかまたはイメージへの URL のいずれかです。色の指定には、標準 Java カラー名または RGB 値を使用します。RGB 値を使用する場合には、0xffffff または #ffffff の形式で入力してください。色のグラジエントを使用する場合、ストリングはコンマで区切られた 2 色のリストである必要があります。

バーにグラジエントを含める場合、グラジエントの方向は、ストリングの最後の項目として、以下の表から適切なグラジエント修飾子を追加することにより指定できます。バーに単一カラーを含める場合には、seriesFill プロパティーは設定しないで、代わりに seriesColorList プロパティーを設定します。

グラジエント方向	修飾子
右方	1 (方向が指定されていない場合のデフォルト)
左方	2
下	3
上	4
下/左	5

上/左	6
下/右	7
上/右	8

seriesFill の指定時にインデックスをスキップした場合、スキップされた系列は以前に指定された seriesFill と同じ seriesFill を使用します。以下の例では、系列 2 および 3 は 1 と同じグラジエントを使用します。

```
<blox:seriesFill index="1" value="green, yellow"/>
<blox:seriesFill index="4" value="blue, green"/>
```

イメージの使用を指定する場合には、以下のいずれかである必要があります。

- アプリケーション・コンテキストからイメージへの相対 URL。例えば、 JSP が「salesApp」というアプリケーションにあり、salesApp/images/ ディレクトリー中のイメージ・ファイル logo.gif を使用する場合には、次のようにします。

  ```
  <blox:seriesFill index="1" value="images/logo.gif" />
  ```

- “http:” で始まる絶対 URL。

  ```
  <blox:seriesFill index="2" value="http://serverName/path/to/image.gif" />
  ```

  参照されるイメージ・ファイルがあるサーバーに認証が必要ないことを確認してください。認証が必要な場合、イメージはロードされず、デフォルト系列の色が使用されます。これは、チャート作成エンジンに、認証に必要なユーザー名およびパスワードがないためです。

- 以下のファイル・プロトコルを使用する “file:” で始まる URL。

  ```
  <blox:seriesFill index="2"
  value="file:///C|/DB2Alphablox/webapps/salesApp/images/logo.gif" />
  ```

  これは、DB2 Alphablox が稼働しているサーバー上のイメージへのファイル・パスです。

イメージは常時タイル表示されます。透過性のある GIF イメージを使用する場合、それは系列の色 (121 ページの『seriesColorList』を参照) の上にオーバーレイされます。

例

```
<blox:chart ...>
   <blox:seriesFill index="1" value="green, yellow, 2"/>
</blox:chart>
```

上記の例では、最初のデータ系列が左方向の緑から黄色のグラジエントに設定されます。

関連項目

103 ページの『footnoteStyle』、108 ページの『labelStyle』、125 ページの『titleStyle』、96 ページの『chartFill』、121 ページの『seriesColorList』

showSeriesBorder

チャートのバーおよび凡例の四角形の周りの枠を表示するかどうかを指定します。

データ・ソース

すべて

構文

```
showSeriesBorder="show"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
show	false	true - バーおよび凡例の四角形の周りに枠を表示する。 false - 枠を表示しない。デフォルトは false です。

使用法

棒グラフにのみ適用されます。アプリケーションのレンダリング・モードが DHTML に設定されている場合、デフォルトは false です。

smallValuePercentage

比較的小さい値を円グラフの「その他」の項目にグループ化するための最小パーセンテージを設定します。このプロパティーは円グラフのみに影響します。

データ・ソース

すべて

構文

```
smallValuePercentage="percentage"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
percentage	5.0	タイプ double の引数。有効な値は 0.01 から 10.0 で、それらの値を含みます。

使用法

小さい値の多い円グラフは読みにくいため、項目をグループ化するとチャートの読みやすさが向上します。

例

```
smallValuePercentage"7.2"
```

関連項目

106 ページの『groupSmallValues』

title

チャートの上にタイトルとして表示されるテキストを指定します。

データ・ソース
マルチディメンション

構文
```
title="text"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
style	空ストリング	スタイル属性を定義するストリング。

例
```
title="My Title"
```

関連項目
125 ページの『titleStyle』

titleStyle
チャートのタイトルのスタイル (前景色およびテキスト書式) を指定します。

データ・ソース
すべて

構文
```
titleStyle="style"
```

または

```
<blox:titleStyle
    font=""
    foreground="">
</blox:titleStyle>
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
style	空ストリング	スタイル属性を定義するストリング。

使用法
スタイル・ストリングの指定方法について詳しくは、79 ページの『スタイルの指定』を参照してください。

例
```
titleStyle="foreground=white, font=Courier:Bold:10"
```

関連項目
91 ページの『axisTitleStyle』、103 ページの『footnoteStyle』、108 ページの『labelStyle』、124 ページの『title』

toolbarVisible

ツールバーを表示するかどうかを指定します。

データ・ソース

すべて

構文

JSP タグ属性

```
toolbarVisible="visible"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
visible	true	true - ツールバーは表示されます。false - ツールバーは表示されません。アプリケーションのレンダリング・モードが DHTML に設定されている場合、デフォルトでツールバーは表示されます。

使用法

デフォルトでは、独立型 ChartBlox でツールバーは表示されます。ネストされた <blox:toolbar> タグが追加された場合、その設定値はこの属性の値を上書きします。たとえば、次のコーディングではツールバーが表示されます。

```
<blox:chart id="myChart" toolbarVisible="false" ....>
   <blox:toolbar visible="true" />
   <blox:data bloxRef="myDataBlox"/>
</blox:chart>
```

ヒント: toolbarVisible は単にタグ属性であり、プロパティーではありません。

totalsFilter

チャートでの合計の表示/非表示、表示の仕方を指定します。

データ・ソース

マルチディメンション

構文

```
totalsFilter="type"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
type	1	0 から 2 の整数で、それらの値も含む。

使用法

有効な値とその意味は以下のとおりです。

0 = フィルターなし、すべての合計が表示されます。
1 = チャートで各ディメンションの最後 (ドリルダウンされた最下の) 世代を表示します。
2 = ユーザー指定の世代を表示します。

ヒント: 値 “2” ではチャートの下部にラジオ・ボタンのセットが表示され、ユーザーはそれを使ってチャートに表示される世代を選択します。使用可能な世代の範囲を制限するには、119 ページの『rowLevel』および 98 ページの『columnLevel』にあるプロパティーを参照してください。

例

```
totalsFilter="0"
```

関連項目

119 ページの『rowLevel』、98 ページの『columnLevel』

trendLines

チャートに傾向線を作成します。

データ・ソース

すべて

構文

```
trendLines="trendLines"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
trendLines	ヌル	コンマで区切られた、傾向線のストリング。 trendLines="[trendLine1],[trendLine2],...,[trendLine*N*]" 各傾向線は、セミコロンで区切られた *parameter=value* の組のストリングです。 param1=value1;param2=value2;...;param*N*=value*N* 有効なパラメーターは以下のとおりです。 • name: 必要。傾向線の名前のストリング。 • member: 必要。1 つ以上の member パラメーターを追加できます。傾向線のプロットに使用する、「*uniqueDimensionName*:*uniqueMemberName*」形式のメンバー。たとえば、member=All Locations:East となります。 • type: 必要。有効なタイプは以下のとおりです。 – exponential – linear – logarithmic – moving average(*N*): ここで、N は少なくとも 2 です。 – polynomial(*N*): ここで N は 2 以上、100 以下です。 – power • drilldownscope: オプショナル。有効な値は descendents および none です。 • replace: オプショナル。傾向線がそれが関連する線またはバーの代わりに引かれるようにするには、true に設定します。置換しないようにするには、false に設定します。 • forecastforward: オプショナル。予測する期間または単位の数。移動平均傾向線タイプではサポートされていません。 • color: オプショナル。Java カラーまたは 16 進値 (例えば、#CCCCFF) を指定します。 • style: オプショナル。線のスタイルに solid または dashed を設定します。

使用法

傾向線は、折れ線グラフ、棒グラフ、面グラフ、または散布図に追加できます。追加された傾向線は、メニュー・バーの「**チャート**」> 「**傾向線...**」で、ユーザーが変更することができます。

次の表に、サポートされる傾向線のタイプを示します。

タイプ	説明	式
線形	表されている線の最小二乗。	$y = c_0 + c_1 x$ C1 は傾き、C0 は切片です。

タイプ	説明	式
対数	データ・ポイントすべてについての最小二乗。	$y = c_0 + c_1 \ln x$ C0 および C1 は定数、ln は自然対数関数です。
多項式	データ・ポイントすべてについての最小二乗。	$y = c_0 + c_1 x + c_2 x^2 + c_3 x^3 + \ldots + c_n x^n$ C0 および C1...Cn は定数で、N は 2 以上、100 以下です。
累乗	データ・ポイントすべてについての最小二乗。	$y = cx^b$ c および b は定数です。
指数	データ・ポイントすべてについての最小二乗。	$y = ce^{bx}$ c および b は定数で、e は自然対数の底です。
移動平均	指定した時間枠内の平均。移動平均傾向線の期間数は、系列のポイント総数と等しいか、期間に指定する数より小さくなります。	$F_{(t+1)} = \frac{1}{N} \sum_{j=0}^{N-1} A_{t-j+1}$ N は移動平均に含める事前期間の数、Aj は時間 j における実際の値、Fj は時間 j における予測値です。

例

以下の例では、一方が 3 次の多項式で他方が直線の、2 本の傾向線が作成されます。

```
trendLines="name=poly(3);member=All Locations:All Locations;
replace=true; type=polynomial(3),
name=line;member=All Locations:Central;type=linear"
```

これにより、すべてのロケーションでそれぞれのメンバーをプロットする多項式傾向線が作成され、中央のロケーションには、線形傾向線が表示されます。元のデータ系列の線/バーは置換 (replace=true) されます。

useSeriesShapes

折れ線グラフの凡例が、チャート中のデータ・ポイントで使用される形状を表示するかどうかを設定します。

データ・ソース

すべて

構文

useSeriesShapes="*display*"

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
display	true	ブール引数。値 true は凡例が異なるデータ・ポイントの形状を表示することを指定し、値 false は形状を表示しない指定をします。アプリケーションのレンダリング・モードが DHTML に設定されている場合、デフォルトは true です。

例

```
useSeriesShapes="true"
```

visible

これは共通の Blox プロパティーです。詳しい説明は、49 ページの『visible』を参照してください。

width

これは共通の Blox プロパティーです。詳しい説明は、49 ページの『width』を参照してください。

x1AxisTitle

X1 軸のタイトルを明示的に定義します。このプロパティーは、棒グラフ、折れ線グラフ、および面グラフにのみ適用されます。

データ・ソース

すべて

構文

```
x1AxisTitle="title"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
title	ヌル	軸タイトルのテキストを示す任意のストリング。

使用法

リレーショナル・データをチャート化する場合、チャートは軸タイトルを自動的には表示しません。チャートに表示するタイトルはすべて定義する必要があります。

マルチディメンションのデータ・ソースでタイトルを指定することもできますが、必要ではありません。この場合のデフォルト値、ヌルが自動的に軸タイトルを設定し、空ストリングはタイトルを表示しません。getter メソッドの戻り値「null」は、チャートがマルチディメンション・データ・ソースから自動的に軸タイトルを判別したことを示します。

例

```
x1AxisTitle="This is the X1 Axis"
```

関連項目

115 ページの『o1AxisTitle』、115 ページの『pieFeelerTextDisplay』、135 ページの『XAxis』、135 ページの『XAxisTextRotation』、137 ページの『y1AxisTitle』、142 ページの『y2AxisTitle』

x1FormatMask

散布図またはバブル・チャートの X1 軸のフォーマット・マスクの値を指定します。

データ・ソース

すべて

構文

```
x1FormatMask="formatMask"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
formatMask	ヌル	251 ページの『数値のフォーマット』および 280 ページの『formatMask』を参照。

使用法

フォーマット・マスクは、チャートの軸値のカスタマイズしたフォーマットを指定できるようにします。このフォーマットは、マウスをチャート・データ項目の上で停止させると表示されるポップアップ・ラベルにも使用されます。例えば、X1 軸がパーセンテージの指標である場合、##0.00% をフォーマット・マスクに指定できます。フォーマット・マスクはグリッドでのフォーマット・マスクと同様に設定します。これらのプロパティーの値の設定方法については、251 ページの『数値のフォーマット』を参照してください。キーワード K および M (千と百万) がサポートされています。割り算 (/) および掛け算 (*) もサポートされます。

例

```
x1FormatMask="##0.00%"
x1FormatMask="$#,###/1000"
xy1FormatMask="#,###K"
```

それに最も近い 1 億単位に丸めたドル値に Y1 軸フォーマットを設定するには、以下のようにします。

```
y1FormatMask="$###,###,###.##"
```

関連項目

137 ページの『y1FormatMask』、143 ページの『y2FormatMask』、251 ページの『数値のフォーマット』

x1LogScale

X1 軸に対数スケールを使用するかどうかを設定します。

データ・ソース

すべて

構文

```
x1LogScale="width"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
logScale	false	X1 軸に対数スケールを使用するかどうかを指定します。

使用法

チャート・エンジンはスケールの最大および最小値を自動的に計算しないため、x1LogScaleが true に設定されている場合、以下のプロパティーの値も指定する必要があります。

- x1ScaleMaxAuto は false に設定する必要があります。
- x1ScaleMax の値を設定する必要があります。
- x1ScaleMinAuto は false に設定する必要があります。
- x1ScaleMin の値を設定する必要があります。
- mustIncludeZero は false に設定する必要があります。

ログ・スケールを 0 から開始することはできず、値は決して負ではないため、mustIncludeZero は false (デフォルトは true) に設定しなければなりません。

例

```
x1LogScale="true"
```

関連項目

133 ページの『x1ScaleMaxAuto』、132 ページの『x1ScaleMax』、134 ページの『x1ScaleMinAuto』、133 ページの『x1ScaleMin』、114 ページの『mustIncludeZero』、138 ページの『y1LogScale』、144 ページの『y2LogScale』

x1ScaleMax

X1 軸の最大値を設定します。

データ・ソース

すべて

構文

```
x1ScaleMax="scale"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
scale	ヌル	X1 軸の最大値。

使用法

このプロパティーは x1ScaleMaxAuto が `true` に設定されている場合、無視されます。 `x1ScaleMax` は常時 `x1ScaleMin` より大きい値に設定する必要があります。そうでない場合、チャートが正しく振る舞わないことがあります。

例

```
x1ScaleMax="500000"
```

関連項目

133 ページの『x1ScaleMaxAuto』、133 ページの『x1ScaleMin』

x1ScaleMaxAuto

X1 軸の最大値を自動的に設定するかどうかを設定します。これが `false` の場合、X1 軸の最大値設定には、x1ScaleMax プロパティーの値が使用されます。

データ・ソース

すべて

構文

```
x1ScaleMaxAuto="auto"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
auto	true	有効な値は `true` または `false` です。これが `false` に設定されている場合、X1 軸の最大値は x1ScaleMax プロパティーの値によって決定されます。デフォルトでは、これは `true` に設定され、x1ScaleMax プロパティーの値は無視されます。

使用法

軸にログ・スケールを使用する場合、すべての対応する *[axis]ScaleMaxAuto* および *[axis]ScaleMinAuto* は `false` に設定する必要があり、*[axis]ScaleMax* および *[axis]ScaleMin* に値を指定する必要があります。ログ・スケールには 0 または負の数値を含められないため、mustIncludeZero は `false` に設定しなければなりません。

例

```
x1ScaleMaxAuto="false"
```

関連項目

131 ページの『x1LogScale』、132 ページの『x1ScaleMax』

x1ScaleMin

X1 軸の最小値を設定します。

データ・ソース

すべて

構文

```
x1ScaleMin="scale"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
scale	ヌル	X1 軸の最小値。

使用法

このプロパティーは x1ScaleMinAuto が true に設定されている場合、無視されます。 mustIncludeZero が true に設定されている場合で、x1ScaleMin がゼロより大きい場合、これは優先されます。x1ScaleMax は常時 x1ScaleMin より大きい値に設定する必要があります。そうでない場合、チャートが正しく振る舞わないことがあります。

例

```
x1ScaleMin="10000"
```

関連項目

134 ページの『x1ScaleMinAuto』、132 ページの『x1ScaleMax』、114 ページの『mustIncludeZero』

x1ScaleMinAuto

X1 軸の最小値を自動的に設定するかどうかを設定します。これが false の場合、X1 軸の最小値設定には、x1ScaleMin プロパティーの値が使用されます。

データ・ソース

すべて

構文

```
x1ScaleMinAuto="auto"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
auto	true	有効な値は true または false です。これが false に設定されている場合、X1 軸の最小値は x1ScaleMin プロパティーの値によって決定されます。デフォルトでは、これは true に設定され、x1ScaleMin プロパティーの値は無視されます。

使用法

軸にログ・スケールを使用する場合、すべての対応する *[axis]ScaleMaxAuto* および *[axis]ScaleMinAuto* は false に設定する必要があり、*[axis]ScaleMax* および

[axis]ScaleMin に値を指定する必要があります。ログ・スケールには 0 または負の数値を含められないため、mustIncludeZero は `false` に設定しなければなりません。

例

```
x1ScaleMinAuto="false"
```

関連項目

131 ページの『x1LogScale』、133 ページの『x1ScaleMin』

XAxis

X 軸上のディメンションを指定します。このプロパティーは、棒グラフ、折れ線グラフ、および面グラフにのみ適用されます。

データ・ソース

すべて

構文

```
XAxis="xAxis"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
xAxis	空ストリング	コンマで区切られた、ディメンション名のストリング。

使用法

デフォルト使用時には、ChartBlox がディメンション配置を決定します。`setXAxis()` メソッドはチャートを自動的にリフレッシュします。

チャート軸について詳しくは、79 ページの『チャートの軸』を参照してください。

例

```
xAxis="All Products"
```

関連項目

130 ページの『x1AxisTitle』、131 ページの『x1FormatMask』、135 ページの『XAxisTextRotation』、136 ページの『y1Axis』、141 ページの『y2Axis』

XAxisTextRotation

X 軸のラベル回転を指定します。このプロパティーは、棒グラフ、折れ線グラフ、および面グラフにのみ適用されます。

データ・ソース

すべて

構文

```
XAxisTextRotation="type"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
type	0	0 から 2 の整数で、それらの値も含む。

使用法

受け入れられる値は以下のとおりです。

0 = 標準
1 = 90 度回転
2 = 互い違い

例

```
xAxisTextRotation="2"
```

関連項目

130 ページの『x1AxisTitle』、135 ページの『XAxis』

y1Axis

Y1 軸のメンバー名を指定します。

データ・ソース

すべて

構文

```
y1Axis="y1Axis"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
y1Axis	空ストリング	メンバー表示名の、コンマで区切られたストリング。メンバーは、同じディメンションからのものである必要があります。

使用法

チャート軸について詳しくは、79 ページの『チャートの軸』を参照してください。

デフォルトでは、ChartBlox がディメンション配置を決定します。setY1Axis() メソッドはチャートを自動的にリフレッシュします。

例

```
y1Axis="Market"
```

関連項目

137 ページの『y1AxisTitle』、137 ページの『y1FormatMask』、141 ページの『y2Axis』

y1AxisTitle

Y1 軸のタイトルを明示的に定義します。

データ・ソース

すべて

構文

```
y1AxisTitle="title"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
title	ヌル	軸タイトルのテキストを示す任意のストリング。

使用法

チャート軸について詳しくは、79 ページの『チャートの軸』を参照してください。

リレーショナル・データをチャート化する場合、チャートは軸タイトルを自動的には表示しません。チャートに表示するタイトルはすべて定義する必要があります。

マルチディメンションのデータ・ソースでタイトルを指定することもできますが、必要ではありません。この場合のデフォルト値、ヌルが自動的に軸タイトルを設定し、空ストリングはタイトルを表示しません。getter メソッドの戻り値「null」は、チャートがマルチディメンション・データ・ソースから自動的に軸タイトルを判別したことを示します。

例

```
y1AxisTitle="This is the Y1 Axis"
```

関連項目

115 ページの『o1AxisTitle』、115 ページの『pieFeelerTextDisplay』、130 ページの『x1AxisTitle』、136 ページの『y1Axis』、137 ページの『y1FormatMask』、142 ページの『y2AxisTitle』

y1FormatMask

チャートの Y1 軸のフォーマット・マスクの値を指定します。

データ・ソース

すべて

構文

```
y1FormatMask="formatMask"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
formatMask	ヌル	251 ページの『数値のフォーマット』および 280 ページの『formatMask』を参照。

使用法

フォーマット・マスクは、チャートの軸値のカスタマイズしたフォーマットを指定できるようにします。このフォーマットは、マウスをチャート・データ項目の上で停止させると表示されるポップアップ・ラベルにも使用されます。例えば、Y1 軸がパーセンテージの指標である場合、##0.00% をフォーマット・マスクに指定できます。フォーマット・マスクはグリッドでのフォーマット・マスクと同様に設定します。これらのプロパティーの値の設定方法については、251 ページの『数値のフォーマット』を参照してください。キーワード K および M (千と百万) がサポートされています。割り算 (/) および掛け算 (*) もサポートされます。

例

```
y1FormatMask="##0.00%"
y1FormatMask="$#,###/1000"
y1FormatMask="#,###K"
```

それに最も近い 1 億単位に丸めたドル値に Y1 軸フォーマットを設定するには、以下のようにします。

```
y1FormatMask="$###,###,###.##"
```

関連項目

143 ページの『y2FormatMask』、131 ページの『x1FormatMask』、251 ページの『数値のフォーマット』

y1LogScale

Y1 軸に対数スケールを使用するかどうかを設定します。

データ・ソース

すべて

構文

```
y1LogScale="width"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
logScale	false	Y1 軸に対数スケールを使用するかどうかを指定します。

使用法

チャート・エンジンはスケールの最大、最小値を自動的に計算しないため、y1LogScale が true に設定されている場合、以下のプロパティーの値も指定する必要があります。

- y1ScaleMaxAuto は false に設定する必要があります。
- y1ScaleMax の値を設定する必要があります。
- y1ScaleMinAuto は false に設定する必要があります。
- y1ScaleMin の値を設定する必要があります。
- mustIncludeZero は false に設定する必要があります。

ログ・スケールを 0 から開始することはできず、値は決して負ではないため、mustIncludeZero は false (デフォルトは true) に設定しなければなりません。

例

```
y1LogScale="true"
```

関連項目

139 ページの『y1ScaleMaxAuto』、139 ページの『y1ScaleMax』、141 ページの『y1ScaleMinAuto』、140 ページの『y1ScaleMin』、114 ページの『mustIncludeZero』、144 ページの『y2LogScale』、131 ページの『x1LogScale』

y1ScaleMax

Y1 軸の最大値を設定します。

データ・ソース

すべて

構文

```
y1ScaleMax="scale"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
scale	null	Y1 軸の最大値。

使用法

このプロパティーは y1ScaleMaxAuto が true に設定されている場合、無視されます。 y1ScaleMax は常時 y1ScaleMin より大きい値に設定する必要があります。そうでない場合、チャートが正しく振る舞わないことがあります。

```
y1ScaleMax="500000"
```

関連項目

138 ページの『y1LogScale』、139 ページの『y1ScaleMaxAuto』、140 ページの『y1ScaleMin』

y1ScaleMaxAuto

Y1 軸の最大値を自動的に設定するかどうかを設定します。これが false の場合、Y1 軸の最大値設定には、y1ScaleMax プロパティーの値が使用されます。

データ・ソース

すべて

構文

y1ScaleMaxAuto="*auto*"

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
auto	true	有効な値は true または false です。これが false に設定されている場合、Y1 軸の最大値は y1ScaleMax プロパティーの値によって決定されます。デフォルトでは、これは true に設定され、y1ScaleMax プロパティーの値は無視されます。

使用法

軸にログ・スケールを使用する場合、すべての対応する [axis]ScaleMaxAuto および [axis]ScaleMinAuto は false に設定する必要があり、[axis]ScaleMax および [axis]ScaleMin に値を指定する必要があります。ログ・スケールには 0 または負の数値を含められないため、mustIncludeZero は false に設定しなければなりません。

例

```
y1ScaleMaxAuto="false"
```

関連項目

138 ページの『y1LogScale』、139 ページの『y1ScaleMax』、140 ページの『y1ScaleMin』、141 ページの『y1ScaleMinAuto』

y1ScaleMin

Y1 軸の最小値を設定します。

データ・ソース

すべて

構文

y1ScaleMin="*scale*"

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
scale	ヌル	Y1 軸の最小値。

使用法

このプロパティーは y1ScaleMinAuto が true に設定されている場合、無視されます。 y1ScaleMax は常時 y1ScaleMin より大きい値に設定する必要があります。そ

うでない場合、チャートが正しく振る舞わないことがあります。y1ScaleMin が 0 より大きい場合には優先されるため、mustIncludeZeo は false に設定する必要があります。

例

```
y1ScaleMin="10000"
```

関連項目

141 ページの『y1ScaleMinAuto』、139 ページの『y1ScaleMax』

y1ScaleMinAuto

Y1 軸の最小値を自動的に設定するかどうかを設定します。これが false の場合、Y1 軸の最小値設定には、y1ScaleMin プロパティーの値が使用されます。

データ・ソース

すべて

構文

```
y1ScaleMinAuto="auto"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
auto	true	有効な値は true または false です。これが false に設定されている場合、Y1 軸の最小値は y1ScaleMin プロパティーの値によって決定されます。デフォルトでは、これは true に設定され、y1ScaleMin プロパティーの値は無視されます。

使用法

軸にログ・スケールを使用する場合、すべての対応する *[axis]ScaleMaxAuto* および *[axis]ScaleMinAuto* は false に設定する必要があり、*[axis]ScaleMax* および *[axis]ScaleMin* に値を指定する必要があります。ログ・スケールには 0 または負の数値を含められないため、mustIncludeZero は false に設定しなければなりません。

例

```
y1ScaleMinAuto="false"
```

関連項目

138 ページの『y1LogScale』、140 ページの『y1ScaleMin』

y2Axis

Y2 軸のメンバー表示名を指定します。

データ・ソース

すべて

構文

y2Axis="*y2Axis*"

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
y2Axis	空ストリング	メンバー表示名の、コンマで区切られたストリング。メンバーは、同じディメンションからのものである必要があります。

使用法

デフォルトでは、ChartBlox がディメンション配置を決定します。setY2Axis() メソッドはチャートを自動的にリフレッシュします。チャート軸について詳しくは、79 ページの『チャートの軸』を参照してください。

例

```
y2Axis="Market"
```

関連項目

136 ページの『y1Axis』、142 ページの『y2AxisTitle』、143 ページの『y2FormatMask』

y2AxisTitle

Y2 軸のタイトルを明示的に定義します。チャート軸について詳しくは、79 ページの『チャートの軸』を参照してください。

データ・ソース

すべて

構文

y2AxisTitle="*title*"

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
title	ヌル	軸タイトルのテキストを示す任意のストリング。

使用法

リレーショナル・データをチャート化する場合、チャートは軸タイトルを自動的には表示しません。チャートに表示するタイトルはすべて定義する必要があります。

マルチディメンションのデータ・ソースでタイトルを指定することもできますが、必要ではありません。この場合のデフォルト値、ヌルが自動的に軸タイトルを設定

し、空ストリングはタイトルを表示しません。getter メソッドの戻り値「null」は、チャートがマルチディメンション・データ・ソースから自動的に軸タイトルを判別したことを示します。

例

```
y2AxisTitle="This is the Y2 Axis"
```

関連項目

115 ページの『o1AxisTitle』、115 ページの『pieFeelerTextDisplay』、130 ページの『x1AxisTitle』、137 ページの『y1AxisTitle』、141 ページの『y2Axis』、143 ページの『y2FormatMask』

y2FormatMask

チャートの Y2 軸のフォーマット・マスクの値を指定します。

データ・ソース

すべて

構文

```
y2FormatMask="formatMask"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
formatMask	ヌル	251 ページの『数値のフォーマット』および 280 ページの『formatMask』を参照。

使用法

フォーマット・マスクは、チャートの軸値のカスタマイズしたフォーマットを指定できるようにします。このフォーマットは、マウスをチャート・データ項目の上で停止させると表示されるポップアップ・ラベルにも使用されます。例えば、Y2 軸がパーセンテージの指標である場合、##0.00% をフォーマット・マスクに指定できます。フォーマット・マスクはグリッドでのフォーマット・マスクと同様に設定します。これらのプロパティーの値の設定方法については、251 ページの『数値のフォーマット』を参照してください。キーワード K および M (千と百万) がサポートされています。割り算 (/) および掛け算 (*) もサポートされます。

例

```
y2FormatMask="##0.00%"
y2FormatMask="$#,###/1000"
y2FormatMask="#,###K;
```

それに最も近い 1 億単位に丸めたドル値に Y2 軸フォーマットを設定するには、以下のようにします。

```
y2FormatMask="$###,###,###.##"
```

関連項目

137 ページの『y1FormatMask』、131 ページの『x1FormatMask』、251 ページの『数値のフォーマット』

y2LogScale

Y2 軸に対数スケールを使用するかどうかを設定します。

データ・ソース

すべて

構文

```
y2LogScale="width"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
logScale	false	Y2 軸に対数スケールを使用するかどうかを指定します。

使用法

チャート・エンジンはスケールの最大、最小値を自動的に計算しないため、y2LogScale が true に設定されている場合、以下のプロパティーの値も指定する必要があります。

- y2ScaleMaxAuto は false に設定する必要があります。
- y2ScaleMax の値を設定する必要があります。
- y2ScaleMinAuto は false に設定する必要があります。
- y2ScaleMin の値を設定する必要があります。
- mustIncludeZero は false に設定する必要があります。

ログ・スケールを 0 から開始することはできず、値は決して負ではないため、mustIncludeZero は false (デフォルトは true) に設定しなければなりません。

例

```
y2LogScale="true"
```

関連項目

145 ページの『y2ScaleMaxAuto』、144 ページの『y2ScaleMax』、146 ページの『y2ScaleMinAuto』、146 ページの『y2ScaleMin』、114 ページの『mustIncludeZero』、131 ページの『x1LogScale』、138 ページの『y1LogScale』

y2ScaleMax

Y2 軸の最大値を設定します。

データ・ソース

すべて

構文

y2ScaleMax="*scale*"

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
scale	ヌル	Y2 軸の最大値。

使用法

このプロパティーは y2ScaleMaxAuto が true に設定されている場合、無視されます。 y2ScaleMax は常時 y2ScaleMin より大きい値に設定する必要があります。そうでない場合、チャートが正しく振る舞わないことがあります。

例

y2ScaleMax="500000"

関連項目

145 ページの『y2ScaleMaxAuto』、146 ページの『y2ScaleMin』

y2ScaleMaxAuto

Y2 軸の最大値を自動的に設定するかどうかを設定します。これが false の場合、Y2 軸の最大値設定には、y2ScaleMax プロパティーの値が使用されます。

データ・ソース

すべて

構文

y2ScaleMaxAuto="*auto*"

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
auto	true	有効な値は true または false です。これが false に設定されている場合、Y2 軸の最大値は y2ScaleMax プロパティーの値によって決定されます。デフォルトでは、これは true に設定され、y2ScaleMax プロパティーの値は無視されます。

使用法

軸にログ・スケールを使用する場合、すべての対応する [axis]ScaleMaxAuto および [axis]ScaleMinAuto は false に設定する必要があり、[axis]ScaleMax および [axis]ScaleMin に値を指定する必要があります。ログ・スケールには 0 または負の数値を含められないため、mustIncludeZero は false に設定しなければなりません。

例

```
y2ScaleMaxAuto="false"
```

関連項目

144 ページの『y2LogScale』、144 ページの『y2ScaleMax』

y2ScaleMin

Y2 軸の最小値を設定します。

データ・ソース

すべて

構文

```
y2ScaleMin="scale"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
scale	ヌル	Y2 軸の最小値。

使用法

このプロパティーは y2ScaleMin が true に設定されている場合、無視されます。y2ScaleMax は常時 y2ScaleMin より大きい値に設定する必要があります。そうでない場合、チャートが正しく振る舞わないことがあります。y2ScaleMin が 0 より大きい場合には優先されるため、mustIncludeZeo は false に設定する必要があります。

例

```
y2ScaleMin="10000"
```

関連項目

146 ページの『y2ScaleMinAuto』、144 ページの『y2ScaleMax』

y2ScaleMinAuto

Y2 軸の最小値を自動的に設定するかどうかを設定します。これが false の場合、Y2 軸の最小値設定には、y2ScaleMin プロパティーの値が使用されます。

データ・ソース

すべて

構文

```
y2ScaleMinAuto="auto"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
auto	true	有効な値は true または false です。これが false に設定されている場合、Y2 軸の最小値は y2ScaleMin プロパティーの値によって決定されます。デフォルトでは、これは true に設定され、y2ScaleMin プロパティーの値は無視されます。

使用法

軸にログ・スケールを使用する場合、すべての対応する *[axis]ScaleMaxAuto* および *[axis]ScaleMinAuto* は false に設定する必要があり、*[axis]ScaleMax* および *[axis]ScaleMin* に値を指定する必要があります。ログ・スケールには 0 または負の数値を含められないため、mustIncludeZero は false に設定しなければなりません。

例

```
y2ScaleMinAuto="false"
```

関連項目

144 ページの『y2LogScale』、146 ページの『y2ScaleMin』

ダイヤル・チャートの概説

カスタム JSP タグをダイヤル・チャートの定義に使用できます。ダイヤル・チャートは、1 つ以上のダイヤル盤のある円形のチャートです。これらはしばしばエグゼクティブ・ダッシュボード、フラッシュ・レポート、または重要業績評価指標 (KPI) などといったタイプのシナリオに使用されます。各ダイヤル盤には、スケール、針が 1 つずつ、また 1 つ以上のダイヤル・セクターがあります。ダイヤル・セクターは、ダイヤル・チャート上の指定された領域を特定の色で強調表示するために使用されます。例えば、最小ダイヤル値 100、最大ダイヤル値 500 の、在庫を作図したダイヤル盤があるとします。100 から 200 の間の領域が赤いダイヤル・セクターであるとすると、針がその領域にある場合、提供する在庫が不足する危険性があることを示します。

ダイヤル・チャートには複数のダイヤル盤が含まれることがあり、それぞれのダイヤル盤に独自の針と複数のセクターがあります。通常、異なった色をそれぞれの領域に割り当てます。各ダイヤル盤には開始角度、終了角度、針、および半径があります。 100% の半径とは、指定のチャート領域で可能な限り大きい長さを意味します。開始/終了角度および半径を組み合わせることにより、ダイヤル・チャートに複数のダイヤル盤を作成することができます。Blox Sampler には、ダイヤル盤の異なるプロパティーを例示するダイヤル・チャートの例が幾つか含まれています。

デフォルトで、ダイヤル・チャートは、指定のない限り、最初の使用可能なデータ値に基づいて作図されます。ダイヤル・チャート作成用の API は com.alphablox.blox.uimodel.core.chart.dial パッケージにあります。カスタム JSP タグは、共通プロパティーの大多数を指定するために使用でき、153 ページの『ダイヤル・チャートのタグ・リファレンス』で説明されています。

ダイヤル・チャートの作成

ダイヤル・チャートを作成するには、以下の手順で行います。

1. ChartBlox の chartType プロパティーを dial に設定します。
2. 各ダイヤル盤に、ネストされた <blox:dial> タグを追加します。
3. 各ダイヤル盤に、針、セクター、およびスケールを指定します。タグの階層は以下のとおりです。

```
<blox:chart chartType="dial">
   <blox:dial ...>
      <blox:needle ...>
      <blox:sector ...>
      <blox:scale ...>
      </blox:dial>
   </blox:chart>
```

これらのタグによって、ダイヤル・チャート中の各コンポーネントに属性を指定できます。タグの詳細なリストは、153 ページの『ダイヤル・チャートのタグ・リファレンス』を参照してください。ダイヤル・チャートの各コンポーネントについての説明は、148 ページの『ダイヤル・チャートのコンポーネント』を参照してください。

注: ネストされた dial タグを <blox:chart> 内に追加すると、chartType プロパティーの設定がないか、または別のものに設定されていたとしても、チャート・タイプは dial に強制されます。

ヒント: タグを使用してダイヤル・チャートを追加する場合、ChartBlox の legendPosition プロパティーは自動的に none に設定されます。ユーザー・インターフェースの ChartBlox ダイアログを通して凡例の位置を変更しても、凡例は表示されないため、影響はありません。凡例はユーザーに意味をなさないため、legendPosition を他の位置にリセットしないでください。

ダイヤル・チャートのコンポーネント

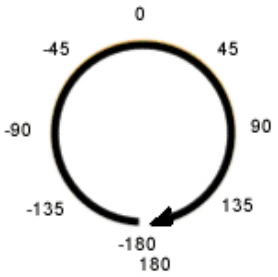
ダイヤル・チャートのキー・コンポーネントには、ダイヤル盤、スケール、セクター、および 針が含まれます。

ダイヤル盤

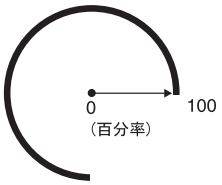
各ダイヤル・チャートには、1 つ以上のダイヤル盤が含まれます。以下のキー・プロパティーをダイヤル・チャートのそれぞれのダイヤル盤に指定するため、<blox:dial> タグが提供されています。

- startAngle: ダイヤル盤領域が始まる位置。
- stopAngle: ダイヤル盤領域が終了する位置。
- radius: 使用可能なスペースのパーセンテージとしてのダイヤル盤の半径。
- color: このダイヤル盤を塗りつぶす色。
- ticPosition: ダイヤル盤上のティック・マークの位置。位置は、inside、outside、または none です。
- borderType: ダイヤル盤の枠のタイプ。solid または none にできます。
- borderColor: ダイヤル盤の枠の色。

以下の図は、startAngle および stopAngle がどのように決定されるかを示しています。



以下の図は、radius がどのように決定されるかを示しています。



スケール

各ダイヤル盤ごとにそのスケールを指定する必要があります。ダイヤル盤のスケールは、最小値、最大値、およびダイヤル盤上のティック・マーク間のステップ・サイズから成っています。このダイヤル盤に作図される針は、このスケールに対して作図されます。ダイヤル盤の最小・最大値は、指定のない限り、最初のデータ値に基づいて自動的に決定されます。ネストされた <blox:scale> タグが供給されており、これにより、以下のキー属性を指定できます。

- maximum: スケール上の最大値。
- minimum: スケール上の最小値。
- stepSize: ティック・マーカーのステップ・サイズ。
- scope: このダイヤル盤のスケール上の値を決定するのに使用されるデータ値。

最小・最大値は比率または実際の値のいずれかを使用して指定できます。データ値に基づいて動的にスケールが設定されるため、比率を使用することをお勧めします。実際の値を指定する場合 (“%” 符号なしで)、針に使用されるデータ値がスケールの最大値を超過し、針の作図に問題が生じることがあります。実際の値は、データ・ドリル・アクションが使用不可で、針の値が変更されない「静的」チャートでのみ指定してください。

セクター

ダイヤル・セクターを使用して、ダイヤル盤を異なる色の異なるセクターに分割することができます。これはしきい値を示すのに役立ちます。ダイヤル・セクターは、以下のもので定義されます。

- 開始値および終了値。
- ダイヤル盤の半径の百分率で表わされる内部半径と外部半径。

ネストされた <blox:sector> タグが供給されており、これにより、以下のキー属性を指定できます。

- startValue: このセクターの開始値。これは、ダイヤル盤のスケールでどのように最小・最大値が指定されているかにより、比率または実際のデータ値のいずれかです。
- stopValue: このセクターの終了値。これは、ダイヤル盤のスケールでどのように最小・最大値が指定されているかにより、比率または実際のデータ値のいずれかです。
- color: このセクターの色。
- innerRadius: このセクターの内部半径。デフォルトは 0 で、円の中心です。
- outerRadius: このセクターの外部半径。デフォルトは 100 で、使用可能な全部の長さです。
- scope: セクター内の値を決定するのに使用される値が属するセル。
- tooltip: マウスをセクター上に移動したときに表示されるテキスト。

上記の出力を生成するコードの断片は 150 ページの『例 1: セクターの指定』にあります。

針

しきい値の数に対して実際のデータ値が現在何であるかを示すために、ダイヤル・チャートで針が使用されます。各ダイヤル盤はそれぞれ 1 本の針を持つことができます。ネストされた <blox:needle> タグが供給されており、これにより、以下のキー・プロパティーを指定できます。

- needleWidth: ピクセルでの針の幅。
- endType: 針先のタイプ。針先には、鋭い矢印、ブロック矢印、円があり、または針先がないこともあります。
- endWidth: 針先の幅。
- color: この針の色。
- tooltip: ユーザーが針上をマウスオーバーする際に表示されるツールチップ。
- scope または value: 針が指すべき値のセル、または実際の値。

ダイヤル・チャートの例

このセクションには、実行して出力を見ることのできる例が含まれています。例には、以下のものがあります。

- 150 ページの『例 1: セクターの指定』
- 151 ページの『例 2: 針と有効範囲の指定』

例 1: セクターの指定

以下の例では、ダイヤル盤の中にセクターを作成する方法を例示します。実例は、Blox Sampler を参照してください。

1. チャート・タイプを初めに “dial” に設定します。ここでは PresentBlox を使用して GridBlox を含め、ダイヤル盤のスケールを決定する方法を示します。
2. このチャートには 2 つのダイヤル盤があります。最初のものは角度が -150 から 150 度で、2 番目のものは完全な円で角度が -180 から 180 度です。

3. このダイヤル盤のスケールは、照会から戻される最初のデータ値の 0 から 150% で、ティック・マーカーが 2500 の増分ごとに付けられます。
4. 4 セクターがそれぞれ異なる色 (赤、黄色、緑、および深緑) でダイヤル盤の内部で作成されます。
5. このチャートの 2 番目のダイヤル盤。こちらは、チャートの外観を良くするために、中央に針のベースとして円を単に追加するためのものです。その needleWidth は 0、needleType は none に設定されています。

上記の出力を生成するコードの全体は次のようになります。

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

      <blox:present id="dialExample1" height="90%" width="90%">
         <blox:chart chartType="dial" y1FormatMask="$#,###"
            titleStyle="font=Arial:bold:10"
            title="Milk Chocolate Truffles Sales for Jan 01">
(1)         <blox:dial startAngle="-150" stopAngle="150" color="#CCCC99">
               <blox:scale minimum="0" maximum="150%" stepSize="2500" />
(2)            <blox:sector startValue="0" stopValue="50%"
(3)               color="red" innerRadius="30" outerRadius="80"
(4)               tooltip="Below expectation" />
               <blox:sector startValue="50%" stopValue="80%"
                  color="yellow" outerRadius="80"
                  tooltip="Marginal performance"/>
               <blox:sector startValue="80%" stopValue="120%"
                  color="green" innerRadius="80"
                  tooltip="Satisfactory performance"/>
               <blox:sector startValue="120%" stopValue="150%"
                   color="#009966" innerRadius="80" />
            </blox:dial>

            <%--creating a blue circle in the center --%>
            <blox:dial startAngle="-180" stopAngle="180" color="black"
               radius="10" ticPosition="none" borderType="none">
(5)            <blox:needle needleWidth="0" endType="none" />
            </blox:dial>
         </blox:chart>
      <blox:data dataSourceName="qcc-essbase" useAliases="true"
            query="<ROW (¥"All Products¥") <CHILD ¥"Truffles¥"
               <COLUMN (¥"All Time Periods¥") <CHILD ¥"Qtr 1 01¥" !" />
      </blox:present>
      </body>
      </html>
```

例 2: 針と有効範囲の指定

以下の例では、4 つのダイヤル盤および 4 つの異なる針タイプのあるダイヤル・チャートを生成します。

1. チャート・タイプを初めに “dial” に設定します。
2. チャートの最初のダイヤル盤は、単に色の枠を作成するためのものです。開始および終了角度は -135 から 135 に設定され、ティック・マーカーや針はありません。

3. 2 番目のダイヤル盤は意味のあるデータを表示する実際のダイヤル盤です。最初のダイヤル盤が枠になるよう、その半径は 90 に設定されています。
4. 2 番目のダイヤル盤のスケールは {scenario:forecast} の値に基づいており、最小値が 0 で最大値が予測の 120% ($16,828,805 は予想された $14,024,008 の 120%) です。
5. 2 番目のダイヤル盤の針は {scenario:actual} の値に基づいています。
6. 予測の 100% で黄色のセクターが終わり、緑のセクターが始まります。これによって、ユーザーは実際の値が予測された目標を果たしたかどうかを見ることができます。
7. 3 番目のダイヤル盤は、チャートの外観を良くするために、単に中央に針のベースとして円を追加するためのものです。その needleWidth は 0、needleType は none に設定されています。

上記の出力を生成するコードの全体は次のようになります。

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

      <blox:present id="dialExample1" height="90%" width="90%">
         <blox:chart chartType="dial" y1FormatMask="$#,###">
            <blox:dial startAngle="-135" stopAngle="135" color="#CCCC99"
               ticPosition="none" showLabels="false">
(1)            <blox:needle needleWidth="0" endType="none" />
(2)         </blox:dial>

            <blox:dial startAngle="-135" stopAngle="135" radius="90">
               <blox:scale minimum="0" maximum="120%"
(3)               scope="{scenario:forecast}" />
(4)            <blox:needle color="blue" needleWidth="3"
                  endType="sharpArrow" scope="{scenario:actual}" />
(5)            <blox:sector startValue="0" stopValue="75%"
                  color="red" />
               <blox:sector startValue="75%" stopValue="100%"
                  color="yellow"/>
(6)            <blox:sector startValue="100%" stopValue="120%"
                  color="green" />
            </blox:dial>

            <%--creating a blue circle in the center --%>
            <blox:dial startAngle="-180" stopAngle="180" color="black"
               radius="10" ticPosition="none" borderType="none">
(7)            <blox:needle needleWidth="0" endType="none" />
            </blox:dial>
         </blox:chart>

         <blox:data dataSourceName="qcc-essbase" useAliases="true"
            query="¥"All Time Periods¥" <COLUMN (¥"Scenario¥") <SYM <ICHILD
            ¥"Scenario¥" <ROW (¥"All Products¥") ¥"All Products¥" !" />

      </blox:present>
      </body>
      </html>
```

ダイヤル・チャートのタグ・リファレンス

このセクションでは、ダイヤル・チャートの作成をサポートするカスタム JSP タグのタグ属性について説明します。情報は以下のように編成されています。

<blox:dial> タグ属性

以下の表に、<blox:dial> タグの属性をリストします。ダイヤル盤とは何か、またその一般的なプロパティーについては、148 ページの『ダイヤル盤』を参照してください。

属性	デフォルト	説明
borderColor	black	ダイヤル盤の枠の色。
borderType	solid	ダイヤル盤の端の周りの枠の枠タイプ。使用可能な値は、none および solid です。たとえば、以下のようにします。 `borderType="none"`
color	チャート作成エンジンのデフォルト	ダイヤル・チャートを塗りつぶす色。Java カラーまたは 16 進値 (例えば、#CCCCFF) を指定します。
formatMask	ヌル	ダイヤル盤のラベルに使用するカスタマイズされたフォーマット。フォーマット・マスクは 131 ページの『x1FormatMask』、137 ページの『y1FormatMask』、および 143 ページの『y2FormatMask』 と同様に設定します。キーワード K および M (千と百万) がサポートされています。割り算 (/) および掛け算 (*) もサポートされます。たとえば、以下のようにします。 `formatMask="$###K"`
radius	100	使用可能なスペースのパーセンテージとしてのダイヤル盤の半径。有効値は 0 から 100 です。
showLabels	true	ティックのラベルを表示します。このダイヤル盤にスケールがある場合、最小値から最大値まで、ラベルがティックに適用されます。欠落しているラベルまたは余分なラベルはブランクになるか、または無視されます。
startAngle	-90	ダイヤル盤領域が始まる位置。この角度は、上向きの縦線で、時計回りに回ります。値の範囲は -180 から 180 です。例えば、開始角度 -90 終了角度 90 では水平な半円のダイヤル盤が作成されます。
stopAngle	90	ダイヤル盤領域が終了する位置。この角度は、上向きの縦線で、時計回りに回ります。値の範囲は -180 から 180 です。例えば、開始角度 -90 終了角度 90 では水平な半円のダイヤル盤が作成されます。

属性	デフォルト	説明
ticPosition	outside	ダイヤル盤軸のティックを配置する位置。可能な値には、以下のものがあります。 • inside: ティックをダイヤル盤の円周上に内側に向けて配置する • outside: ティックをダイヤル盤の円周上に外側に向けて配置する • none: ティックを付けない

<blox:needle> タグ属性

以下の表に、<blox:needle> タグの属性をリストします。ダイヤル盤とは何か、またその一般的なプロパティーについては、150 ページの『針』を参照してください。例については、151 ページの『例 2: 針と有効範囲の指定』を参照してください。

属性	デフォルト	説明
color	black	針の色 (針と針先の両方)。
endType	sharpArrow	針先に何を描画するかを指定します。可能な値は以下のとおりです。 • sharpArrow: 鋭い角が後方にある三角形 • blockArrow: 針先が三角形 • round: 針先が円 • none: 単なる線の針 針の幅と同じ直径の円は見分けられないため、丸い針先を使用する場合には、endWidth は needleWidth より大きい設定にする必要があります。
endWidth	5	ピクセル単位の針先の幅。endType が none に設定されている場合、endWidth は無視されます。
needleWidth	2	ピクセル単位の針の幅。

属性	デフォルト	説明
scope	最初のデータ値	針が指す値が属するセル。 scope は、中括弧に囲まれた、一連のディメンションとメンバーのセットとして指定する必要があります。 表示名または固有の名前のいずれかを使用します。scope は、行と列の軸にのみ適用されます。scope 中にディメンションがない場合でも、scope はマッチングします。 IBM DB2 OLAP Server および Hyperion Essbase データ・ソースには、以下のように scope を指定します。 `scope="{d0:m0} {d1:m1} ..."` d0 はディメンションを表し、m0 はそのディメンション内のメンバーを表します。例えば、IBM DB2 OLAP Server および Hyperion Essbase データ・ソースでは、以下のようにします。 `scope="{scenario:budget}"` MS OLAP、Alphablox Cube、および SAP BW データ・ソースの場合、以下のように固有の名前を使用して有効範囲を指定します。 `scope="{[My Cube].[Measures]: [Measures].[Profit]}"`
tooltip	関連したメンバー名および値	エンド・ユーザーが針の先の上を移動するときに表示されるツールチップ。
value	なし	ダイヤル盤で針が指す値。

<blox:scale> タグ属性

以下の表に、<blox:scale> タグの属性をリストします。スケールは何か、またその指定法については、149 ページの『スケール』を参照してください。

属性	デフォルト	説明
maximum	200%	ダイヤル盤上の最大値。実際の最大値または比率 (% 符号を含める) を指定できます。たとえば、データ値が 100,000 で、maximum が 150% に設定されている場合、スケールの最大値は 150,000 です。 データ値がダイヤル盤の最大値を超過する場合、ポインターはスケールの最後を指します。
minimum	0	ダイヤル盤上の最小値。実際の最小値または比率 (% 符号を含める) を指定できます。たとえば、データ値が 100,000 で、minimum が 50% に設定されている場合、スケールの最小値は 50,000 です。 データ値がダイヤル盤の最小値よりも低い場合、ポインターはスケールの始めを指します。

属性	デフォルト	説明
scope	最初のデータ値	スケール上の値を決定するのに使用される値が属するセル。scope は、中括弧に囲まれた、一連のディメンションとメンバーのセットとして指定する必要があります。 表示名または固有の名前のいずれかを使用します。 scope は、行と列の軸にのみ適用されます。scope 中にディメンションがない場合でも、scope はマッチングします。 IBM DB2 OLAP Server および Hyperion Essbase データ・ソースには、以下のように scope を指定します。 `scope="{d0:m0} {d1:m1}...]"` d0 はディメンションを表し、m0 はそのディメンション内のメンバーを表します。例えば、IBM DB2 OLAP Server および Hyperion Essbase データ・ソースでは、以下のようにします。 `scope="{scenario:budget}"` MS OLAP、Alphablox Cube、および SAP BW データ・ソースの場合、以下のように有効範囲を指定します。 `scope="{[My Cube].[Measures]: [Measures].[Profit]}"`
stepSize	スケールを 5 つの領域に分け、自動的に決定されます。	ダイヤル盤上のティック・マーク間のステップ・サイズ。

<blox:sector> タグ属性

以下の表に、<blox:sector> タグの属性をリストします。セクターは何か、またセクターの指定法については、149 ページの『セクター』を参照してください。例については、150 ページの『例 1: セクターの指定』を参照してください。

属性	デフォルト	説明
color		ダイヤル・セクターの色。Java カラーまたは 16 進値を指定します。
innerRadius	0	ダイヤル盤の半径に対する比率としてのこのダイヤル・セクターの内部半径。有効値は 0 から 100 です。
outerRadius	100	ダイヤル盤の半径に対する比率としてのこのダイヤル・セクターの外部半径。有効値は 0 から 100 です。

属性	デフォルト	説明
scope	最初のデータ値	セクター中の値を決定するのに使用される値が属するセル。scope は、中括弧に囲まれた、一連のディメンションとメンバーのセットとして指定する必要があります。 表示名または固有の名前のいずれかを使用します。 scope は、行と列の軸にのみ適用されます。scope 中にディメンションがない場合でも、scope はマッチングします。 IBM DB2 OLAP Server および Hyperion Essbase データ・ソースには、以下のように scope を指定します。 scope="{d0:m0} {d1:m1}...]" d0 はディメンションを表し、m0 はそのディメンション内のメンバーを表します。 MS OLAP、Alphablox Cube、および SAP BW データ・ソースの場合、以下のように有効範囲を指定します。 scope="{[My Cube].[Measures]: [Measures].[Profit]}"
startValue		このセクターの開始値。これはダイヤル盤のスケールに基づいて設定する必要があります。例えば、ダイヤル盤のスケールの最小値が 100 で、最大値が 500 である場合、赤いセクターの開始値を 300、終了値を 500 にするとダイヤル盤の 300 と 500 の間の領域が赤になります。 この値はダイヤル盤のスケールの最小と最大の間である必要があります。
stopValue		このセクターの終了値。これはダイヤル盤のスケールに基づいて設定する必要があります。例えば、ダイヤル盤のスケールの最小値が 100 で、最大値が 500 である場合、赤いセクターの開始値を 300、終了値を 500 にするとダイヤル盤の 300 と 500 の間の領域が赤になります。 この値はダイヤル盤のスケールの最小と最大の間である必要があります。
tooltip		マウスをセクター上に移動したときに表示されるテキスト。

第 8 章 CommentsBlox タグ・リファレンス

この章には、CommentsBlox タグ属性の参照資料が含まれています。Blox についての一般的な参照情報は、21 ページの『第 3 章 一般 Blox リファレンス情報』を参照してください。このリファレンスの使用方法については、1 ページの『第 1 章 このリファレンスの使用法』を参照してください。

CommentsBlox の概説

CommentsBlox を使用することにより、アプリケーションにセル・コメント (セル注釈とも言う) 機能を提供することができます。さらに、他の Blox には結びつかない一般コメント用に、CommentsBlox を使用することができます。たとえば、ユーザーがサイト、アプリケーション、レポート、または Web ページにコメントを追加することを許可できます。

コメントは、JDBC のアクセス可能なリレーショナル・データベースに格納されます。対応するデータベースには、IBM DB2® UDB、Sybase、Microsoft SQL Server、および Oracle が含まれます。このデータ・ソースを、DB2 Alphablox に定義する必要があります。 DB2 Alphablox は、コメントを格納するのに使用するリレーショナル・データ・ソースを指定できる「DB2 Alphablox **管理**」タブにある「**サーバー・リンク**」の下の「**コメント管理**」ページを提供します。そのページから、「コレクション」(データ表) を作成し、コメントを格納できます。セル・レベル・コメントの場合、GridBlox で使用されるマルチディメンション・データ・ソース、(Microsoft Analysis Services で) 使用するキューブ、および含まれるディメンションを指定する必要があります。一般コメントの場合、名前を指定することだけが必要です。

注: コメントを保管するのに z/OS システム上で DB2 UDB を使用する場合は、DB2 Alphablox にこのデータ・ソースを定義するときに、「z/OS 用 IBM DB2 JDBC タイプ 4 ドライバー (IBM DB2 JDBC Type 4 Driver for z/OS)」データ・アダプターを選択する必要があります。「IBM DB2 JDBC タイプ 4 ドライバー (IBM DB2 JDBC Type 4 Driver)」ドライバーを選択すると、コメント・コレクションを作成できません。

ユーザー・インターフェース

GridBlox のユーザー・インターフェースにコメント機能が設定され、使用可能であるとき、コメント・メニュー項目は右マウス・ボタン・クリック・メニューから使用可能です。コメント標識が、コメントのあるセルの隅に表示されます。

CommentsBlox は、同じフィールドの設定および同じアドレス体系を共有するコメントのコンテナーです。セル・レベル・コメントの場合、コメントには、セルを識別

するのに必要なディメンションおよびキューブ情報のサブセットを取り込む、アドレッシング体系があります。データ・セルと結びつかない一般コメントの場合、アドレッシング体系は単にコメント・コレクションの名前を含むストリングです。これらのコメントは、「名前付きコメント」と呼ばれます。各 CommentsBlox には、複数の名前付きコメントのセットがあります。

ユーザーがセルのコメントの表示を選択するとき、ポップアップ・ウィンドウは、セルとセル上で作成されたすべてのコメントのアドレス、およびコメントを追加した作成者と時間を表示します。コメントの作成者のみが、コメントを削除できます。

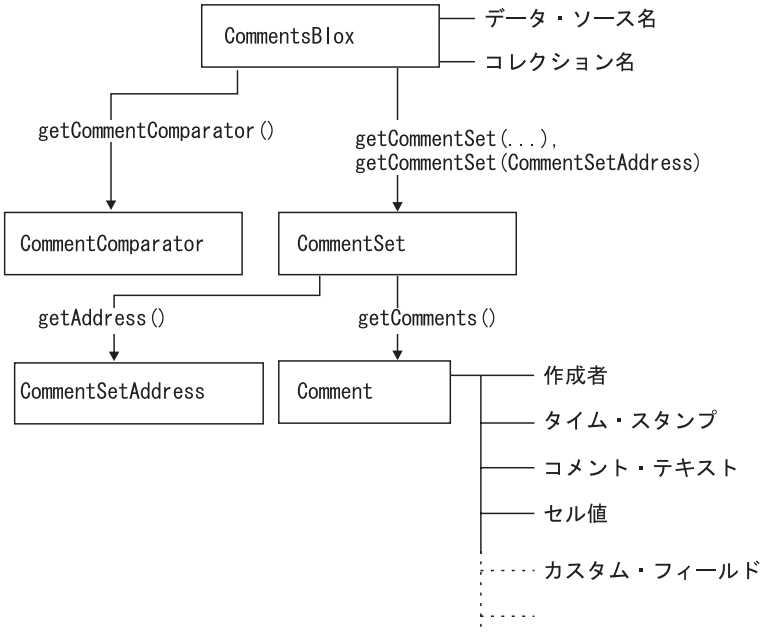
デフォルトでは、コメントは日付でソートされます。ユーザーは列ヘッダーをクリックし、その列の値を基にしてコメントをソートできます。ソート順序は、トグル・モードで作動します。アプリケーション開発者はソートするフィールド、およびタグを使用したソート順序を指定できます。

CommentsBlox のオブジェクト階層および API

CommentsBlox を使って、特定のセルまたは名前付きコメントのセットと関連したコメントのサブセットにアクセスし、続いて、作成者、コメント・テキスト、コメントが追加された時間など、個々のコメントについてのより多くの情報を設定、または得ることができます。

各コメントには、作成者、タイム・スタンプ、セル値、およびコメント・テキストの 4 つの必要フィールドがあります。「コメント管理」ページからコメント・コレクションを作成するとき、独自のカスタム・フィールドを追加することができます。

以下の図は、CommentsBlox のオブジェクト階層を示しています。



CommentSet オブジェクトは、コメントが追加、更新され、コレクションから除去されるのに使うインターフェースです。各 CommentSet にもアドレスがあります。先

に説明されているとおり、セル・レベル・コメントの場合、CommentSet のアドレスは、セルを識別するのに必要なディメンションおよびキューブ情報で構成されます。データ・セルと結びつかない一般コメントの場合、アドレッシング体系は単にコメント・コレクションの名前を含むストリングです。 CommentsBlox を使って、名前付きアドレス内のセル上に保管されているコメントを含む、CommentSet にアクセスできます。

Comment オブジェクトには、コメントごとの情報を格納する以下の静的フィールドがあります。

- `FIELD_AUTHOR`
- `FIELD_CELLVALUE`
- `FIELD_COMMENTTEXT`
- `FIELD_TIMESTAMP`

コメント・コレクションを作成するときに定義されているように、その他のカスタム・フィールドを含む場合もあります。

Comment、CommentSet、CommentSetAddress、および CommentComparator オブジェクトの Javadoc は、 `com.alphablox.blox.comments` パッケージの下にあります。

CommentsBlox のイベント

CommentsBlox は CommentsListener を使用して、割り当てられた CommentEvent リスナー (CommentAddedEvent、CommentDeletedEvent、および CommentUpdatedEvent) に、コメントがコメント・コレクション内で変更されたことを通知します。これによって、イベントがサーバーによって処理された後に、コメントの変更を記録するなどのカスタム・アクションを実行できます。

CommentsBlox の `addCommentsListener()` メソッドを使用して、コメント・リスナーを追加することができます。 CommentsListener には、聴取するコメントのイベントを指定できる `CommentChanged()` メソッドがあります。
CommentAddedEvent、CommentDeletedEvent、CommentUpdatedEvent の各々により、関連するイベントによって影響を受けた Comment または CommentSet にアクセスできます。例については、167 ページの『例 4: CommentAddedEvent リスナーの追加』を参照してください。

データベースの操作および許可

CommentsBlox の使用には、コメント・コレクションの作成、コレクションの編集、およびコメントの追加、表示、削除をサポートする、さまざまなデータベースの操作が含まれます。以下の表は、実行されるタスクに依存した場面の背後のデータ操作を示しています。これは、アプリケーションが作動するのに必要な、適切な許可のセットアップを計画するのに助けとなります。

実行されるタスク	含まれるデータ操作
コメント・コレクションの作成 :	表および索引の作成
既存のコメント・コレクションの編集 :	古い表のドロップ、および新しい表の作成

コメント・コレクションの削除 :
関連する表の削除

コメントの追加 : 更新および挿入

コメントの削除 : 削除

コメントの表示 : 選択

CommentsBlox の JSP カスタム・タグ構文

DB2 Alphablox Tag Libraries は、各 blox を作成する JSP ページで使用するカスタム・タグを提供します。このセクションでは、CommentsBlox を作成するためのカスタム・タグの使用方法を説明します。すべての属性を含むタグのコピー・アンド・ペースト・バージョンについては、522 ページの『CommentsBlox JSP カスタム・タグ』を参照してください。セル・レベル・コメントを提供するときには、CommentsBlox タグが DataBlox カスタム・タグ内にネストされたタグであることに注意してください。名前付きコメントの場合、コメントは DataBlox と結びついていないので、CommentsBlox タグは独立型のタグとして使用されます。

構文

セル・レベル・コメント (DataBlox と関連) の場合、次のようになります。

```
<blox:data id = "myData1"
    dataSourceName = "foodmart"
    query = " <%=myQueryString %>"
    ... >
          <blox:comments
    [attribute="value"] >
        <blox:sortComments
          field="" order="" />
    </blox:comments>
       </blox:data>
```

または、DataBlox で参照される独立型の CommentsBlox もあります。

```
<blox:comments id="myComments"
    [attribute="value"] >
        <blox:sortComments
          field="" order="" />
    </blox:comments>

<blox:data id = "myData1"
    dataSourceName = "foodmart"
    query = " <%=myQueryString %>"
    ... >
          <blox:comments
        bloxRef="myComments">
    </blox:comments>
       </blox:data>
```

注: bloxRef 属性を使用している DataBlox タグ内に、CommentsBlox タグを追加することはできません。dataSourceName および query 属性が定義される、実際の DataBlox タグ内にネストされる必要があります。

名前付きコメント (DataBlox と関連のない) の場合、次のようになります。

```
      <blox:comments
  [attribute="value"] >
      <blox:sortComments
        field="" order="" />
  </blox:comments>
```

ここで、それぞれ以下のとおりです。

attribute 属性表にリストされている属性の 1 つです。

value 属性の有効な値です。

属性は以下のいずれかになります。

属性
id
bloxName
bloxRef
commentsOnBaseMember
collectionName
dataSourceName
password
userName

ネストされた <blox:sortComments> タグはオプションであり、次の 2 つの属性があります。

属性
field
sort

使用法

各カスタム・タグには 1 つ以上の属性を含めることができ、それぞれを 1 つ以上のスペースまたは改行文字で区切ります。余分のスペースまたは改行文字は無視されます。読みやすくするため、同じインデントでそれぞれ別々の行に属性を並べることができます。

次にあるような属性リストの終わりでタグを閉じ、省略表現を使用して、終了 </blox:comments> タグを置き換えることができます。

```
    dataSourceName = "comments_mssql" />
```

例

セル・レベル・コメント (DataBlox と関連) の場合、次のようになります。

```
<blox:data id = "myData1"
    dataSourceName = "foodmart"
    query = " <%=myQueryString %>" >
<blox:comments id = "myComments1"
    collectionName = "sales_comments"
    dataSourceName = "comments_mssql" />
       </blox:data>
```

名前付きコメント (DataBlox と関連のない) の場合、次のようになります。

```
<blox:comments id = "myComments1"
    collectionName = "sales_comments"
    dataSourceName = "comments_mssql" />
```

「コメントの表示」ウィンドウがポップアップするとき、ソートする特定のフィールドのあるセル・レベル・コメントは、次のようになります。

```
<!--import the following package in order to use the field constants-->
<%@ page import="com.alphablox.blox.comments.*" %>

<blox:data id = "myData2"
    dataSourceName = "QCC-MSAS"
    query = " <%=myQueryString %>" >
    <blox:comments id = "myComments2"
        collectionName = "planning_comments"
        dataSourceName = "comments_mssql" >
        <blox:sortComments
            field="<%= Comment.FIELD_COMMENTTEXT %>"
            order="<%= CommentComparator.DESCENDING %>" />
    </blox:comments>
       </blox:data>
```

CommentsBlox の例

このセクションには、セル・レベル・コメント (DataBlox および GridBlox と関連) を使用可能にするための CommentsBlox の使用方法、MDBResultSet を使って個々のセルのコメントにアクセスする方法、およびコメントが追加 (または削除、更新) されるときにイベント・リスナーを追加する方法を示す例があります。

- 例 1: セルのコメントの使用可能化
- 例 2: ソートするフィールドおよびソート順序の指定
- 例 3: MDBResultSet を使用したセル・コメントへのアクセス
- 例 4: CommentAddedEvent リスナーの追加

ページ・レベルのコメント (DataBlox と関連しない)、またはセル・コメントの追加および表示のためのカスタム・ページの使用については、「*開発者用ガイド*」の『情報の強調とコメント』の章を参照してください。 CommentsBlox の完全な実例については、DB2 Alphablox のホーム・ページの「Assembly」タブにある「Blox Sampler」の「Commenting on Data」セクションを参照してください。

例 1: セルのコメントの使用可能化

この例では、commentsEnabled 属性を GridBlox で true に設定し、DataBlox 内にネストされた CommentsBlox タグを追加することによって、セルのコメントを使用可能にする方法を示します。コメントを格納するのに使用されるリレーショナル・データ・ソースが、DB2 Alphablox で定義されている必要があり、コレクション名が DB2 Alphablox の「管理 (Administrative)」タブにある「コメント管理」ページによって作成されている必要があることに注意してください。

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%
    String query = "your_data_query_here";
%>

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

```
</head>
<body>
    <blox:present id="presentBlox">
        //Enable cell commenting UI in GridBlox
        <blox:grid
            commentsEnabled="true" />

       <blox:data
            dataSourceName="foodmart"
            query="<%=query%>">
            //The datasource and collection names are defined and
            //and created via the DB2 Alphablox Admin Pages
            <blox:comments
                collectionName="sales_comments"
                dataSourceName="comments_mssql" >
            </blox:comments>
       </blox:data>
 </blox:present>
</body>
</html>
```

例 2: ソートするフィールドおよびソート順序の指定

この例は、ユーザーがセル上でコメントを表示することを選択するときに、コメントをソートするデフォルトのフィールドを指定すること以外は、前の例と同じです。ソート順序は、昇順に設定されています。 com.alphablox.blox.comments パッケージは、フィールド名およびソート順序に定数を使用する目的で、インポートされています。

```
<%@ page import="com.alphablox.blox.comments.*" %>
<%@ taglib uri="bloxtld" prefix="blox"%>
<%
    String query = "your_data_query_here";
%>
<html>
<head>
    <blox:header />
</head>
<body>
    <blox:present id="presentBlox" mayscriptEnabled="true" >
        //Enable cell commenting UI in GridBlox
        <blox:grid
            commentsEnabled="true" />
       <blox:data
            dataSourceName="foodmart"
            query="<%=query%>">
            //The datasource and collection names are defined and
            //and created via the DB2 Alphablox Admin Pages
            <blox:comments
                collectionName="sales_comments"
                dataSourceName="comments_mssql" >
                <blox:sortComments
                   field="<%= Comment.FIELD_AUTHOR %>"
                   order="<%= CommentComparator.ASCENDING %>" />
            </blox:comments>
       </blox:data>
 </blox:present>
</body>
</html>
```

例 3: MDBResultSet を使用したセル・コメントへのアクセス

この例は、MDBResultSet オブジェクトを使って、セルと関連した個々のコメントにアクセスする方法を示します。以下の GridBlox の Truffles for FY2001 と関連したコメントにアクセスするには、次のようになります。

製品 (カテゴリー)	FY2000	FY2001	FY2002
Chocolate Blocks	178,148	3,282,371	3,052,920
Chocolate Nuts		6,686,802	9,390,529
Specialties	295,497	7,769,125	7,909,836
Truffles		749,789	789,908
All Products	473,645	18,488,088	21,143,193

基礎となる DataBlox の MDBResultSet にアクセスして、その後セル (1,3) にアクセスします。

```
   MDBResultSet resultSet = (MDBResultSet)
myCommentGrid.getDataBlox().getResultSet();
   Cells cells = resultSet.getCells();
   Cell cell = cells.getCell(1,3);
```

ここで、セルのすべてのコメントにアクセスできます。

```
   CommentSet truffleCommentSet = cell.getCommentSet();
```

完全なコードは次のようになります。

```
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ page import="com.alphablox.blox.data.mdb.*" %>
<%@ page import="com.alphablox.blox.*" %>
<%@ page import="com.alphablox.blox.comments.*" %>
<%@ taglib uri="bloxtld" prefix="blox" %>
<html>
<head>
   <blox:header/>
</head>
<body>
<blox:grid id="myCommentGrid"
   width="60%"
   height="50%"
   commentsEnabled="true"
   defaultCellFormat="#,###"
   bandingEnabled="true">
   <blox:data dataSourceName="QCC-MSAS"
      query="SELECT {[Time.Fiscal].[All Time Periods].Children} ON COLUMNS,
         {[Products.Category].[All Products].Children,
         [Products.Category].[All Products]} ON ROWS FROM QCC
          WHERE ([Measures].[Sales])">
      <blox:comments
         collectionName="CommentsCollectionMSAS"
         dataSourceName="CommentsCollectionMSAS" />
   </blox:data>
</blox:grid>

<%
   //Access the comments associated with a cell from the result set
   MDBResultSet resultSet = (MDBResultSet)
myCommentGrid.getDataBlox().getResultSet();
   Cells cells = resultSet.getCells();
   Cell cell = cells.getCell(1,3);
   CommentSet truffleCommentSet = cell.getCommentSet();
```

```
   //Now get the address of the CommentSet for this cell
   CommentSetAddress truffleAddress = truffleCommentSet.getAddress();
   out.write("<BR>Address of CommentSet for Truffles: "+truffleAddress +
"<br>");

   //Now get the comment text for each comment in the CommentSet
   Comment[] comments = truffleCommentSet.getComments();
   out.write("<BR>The number of comments is: "+comments.length);
   for(int i = 0; i < comments.length; i++) {
      out.write("<BR>Comment Text: "+comments[i].getCommentText()+" for
comment: "+ i + "<br>");
   }

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

出力は、次のようになります。

Address of CommentSet for Truffles: CellCommentAddress: [Locations]:[Locations].[All Locations];[Measures]:[Measures].[Sales];[Products].[Category]:[Products].[Category].[All Products].[Truffles];[Products].[Code]:[Products].[Code].[All Products];[Products].[Seasonal]:[Products].[Seasonal].[All Products];[Scenario]:[All Scenario];[Seasonal]:[Seasonal].[All Seasonal];[Time].[Calendar]:[Time].[Calendar].[All Time Periods];[Time].[Fiscal]:[Time].[Fiscal].[All Time Periods].[FY2001];

The number of comments is: 2

Comment Text for comment 0: The sales in the East region were 32% higher than projected, making up the lost of sales in the West due to machine breakdown.

Comment Text for comment 1: There was a machine breakdown in the west region for two weeks that impacted the sales of seasonal items.

例 4: CommentAddedEvent リスナーの追加

次の例は、CommentAddedEvent をキャプチャーし、その後 Alphablox コンソールに作成者、コメント・テキスト、およびタイム・スタンプを印刷する方法を示します。コメントのイベント・リスナーを追加するには、次のようにします。

1. CommentsBlox の `addCommentsListener()` メソッドを使用して、コメント・リスナーを追加します。
2. コメント・リスナーは、CommentsListener インターフェースをインプリメントする必要があります。
3. コメントが変更されるときに、実行すべきアクションを追加します。`CommentChanged()` メソッドで、 listen する CommentAddedEvent、CommentDeletedEvent、または CommentUpdatedEvent のいずれかを指定します。
4. 次に、`getComment()` または `getCommentSet()` を使用して、関連した Comment または CommentSet にアクセスすることができます。

```
<%@ taglib uri = "bloxtld" prefix = "blox"%>
<%@ page import="com.alphablox.blox.comments.*,
                 com.alphablox.blox.uimodel.core.MessageBox,
                 com.alphablox.blox.uimodel.BloxModel,
                 com.alphablox.blox.*" %>
<blox:comments id="myComments"
    collectionName="CommentsCollection"
```

```
    dataSourceName="CommentsCollection" />

<%! public abstract class CListener implements CommentsListener
   {
   BloxModel model;
   public void commentsChanged(com.alphablox.blox.comments.CommentAddedEvent cadded)
     throws Exception
     {
        Comment comment = cadded.getComment();
        StringBuffer msg = new StringBuffer("---------" + "¥n");
        msg.append("Author: " + comment.getAuthor() + "¥n");
        msg.append("Comment text: " + comment.getCommentText() + "¥n");
        msg.append("Time: " + comment.getTimestamp( ));
        MessageBox msgBox = new MessageBox(msg.toString(),"Comments Added",
           MessageBox.MESSAGE_OK, null);
        model.getDispatcher().showDialog(msgBox);
     }
  } %>
  <blox:present id="CommentsPresentBlox"
    ...
     >
    <blox:grid
       commentsEnabled="true" />
       <blox:data
          dataSourceName="QCC-Essbase"
          query="!">
          <blox:comments
             bloxRef="myComments"/>
       </blox:data>
       <% myComments.addCommentsListener( CListener() );  %>
 </blox:present>
```

CommentsBlox タグ属性

以下は、固有の CommentsBlox タグ属性です。

- commentsOnBaseMember
- collectionName
- dataSourceName
- userName

id

これは共通の Blox タグ属性です。詳しい説明は、43 ページの『id』を参照してください。

bloxName

これは共通の Blox タグ属性です。詳しい説明は、39 ページの『bloxName』を参照してください。

bloxRef

これは共通の Blox タグ属性です。詳しい説明は、41 ページの『bloxRef』を参照してください。

commentsOnBaseMember

共用メンバーで作成されたコメントを、ベース・メンバーまたは共用メンバーのインスタンスと関連付けるかどうかを指定します。そのようにすると、ベース・メンバーとそのメンバーの共用バージョンすべての両方で、または共用メンバーの特定のインスタンスでコメントを表示できるようになります。

データ・ソース

DB2 OLAP Server および Hyperion Essbase

構文

```
commentsOnBaseMember="flag"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
flag	false	このプロパティーを false (デフォルト) に設定すると、共用メンバーに配置されているコメントは共用メンバーの特定インスタンスにのみ表示されます。true の場合、共用メンバーで作成されたコメントはベース・メンバーと関連付けられ、ベース・メンバー、およびすべての共用メンバーのインスタンスの両方に表示されます。

使用法

commentsOnBaseMember の場合、このフラグは、コメントを表示するときではなく、コメントを作成するときに使用します。 関連する Java メソッドを使用して実行時にこのフラグの値を変更し、共用メンバー (setCommentsOnBaseMember(false)) の指定のインスタンスまたはベース・メンバー (setCommentsOnBaseMember(true)) に明確にコメントを関連付けることができます。このプロパティーは、DB2 OLAP Server および Hyperion Essbase データ・ソースにのみ影響を及ぼします。

collectionName

コメント・コレクションの名前。各 CommentsBlox は、リレーショナル・データ・ソースおよびそのデータ・ソースにあるコレクション名と関連付けられている必要があります。

データ・ソース

リレーショナル (コメントを格納する場合)

構文

```
collectionName="name"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
name	null	コメント・コレクションの名前。

dataSourceName

格納されたコメントに使用するデータ・ソースの名前。

データ・ソース

リレーショナル (コメントを格納する場合)

構文

```
dataSourceName="name"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
name	null	データ・ソースの名前。

password

コメントの格納に使用されるデータ・ソースに接続するために使用するパスワード。

データ・ソース

リレーショナル (コメントを格納する場合)

構文

```
password="password"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
password	DB2 Alphablox 管理ページを介してデータ・ソース定義で指定されたパスワード。	データ・ソースに接続するために使用されるパスワード。

userName

コメントの格納に使用されるデータ・ソースに接続するために使用するユーザー名。

データ・ソース

リレーショナル (コメントを格納する場合)

構文

```
userName="username"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
username	DB2 Alphablox 管理ページを介してデータ・ソース定義で指定されたユーザー名。	データ・ソースに接続するために使用されるユーザー名。

使用法

コレクションが最初に作成されるとき、このユーザー名は表および索引の十分な作成権を持っていなければなりません。コメントを検索し、書き込む場合、このユーザーはデータベースへの**接続**特権を持っている必要があり、ユーザーが単にコメントを読み取る場合には、**選択**特権を持っていなければなりません。ユーザーがコメントを追加または変更する場合、**挿入**および**更新**特権が必要です。

第 9 章 ContainerBlox タグ・リファレンス

この章では、ContainerBlox のプロパティー、メソッド、およびオブジェクトに関する参照資料を記載します。このリファレンスの使用方法については、1 ページの『第 1 章 このリファレンスの使用法』を参照してください。

ContainerBlox の概説

ContainerBlox は、事前定義された振る舞いのない空の Blox です。 DataBlox およびすべてのプレゼンテーション Blox (PresentBlox、 GridBlox、 ChartBlox、 ToolbarBlox、 PageBlox、および DataLayoutBlox) は、ViewBlox の拡張であり、ViewBlox は ContainerBlox の拡張です。 ContainerBlox により、ページ上に領域を作成して、メニューやボタンなどの DB2 Alphablox が提供する任意のユーザー・インターフェース・コンポーネントを使用することができます。

ページ・リフレッシュなし、完全なサーバー・サイド・コントロールとサーバー・サイド・ロジック、すべてのコア・コンポーネント (たとえば、ツリー・コントロール、メニュー、およびツールバー) の使用、同じユーザー・インターフェースのプログラミング・モデル、簡単な分散、ローカライズ可能なリソース・ファイル、およびダイアログなどの提供されるサービスを利用するために、ContainerBlox を拡張して、独自のカスタム Blox を作成することができます。 Java 開発者は、ContainerBlox を拡張し、DataLayoutBlox にドロップダウン・リストを追加するなど、Blox 内にコントロールを追加することができます。組み込まれた Blox に、再使用可能で内蔵タイプの拡張機能を作成することもできます。たとえば Java 開発者は、ユーザーが既存の ChartBlox のカラー・プロパティーを使用して円グラフの扇形スライスに色を割り当てることのできる、カラー・ピッカー・ダイアログを追加することができます。

ContainerBlox の JSP カスタム・タグ構文

DB2 Alphablox タグ・ライブラリーは、それぞれの Blox を作成するために JSP ページで使用するカスタム・タグを提供します。このセクションでは、ContainerBlox を作成するためのカスタム・タグの作成方法を説明します。すべての属性を含むタグのコピー・アンド・ペースト・バージョンについては、523 ページの『ContainerBlox JSP カスタム・タグ』を参照してください。

構文

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

ここで、それぞれ以下のとおりです。

attribute 属性表にリストされている属性の 1 つです。

value 属性の有効な値です。

属性は以下のいずれかになります。

属性
id
bloxName
enablePoppedOut
height
poppedOut
poppedOutHeight
poppedOutTitle
poppedOutWidth
render
visible
width

使用法

各カスタム・タグには 1 つ以上の属性を含めることができ、それぞれを 1 つ以上のスペースまたは改行文字で区切ります。余分のスペースまたは改行文字は無視されます。読みやすくするため、同じインデントでそれぞれ別々の行に属性を並べることができます。

例

```
<blox:container id="myContainer"
    width="200"
    height="100"
/>
```

ContainerBlox タグ属性

このセクションでは、ContainerBlox によってサポートされているタグ属性をアルファベット順に説明します。ContainerBlox メソッドについては、Javadoc 内にある com.alphablox.blox パッケージの ContainerBlox クラスを参照してください。ほとんどの Blox は ContainerBlox から継承しているので、ここにリストされているタグ属性は、 36 ページの『複数の Blox に共通するタグ属性』でも見つけることができます。

id

これは共通の Blox タグ属性です。詳しい説明は、43 ページの『id』を参照してください。

bloxName

これは共通の Blox プロパティーです。詳しい説明は、39 ページの『bloxName』を参照してください。

enablePoppedOut

別個のウィンドウ (アプリケーション・ページの「ポップアウト」) に Blox を開くことができるかどうかを指定します。

データ・ソース

すべて

構文

```
enablePoppedOut="enablePoppedOut"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
enablePoppedOut	true	ブール引数。 true の値は blox を他のウィンドウにポップアウトできるように、false の値はこの機能を使用不可に指定します。

使用法

デフォルトでは、enablePoppedOut は true に設定されており、ユーザーはツールバーの「ポップアウト」ボタンをクリックするか、または「表示」メニューから「ポップアウト」オプションを選択して、ポップアウトされたブラウザー・ウィンドウに Blox を表示できます。 enablePoppedOut が false に設定されているときは、このボタンおよびメニュー・オプションは使用不可です。ボタンおよびメニュー項目を除去するには、Blox UI タグを使用して UI コンポーネントを除去します。 446 ページの『Menubar、Menu、および MenuItem』を参照してください。

poppedOut および関連するプロパティーは、 PresentBlox および独立型の GridBlox/ChartBlox に適用されます。

例

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

関連項目

176 ページの『poppedOut』、176 ページの『poppedOutHeight』、177 ページの『poppedOutTitle』、178 ページの『poppedOutWidth』

height

これは共通の Blox プロパティーです。詳しい説明は、42 ページの『height』を参照してください。

poppedOut

Blox がロードされるときに、Blox が別個のウィンドウか、アプリケーション・ページの「ポップアウト」か、どちらに表示されるかを指定します。

データ・ソース

すべて

構文

```
poppedOut="popOut"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
popOut	false	ブール引数。 true の値は、 Blox がロードされるときに、Blox がポップアウト・ウィンドウに表示されるように指定します。false の値は、Blox がページと同じウィンドウに表示されるように指定します。

使用法

PresentBlox、独立型の GridBlox、または独立型の ChartBlox に適用します。

例

```
<blox:present id="myPresentBlox"
   poppedOut="true"
   poppedOutHeight="800"
   poppedOUtWidth="1000"
   poppedOutTitle="Sales Analysis Window"
   ...>
   ...
</blox:present>
```

関連項目

175 ページの『enablePoppedOut』、176 ページの『poppedOutHeight』、177 ページの『poppedOutTitle』、178 ページの『poppedOutWidth』

poppedOutHeight

別個のウィンドウ、またはポップアウト・ウィンドウの Blox の高さ (ピクセル) を指定します。

データ・ソース

すべて

構文

```
poppedOutHeight="newHeight"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
newHeight	600	ポップアウト・ウィンドウの高さをピクセルで指定する整数。

例

```
<blox:present id="myPresentBlox"
   poppedOutHeight="800"
   poppedOUtWidth="1000"
   poppedOutTitle="Sales Analysis Window"
   ...>
   ...
</blox:present>
```

関連項目

175 ページの『enablePoppedOut』、176 ページの『poppedOut』、177 ページの『poppedOutTitle』、178 ページの『poppedOutWidth』

poppedOutTitle

Blox が表示される別個の、またはポップアウトのウィンドウのタイトルを指定します。

データ・ソース

すべて

構文

```
poppedOutTitle="title"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
title	Popout Blox Window	任意のストリング。指定されたストリングは、ポップアウト・ウィンドウのタイトルとして表示されます。

使用法

デフォルトでは、ウィンドウ・タイトルとしてアプレットの名前を表示します。

例

```
<blox:present id="myPresentBlox"
   poppedOutHeight="800"
   poppedOUtWidth="1000"
   poppedOutTitle="Sales Analysis Window"
   ...>
   ...
</blox:present>
```

関連項目

175 ページの『enablePoppedOut』、176 ページの『poppedOut』、176 ページの『poppedOutHeight』、178 ページの『poppedOutWidth』

poppedOutWidth

別個のウィンドウ、またはポップアウト・ウィンドウの Blox の幅をピクセルで指定します。

データ・ソース

すべて

構文

```
poppedOutWidth="newWidth"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
newWidth	800	ポップアウト・ウィンドウの幅をピクセルで指定する整数。

使用法

Blox がすでにポップアウトされている場合、setPoppedOutWidth メソッドには効果がありません。

例

```
<blox:present id="myPresentBlox"
   poppedOutHeight="800"
   poppedOUtWidth="1000"
   poppedOutTitle="Sales Analysis Window"
   ...>
   ...
</blox:present>
```

関連項目

175 ページの『enablePoppedOut』、176 ページの『poppedOut』、176 ページの『poppedOutHeight』、177 ページの『poppedOutTitle』

render

これは共通の Blox プロパティーです。詳しい説明は、47 ページの『render』を参照してください。

visible

これは共通の Blox プロパティーです。詳しい説明は、49 ページの『visible』を参照してください。

width

これは共通の Blox プロパティーです。詳しい説明は、49 ページの『width』を参照してください。

第 10 章 DataBlox タグ・リファレンス

この章には、DataBlox の参照資料が含まれています。Blox についての一般的な参照情報は、21 ページの『第 3 章 一般 Blox リファレンス情報』を参照してください。このリファレンスの使用方法については、1 ページの『第 1 章 このリファレンスの使用法』を参照してください。

DataBlox の概説

DataBlox は以下の機能を提供します。

- リレーショナルまたはマルチディメンションのいずれかでアセンブラーがアクセスできるように、データ・セットの表記を (グリッド形式で) 提供する機能
- アプリケーション・スクリプトを使用可能にする機能 (照会の実行など)
- 他の Blox (ChartBlox または GridBlox など) のデータ・ソースとなる機能

DataBlox は、データにアクセスして照会を行う手段を提供するだけでなく、ResultSet オブジェクトおよび MetaData オブジェクトを戻します。戻される結果セットには、照会からの実際のデータ値が含まれ、計算やカスタム・データの表示などのタスクを実行できるようにします。 metadata オブジェクトには、データ・ソースのキューブ、ディメンション、およびメンバー (一括表示) に関する情報が含まれます。 DataBlox オブジェクト・モデルについて詳しくは、10 ページの『DataBlox - メタデータおよび結果セットへのアクセス』を参照してください。

RDBMetaData および RDBResultSet と関連した API を使用するには、次に示すように JSP ページに com.alphablox.blox.data.rdb パッケージをインポートする必要があります。

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

MDBMetaData および MDBResultSet と関連した API を使用するには、次に示すように JSP ページに com.alphablox.blox.data.mdb パッケージをインポートする必要があります。

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

DataBlox の JSP カスタム・タグ構文

DB2 Alphablox タグ・ライブラリーは、それぞれの Blox を作成するために JSP ページで使用するカスタム・タグを提供します。このセクションでは、カスタム・タグを作成して DataBlox を作成する方法を説明します。すべての属性を含むタグのコピー・アンド・ペースト・バージョンについては、523 ページの『DataBlox JSP カスタム・タグ』を参照してください。

構文

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

ここで、それぞれ以下のとおりです。

attribute 属性表にリストされている属性の 1 つです。

value 属性の有効な値です。

属性は以下のいずれかになります。

属性
id
bloxName
bloxRef
aliasTable
applyPropertiesAfterBookmark
autoConnect
autoDisconnect
bookmarkFilter
calculatedMembers
catalog
columnSort
connectOnStartup
credential
dataSourceName
dimensionRoot
drillDownOption
drillKeepSelectedMember
drillRemoveUnselectedMembers
enableKeepRemove
enableShowHide
hiddenMembers
hiddenTuples
leafDrillDownAvailable
memberNameRemovePrefix
memberNameRemoveSuffix
mergedDimensions
mergedHeaders
onErrorClearResultSet
parentFirst
password
performInAllGroups

属性
provider
query
retainSlicerMemberSet
rowSort
schema
selectableSlicerDimensions
showSuppressDataDialog
suppressDuplicates
suppressMissingColumns
suppressMissingRows
suppressNoAccess
suppressZeros
textualQueryEnabled
useAASUserAuthorizationEnabled
useAliases
useOlapDrillOptimization
userName

使用法

各カスタム・タグには 1 つ以上の属性を含めることができ、それぞれを 1 つ以上のスペースまたは改行文字で区切ります。余分のスペースまたは改行文字は無視されます。読みやすくするため、同じインデントでそれぞれ別々の行に属性を並べることができます。

終了タグ </blox:data> は、省略表現を使用して置き換えられます。以下のようなタグを使用して属性リストを閉じることができます。

```
useAliases="true" />
```

例

```
<blox:data
     dataSourceName="QCC-Essbase"
     query="<ROW (¥"All Products¥") <ICHILD ¥"All Products¥"
             <COLUMN (¥"All Time Periods¥") <CHILD ¥"All Time Periods¥"
             <PAGE (Measures) Sales !"
     useAliases="true"
     selectableSlicerDimensions="All Locations" >
</blox:data>
```

data タグがデータ・プレゼンテーション Blox (PresentBlox、 GridBlox、 ChartBlox、 DataLayoutBlox、 PageBlox、または MemberFilterBlox) のタグの内部でネストされる場合、id を持つことはできません。一般的には、以下のように独立型 DataBlox を id とともに定義し、後でそれをプレゼンテーション Blox で参照します。

```
<blox:data id="myDataBlox"
     dataSourceName="QCC-Essbase"
     query="<ROW (¥"All Products¥") <ICHILD ¥"All Products¥"
```

```
            <COLUMN (¥"All Time Periods¥") <CHILD ¥"All Time Periods¥"
            <PAGE (Measures) Sales !"
     useAliases="true"
     selectableSlicerDimensions="All Locations" >
</blox:data>

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

これにより、複数のプレゼンテーション Blox で同じ DataBlox を使用して同じデータの同期化されたビューを表示したり、Javaスクリプトレットで DataBlox id を使用して DataBlox プロパティーに直接アクセスし、変更したりすることができます。

カテゴリー別の DataBlox タグ属性

このセクションでは、GridBlox に固有のタグ属性をリストします。複数の Blox に共通するタグ属性については、35 ページの『カテゴリー別の共通の Blox タグ属性』を参照してください。 DataBlox によってサポートされるタグ属性は、以下のように編成されています。

- 182 ページの『データの外観』
- 182 ページの『データ・ソース』
- 183 ページの『データ操作』

DataBlox メソッドについては、Javadoc の com.alphablox.blox パッケージの DataBlox クラスを参照してください。

データの外観

以下は、データの外観に関連する DataBlox タグ属性です。

- memberNameRemovePrefix
- memberNameRemoveSuffix
- mergedDimensions
- mergedHeaders
- showSuppressDataDialog
- suppressDuplicates
- suppressMissingColumns
- suppressMissingRows
- suppressNoAccess
- suppressZeros
- textualQueryEnabled
- useOlapDrillOptimization

データ・ソース

以下は、データ・ソースに関連する DataBlox タグ属性です。

- aliasTable
- autoConnect

- autoDisconnect
- catalog
- credential
- dataSourceName
- onErrorClearResultSet
- password
- provider
- query
- schema
- useAASUserAuthorizationEnabled
- userName

データ操作

以下は、データを操作するために使用する DataBlox タグ属性です。

- calculatedMembers
- columnSort
- dimensionRoot
- drillDownOption
- drillKeepSelectedMember
- drillRemoveUnselectedMembers
- enableKeepRemove
- enableShowHide
- hiddenMembers
- hiddenTuples
- leafDrillDownAvailable
- parentFirst
- performInAllGroups
- retainSlicerMemberSet
- rowSort
- selectableSlicerDimensions
- useAliases

DataBlox タグ属性

このセクションでは、DataBlox によってサポートされるタグ属性をアルファベット順に説明します。DataBlox メソッドについては、com.alphablox.blox パッケージの DataBlox クラスを参照してください。 共通 Blox タグ属性の詳しい説明は、36 ページの『複数の Blox に共通するタグ属性』を参照してください。

id

これは共通の Blox タグ属性です。詳しい説明は、43 ページの『id』を参照してください。

bloxName

これは共通の Blox タグ属性です。詳しい説明は、39 ページの『bloxName』を参照してください。

bloxRef

これは共通の Blox タグ属性です。詳しい説明は、41 ページの『bloxRef』を参照してください。

aliasTable

データ・ソースで使用する IBM DB2 OLAP Server または Hyperion Essbase の別名表を指定します。

データ・ソース

IBM DB2 OLAP Server、Hyperion Essbase

構文

```
aliasTable="aliasTable"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
aliasTable	空ストリング	IBM DB2 OLAP Server または Hyperion Essbase 別名表の名前

使用法

aliasTable の値は、データ・ソースを DB2 Alphablox に定義したときに提供された値の 1 つです。 DataBlox に aliasTable プロパティーが指定されない場合、値は対応するデータ・ソース定義から取られます (値が存在する場合)。

IBM DB2 OLAP Server または Hyperion Essbase データベースの別名表にある名前はすべて ASCII 文字でなければなりません。

関連項目

208 ページの『dataSourceName』

applyPropertiesAfterBookmark

これは共通の Blox プロパティーです。詳しい説明は、36 ページの『applyPropertiesAfterBookmark』を参照してください。

autoConnect

データベース接続が必要なときには常に DataBlox がデータベースに自動接続できるようにします。

データ・ソース

リレーショナルのみ

構文

autoConnect="*autoConnect*"

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
autoConnect	false	autoConnect を使用可能にする場合は true を指定し、使用不可にする場合は false を指定します。

使用法

autoConnect プロパティーを使用すると、アプリケーションが接続を必要とするたびに DataBlox が自動的にデータ・ソースに再接続できるようにします。開始時にはデータ・ソースに接続しません。開始時に接続する場合は DataBlox connect() メソッドまたは connectOnStartup を使用します。

結果セットを消去することなく、必要な場合にのみ接続し、要求されたデータが取り出されると接続を切断するアプリケーションを作成するには、autoConnect と autoDisconnect を同時に使用できます。 autoConnect が使用可能でユーザーがデータ・ソース接続を必要とする操作を実行すると、DataBlox はデータ・ソースに接続し、現行の照会をリストアした後、操作を実行します。 autoDisconnect も使用可能になっている場合、DataBlox は操作の完了時にデータ・ソースから切断します。

重要: データ・ソースの接続と照会のリストアは時間の掛かるプロセスです。 autoConnect および autoDisconnect プロパティーは、データベースへの接続数が限られているなど、特定の場合にのみ使用してください。

autoConnect が使用不可で autoDisconnect が使用可能になっている場合、ユーザーは最初に切断された後、結果セットに対して操作を行うことができません。

関連項目

185 ページの『autoDisconnect』、205 ページの『connectOnStartup』、connect()、disconnect()

autoDisconnect

データ・アクセス操作の完了時に常に DataBlox がデータベースから自動切断できるようにします。

データ・ソース

リレーショナル、Microsoft Analysis Services、SAP BW

構文

autoDisconnect="*autoDisconnect*"

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
autoDisconnect	false	autoDisconnect を使用可能にする場合は true を指定し、使用不可にする場合は false を指定します。

使用法

autoDisconnect プロパティーを使用すると、アプリケーションが接続を必要となくなるときに DataBlox が現行の結果セットを消去することなく自動的にデータ・ソースから切断できるようにします。開始時に DataBlox が接続されるように connect() メソッドが設定され (または connectOnStartup プロパティーが true に設定され) 、autoDisconnect が使用可能になっている場合、DataBlox は接続し、照会をリストアした後 (適用可能な場合)、切断します。

必要な場合にのみ接続し、要求されたデータが取り出されると接続を切断するアプリケーションを作成するには、autoDisconnect と autoConnect を同時に使用できます。 autoDisconnect が使用可能な場合、DataBlox は接続を必要とする操作の完了時にデータ・ソースから切断します。 autoConnect も使用可能になっている場合、DataBlox は自動的にデータ・ソースに再接続します (必要な場合)。

重要: データ・ソースの接続と照会のリストアは時間の掛かるプロセスです。autoConnect および autoDisconnect プロパティーは、接続に対して大量のクライアント・キャッシュ・メモリーが消費されるために Microsoft Analysis Services でスケーラビリティーの問題が発生しているなど、特別な場合にのみ使用してください。この場合、多数の resolveMember() 呼び出しを伴う for ループなどのメタデータ操作を実行するカスタム・コードがある場合、メモリーを解放した後で clearClientCache() メソッドを呼び出す必要があります。「*開発者用ガイド*」にある『データへの接続』を参照してください。

autoDisconnect プロパティーは、サーバー・サイドの DataBlox 上の RDB メタデータ・オブジェクトには適用されません。 DataBlox は、autoDisconnect が true に設定されている場合でも、メタデータ要求の後 RDB データ・ソースから切断しません。アプリケーションがこのオブジェクトを使用している場合はアプリケーションを明示的に切断する必要があります。ただし、autoConnect を使用して将来のメタデータ要求でデータ・ソースに再接続することは依然可能です。

autoDisconnect が使用可能で autoConnect が使用不可になっている場合、ユーザーは最初に切断された後、結果セットに対して操作を行うことができません。

関連項目

184 ページの『autoConnect』、205 ページの『connectOnStartup』、connect()、disconnect()

bookmarkFilter

これは共通の Blox プロパティーです。詳しい説明は、37 ページの『bookmarkFilter』を参照してください。

calculatedMembers

データ・ソースから取り出される結果セットを使用して DB2 Alphablox によって計算される新規メンバーを指定します。計算で使用されるメンバーは、結果セットに存在していなければなりません。存在していないと、算出メンバーは現れません。

データ・ソース

すべて

構文

calculatedMembers="*definitionString*"

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
definitionString	空ストリング	1 つ以上の算出メンバー定義のコンマ区切りリスト。以下に示すように各定義を指定します。

以下のように、definitionString 内に各定義を指定します。

dim:*calc*{*refMember*:*gen:*missingIsZero:*drillDownMember*:*drillUpMember*}=
expression{*scopeDim*:*scopeMember*}

ここで、それぞれ以下のとおりです。

definitionString コンポーネント	説明
dim	算出メンバーを作成するディメンションの名前。 **注:** mergedHeaders プロパティーを使用してヘッダーをマージしている場合、マージされるディメンション・ヘッダーではなく元のディメンション名を使用する必要があります。というのは、calculatedMembers はディメンション・ヘッダーがマージされる前に実行されるからです。
calc	算出メンバーの名前。 この名前は既存のメンバーまたはディメンション名と同じにすることもできます。というのは、算出メンバーは ABXCalc_*dimName_calc* という形式の内部的な固有名を持つからです。たとえば、“Total” 算出メンバーが Product ディメンションに追加される場合、ABXCalc_Product_Total という固有名を持ちます。これは内部的に使用され、ユーザーが Dimension Explorer で固有名をオンにする場合にのみユーザーに表示されます。これはグリッドには表示されません。「別名の使用」データ・オプションがオフになっていても表示されません。

definitionString コンポーネント	**説明**
refMember	他の算出メンバーを含む、既存のメンバーの名前。この算出メンバーはこれの前に置かれます。参照メンバーの指定はオプションです。 特殊文字を含むメンバー名の場合は二重引用符で囲む必要があります。以下に例を示します。 `calculatedMembers="Product:¥"Profit %¥"{missingIsZero} = Gross Margin/Sales*100"` 算出メンバー *calc* は、グリッドで *refMember* の前に置かれます。参照メンバーを指定しない場合、算出メンバー *calc* は最後の行または列に置かれます。 ユーザーが参照メンバーをドリルまたは非表示にする場合、算出メンバーはその位置を保持します。しかし、ユーザーが参照メンバーを除去する場合、算出メンバーは最後の行または列に移動します。 198 ページの『例 2: 算出メンバーの位置を指定する』を参照してください。
gen	算出メンバーの世代番号。世代番号の指定はオプションです。 世代は絶対として定義することもできますし、refMember に対する相対として定義することもできます。これは軸上の算出メンバーのインデントを決定します。デフォルトの世代番号は 1 です。 絶対世代を指定するには、*gen* を正の整数として定義します。参照メンバーが定義されていない場合でも世代番号の前にコロンを置く必要があります。 相対世代を指定するには、+ (plus) または - (minus) 演算子を前に付けた整数として *gen* を定義します。算出メンバーの世代は、参照番号の世代番号に gen で定義された整数を加えたものまたは引いたものになります。相対世代を使用するには参照メンバーを定義する必要があり、それぞれをコロンで区切る必要があります。
missingIsZero	計算に関係するメンバーのすべての欠落値をゼロとして処理したい場合に使用するオプション・キーワード (大/小文字を区別しない)。デフォルトでは、計算におけるすべての欠落値は欠落として処理されます。デフォルトの振る舞いを変更するには、この特殊キーワードを使用します。以下の例では、Product1、Product2、および Product3 の欠落値がすべてゼロとして処理されています。 `Product:Total Sales{missingIsZero} = Product1 + Product2 + Product3` **注:** このキーワードは、メンバー変数を使用する計算のみに影響します。計算関数には影響はありません。
drillDownMember	オプション。ユーザーがこの算出メンバーに対してドリルダウンを試行する際にドリルダウンを実際に行うメンバー。
drillUpMember	オプション。ユーザーがこの算出メンバーに対してドリルアップを試行する際にドリルアップを実際に行うメンバー。以下に例を示します。 `Year:Total{:::Qtr3:Jul} = sum()` これは、算出メンバー “Total” から Qtr3 に対してドリルダウンし、Jul に対してドリルアップします。オプションの *refMember*、*gen*、および MissingIsZero が中括弧内で指定されない場合、コロンを組み込む必要があります。これは、*drillDownMember* が *refMember* として見なされないようにするためです。

definitionString コンポーネント	説明
expression	*dim* のメンバーを含む算術式または他のディメンションからの値。たとえば、 `calculatedMembers="All Products:Products 1 and 2 = Product1 + Product2"` は、“Products 1 and 2” という算出メンバーを “All Products” ディメンションに追加します。この式には、算出メンバーが追加される同じディメンションからのメンバーのみを含めます。 計算に、交差するディメンションまたは複数のディメンションからの値が含まれる場合、各メンバーの出身ディメンションを、コロンによってディメンションとメンバーを区切り、その後セミコロンによって各 *dimesnion:member* の対を区切って指定します。 *dim1:member1;dim2:member2;...;dimN:memberN* ここで、それぞれ以下のとおりです。 • 少なくとも 1 つのディメンションは行軸からのディメンションである • 少なくとも 1 つのディメンションは列軸からのディメンションである • セミコロンで区切られたそれぞれの *dimension:member* でディメンション名を指定し、その後にコロンとそのディメンションのメンバーを指定する 以下の例では、算出メンバー “Percentage of Total” が All Products ディメンションに追加されており、All Products と All Locations の交差の値が除数として使用されています。 `calculatedMembers="All Products: Percentage of Total= All Products / All Locations:All Locations; All Products:All Products` 詳しくは、200 ページの『例 5: 異なるディメンションからのメンバーを含む計算』を参照してください。計算でサポートされる関数については、191 ページの『CalculatedMembers の関数』 の詳細リストと説明を参照してください。

definitionString コンポーネント	説明
scopeDim: scopeMember	算出メンバーが表示されるディメンションおよびメンバーを定義します。追加の有効範囲のメンバーはコンマで区切られます。追加の対は、中括弧内でセミコロンで分けられます。有効範囲の指定はオプションです。特殊文字を含むメンバー名は二重引用符で囲む必要があります (内部二重引用符はエスケープする必要があります)。 有効範囲が定義されると、有効範囲で指定されたメンバーの算出メンバーのみが表示されます。しかし、算出メンバーと有効範囲メンバーが異なる軸上にある場合、有効範囲にない交差するセルは依然塗りつぶされたままとなります。この値は、missingValueString GridBlox プロパティーによって決定されます。 有効範囲が定義されない場合、算出メンバーは *expression* に関係するメンバーが現れるたびに表示されます。 Microsoft Analysis Services、Alphablox Cube、および SAP BW データ・ソースの場合は、以下のように固有の名前を使用します。 • ディメンション名は大括弧で囲み、さらにキューブ名 (MSAS、Alphablox Cube) または照会名 (SAP BW) を含める必要があります。別の方法としては、ディメンション名を大括弧なしで指定します。たとえば、 – Market – [My Cube].[Market] • メンバー名は大括弧で囲むか、キューブ (MSAS、Alphablox Cube) または照会 (SAP BW) から始まるメンバーの完全修飾された固有名を使用してください。たとえば、 – [California] – [My Cube].[Market].[West].[California] メンバー名を大括弧で囲まないと、名前はすべて大文字のメンバーとして扱われます。 算出メンバーの表示レベルの指定には、以下のメンバー検索関数を使用できます。 • `Leaf()`: 指定されたメンバーのリーフ・レベルの子孫を検索します。例: `Market: leaf(East)` (Essbase) • `Child()`: 指定されたメンバーの子を検索します。例: `Market: child(East)` (Essbase) • `Descendants()`: 指定のメンバーの子孫すべて。関数に指定できるメンバーは 1 つだけです。例: `[My Cube].[Market]:descendants([East])"` (MSAS、Alphablox Cube、SAP BW) • `Gen()`: 指定された世代のすべてのメンバーを検索します。例: `Market: gen(2)` • `Not()`: 計算が適用されないメンバーを検索します。例: `[My Cube].[Market]: not([East], [West])` (MSAS、Alphablox Cube、SAP BW) 指定された基準を満たすメンバーを検索する `Find()` という関数もあります。たとえば、`Find(Sales < 10000)` のように指定します。 関数名は大文字小文字を区別しません。 197 ページの『有効範囲』および 199 ページの『例 3: 世代番号および有効範囲を追加する』の使用法を参照してください。

CalculatedMembers の関数: 計算式に関数を使用できます。関数の構文は以下のようになっています。ただしこれは、 Abs、Sqrt、Round、ifNotNumber、 Power、Rank、RunningTotal、および検索に関係する関数には適用されません。

- *functionName(gen(generation))*。指定した世代にあるすべての項目のメンバーに基づいて値を計算します。世代 1 は、ルート・メンバーに基づいて値を計算することを意味します。以下の例は、Product ディメンションで “Standard Deviation” という算出メンバーを作成します。世代 2 のすべてのメンバーの標準偏差がその値です。

  ```
  Product: Standard Deviation = Stdev(gen(2))
  ```

  世代 0 は、すべての世代が計算に組み込まれることを意味します。

- *functionName(member1, member2, ..., memberN)*。算出メンバーの追加先ディメンションの指定されたメンバーに基づいて値を計算します。以下の例は、Product ディメンションで “Standard Deviation” という算出メンバーを追加します。
  CD、Cassette、および TV の標準偏差がその値です。

  ```
  Product: Standard Deviation = Stdev(CD, Cassette, TV).
  ```

- *functionName(searchFunction(member))*。検索関数 (Child、Descendants、Leaf、および find) からの結果に基づいて値を計算します。以下の例は、Product ディメンションで “Standard Deviation” という算出メンバーを追加します。 Audio のすべての子の標準偏差がその値です。

  ```
  Product: Standard Deviation = Stdev(Child(Audio))
  ```

- 関数は別の計算からの結果を取ることもできます。以下の例は、Product ディメンションで “Absolute Total Values” という算出メンバーを追加します。世代 2 メンバーの絶対値の合計がその値です。

  ```
  Product: Absolute Total Values = Abs(Sum(gen(2)))
  ```

サポートされる関数は以下のとおりです。

- 191 ページの『算術関数』
- 192 ページの『検索関数』
- 193 ページの『特殊計算関数』
- 195 ページの『条件関数および欠落値に関連した関数』

算術関数:

算術関数	説明
Abs	番号の絶対値を戻します。これを使用できるのは、別の計算の結果や単一のメンバーなどの単一数値項目に対してだけです。以下に例を示します。 `Product: Average for Audio = Abs(Average(Audio)` `Product: Absolute Value for CD Sales = Abs(CD)`
Average	定義内のすべての数値の平均を戻します。これは、sum を count で除算したものです。 count がゼロの場合、平均は欠落値として戻されます。
Count	定義に含まれるすべての数値の個数を戻します。欠落値は無視されます。カウントする値がない場合、ゼロが戻されます。

Max	定義内のすべての数値の中の最大値を戻します。
Median	セットの中央の数値を戻します。つまり、半分の数値が中央値より大きな値を持ち、半分が中央値より小さな値を持つことになります。
Min	定義内のすべての数値の中の最低値を戻します。
Power	最初のパラメーターを 2 番目のパラメーターで累乗したものを計算します。パラメーターは、メンバー名か数値定数のどちらかです。
Product	定義内のすべての値の積を戻します。
Round	数値をそれに最も近い整数に丸めた結果の整数を戻します。これを使用できるのは、別の計算の結果や単一のメンバーなどの単一数値項目に対してだけです。以下に例を示します。 `Product: Total Sales = Round(Sum(gen(2)))` `Product: Rounding value for CD Sales = Round(CD)`
Sqrt	数値の平方根を戻します。これを使用できるのは、別の計算の結果や単一のメンバーなどの単一数値項目に対してだけです。以下に例を示します。 `Product: My Calculation 2 = Sqrt(Average(gen(2)))` `Product: My Calculatioin 1 = Sqrt(CD)`
Stdev	定義内のすべての数値の標準偏差を戻します。標準偏差とは、どれだけ広く値が平均値から分散しているかを示す測度です。
Sum	定義内のすべての数値の加算を戻します。欠落値は無視されます。加算する値がない場合、ゼロが戻されます。
Var	分散 (セット中の各数値ごとに平均値からの偏差の 2 乗を計算し、その平均を求めた値) を戻します。

注: Microsoft Analysis Services および SAP BW データ・ソースの場合、検索関数を使用するときは固有の名前を指定します。

検索関数:

検索関数	**説明**
Child	指定されたメンバーのすべての子を戻します。以下に例を示します。 `Product: Average = Average(Child(Visual))` これは、Visual のすべての子の平均を計算します。 Microsoft Analysis Services、Alphablox Cube、および SAP BW データ・ソースの場合、メンバー名の絶対パスを使用する必要があります。たとえば、`child([2000])` ではなく、

child([Time].[Calendar].[All Time Periods].[2000]) を使用します。

Child

指定されたメンバーの子孫をすべて戻します。以下に例を示します。

```
Product: Average = Average(Descendants(Visual))
```

これは、Visual のすべての子孫の平均を計算します。

Microsoft Analysis Services、Alphablox Cube、および SAP BW データ・ソースの場合、メンバー名の絶対パスを使用する必要があります。たとえば、descendants([2000]) ではなく、descendants([Time].[Calendar].[All Time Periods].[2000]) を使用します。

Leaf

指定されたメンバーの子孫のうちリーフ・レベルのものをすべて戻します。以下に例を示します。

```
Product: Average = Average(Leaf(Visual))
```

これは、Visual のすべてリーフ・メンバーの平均を計算します。

Microsoft Analysis Services、Alphablox Cube、および SAP BW データ・ソースの場合、メンバー名の絶対パスを使用する必要があります。たとえば、leaf([2000]) ではなく、leaf([Time].[Calendar].[All Time Periods].[2000]) を使用します。

Find

検索条件を満たすすべてのメンバーを戻します。以下に例を示します。

```
Sum(Find(All Products:Rank>5)
```

これは、ランキングが 5 より下のすべての合計を計算します。有効な比較は、>、<、>=、<=、=、および != です (<> も非等価のテストで使用できます)。

Microsoft Analysis Services、Alphablox Cube、および SAP BW データ・ソースの場合、メンバー名の絶対パスを使用する必要があります。

特殊計算関数: 特殊計算関数には、LookupCount、Rank、および RunningTotal の 3 つがあります。

特殊計算関数	説明
LookupCount	Excel の LOOKUP 関数と同様、この関数はコンマ区切りの数値のリストを含む 2 つのパラメーター (2 つのストリング) を受け入れます。この関数は指

定された値を求めて最初のリストを調べ、2 番目のリストの同じ位置から値を戻します。

この関数は、固定値の検索に基づくカスタム統計関数の作成に使用できます。たとえば、D3() を以下のように定義するとします。

```
LookupCount("2,3,4,5", "8.3, 6.4, 4.3, 2.1")
```

2 項目の個数が D3() 列に値を持つ場合、結果セットは 8.3 になります。個数に 5 項目ある場合、計算の結果は 2.1 になります。 5 より大または 2 より小の場合、結果は NaN (数値なし) になります。

この関数を使用すると、制御チャートを作成するために算出メンバーで必要な統計関数をインプリメントできます。制御チャートは通常、プロセスが制御下にあるかどうかを把握するために、バリエーションを視覚的に探す目的で用いられます。

Rank

指定されたディメンションの値を、指定されたメンバーの値についての昇順または降順で戻します。 Rank の構文は以下のとおりです。

Rank(*member*, *dimension*, *generation*, *order*, *grouping*, *number*)

ここで、それぞれ以下のとおりです。

- *member* は、ランク付けするこのディメンションのメンバーです。
- *dimension* は、メンバーがランク付けの生成のために使用される反対軸上のディメンションです。
- *generation* は、ランク付けされるディメンションのメンバーの世代です。 0 を指定すると、すべてのメンバーがランク付けされます。
- *order* は、昇順の場合の ASC かまたは降順の場合の DESC のいずれかです。降順では、最も大きな数値が 1 とランク付けされ、昇順では最も小さな数値が 1 とランク付けられます。
- *grouping* はオプションです。これが存在し、GROUPDIM に設定される場合、反対軸上に複数のディメンションがある際に、ランキング定義の一部ではないディメンションの各グループごとにランク付けが個別に実行されます。これが設定されないか、または NOGROUP に設定される場合、ランキングはグループ全体に実行されます。これは軸上に複数のディメンションが存在する場合にのみ有効です。
- *number* はオプションです。ランク付けする項目の数を指定します。たとえば 5 を指定すると、上から 5 つのメンバーをランク付けします。こ

のパラメーターを指定する場合、grouping パラメーター (GROUPDIM または NOGROUP) も指定する必要があります。

200 ページの『例 6: ランキングを追加する』 および 201 ページの『例 7: 各グループ内に個別のランキングを追加する』を参照してください。

RunningTotal

指定されたメンバーに対して、指定されたディメンションの値の累積合計を戻します。 RunningTotal の構文は以下のとおりです。

RunningTotal(*member*, *dimension*, *generation*, *grouping*)

ここで、それぞれ以下のとおりです。

- *member* は、現在高を計算するこのディメンションのメンバーです。
- *dimension* は、メンバーが現在高の計算のために使用される反対軸上のディメンションです。
- *generation* は、計算されるディメンションのメンバーの世代です。 1 を指定すると、ルート・メンバーのみが計算に組み込まれます。 0 を指定すると、すべてのメンバーが組み込まれます。
- *grouping* はオプションです。これが存在し、GROUPDIM に設定される場合、反対軸上に複数のディメンションがある際に、現在高定義の一部ではないディメンションの各グループごとに現在高が個別に実行されます。これは軸上に複数のディメンションが存在する場合にのみ有効です。

201 ページの『例 8: 各グループ内に現在高を追加する』を参照してください。

条件関数および欠落値に関連した関数:

条件関数および欠落値に関連した関数 説明

If

この関数は、コンマで区切られる 3 つのパラメーターを取ります。

if(*condition*, *result_if_true*, *result_if_false*)

condition には、演算子 <=、>=、=、<、>、または != (<> も非等価のテストで使用できる) のうちの 1 つによって区切られる、左辺と右辺の部分があります。以下に例を示します。

```
Scenario:Act/Bdgt{MissingisZero} =
    If(Budget=0, 0, Actual - Budget*100 / Budget)
```

これは、Scenario ディメンションに “Act/Bdgt” という名前の算出メンバーを作成します。 Budget がゼロの場合 (または MissingIsZero キーワードが示すように Budget が欠落している場合)、算出メンバ

ーの値はゼロになります。 Budget がゼロでない場合、“Act/Bdgt” の値は Actual-Budget*100/Budget になります。

ifNotNumber

デフォルトでは、欠落値または NULL 値は欠落として処理されます。メンバー値のところを関数 ifNotNumber に置き換えることにより、計算で使用される結果セット内の欠落値または NULL 値を処理するための特殊ケース・ロジックを用意することができます。 ifNotNumber 関数の構文は次のとおりです。

ifNotNumber(*memberName*,*value*)

ここで、それぞれ以下のとおりです。

- *memberName* は、関数の対象となるメンバーの名前です。
- *value* は、欠落値または NULL のメンバー値を置き換える数値です。指定する値の中にコンマが含まれていてはなりません。

注: この関数は 1 度に 1 つのメンバーに対して有効です。計算に含まれるすべてのメンバーについて欠落値をゼロとして処理する場合、キーワード missingIsZero を使用します。 198 ページの『欠落値を含む計算』の使用法を参照してください。例については、199 ページの『例 4: 欠落値または NULL 値を 0 で置き換える』を参照してください。

使用法

算出メンバーには以下の制約事項が適用されます。

- コンマなどの特殊文字を含むメンバー名の場合は二重引用符で囲む必要があります。
- 展開/縮小モードでは算出メンバーに関して相対位置を指定できません。
- 追加された算出メンバーを表示するために、計算で使用される値は結果セットに存在している必要があります。たとえば、計算に世代 3 メンバーが含まれる場合、世代 3 メンバーが結果セットに存在していないと算出メンバーは表示されません。
- Microsoft Analysis Services、SAP BW、および DB2 Alphablox cube を使用する場合、構文は固有のメンバー名を使用する必要があります。
- リレーショナル・データ・ソースを使用する場合、使用可能な “dimensions” は Record # および Columns です。
- 算出メンバーで保持/除去オプションは使用できませんが、非表示/表示オプションは使用できます。

プロパティーの値に指定されるディメンション名およびメンバー名には、固有の名前 (IBM DB2 OLAP Server または Hyperion Essbase のベース名) または表示名を使用できます。これにより、同じ表示名を持つ異なるメンバーまたはディメンショ

ンを区別できます。 IBM DB2 OLAP Server または Hyperion Essbase では、別名表に関係なく、ベース名を使用することによりメンバーを指定できます。

算出メンバーをクリアするには、空ストリングを `setCalculatedMembers()` メソッドに渡します。

有効範囲: 複数の算出メンバーを作成しており、有効範囲に他の算出メンバーを組み込むかまたは除外する場合、算出メンバーの配列が重要となり、表示される結果の有効範囲に影響を与えます。たとえば、有効範囲を与えて算出メンバーが特定のメンバーのセットのみを計算するように限定する場合、有効範囲内の算出メンバーの後に作成される算出メンバー (つまり *definitionString* の右に現れる算出メンバー) はすべて、最初に有効範囲内の算出メンバーに組み込まれます。これは有効範囲内の算出メンバーを評価する時点で他の算出メンバーが存在しないためそれらを有効範囲から除去できないからです。このような場合、他の算出メンバーで使用される有効範囲外のメンバーの定義を有効範囲内の算出メンバーの前に置くこともできます。以下の 2 つの定義は同等ではなく、それぞれ異なる結果を生成します。

```
calculatedMembers="All Products: Product1 and Product2=Product1 + Product2
{Measures:Sales,COGS}, Measures:Gross Margin=Sales - COGS"
```

これは以下の出力を生成します。

	Sales	**COGS**	**Gross Margin**
Product1	500.00	300.00	200.00
Product2	1500.00	1000.00	500.00
Product1 + Product2	2000.00	1300.00	700.00

一方、以下の定義は上記とは異なる出力を生成します。

```
calculatedMembers="Measures:Gross Margin=Sales - COGS, All Products:
Product1 and Product2=Product1 + Product2 {Measures:Sales,COGS}"
```

これは以下の出力を生成します。

	Sales	**COGS**	**Gross Margin**
Product1	500.00	300.00	200.00
Product2	1500.00	1000.00	500.00
Product1 + Product2	2000.00	1300.00	

これら 2 つの例の間の違いは、最初の例の算出メンバーの有効範囲 `Product1 + Product2` には算出メンバー `Gross Margin` が組み込まれるのに対し (なぜなら有効範囲付けの時点で存在していなかったから)、2 番目の例の算出メンバーの有効範囲 `Product1 + Product2` には算出メンバー `Gross Margin` が組み込まれない、という点です。

メンバー検索関数を使用して、算出メンバーをメンバーのすべての子またはすべてのリーフ・レベルの子孫で表示するか、または特定の世代のメンバーで表示するかを指定することもできます。以下の例は、Scenario ディメンション上に算出メンバー Difference (Actual と Budget の間の相違) を Products のすべての子に関して作成します。

```
Scenario:Difference  = Actual - Budget {Products: Child(Products)}
```

欠落値を含む計算: 計算に欠落値が含まれる場合、デフォルトでは欠落データとして処理され、計算は失敗します。欠落データが GridBlox に表示されると、グリッド・セルはデフォルトではブランクになります。これは GridBlox に missingValueString と呼ばれるプロパティーがあるためです。このプロパティーのデフォルト値は空ストリングです。計算ですべての欠落値をゼロとして処理するには、missingIsZero キーワードを使用します。たとえば、

```
All Locations:East+Central {West:missingIsZero} = East + Central
```

これは、メンバー West の前に “East+Central” と呼ばれる算出メンバーを追加し、計算において欠落値はすべて 0 として処理されます。ただし、このキーワードはメンバー変数にのみ適用され、計算関数には影響を与えません。

すべてではなく一部のメンバーで欠落値を欠落として処理する場合は、欠落値をゼロとして処理するメンバーに対してそれぞれ ifNotNumber 関数を使用します。たとえば、

```
All Locations:East+Central {West} = ifNotNumber(East,0) + Central
```

これは、メンバー West の前に “East+Central” と呼ばれる算出メンバーを追加し、East の欠落値は 0 として処理され、Central の欠落値は欠落として処理されます。その結果、Central に欠落値が含まれていると計算は失敗して欠落を戻し、それらのグリッド・セルに空ストリングが表示されます。これが表示されないように GridBlox の missingValueString プロパティーを設定することもできます。

注: 計算が行または列全体で欠落値となる場合、行または列はまったく現れません。

199 ページの『例 4: 欠落値または NULL 値を 0 で置き換える』を参照してください。

例

例 1: Profit という名前の算出メンバーを Measures ディメンションの末尾に追加する: Profit メンバーの各セルの値は、Expenses メンバーおよび Sales メンバーの対応するセルの値を減算することで導出します。

```
setCalculatedMembers("Measures : Profit = Sales - Expenses");
```

	Actual		**Budget**	
	East	West	East	West
Sales	*value*	*value*	*value*	*value*
Expenses	*value*	*value*	*value*	*value*
Profit	*calc value*	*calc value*	*calc value*	*calc value*

例 2: 算出メンバーの位置を指定する: Expenses を参照メンバーとして追加することにより、グリッドでメンバー Expenses の前に算出メンバー Profit を配置できます。

```
setCalculatedMembers("Measures : Profit {Expenses} = Sales - Expenses");
```

	Actual		**Budget**	
	East	West	East	West

	Actual		Budget	
Sales	*value*	*value*	*value*	*value*
Profit	*calc value*	*calc value*	*calc value*	*calc value*
Expenses	*value*	*value*	*value*	*value*

例 3: 世代番号および有効範囲を追加する:

```
setCalculatedMembers("Measures : Profit {Expenses:2} = Sales - Expenses
{Actual:West}");
```

	Actual		Budget	
	East	West	East	West
Sales	*value*	*value*	*value*	*value*
Profit	*#missing*	*calc value*	*#missing*	*#missing*
Expenses	*value*	*value*	*value*	*value*

例 4: 欠落値または NULL 値を 0 で置き換える: 以下のような JSP タグがあるとします。

```
calculatedMembers = "All Locations:East+Central {West:2} = East + Central"
```

このコードは、世代 2 メンバーと同じインデント・レベルでメンバー West の前に配置される “East+Central” と呼ばれる算出メンバーを生成します。出力は次のようになります。

All Time Periods	All Locations	Truffles	Chocolate Blocks	Chocolate Nuts
2000	**Central**	6,119	29,068	
	East	883	3,679	
	East+Central	7,002	32,747	
	West	44,029	268,398	
	All Locations	51,031	301,145	

Chocolate Nuts にはデータがないため、Chocolate Nuts の算出メンバー “East+Central” にもデータはありません。

たとえば、欠落値をゼロとして処理するために `missingIsZero` キーワードを追加するとします。

```
calculatedMembers = "All Locations:East+Central {West:3:missingIsZero} =
East + Central"
```

生成される出力は次のようになります。

All Time Periods	All Locations	Truffles	Chocolate Blocks	Chocolate Nuts
2000	**Central**	6,119	29,068	
	East	883	3,679	
	East+Central	7,002	32,747	0
	West	44,029	268,398	
	All Locations	51,031	301,145	

例 5: 異なるディメンションからのメンバーを含む計算: 以下の JSP タグの例では、算出メンバー “Percent Total” が All Products ディメンションに追加されており、All Products の値が All Products と All Locations の交差によって除算されています。

```
calculatedMembers="All Products: Percent Total=
All Products / All Locations:All Locations; All Products:All Products"
```

生成される出力は次のようになります。

All Time Periods	All Locations	Specialties	Seasonal	All Products	Percent Total
2000	Central	72455.09	6849.592	107642.24	0.105
	East	10381.12	1620.36	14943.32	0.015
	West	593387.76	47641	905814.55	0.881
	All Locations	676223.97	56110.95	1028400.11	1
2001	Central	855542.96	29691.17	2384096.835	0.183
	East	6288893.64	15333.12	177250.755	0.136
	West	3266694.67	97033.65	8887288.195	0.681
	All Locations	4751131.27	142057.94	13043886.785	1
All Time Periods	Central	927998.05	36540.76	2491739.075	0.177
	East	639274.76	16953.48	1787445.075	0.127
	West	3860082.43	144674.65	9793102.745	0.695
	All Locations	5427355.24	198168.89	14072286.895	1

Percent Total 列の各値は、被除数としての All Products の値と除数としての All Products と All Locations の交差の値 (2000 年は 1028400.11、2001 年は 13043886.785) を使用して計算されます。

例 6: ランキングを追加する: この例では、算出メンバー “Rank” が All Products ディメンションに追加され、世代 2 メンバーの最も大きい数が第 1 としてランク付けされています。

```
All Products:Rank = Rank(All Products, All Locations, 2, DESC)
```

生成される出力は次のようになります。

All Time Periods	All Locations	Specialties	Seasonal	All Products	Rank
2000	Central	72455.09	6849.592	107642.24	8
	East	10381.12	1620.36	14943.32	9
	West	593387.76	47641	905814.55	7
	All Locations	676223.97	56110.95	1028400.11	
2001	Central	855542.96	29691.17	2384096.835	4
	East	6288893.64	15333.12	177250.755	6
	West	3266694.67	97033.65	8887288.195	2
	All Locations	4751131.27	142057.94	13043886.785	

All Time Periods	All Locations	Specialties	Seasonal	All Products	Rank
All Time Periods	Central	927998.05	36540.76	2491739.075	3
	East	639274.76	16953.48	1787445.075	5
	West	3860082.43	144674.65	9793102.745	1
	All Locations	5427355.24	198168.89	14072286.895	

上位 N または下位 N のメンバーのみをランク付けするには、6 番目のパラメーターとして末尾に整数を追加します。ランキングの番号を指定するには、5 番目のパラメーター (NOGROUP または GROUPDIM) を指定する必要があります。

```
calculatedMembers="All Products:Rank = Rank(All Products, All Locations, 2,
DESC, NOGROUP, 5)"
```

例 7: 各グループ内に個別のランキングを追加する:

```
calculatedMembers = "All Products:Rank =
Rank(All Products, All Locations, 2, DESC, GROUPDIM)"
```

上記の例は以下の出力を生成します。

All Time Periods	All Locations	Specialties	Seasonal	All Products	Rank
2000	Central	72455.09	6849.592	107642.24	2
	East	10381.12	1620.36	14943.32	3
	West	593387.76	47641	905814.55	1
	All Locations	676223.97	56110.95	1028400.11	
2001	Central	855542.96	29691.17	2384096.835	2
	East	6288893.64	15333.12	177250.755	3
	West	3266694.67	97033.65	8887288.195	1
	All Locations	4751131.27	142057.94	13043886.785	
All Time Periods	Central	927998.05	36540.76	2491739.075	2
	East	639274.76	16953.48	1787445.075	3
	West	3860082.43	144674.65	9793102.745	1
	All Locations	5427355.24	198168.89	14072286.895	

算出メンバー “Rank” は All Locations ディメンションの世代 2 メンバーの値に基づいて All Products ディメンションに追加され、最も大きい数が第 1 としてランクされて、ディメンションの各グループ内で個別にランク付けされています。

例 8: 各グループ内に現在高を追加する:

```
calculatedMembers = "All Products:Running Totals =
RunningTotal(All Products, All Locations, 2, GROUPDIM)"
```

上記の例は以下の出力を生成します。

All Time Periods	All Locations	Specialties	Seasonal	All Products	Running Totals
2000	Central	72455	6850	107642	107642
	East	10381	1620	14943	122586
	West	593388	47641	905815	1028400
	All Locations	676224	56111	1028400	
2001	Central	855543	29691	2384097	2384097
	East	6288894	15333	1772502	4156599
	West	3266695	97034	8887288	13043887
	All Locations	4751131	142058	13043887	
All Time Periods	Central	927998	36541	2491739	2491739
	East	639275	16953.48	1787445	4279184
	West	3860082	144675	9793102.745	14072287
	All Locations	5427355	198169	14072287	

“Running Totals” と呼ばれる算出メンバーが All Products ディメンションに追加され、これにはそれぞれの All Locations ディメンション上の世代 2 メンバーのグループごとの累積合計が入ります。

関連項目

「*開発者用ガイド*」の『データの入力および変更』。

catalog

この DataBlox のデータベース・カタログを指定します。

データ・ソース

すべて

構文

```
catalog=catalog
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
catalog	空ストリング	データベース・カタログの名前。

使用法

catalog の値は、データ・ソースを Analysis Server に定義したときに提供された値の 1 つです。 DataBlox に catalog プロパティーが指定されない場合、値は対応するデータ・ソース定義から取られます (値が存在する場合)。

IBM DB2 OLAP Server または Hyperion Essbase 用語では、カタログは「アプリケーション」と呼ばれます。

columnSort

列軸上のメンバーのデータ値をソートする方法を指定します。

データ・ソース

すべて

構文

```
columnSort=sortString
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
column	なし	リレーショナル結果セットの列。
ascending	なし	昇順にソートする場合は true を指定し、降順にソートする場合は false を指定します。
tuple	なし	ソートされる列を指定する列軸上のタプル。
dimension	なし	グループ化が保存される行軸上のディメンション。行軸上にグループ化が保存されないことを指定するには、null を指定します。
preserveHierarchy	false	ソート操作の後にメンバーとその親を保持しつつ行軸の階層を保存する場合は true を指定し、階層を保存しない場合は false を指定します。

引数	デフォルト	説明
sortString	なし	以下の形式のいずれかのコンマで区切られたストリング。 • *tupleIndex*, *direction* • *tupleIndex*, *groupingNestLevel*, *direction* • *tupleIndex*, *groupingNestLevel*, *direction, preserveHierarchy* *tupleIndex* - ゼロ・ベースのソート対象タプル索引メンバー (列) を表す整数のストリング表記。0 は左端列を示します。 *groupingNestLevel*- グループ化が保存される行軸上のディメンションを表す整数のストリング表記。たとえば、Product および Market が行軸上にある場合、1 は Market ディメンション内のシーケンスにソートされます。 -1 を指定すると、行のグループ化に関係なくソートします。デフォルトは -1 です。 *direction* - "Ascending"、"Asc"、"Descending"、または "Desc" のいずれかのストリング。大/小文字を区別しません。 *preserveHierarchy* - ブールのストリング表記。preserveHierarchy 引数を参照してください。デフォルトは false になります。 以下に例を示します。 `setColumnSort("1,0,asc,true");` `setColumnSort("1,0,asc");` `setColumnSort("0,descending");`

使用法

getColumnSort メソッドは、コンマで区切られた 4 つの項目のストリング *tupleIndex*、*groupingNestLevel*、*direction*、および *preserveHierarchy* を戻します。

以下のスクリーン・ショットは、1) 階層が保存されるかどうか 2) 指定されたレベル/ディメンション内のグループ化が保存されるかどうかに応じた Qtr1 列に対する昇順のソート操作の結果を示しています。最初の例は、Market ディメンションへの階層の保存は行われますが、Product ディメンションへのグループ化の保存は行われません。

Product	Market	Qtr1
200	East	562
	New Mexico	-14
	Louisiana	336
	Oklahoma	468
	Texas	675
	South	1465
	West	2325
	Central	2369
	Market	6721
100	West	1042
	New Mexico	-27
	Oklahoma	171
	Louisiana	212
	Texas	695
	South	1051
	Central	2208
	East	2747
	Market	7048

以下の例は、Market ディメンションへの階層の保存も Product ディメンションへのグループ化の保存も行いません。

Product	Market	Qtr1
100	New Mexico	-27
200	New Mexico	-14
100	Oklahoma	171
	Louisiana	212
200	Louisiana	336
	Oklahoma	468
	East	562
	Texas	675
100	Texas	695
	West	1042
	South	1051
200	South	1465
100	Central	2208
200	West	2325
	Central	2369
100	East	2747
200	Market	6721
100	Market	7048

例

以下の例は、`columnSort` タグ属性の使用を示しています。

```
columnSort="1, 0, asc"
```

関連項目

227 ページの『rowSort』、Javadoc 内の DataBlox `removeColumnSort()` メソッド

connectOnStartup

Blox をインスタンス化するときに DataBlox をそのデータ・ソースに自動接続するかどうかを指定します。

データ・ソース

すべて

構文

```
connectOnStartup="connectOnStartup"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
connectOnStartup	true	ブール引数。 Blox がインスタンス化されるときにデータ・ソースに自動接続する場合は true を指定し、自動接続しない場合は false を指定します。

使用法

DataBlox がデータ・ソースに接続しないようにするには、このプロパティーをfalseに設定します。 query プロパティーを後で設定する場合、connect() メソッドを呼び出してデータ・ソースに接続する必要があります。

connectOnStartup が true に設定すると、autoConnect プロパティーをオーバーライドします。照会が定義されていない場合でも connectOnStartup プロパティーによってデータベースに接続されます。

関連項目

226 ページの『query』、184 ページの『autoConnect』、185 ページの『autoDisconnect』

credential

ユーザー名およびパスワードではなく、指定のデータ・ソースで使用する証明書オブジェクトを指定します。

データ・ソース

DB2 OLAP Server および Hyperion Essbase

構文

```
credential="credential"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
credential		データ・ソースに送られる証明書オブジェクト。

使用法

このプロパティーは、ユーザー名およびパスワードではなく、DB2 Alphablox が DB2 OLAP Server または Hyperion Essbase データ・ソースに受け渡す証明書オブジェクトを受け入れます。DB2 OLAP Server または Hyperion Essbase のシングル・サインオン機能を使用すると、ユーザーは DB2 Alphablox リポジトリーにユーザー名とパスワードを保管しなくてもデータ・ソースにログインできます。Hyperion の共通セキュリティー・サービスを使用すると、DB2 Alphablox に渡すトークンを

取得できます。credential 属性を設定すると、DB2 Alphablox のデータ接続用のユーザー名とパスワードが無視されます。このプロパティーは、ブックマークでは保管されません。

例

以下の断片的な例は、トークンを共通セキュリティー・サービスから取得して、DataBlox に渡す方法について示しています。

```
<%@ 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 omittend here.
}
%>

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

<%
    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="myData" credential="<%=credential %>"
    dataSourceName="EssbaseSSO" useAliases="true"
    query="!" visible="false" />

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

dataSourceName

この DataBlox がアクセスする外部データ・ソースを識別します。

データ・ソース

すべて

構文

```
dataSourceName="dataSourceName"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
dataSourceName	空ストリング	DB2 Alphablox に定義されているデータ・ソース名。

使用法

データ・ソース名を DB2 Alphablox に定義する必要があります。データ・ソースを指定しない場合、Blox は初期データ・ソースなしでロードされます。この機能により、ユーザーのプロパティーまたはアクションに基づいてデータ・ソースをプログラマチックに設定できます。ただし、ユーザーにエラー・メッセージが表示されないようにするには、autoConnect プロパティーの値を `false` に設定する必要があります。

この `setDataSourceName()` メソッドは、username、password、catalog、schema、query などのデータ・ソースのプロパティーやページ軸上のディメンションでも読み取れます。そのため、`setUserName()` や `setPassword()` などの Javaメソッドを使用してこれらのプロパティーのいずれかを設定する場合は、それらを `setDataSourceName()` メソッドの後で設定してください。

ヒント: これらのデータ・ソースのプロパティーが設定される順序は、Blox タグを使用する場合には問題となりません。タグは、他のデータ・ソース・プロパティーを設定する呼び出しの前にデータ・ソースの設定を施行するよう設計されているので、副次作用は自動的に解決されます。

ヒント: セキュリティー上の理由により、JSP ページ上で `userName` プロパティーや `password` プロパティーに値を指定するのは得策ではありません。これらのプロパティーについて詳しくは、237 ページの『userName』 および 224 ページの『password』 を参照してください。関連情報については、226 ページの『query』、229 ページの『schema』、235 ページの『useAASUserAuthorizationEnabled』、225 ページの『provider』、および 202 ページの『catalog』 も参照してください。

例

以下の例は、IBM DB2 OLAP Server または Hyperion Essbase データ・ソースの値を持つカスタム・タグ属性を示しています。

```
dataSourceName="MyEssbaseDataSource"
schema="Basic"
catalog="Demo"
query="<SYM <ROW (Product) <ICHILD Product <COL (Market) <ICHILD Market !"
```

以下の例は、Alphablox キューブの値を持つプロパティーを示しています。

```
dataSourceName="MyAlphabloxCube"
query="SELECT Measures.MEMBERS ON COLUMNS,
         {[Store].[Store State].[CA], [Store].[Store State].[WA]}
         ON ROWS
       FROM [Sales]"
```

以下の例は、Microsoft OLAP Services データ・ソースの値を持つプロパティーを示しています。

```
dataSourceName="MyOLAPDataSource"
catalog="MySchema"
schema="MyCatalog"
query="SELECT Measures.MEMBERS ON COLUMNS,
         {[Store].[Store State].[CA], [Store].[Store State].[WA]}
         ON ROWS
       FROM [Sales]"
```

以下の例は、IBM DB2 UDB データ・ソースの値を持つプロパティーを示しています。

```
dataSourceName="MyDB2"
catalog="MyCatalog"
query="SELECT Actual.SalesQty, Actual.ProductID FROM MyCatalog"
```

以下の例は、Oracle データ・ソースの値を持つプロパティーを示しています。

```
dataSourceName="MyOracleDataSource"
schema="MySchema"
catalog="MyCatalog"
query="SELECT Actual.SalesQty, Actual.ProductID,
          Projected.SalesQty, Projected.ProductID
       FROM Actual, Projected
       WHERE Actual.SalesQty <Projected.SalesQty"
```

dataSourceName ストリングの値は、DB2 Alphablox ですでに定義済みのデータ・ソースでなければなりません。データ・ソースをセットアップする方法について詳しくは、「*管理者用ガイド*」を参照してください。

関連項目

202 ページの『catalog』、226 ページの『query』、229 ページの『schema』、235 ページの『useAASUserAuthorizationEnabled』

dimensionRoot

ルートとして使用するディメンションおよび単一のメンバーを指定します。

データ・ソース

マルチディメンション

構文

```
dimensionRoot="dimensionNameAndNewRootMember"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
dimensionNameAndNewRootMember	空ストリング	指定されたディメンション (複数の場合あり) の新規ルート・メンバーを指定するストリング。ストリングの形式は以下のとおりです。 `"DimA:NewRootMemberA;DimB:NewRootMemberB;"` 新規ルート・メンバーなしでディメンションを指定する場合、ディメンション・ルートはそのディメンションのデータベースのデフォルト・ルート・メンバーにリセットされます。
dimension	なし	Dimension オブジェクト。 dimension がページ・フィルターまたはメンバー・フィルターに現れるか、またはメンバーが null の場合、データベースはそのデフォルトのディメンション・ルートを使用し、ディメンションのルートとして機能します。
member	なし	Member オブジェクト。 member が null の場合、ディメンション・ルートはデータベース・デフォルトにリセットされます。

使用法

名前付きディメンションがページ・フィルターまたはメンバー・フィルターに現れる場合、選択されるメンバーはディメンションのルートとして機能します。このプロパティー値と照会の間で競合がある場合、照会がプロパティーをオーバーライドします。

プロパティーの値に指定されるディメンションおよびメンバー名ストリングには、固有の名前 (IBM DB2 OLAP Server または Hyperion Essbase のベース名) または表示名を使用できます。これにより、アセンブラーは同じ表示名を持つ異なるメンバーまたはディメンションを区別できます。 IBM DB2 OLAP Server または Hyperion Essbase では、アセンブラーはベース名を使用することによって、使用中の別名表とは関係なくメンバーを指定できます。

複数のディメンションを指定できますが、ディメンションにつき 1 メンバーのみです。

String getDimensionRoot() メソッドは以下の形式のストリングを戻します。

```
"DimA:RootMemberA;DimB:RootMemberB;"
```

例

以下の例は、DataBlox カスタム・タグの属性として、Products ディメンションのルートが Tools であり、Market ディメンションのルートが East であることを指定します。

```
dimensionRoot="Products: Tools;Markets: East"
```

関連項目

226 ページの『query』

drillDownOption

実行するドリル操作のタイプを指定します。

データ・ソース

マルチディメンション

構文

```
drillDownOption="drillDown"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
drillDown	1	ドリルダウンするレベルを指定する 1 から 5 までの整数。可能な値は以下のとおりです。 1: 次の世代にドリルダウン 2: すべての子孫にドリルダウン (「すべて展開」と同じ) 3: 最低世代にドリルダウン 4: 兄弟にドリル 5: 同じ世代にドリル

使用法

子孫、兄弟、世代などの用語の説明については、「*管理者用ガイド*」の『OLAP の用語および概念』を参照してください。

例

```
drillDownOption="4"
```

関連項目

211 ページの『drillKeepSelectedMember』、212 ページの『drillRemoveUnselectedMembers』、DataBlox `drillDown()` および `drillToAllDescendants()` メソッド

drillKeepSelectedMember

再表示の際に、ドリル操作されるメンバーを保存するかまたは除去するかを指定します。

データ・ソース

マルチディメンション

構文

```
drillKeepSelectedMember="keepSelected"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
keepSelected	true	ブール引数。ドリル操作されるメンバーを保存する場合は true を指定し、除去する場合は false を指定します。

例

```
drillKeepSelectedMember="false"
```

関連項目

212 ページの『drillRemoveUnselectedMembers』

drillRemoveUnselectedMembers

再表示の際にドリル操作されていないすべてのメンバーを除去するかどうかを指定します。

データ・ソース

マルチディメンション

構文

```
drillRemoveUnselectedMembers="removeUnselected"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
removeUnselected	false	ドリル操作されていないすべてのメンバーを除去する場合は true を指定し、保持する場合は false を指定します。

例

```
setDrillRemoveUnselectedMembers(true);
```

関連項目

211 ページの『drillKeepSelectedMember』

enableKeepRemove

GridBlox と ChartBlox 両方のメニューのコンテキストでエンド・ユーザーが「選択的保持」および「選択的除去」オプションを使用できるようにするかどうかを指定します。

データ・ソース

すべて

構文

enableKeepRemove="*enable*"

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
enable	false	「選択的保持」および「選択的除去」オプションを使用可能にする場合は true を指定し、使用不可にする場合は false を指定します。

使用法

「選択的保持」および「選択的除去」オプションによって、エンド・ユーザーはグリッドに表示できるメンバーおよび列を制御することができます。

このプロパティーを使用して「選択的除去」および「選択的保持」オプションを使用可能または使用不可にした場合でも、エンド・ユーザーは「オプション」ダイアログの「データ」タブの下の「保持および除去を使用可能にする (Enable keep and remove)」チェック・ボックスを使用することによってこれを使用可能または使用不可にできます。

例

```
enableKeepRemove="true"
```

関連項目

213 ページの『enableShowHide』

enableShowHide

GridBlox および ChartBlox 両方のメニューのコンテキストでエンド・ユーザーが「選択的表示」、「すべて表示」、および「選択的非表示」オプションを使用できるようにするかどうかを指定します。

データ・ソース

すべて

構文

enableShowHide="*enable*"

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
enable	true	「選択的表示」、「すべて表示」、および「選択的非表示」オプションを使用可能にする場合は true を指定し、使用不可にする場合は false を指定します。

使用法

「表示/非表示 (Show/Hide)」オプションによって、エンド・ユーザーはグリッドに表示できるメンバーを制御することができます。「保持/除去 (Keep/Remove)」オプションと動作は似ていますが、置き換えることはできません。「表示/非表示 (Show/Hide)」機能を「保持/除去 (Keep/Remove)」と同じ場面で使用することができます。

エンド・ユーザーは「データ」タブの下の「オプション」ダイアログ・ボックスを介して「表示/非表示 (Show/Hide)」機能をオン/オフにすることができます。これが使用可能なときに使用できるオプションは、メンバーを個別に表示または非表示にすること、およびディメンションのメンバーをすべて表示することです。メンバーが一度非表示になると、ユーザーは選択的にメンバーを表示したり、すべてのメンバーを非表示にしたりできません。各ディメンションには、常に 1 メンバーが含まれていなければなりません。

非表示のメンバーは引き続きメンバー・フィルターに現れます。さらに、特定のメンバーが非表示のときにブックマークを保管すると、その非表示の状態が保存されます。

「表示/非表示 (Show/Hide)」と「保持/除去 (Keep/Remove)」の比較

「表示/非表示 (Show/Hide)」機能は、いくつかの点で「保持/除去 (Keep/Remove)」機能と異なります。「表示/非表示 (Show/Hide)」機能は、「すべて表示」によって非表示が解除されるまで、後続の GUI 操作を介してメンバーの非表示の状態を保存します。たとえば、ユーザーは非表示メンバーのレベルにドリルアップおよびドリルダウンでき、メンバーは非表示のままです。この点は「保持/除去 (Keep/Remove)」機能と異なります。同じ環境で「保持/除去 (Keep/Remove)」機能を実行すると、除去されるメンバーは表示されます。

「表示/非表示 (Show/Hide)」は、DB2 Alphablox 算出メンバーの出力に干渉しません。メンバーはビューからは非表示になりますが、データは引き続き計算で使用できます。「保持/除去 (Keep/Remove)」を使用してメンバーを除去すると、算出メンバーは除去されたデータにアクセスできなくなり、“#missing” の値 (または指定したいずれかの欠落値ストリング) を戻します。

例

```
enableShowHide="true"
```

関連項目

212 ページの『enableKeepRemove』

hiddenMembers

「表示/非表示 (Show/Hide)」機能を使用して非表示にするメンバーを指定します。

データ・ソース

すべて

構文

```
hiddenMembers="membersToHide"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
membersToHide	空ストリング	最初に非表示にされるメンバーのリスト。ストリングを以下の形式にします。 `DimensionName1:MemberNameA,MemberNameB;` `DimensionName2:MemberNameD,MemberNameD;` `...` `DimensionName`*`N`*`:MemberNameX,MemberNameY` 異なるディメンションにあるメンバーを分けるにはセミコロンが必要です。 **注:** 220 ページの『mergedHeaders』 を使用して複数のディメンションのヘッダーを 1 つにマージする場合 (mergedHeaders は最初に実行される)、非表示にするメンバーを指定する際には新しくマージされたヘッダーを使用する必要があります。

使用法

各ディメンションの 1 メンバーがグリッドに存在していなければなりません。ディメンションのすべてのメンバーが非表示になるように指定する場合、指定された最後のメンバーは非表示になりません。どのディメンションにも、除去していないメンバーが存在することを確認するのが最善です。

例

Market ディメンションのメンバー *East* および *North* と、*Product* のメンバー *Audio* を非表示にする場合、`hiddenMembers` プロパティーを以下のように定義します。

```
hiddenMembers="Market:East,North; Product:Audio"
```

関連項目

213 ページの『enableShowHide』、DataBlox `hideMembers()` および `showMembers()` メソッド。

hiddenTuples

「表示/非表示 (Show/Hide)」機能を使用して非表示にする結果セットのタプルを指定します。

データ・ソース

マルチディメンション

構文

```
hiddenTuples="selectedTuples"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
selectedTuples	空ストリング	最初に非表示にされるタプルのリスト。ストリングを以下の形式にします。 `Dimension1,Dimension2,...,DimensionN:` `Dim1Member1, Dim2Member1,..,DimNMember1;` `Dim1Member2,Dim2Member2,...,DimNMember2;...` `Dim1MemberM,Dim2MemberM,...,DimNMemberM` 各タプル・リストには、コンマで区切られたディメンション名と、その後に続くコロン、タプルのリストが含まれています。各タプルは、指定されたそれぞれのディメンションからの 1 メンバーで構成され、セミコロンによって区切られます。タプルのメンバーはそれぞれコンマで区切られます。 複数のタプル・リストを指定することもでき、その場合それぞれのリストを中括弧で囲み、コンマで区切ります。これにより、同じ構文の反対軸上にあるディメンションのタプルの指定が可能になります。詳しくは、以下の『使用法』を参照してください。 **注:** 220 ページの『mergedHeaders』 を使用して複数のディメンションのヘッダーを 1 つにマージする場合 (mergedHeaders は最初に実行される)、非表示にするタプルを指定する際には新しくマージされたヘッダーを使用する必要があります。

使用法

指定されたタプルを非表示にするには、タプルが結果セットに存在していなければなりません。

同じ構文の反対軸上のタプルを指定するには、以下のように 1 つの軸上のディメンションからのタプル・リストを中括弧で囲み、コンマでリストを区切ります。

```
{Period,Product:Q1,Audio;Q2,Visual},{Accounts,Market:Profit,East}
```

上記の例は以下の出力を生成します。

		Margin				Profit		
Period	Product	East	West	South	Market	West	South	Market
Q1	Visual	4950.9	23995	15344	44289.9	8042	4474	4373.9
	Product	1461	37890	15344	54495	12917	4474	834
Q2	Audio	9331	13354		22685	4320	0	6852
	Product	28232	36785	14895	79912	11449	4199	22924
Q3	Audio	9390	13745		23135	5324	0	8300
	Visual	22282	24740	15675	62697	8946	5430	22680
	Product	31672	38485	15675	85832	14270	5430	30980

注: setHiddenTuples() メソッドは非表示にされたタプルの新規リストを設定し、以前に設定された非表示のタプル・リストをすべてオーバーライドします。追加のタプルを非表示にするには、DataBlox hideTuples() メソッドを使用します。

関連項目

DataBlox hideTuples() メソッド

leafDrillDownAvailable

ユーザーによるリーフ・メンバーに対するドリルダウンの許可を指定します。

データ・ソース

マルチディメンション

構文

leafDrillDownAvailable="*available*"

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
available	false	リーフのドリルダウンを使用可能にする場合は true を指定し、使用不可にする場合は false を指定します。

使用法

このプロパティーは、ユーザーがリーフ・メンバーにドリルダウンする際にカスタム・アクションを実行する場合にのみ有効です。この場合にのみ、値を true に設定します。関数を呼び出してリーフ・メンバーへのドリルダウンを使用可能にする必要がない場合、このプロパティーをデフォルト値の false のままとしてください。

memberNameRemovePrefix

データ・ソースから戻されたときのメンバー名の開始点を指定します。

データ・ソース

マルチディメンション

構文

memberNameRemovePrefix="*prefix*"

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
prefix	空ストリング	メンバー名の開始点。

使用法

memberNameRemovePrefix プロパティーは、指定されたストリングで始まる、またはそれを含むデータ・ソースから戻されたメンバー名ストリングのテキストを除去します。

注: このメソッドは、結果セットにのみ影響します。メタデータには影響を与えません。つまり、メンバーの表示名を取得するための後続のメタデータ呼び出しには依然接頭部が含まれます。

このプロパティーは IBM DB2 OLAP Server または Hyperion Essbase データ・ソースを使用する場合にのみ使用され、その際メンバー名は固有でなければなりません。固有の名前はしばしばメンバー名に接尾部または接頭部として固有のストリングを追加することによって作成されます。このプロパティーを使用すると、メンバー名を表示する前に接頭部のストリングを取り除くことができます。

メンバー名にのみこの除去は行われ、ディメンション名には行われません。さらに、引数としてメンバー名を取るプロパティーおよびメソッドは、接頭部および接尾部を除去する前に固有のメンバー名を使用します。

例

memberNameRemovePrefix プロパティーが ″##″ の場合、メンバー ″123##Year″ は ″Year″ として表示されます。

このプロパティーは memberNameRemoveSuffix プロパティーと併用できます。たとえば、memberNameRemovePrefix ストリングが ″$$″ で memberNameRemoveSuffix ストリングが ″##″ の場合、メンバー ″123$$Year##978-9″ は ″Year″ として表示されます。

関連項目

218 ページの『memberNameRemoveSuffix』

memberNameRemoveSuffix

データ・ソースから戻されたときのメンバー名の終点を指定します。

データ・ソース

マルチディメンション

構文

memberNameRemoveSuffix="*suffix*"

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
suffix	空ストリング	メンバー名の終点。

使用法

memberNameRemoveSuffix プロパティーは、指定されたストリングで始まる、またはそれを含むデータ・ソースから戻されたメンバー名ストリングのテキストを除去します。

注: このメソッドは、結果セットにのみ影響します。メタデータには影響を与えません。つまり、メンバーの表示名を取得するための後続のメタデータ呼び出しには依然接尾部が含まれます。

このプロパティーは IBM DB2 OLAP Server または Hyperion Essbase データ・ソースを使用する場合にのみ使用され、その際メンバー名は固有でなければなりません。固有の名前はしばしばメンバー名に接尾部または接頭部として固有のストリングを追加することによって作成されます。このプロパティーを使用すると、メンバー名を表示する前に接尾部のストリングを取り除くことができます。

メンバー名にのみこの除去は行われ、ディメンション名には行われません。さらに、引数としてメンバー名を取るプロパティーおよびメソッドは、接頭部および接尾部を除去する前に固有のメンバー名を使用します。

例

memberNameRemoveSuffix が ″##″ の場合、メンバー ″Year##978-9″ は ″Year″ として表示されます。

このプロパティーは memberNameRemovePrefix プロパティーと併用できます。たとえば、memberNameRemovePrefix ストリングが ″$$″ で memberNameRemoveSuffix ストリングが ″##″ の場合、メンバー ″123$$Year##978-9″ は ″Year″ として表示されます。

関連項目

217 ページの『memberNameRemovePrefix』

mergedDimensions

ディメンションの複数の階層を「データ・レイアウト」パネルの「その他」軸およびメンバー・フィルターにマージするかどうかを指定します。

データ・ソース

Microsoft Analysis Services、SAP BW

構文

mergedDimensions="*dimensionString*"

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
dimensionString	null	「データ・レイアウト」パネルの「その他」軸にマージするディメンションの接頭部を表すコンマ区切りのストリング。

使用法

Microsoft Analysis Services、および SAP BW は複数の階層をサポートするため、キューブ・データの代替ビューが可能です。複数の階層とは、同じ接頭部とそれに続くピリオドを共有し、接尾部は異なる名前を持つ 2 つ以上のディメンションのことです。たとえば、Time.Calendar と Time.Fiscal は 2 つの異なるディメンションですが、「論理」ディメンション (実際には存在しない) にそれらをマージすると、ユーザーが混乱することが少なくなると考えられるため、アプリケーションのユーザビリティーを強化できます。

いったん複数の階層がマージされると、それらは「データ・レイアウト」パネルの「その他」軸のユーザー・インターフェースに 1 つのディメンションとして現れます。たとえば、接頭部 “Time” を使用してすべてのディメンションをマージすることを指定すると、Time.Calendar と Time.Fiscal は「データ・レイアウト」パネルに “Time” ディメンションとして現れます。ユーザーが Time ディメンションを行軸、列軸、またはページ軸にドラッグすると、ユーザーに 2 つの階層のうち使用する 1 つを選択することを求めるダイアログが自動的にポップアップされます。メンバー・フィルターでは、Time ディメンションの下に対応するすべての階層が表示されますが、ユーザーが選択できるのは 1 つの階層からのみです。

注: MDBMetaData オブジェクトの `resolveDimension()` メソッドなどのメソッドを介してディメンションにアクセスする際、データ・ソースに実際に保管されている実際のディメンション名 (たとえば [Time].[Calendar] や [Time].[Fiscal]) を指定する必要があります。マージされたディメンションは実際にはデータ・ソースに存在しないため、マージされたディメンション名を使用するとエラーが発生します。マージされたディメンションを構成するディメンションを調べるには、`getCube().getMultipleHierarchies()` メソッドを使用します。

例

以下の例は、接頭部 “Time” を持つすべての階層を Time と呼ばれる存在しないディメンションにマージし、接頭部 “Products” を持つすべての階層を Products と呼ばれる存在しないディメンションにマージする方法を示しています。

```
mergedDimensions="Time,Products"
```

関連項目

Javadoc 内の `getCube().getMultipleHierarchies()` 。

mergedHeaders

ヘッダーがマージされる同じ軸上のディメンションを指定します。

データ・ソース

すべて

構文

```
mergedHeaders="mergedString"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
mergedString	null	コロンで区切られた *dimensionString:matchPatterns:drillableDim* ストリング。 詳しくは、以下の『使用法』を参照してください。

使用法

- *dimensionString* - このストリングは、ヘッダーがマージされるディメンションと、マージされるヘッダーの新規メンバー名を以下の形式で指定します。

  dimensionList = *newMemberName*

  dimensionList は、ヘッダーがマージされるディメンションのコンマ区切りリストです。 *newMemberName* は、マージされるヘッダーの名前です。これは、メンバー、行、またはタプルを非表示にする場合に使用する名前です (DataBlox の hiddenMembers プロパティーまたは hiddenTuples プロパティーを使用します)。デフォルトでは、マージされるヘッダーはマージされるディメンション・ヘッダーの区切り文字としてスペースを追加します。たとえば、Scenario と Measures のヘッダーがマージされる場合、新規ヘッダーは ″Scenario Measures″ のように、間にスペースが入ります。

  注: ディメンションの指定の順序は結果セットの指定の順序で同じでなければならず、連続していなければなりません。たとえば、 All Time Periods、Measures、および Scenario をこの順序で戻す照会がある場合、以下の例が有効です。

  ```
  mergedHeaders="All Time Periods, Measures, Scenario = Measures and
  Scenario by Year" mergedHeaders="Measures, Scenario = Measures &
  Scenario"
  ```

  しかし、以下は無効です。

  ```
  mergedHeaders="All Time Periods, Scenario = Scenario by Period" (ディメンションが連続していない) mergedHeaders="Measures, All Time Periods =
  Measures by Period" (順序が間違っている)
  ```

  注: 算出メンバー (calculatedMembers プロパティーを使用して指定される) は、ヘッダーをマージする前に実行されます。そのため、算出メンバー (複数の場合あり) を追加する必要があるときは、元のディメンション名を使用してください。一方、hiddenMembers と hiddenTuples は mergedHeaders の後に実行されるため、新しくマージされたヘッダーを使用する必要があります。

- *matchPatterns* - オプション。一致するヘッダー・パターンの対のコンマ区切りリストと、置き換えるヘッダー。古いヘッダーと新しいヘッダーの各対の形式は、*olderHeader* = *newHeader* です。以下の例は、Measures および All Time Periods のヘッダーをマージし、すべてのヘッダーで検出されたストリング “Qtr 1” を “Q1”、“Qtr 2” を “Q2”、“Qtr 3” を “Q3”、“Qtr 4” を “Q4” で置き換えます。

```
mergedHeaders="All Time Periods, Measures = Measures by Period:  Qtr 1 =
Q1, Qtr 2 = Q2, Qtr 3 = Q3, Qtr 4 = Q4"
```

この結果、Qtr 1 00 Sales は Q1 00 Sales となり、Qtr 1 01 Forecast は Q1 01 Forecast となります。

- *drillableDim* - オプション。ユーザーがマージされるヘッダーにドリルするときにドリル可能なディメンション。ドリル可能なディメンションが指定されないと、*dimensionString* で最初にリストされているディメンションがデフォルトでドリル可能ディメンションとなります。

  注: 一致パターンなしでドリル可能ディメンションが指定される場合、2 つのストリングを分けるコロンが引き続き組み込まれます。以下に例を示します。

  ```
  mergedHeaders="Measures, Scenario::Scenario"
  ```

  注: ヘッダーがマージされるときにドリル可能なディメンションは 1 つのみです。ドリル可能なディメンションを設定すると、*dimensionString* の他のディメンションはドリル可能ではなくなります。

例

以下の例は、ディメンション All Time Periods および Measures のヘッダーをマージします。新しくマージされるディメンションの名前は Measures by Period です。4 つのヘッダー名の置換一致パターンが指定されています。ドリル可能なディメンションは All Time Periods に設定されます (これはリストの最初のディメンションなので、指定されない場合のデフォルトでもあります) 。

```
mergedHeaders="All Time Periods, Measures= Measures by Period:
Qtr 1 = Q1, Qtr 2 = Q2, Qtr 3 = Q3, Qtr 4 = Q4:
All Time Periods"/>
```

onErrorClearResultSet

以降のデータベース操作に失敗した場合に既存の結果セットを消去するかどうかを指定します。

データ・ソース

すべて

構文

```
onErrorClearResultSet="clearResultSet"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
clearResultSet	false	結果セットを消去する場合は true を指定し、消去しない場合は false を指定します。

parentFirst

子に相対して親を戻す方法を指定します。

データ・ソース

マルチディメンション

構文

parentFirst="*parentFirst*"

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
parentFirst	照会から戻されるメンバーの順序を尊重します。	Blox タグで、子の前に親を置く場合は true を指定し、子を最初に置く場合は false を指定します。このタグ属性が指定されない場合、照会から戻されるメンバーの順序が尊重されます。 関連する Javaメソッドは、整数を取り、戻します。 setParentFirst() は以下の値をとります。 • DataBlox.PARENT_DEFAULT - 照会から戻されるメンバーの順序が尊重されます。 • DataBlox.PARENT_FIRST - 照会から戻されるメンバーの順序に関係なく、親メンバーがその子メンバーの前に置かれます。 • DataBlox.PARENT_LAST - 照会から戻されるメンバーの順序に関係なく、親メンバーがその子メンバーの後に置かれます。

使用法

Blox タグで値を true に設定すると、親を最初に置いた状態 (子の上または左) でデータを戻し、値を false に設定すると、親を最後に置いた状態 (子の下または右) でデータを戻します。 DataBlox でこの属性が指定されない場合、照会から戻されるメンバーの順序が尊重されます。 GridBlox の展開/縮小モード (expandCollapseMode ="true") を使用し、親を最初に表示する場合、parentFirst を true に設定してください。これは照会では行わないでください。これは、展開/縮小モードで、結果セットが正しく検索されるようにし、ベース・メンバーおよび共用メンバーを判別できるようにするためです。

重要: 戻されるメンバーの前または後に親メンバーを置くように設定する場合、デフォルトの順序を尊重するようにリセットすることはできません。

例

以下の例は、JSP タグと Java メソッドを使用して親メンバーが子の前に来るように設定する方法を示しています。

```
<blox:data ..
   parentFirst="true" />

<% myDataBlox.setParentFirst(DataBlox.PARENT_FIRST); %>
```

次の例は、親とその子の現行の順序を取得する方法を示しています。

```
<% String message;
   int order;
   order = myDataBlox.getParentFirst();
```

```
   if (order == myDataBlox.PARENT_FIRST) {
      message = "Parent First";
   } else if (order == myDataBlox.PARENT_LAST) {
      message = "Parent Last";
   } else message="Default Order";
   out.write("The current parent-child order is: " + message);
%>
```

password

データ・ソースのアクセスに使用するデータベース・パスワードを指定します。

データ・ソース

すべて

構文

password="*password*"

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
password	空ストリング	データ・ソースのアクセス用パスワード。

使用法

デフォルトのパスワードは、データ・ソースを DB2 Alphablox に定義したときに提供された値の 1 つです。 DataBlox に password プロパティーが指定されない場合、AASUserAuthorizationEnabled が true に設定されていなければ、値はデータ・ソース定義からとられます (この場合、ユーザーが入力したパスワードが使用されます)。

関連する setPassword() メソッドと setDataSourceName() メソッドを併用する場合、パスワードは setDataSourceName() を呼び出した後に設定する必要があります。そうしないと、DataBlox は指定されたデータ・ソースのすべてのプロパティーに接続し、以前に設定されたすべてのプロパティーをオーバーライドします。これは、この setDataSourceName() メソッドは、
username、password、catalog、schema、query などのデータ・ソースのプロパティーやページ軸上のディメンションでも読み取れるからです。そのため、Java メソッドを使用してこれらのプロパティーのいずれかを設定する場合は、それらを setDataSourceName() を呼び出した後で設定してください。

ヒント: これらのデータ・ソースのプロパティーが設定される順序は、Blox タグを使用する場合には問題となりません。タグは、他のデータ・ソース・プロパティーを設定する呼び出しの前にデータ・ソースの設定を施行するよう設計されています。副次作用は自動的に解決されます。

例

```
password="secret"
```

関連項目

208 ページの『dataSourceName』、237 ページの『userName』

performInAllGroups

ドリル操作が、ディメンションを含むネストされたそれぞれの外部グループの選択されたメンバーのすべてのオカレンスに対して実行されるか、それとも選択されたメンバーの単一のオカレンスに対してのみ実行されるかを指定します。

データ・ソース

マルチディメンション

構文

performInAllGroups="*perform*"

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
perform	true	選択されたメンバーのすべてのオカレンスにドリルを適用する場合は true を指定し、メンバーの単一のオカレンスのみドリルする場合は false を指定します。

使用法

このプロパティーが true に設定される場合でも、ドリルが行われる他のグループでもメンバー名が同じでなければなりません。たとえば、Period が Product 内にネストされているとします。 VCR の Qtr1 に対するドリルは、メンバー名が同じなので、TV の Qtr1 を拡張します。しかし、VCR および TV の Qtr2 から Qtr4 は、メンバー名が異なるので、拡張されません。

provider

データ・ソースにアクセスするためのプロバイダー・ストリングを設定します。

データ・ソース

OLE DB for OLAP のみ

構文

provider="*provider*"

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
provider	データ・ソース定義で指定されたストリング	データベースの供給業者名。たとえば、Microsoft Analysis Server の場合にはストリングは MSOLAP となります。

使用法

データ・ソースが Alphablox に対して定義されていると、OLE DB for OLAP データ・ソースのデフォルトのプロバイダー・ストリングが指定されます。DataBlox に provider プロパティーを指定しない場合、値はデータ・ソース定義から取られます。

setProvider() Java メソッドを使用する場合、setDataSourceName() を呼び出した後にプロバイダー・ストリングを設定する必要があります。そうしないと、DataBlox は指定されたデータ・ソースのすべてのプロパティーに接続し、以前に設定されたすべてのプロパティーをオーバーライドします。これは、この setDataSourceName() メソッドは、username、password、catalog、schema、query などのデータ・ソースのプロパティーやページ軸上のディメンションでも読み取れるからです。これらのデータ・ソースのプロパティーが設定される順序は、Blox タグを使用する場合には問題となりません。タグは、適切な順序になるように設計されているからです。

query

データ・ソースに渡される初期照会ストリングを指定します。

データ・ソース

すべて

構文

query="*queryString*"

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
queryString	空ストリング	データ・ソースによって理解される言語の照会ステートメント。リレーショナル・データ・ソースの場合、SQL SELECT ステートメントを使用します。マルチディメンション・データ・ソースの場合、 MicrosoftMDX または Essbase レポート仕様などの適切な言語を使用します。

使用法

getQuery() メソッドは、現行データ・ソースに対して設定された最後の照会ストリングを戻します。最後の照会以来実行されたソートやドリルなどのユーザー処置は、戻り値には反映されません。

setQuery() メソッドは、照会ストリングを設定します。 connect() メソッドが呼び出されると照会が実行されます。

例

Microsoft MDX 照会言語を使用した照会の例については、「*開発者用ガイド*」の『データの取り出し』にある『MDX ステートメント』を参照してください。MDX に関する Microsoft の特定情報については、以下の Web リンクを参照してください。

http://www.microsoft.com/data/oledb/

および

http://msdn.microsoft.com/library/techart/intromdx.htm

Essbase レポート仕様を使用した照会の例については、「*開発者用ガイド*」の『データの取り出し』にある『Essbase レポート仕様』を参照してください。特定の情報については、Essbase インストール・ディレクトリーのオンライン文書を参照してください。

¥docs¥techref¥RPTIND.HTM

関連項目

229 ページの『selectableSlicerDimensions』、DataBlox generateQuery() メソッド。

retainSlicerMemberSet

グリッドでのメンバー選択を保存するかどうかを指定します。

データ・ソース

マルチディメンション

構文

retainSlicerMemberSet="*persistMemberSelection*"

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
persistMemberSelection	true	グリッドでのメンバー選択を保持する場合、true を指定します。

使用法

true (デフォルト) のときは、グリッドでのメンバー選択が保存されて、ページ・フィルターには選択されたメンバーの子が表示されます。 false のときは、ユーザーがディメンションをページ・ディメンションと行または列ディメンションとの間で相互に移動すると、グリッドでのメンバー選択は保存されません。

rowSort

行軸上のメンバーのデータ値をソートする方法を指定します。

データ・ソース

すべて

構文

rowSort="*sortString*"

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
tuple	なし	ソートされる行を指定する行軸上のタプル。
dimension	なし	グループ化が保存される列軸上のディメンション。列軸上にグループ化が保存されないことを指定するには、null を指定します。
ascending	なし	昇順にソートする場合は true を指定し、降順にソートする場合は false を指定します。
preserveHierarchy	false	ソート操作の後にメンバーとその親を保持しつつ列軸の階層を保存する場合は true を指定し、階層を保存しない場合は false を指定します。この引数は Java メソッドでのみ有効です。
sortString	なし	以下の形式のいずれかのコンマで区切られたストリング。 • *tupleIndex*, *direction* • *tupleIndex*, *groupingNestLevel*, *direction* • *tupleIndex*, *groupingNestLevel*, *direction, preserveHierarchy* *tupleIndex* - ゼロ・ベースのソート対象タプル索引メンバー (行) を表す整数のストリング表記。0 は最上位の行を示します。 *groupingNestLevel*- グループ化が保存される列軸上のディメンションを表す整数のストリング表記。たとえば、Time および Measures が列軸上にある場合、1 は Measures ディメンション内のシーケンスにソートされます。 -1 を指定すると、列のグループ化に関係なくソートします。デフォルトは -1 です。 *direction* - "Ascending"、"Asc"、"Descending"、または "Desc" のいずれかのストリング。大/小文字を区別しません。 *preserveHierarchy* - ブールのストリング表記。preserveHierarchy 引数を参照してください。デフォルトは false になります。 以下に例を示します。 `setRowSort("1,0,asc");` `setRowSort("1,0,asc,true");` `setRowSort("0,descending");`

使用法

getRowSort メソッドは、コンマで区切られた 4 つの項目のストリング *tupleIndex*、*groupingNestLevel*、*direction*、および *preserveHierarchy* を戻します。

例

以下の例は、rowSort タグ属性の使用を示しています。

```
rowSort="1, 0, asc"
```

関連項目

203 ページの『columnSort』、Javadoc 内の DataBlox `removeRowSort()` メソッド。

schema

アクセスするスキーマの名前を指定します。

データ・ソース

すべて

構文

```
schema="schema"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
schema	空ストリング	スキーマの名前

使用法

schema の値は、データ・ソースを DB2 Alphablox に定義したときに提供された値の 1 つです。 DataBlox に schema プロパティーを指定しない場合、値はデータ・ソース定義から取られます。

IBM DB2 OLAP Server または Hyperion Essbase 用語では、スキーマは「データベース」と呼ばれます。

関連項目

202 ページの『catalog』

selectableSlicerDimensions

ページ (スライサー) 軸上に現れるディメンションを指定します。スライサー・ディメンションはデータ上でフィルターとして機能します。

データ・ソース

マルチディメンション

構文

```
selectableSlicerDimensions="dimensionString"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
dimensionString	空ストリング	固有のディメンション名のコンマで区切られたストリング。

使用法

selectableSlicerDimensions プロパティーは、すでに行または列軸に存在するディメンションには影響を与えません。これは現在「その他」(未使用) 軸上にあるディメンションに対してのみ作用します。このメソッドは、マルチディメンション・データ・ソースにのみ関係します。

showSuppressDataDialog

useOlapDrillOptimization プロパティーが true に設定されると、suppressMissingColumns または suppressMissingRows も true に設定される場合に、このプロパティーは警告ダイアログがポップアップされるかどうかを指定します。このダイアログは、ドリルダウンしてから詳細なデータ分析操作を実行すると不完全なデータが生成される可能性をユーザーに警告します。

データ・ソース

Microsoft Analysis Services、SAP BW

構文

showSuppressDataDialog="*showDialog*"

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
showDialog	true	アラート・ダイアログをポップアップする場合は true を指定し、ポップアップ・ダイアログを抑制する場合は false を指定します。

使用法

このプロパティー (Blox タグ、Java メソッドを介してアセンブラーによって設定されるか Blox ユーザー・インターフェースを介してユーザーによって設定される) および useOlapDrillOptimization プロパティーがどちらも true に設定されると、ユーザーには一部のデータしか表示されない場合があります。これは、ユーザーがドリルダウンしてページ・フィルターの変更、ドリルアップ、またはメンバー・フィルターの使用など、他のアクションを実行するときに発生します。この一連のユーザー処置をとると、ダイアログがポップアップ表示され、ユーザーにこのことが発生する可能性があることを警告し、「欠落抑制」をオフにするように推奨します。 showSuppressDataDialog プロパティーが false に設定されると、ダイアログはポップアップされません。

関連項目

236 ページの『useOlapDrillOptimization』、231 ページの『suppressMissingColumns』、232 ページの『suppressMissingRows』

suppressDuplicates

重複したヘッダー値を含む行または列をグリッドから除去するかどうかを指定します。

データ・ソース
マルチディメンション

構文
suppressDuplicates="*suppress*"

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
suppress	true	重複したヘッダー値を抑制する場合は true を指定し、そのままにする場合は false を指定します。

使用法
IBM DB2 OLAP Server または Hyperion Essbase 結果セットの重複した共有メンバーを抑制するには、レポート・スクリプト照会で SUPSHARE コマンドを使用します。このコマンドについての詳細は、IBM DB2 OLAP Server または Hyperion Essbase の文書を参照してください。

関連項目
231 ページの『suppressMissingColumns』、232 ページの『suppressMissingRows』、233 ページの『suppressNoAccess』、233 ページの『suppressZeros』

suppressMissingColumns
データをまったく含まない列をグリッドから除去するかどうかを指定します。

データ・ソース
マルチディメンション

構文
suppressMissingColumns="*suppress*"

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
suppress	false	データを含まない列を抑制する場合は true を指定し、そのままにする場合は false を指定します。

使用法
値を持たないセルで表示するものを指定するには、GridBlox 上で missingValueString プロパティーを使用します。

データ・ソースが Microsoft Analysis Services または SAP BW の場合、このプロパティーと useOlapDrillOptimization プロパティーを併用する際には注意が必要です。両方のプロパティーが true に設定されると、ユーザーがドリルダウンしてペ

ージ・フィルターの変更、ドリルアップ、またはメンバー・フィルターの使用など、他のアクションを実行するときに一部のデータしか表示されない場合があります。詳しくは、236 ページの『useOlapDrillOptimization』を参照してください。

IBM DB2 OLAP Server または Hyperion Essbase 結果セットの重複した共有メンバーを抑制するには、レポート・スクリプト照会で SUPSHARE コマンドを使用します。このコマンドについての詳細は、IBM DB2 OLAP Server または Hyperion Essbase の文書を参照してください。

関連項目

232 ページの『suppressMissingRows』、230 ページの『suppressDuplicates』、233 ページの『suppressNoAccess』、233 ページの『suppressZeros』

suppressMissingRows

データをまったく含まない行をグリッドから除去するかどうかを指定します。

データ・ソース

マルチディメンション

構文

```
suppressMissingRows="suppress"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
suppresss	false	データを含まない行を抑制する場合は true を指定し、そのままにする場合は false を指定します。

使用法

値を持たないセルで表示するものを指定するには、GridBlox 上で missingValueString プロパティーを使用します。

データ・ソースが Microsoft Analysis Services または SAP BW の場合、このプロパティーと useOlapDrillOptimization プロパティーを併用する際には注意が必要です。両方のプロパティーが true に設定されると、ユーザーがドリルダウンしてページ・フィルターの変更、ドリルアップ、またはメンバー・フィルターの使用など、他のアクションを実行するときに一部のデータしか表示されない場合があります。詳しくは、236 ページの『useOlapDrillOptimization』を参照してください。

IBM DB2 OLAP Server または Hyperion Essbase 結果セットの重複した共有メンバーを抑制するには、レポート・スクリプト照会で SUPSHARE コマンドを使用します。このコマンドについての詳細は、IBM DB2 OLAP Server または Hyperion Essbase の文書を参照してください。

関連項目

231 ページの『suppressMissingColumns』、232 ページの『suppressMissingRows』、230 ページの『suppressDuplicates』、233 ページの『suppressNoAccess』、233 ページの『suppressZeros』

suppressNoAccess

ユーザーがアクセスできないデータを含む行または列をグリッドから除去するかどうかを指定します。

データ・ソース

マルチディメンション

構文

```
suppressNoAccess="suppress"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
suppress	false	ユーザーがアクセスできないデータを抑制する場合は true を指定し、そのままにする場合は false を指定します。

使用法

IBM DB2 OLAP Server または Hyperion Essbase 結果セットの重複した共有メンバーを抑制するには、レポート・スクリプト照会で SUPSHARE コマンドを使用します。このコマンドについての詳細は、IBM DB2 OLAP Server または Hyperion Essbase の文書を参照してください。

関連項目

230 ページの『suppressDuplicates』、231 ページの『suppressMissingColumns』、232 ページの『suppressMissingRows』、233 ページの『suppressZeros』

suppressZeros

すべてがゼロの行または列をグリッドから除去するかどうかを指定します。

データ・ソース

マルチディメンション

構文

```
suppressZeros="suppress"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
suppress	false	すべてがゼロの列または行を抑制する場合は true を指定し、そのままにする場合は false を指定します。

使用法

IBM DB2 OLAP Server または Hyperion Essbase 結果セットの重複した共有メンバーを抑制するには、レポート・スクリプト照会で SUPSHARE コマンドを使用します。このコマンドについての詳細は、IBM DB2 OLAP Server または Hyperion Essbase の文書を参照してください。

関連項目

230 ページの『suppressDuplicates』、231 ページの『suppressMissingColumns』、232 ページの『suppressMissingRows』、233 ページの『suppressNoAccess』

textualQueryEnabled

逐次化照会ではなくテキスト形式の照会を使用してデータ照会をリストアすることを指定します。

データ・ソース

すべて

構文

```
textualQueryEnabled="textualQuery"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
textualQuery	false	テキスト形式の照会ファイルに保管されている照会ストリングを使用してデータ照会をロードするかどうかを指定します。デフォルトでは、ブックマークは逐次化照会を使用してロードされます。

使用法

最初にブックマークが追加されると、DataBlox の照会セットと現行データ・ビューを生成する照会の間の差分 が、<bookmark_name>.data テキスト・ファイルおよび <bookmark_name>.query ファイルに保管されます。これらのファイルは、照会を逐次化オブジェクトとして保管します。その後ブックマークがロードされる際、このプロパティーが true に設定されていなければ逐次化照会が使用されます。これは、データ一括表示やメンバー名の変更があり、それに従ってテキスト形式の照会を変更する場合に役に立ちます。 DB2 Alphablox では結果セットと逐次化オブジェクトを一致させるための操作は必要ないので、テキスト形式の照会の操作はさらに効果的です。

ただし、テキスト形式の照会は、ブックマークが異なるデータ・ビューとともに再保管される際に更新されません。テキスト形式の照会を使用する必要がある場合には、テキスト形式の照会を最新のものにすることもできます。この場合、ブックマーク保管イベントを取り込み、DataBlox `generateQuery()` メソッドを使用して現行のテキスト形式の照会を取得して、ブックマークの照会を更新することができます。これには、`addEventFilter()` 共通 Blox メソッドを使用して `BookmarkSaveFilter` インターフェースをインプリメントするメソッドを追加することも必要となります。

関連項目

64 ページの『逐次化照会とテキスト形式の照会』、63 ページの『ブックマーク・イベントとイベント・フィルター』、Javadoc 内の DataBlox `generateQuery()` メソッド。

useAASUserAuthorizationEnabled

IBM DB2 OLAP Server または Hyperion Essbase データ・ソースに対する認証のために DB2 Alphablox のログイン時に入力されるユーザー名およびパスワードを使用するかどうか指定します。

データ・ソース

IBM DB2 OLAP Server、Hyperion Essbase

構文

`useAASUserAuthorizationEnabled="`*`useIt`*`"`

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
`useIt`	`false`	IBM DB2 OLAP Server または Hyperion Essbase の認証で DB2 Alphablox ログイン情報を使用する場合は `true` を指定し、通常の認証プロセスを使用する場合は `false` を指定します。

使用法

このプロパティーは、外部 Web サーバー・セキュリティーを使用せずに DB2 Alphablox をスタンドアロン構成で使用する場合のみ有効です。

`true` に設定すると、データ・ソースはユーザーがデータ・ソースへのアクセスのためにログインした際に入力される値を使用します。 `false` に設定すると、データ・ソースは通常の認証プロセスを使用します。

useAliases

行見出しまたは列見出しで別名またはデータベース・メンバー値を使用するかどうかを指定します。

データ・ソース

IBM DB2 OLAP Server、Hyperion Essbase

構文

useAliases="*useAliases*"

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
useAliases	false	別名を使用する場合は true を指定し、メンバー名を使用する場合は false を指定します。

使用法

データベース・メンバー値は通常コードで (たとえば 001 から 200)、別名は名前です (Diet Cola など)。

このプロパティーは IBM DB2 OLAP Server または Hyperion Essbase レポート・スクリプトの {OUTALTNAMES} コマンドの使用をオーバーライドします。

useOlapDrillOptimization

Microsoft Analysis Services および SAP BW データ・ソースでドリルの最適化を使用可能にするかどうかを指定します。

データ・ソース

Microsoft Analysis Services、SAP BW

構文

useOlapDrillOptimization="*optimize*"

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
optimize	true	照会の最適化を使用する場合は true を指定します。

使用法

デフォルトでは、より高い照会パフォーマンスを発揮するために、このプロパティーは Microsoft Analysis Services または SAP BW データ・ソースで true に設定されています。このプロパティーを suppressMissing プロパティーと併用する際には注意が必要です。両方のプロパティーが true に設定されると、ユーザーがドリルダウンしてページ・フィルターの変更、ドリルアップ、またはメンバー・フィルターの使用など、他のアクションを実行するときに一部のデータしか表示されない場合があります。この一連のユーザー処置をとると、ダイアログがポップアップ表示され、ユーザーにこのことが発生する可能性があることを警告し、「欠落抑制」をオフにするように推奨します。 このダイアログは showSuppressDataDialog プロパティーを使用してオフにできます。

関連項目

230 ページの『showSuppressDataDialog』、231 ページの『suppressMissingColumns』、232 ページの『suppressMissingRows』

userName

データ・ソースのアクセスに使用するデータベース・ユーザー名を指定します。

データ・ソース

すべて

構文

```
userName="userName"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
userName	空ストリング	データベースのユーザー名。

使用法

デフォルトのユーザー名は、データ・ソースを DB2 Alphablox に定義したときに提供された値の 1 つです。 DataBlox に userName プロパティーを指定しない場合、値はデータ・ソース定義から取られます。

関連する setUserName() メソッドと setDataSourceName() メソッドを併用する場合、ユーザー名は setDataSourceName() を呼び出した後に設定する必要があります。そうしないと、DataBlox は指定されたデータ・ソースのすべてのプロパティーに接続し、以前に設定されたすべてのプロパティーをオーバーライドします。これは、この setDataSourceName() メソッドは、username、password、catalog、schema、query などのデータ・ソースのプロパティーやページ軸上のディメンションでも読み取れるからです。そのため、Java メソッドを使用してこれらのプロパティーのいずれかを設定する場合は、それらを setDataSourceName() を呼び出した後で設定してください。

ヒント: これらのデータ・ソースのプロパティーが設定される順序は、Blox タグを使用する場合には問題となりません。タグは、他のデータ・ソース・プロパティーを設定する呼び出しの前にデータ・ソースの設定を施行するよう設計されています。副次作用は自動的に解決されます。

関連項目

208 ページの『dataSourceName』

第 11 章 DataLayoutBlox タグ・リファレンス

この章には DataLayoutBlox の参照資料が含まれています。 Blox についての一般的な参照情報は、21 ページの『第 3 章 一般 Blox リファレンス情報』を参照してください。このリファレンスの使用方法については、1 ページの『第 1 章 このリファレンスの使用法』を参照してください。

DataLayoutBlox の概説

DataLayoutBlox は、マルチディメンション・データベースのディメンションをグラフィカルに表現します。これはマルチディメンション・データベースのディメンションが表示される軸 (行、列、またはページ) によって編成されます。一方、4 番目の軸は、他のどの軸にも現れない (しかし現行の結果セットで使用可能な) ディメンションをリストします。

グラフィカル・ユーザー・インターフェース

DataLayoutBlox GUI を使用すると、以下のタスクを実行できます。

- ディメンションの表示と軸の間の移動
- GridBlox との間のディメンションのドラッグ・アンド・ドロップ
- メンバー・フィルターへのアクセス

DataLayoutBlox には、ツリーおよびドロップ・リストという 2 つのインターフェース・タイプがあります。デフォルトはツリーです。ツリー・インターフェースには展開縮小可能なツリー・メニューが含まれており、ドラッグ・アンド・ドロップ操作をサポートします。ドロップ・リスト・インターフェースでは、軸の間のディメンションの移動をサポートするためにドロップ・リストが使用されます。

DataLayoutBlox ユーザー・インターフェースの使用に関する説明は、DataLayoutBlox ユーザー・ヘルプを参照してください。ユーザー・ヘルプにアクセスするには、Blox ユーザー・インターフェースにあるツールバーの「ヘルプ」ボタンをクリックします。

DataLayoutBlox の JSP カスタム・タグ構文

DB2 Alphablox タグ・ライブラリーは、それぞれの Blox を作成するために JSP ページで使用するカスタム・タグを提供します。このセクションでは、カスタム・タグを作成して DataLayoutBlox を作成する方法を説明します。すべての属性を含むタグのコピー・アンド・ペースト・バージョンについては、524 ページの『DataLayoutBlox JSP カスタム・タグ』を参照してください。

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

ここで、それぞれ以下のとおりです。

attribute　　属性表にリストされている属性の 1 つです。

value　　属性の有効な値です。

属性は以下のいずれかになります。

属性
id
applyPropertiesAfterBookmark
bloxEnabled
bloxName
bookmarkFilter
height
helpTargetFrame
hiddenDimensionsOnOtherAxis
interfaceType
localeCode
maximumUndoSteps
noDataMessage
render
visible
width

使用法

各カスタム・タグには 1 つ以上の属性を含めることができ、それぞれを 1 つ以上のスペースまたは改行文字で区切ります。余分のスペースまたは改行文字は無視されます。読みやすくするため、同じインデントでそれぞれ別々の行に属性を並べることができます。

終了タグ </blox:dataLayout> は、省略表現を使用して置き換えられます。以下のようなタグを使用して属性リストを閉じることができます。

```
width="650" />
```

例

```
<blox:dataLayout
     id="namedDataLayoutBlox"
     width="100"
     height="400" />
```

DataLayoutBlox タグ属性

このセクションでは、DataLayoutBlox で使用可能なタグ属性をアルファベット順にリストします。共通 Blox プロパティーの詳しい説明は、36 ページの『複数の Blox に共通するタグ属性』を参照してください。 DataLayoutBlox メソッドの参照情報は、Javadoc 内にある com.alphablox.blox パッケージの DataLayoutBlox クラスを参照してください。

id

これは共通の Blox プロパティーです。詳しい説明は、43 ページの『id』を参照してください。

applyPropertiesAfterBookmark

これは共通の Blox プロパティーです。詳しい説明は、36 ページの『applyPropertiesAfterBookmark』を参照してください。

bloxEnabled

これは共通の Blox プロパティーです。詳しい説明は、38 ページの『bloxEnabled』を参照してください。

bloxName

これは共通の Blox プロパティーです。詳しい説明は、39 ページの『bloxName』を参照してください。

bookmarkFilter

これは共通の Blox プロパティーです。詳しい説明は、37 ページの『bookmarkFilter』を参照してください。

height

これは共通の Blox プロパティーです。詳しい説明は、42 ページの『height』を参照してください。

helpTargetFrame

これは共通の Blox プロパティーです。詳しい説明は、42 ページの『helpTargetFrame』を参照してください。

hiddenDimensionsOnOtherAxis

「データ・レイアウト」パネルの「その他」軸で非表示にされるディメンションを指定します。

データ・ソース

マルチディメンション

構文

```
hiddenDimensionsOnOtherAxis="dimensionsToHide"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
dimensionsToHide	ヌル	「その他」軸で非表示にするディメンションを表すコンマ区切りのストリング。

使用法

デフォルトでは、データ照会で指定されていないキューブ内のディメンションはいずれも「データ・レイアウト」パネルの「その他」軸に置かれます。「その他」軸のディメンションを非表示にすることによって、ユーザーがこれらのディメンションに対してピボット操作を行ったり、これらのディメンションで現在選択されているメンバーを変更したりするのを防げます。たとえば、IBM DB2 OLAP Server ディメンションまたは Hyperion Essbase Attribute ディメンションを非表示にしたり、ユーザーが Measures フィルターを変更するのを防止したりすることもできます。あるいは、 Microsoft Analysis Services データ・ソースに複数の階層 [Time].[Fiscal] および [Time].[Calendar] を作ることもできます。照会ではすでに「列」軸に [Time].[Fiscal] が指定されているため、混乱を避けるために「その他」軸の [Time].[Calendar] を非表示にする必要があります。

ディメンションは UI で非表示となるだけで、データには引き続き存在します。照会において、たとえばディメンションが「列」軸に表示されるように指定され、それを「その他」軸で再び非表示にする場合、ディメンションは引き続き「列」軸で表示されます。ただし、いったんユーザーがディメンションを「その他」軸にピボットすると、それ以降非表示となります。ディメンションを再び表示するには、hiddenDimensionsOnOtherAxis プロパティーを再びリセットする必要があります。

例

以下の例は、「その他」軸上から [Time].[Calendar]、Measures、および [Attribute Calcs] の各ディメンションを非表示にします。

```
hiddenDimensionsOnOtherAxis="[Time].[Calendar], Measures,
[Attribute Calcs]"
```

interfaceType

「データ・レイアウト」パネルのインターフェース・タイプを設定します。

データ・ソース

マルチディメンション

構文

```
interfaceType="type"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
type	tree	有効な値は tree および dropList です。 tree に設定すると、「データ・レイアウト」パネルに、展開縮小が可能な 3 つのメニューを持つツリー・ナビゲーション・インターフェースが現れます。 239 ページの『DataLayoutBlox の概説』を参照してください。

localeCode

これは共通の Blox プロパティーです。詳しい説明は、43 ページの『localeCode』を参照してください。

maximumUndoSteps

これは共通の Blox プロパティーです。詳しい説明は、44 ページの『maximumUndoSteps』を参照してください。

noDataMessage

これは共通の Blox プロパティーです。詳しい説明は、45 ページの『noDataMessage』を参照してください。

render

これは共通の Blox プロパティーです。詳しい説明は、47 ページの『render』を参照してください。

visible

これは共通の Blox プロパティーです。詳しい説明は、49 ページの『visible』を参照してください。

width

これは共通の Blox プロパティーです。詳しい説明は、49 ページの『width』を参照してください。

第 12 章 GridBlox タグ・リファレンス

この章には、GridBlox タグおよびタグ属性の参照資料が含まれています。Blox についての一般的な参照情報は、21 ページの『第 3 章 一般 Blox リファレンス情報』を参照してください。このリファレンスの使用方法については、1 ページの『第 1 章 このリファレンスの使用法』を参照してください。

GridBlox の概説

GridBlox のユーザー・インターフェースは、データ・セルの行および列で構成される表データ表示域と、オプションのグリッド・コントロールで構成されています。ユーザーは、行、列、およびページ軸上のマルチディメンション・データを表示できます。ユーザーは、データ階層間のドリルアップおよびドリルダウンによるデータ表示の操作、データ・ディメンションの別の軸への移動、データ・ディメンションの組み込みおよび省略などを実行できます。

GridBlox は、リレーショナル・データ・ソースとマルチディメンション・データ・ソースの両方のデータを表示します。リレーショナル・データは 2 次元の行と列の形式で表示されます。マルチディメンション・データは対話式のマルチディメンション・グリッド形式で表示されるため、ユーザーは多次元分析を実行できます。DB2 Alphablox には、リレーショナル・データをマルチディメンション・キューブにトランスフォームするキューブ・サーバーが組み込まれているため、GridBlox はデータをマルチディメンション形式で表示できます。

注: 多次元分析について詳しくは、「*管理者用ガイド*」の『OLAP の用語および概念』を参照してください。

DHTML クライアントは、GridBlox を `<blox:grid>` タグに `id` を指定した HTML エレメントとして表示します。HTML エレメント内の各グリッドは `DIV` エレメントであり、通常 `DIV` に関連付けられるすべての属性、メソッド、およびイベントを持っています。さらに、現在選択されているセルを表す `Selection` オブジェクトが使用可能です。DHTML Client API についての詳細は、「*開発者用ガイド*」を参照してください。

GridBlox JSP カスタム・タグ構文

DB2 Alphablox タグ・ライブラリーは、それぞれの Blox を作成するために JSP ページで使用するカスタム・タグを提供します。ここでは、GridBlox を作成するためにカスタム・タグを作成する方法について説明します。すべての属性を含むタグのコピー・アンド・ペースト・バージョンについては、525 ページの『GridBlox JSP カスタム・タグ』を参照してください。

構文

```
<blox:grid
   [attribute="value"] >
      [<blox:cellAlert
           [attribute="value"] />]
      [<blox:cellEditor
           [attribute="value"] />]
      [<blox:cellFormat
           [attribute="value"] />]
      [<blox:cellLink
           [attribute="value"] />]
      [<blox:drillThroughWindow
           [attribute="value"] />]
      [<blox:editableCellStyle
           [attribute="value"] />]
      [<blox:editedCellStyle
           [attribute="value"] />]
      [<blox:formatMask
           [attribute="value"] />]
      [<blox:formatName
           [attribute="value"] />]
</blox:grid>
```

ここで、それぞれ以下のとおりです。

attribute 属性表にリストされている属性の 1 つです。

value 属性の有効な値です。

属性は以下のいずれかになります。

<blox:grid> タグ
属性
id
autosizeEnabled
applyPropertiesAfterBookmark
bandingEnabled
bloxEnabled
bloxName
bookmarkFilter
columnHeadersWrapped
columnWidths
commentsEnabled
defaultCellFormat
drillThroughEnabled
drillThroughWindow
editableCellStyle
editedCellStyle
enablePoppedOut
expandCollapseMode
gridLinesVisible
headingsEnabled

<blox:grid> タグ
属性
height
helpTargetFrame
informationWindowName
localeCode
maximumUndoSteps
menubarVisible
missingValueString
noAccessValueString
noDataMessage
poppedOut
poppedOutHeight
poppedOutTitle
poppedOutWidth
relationalRowNumbersOn
removeAction
render
rightClickMenuEnabled
rowHeadersWrapped
rowHeadingsVisible
rowHeadingWidths
rowHeight
rowIndentation
showColumnDataGeneration
showColumnHeaderGeneration
showRowDataGeneration
showRowHeaderGeneration
toolbarVisible
visible
width
writebackEnabled

<blox:cellAlert> ネスト・タグ **254 ページの『cellAlert』を参照。**
属性
index
apply
background
condition
description

<blox:cellAlert> ネスト・タグ **254 ページの『cellAlert』を参照。**
属性
enabled
font
foreground
format
group
link
image_align
image
scope
value
value2

<blox:cellEditor> ネスト・タグ **261 ページの『cellEditor』を参照。**
属性
index
scope

<blox:cellFormat> ネスト・タグ **264 ページの『cellFormat』を参照。**
属性
index
background
font
foreground
format
group
scope

<blox:cellLink> ネスト・タグ **267 ページの『cellLink』を参照。**
属性
index
description
link
scope
image_align

<blox:cellLink> ネスト・タグ **267 ページの『cellLink』を参照。**
属性
image

<blox:drillThroughWindow> ネスト・タグ **275 ページの『drillThroughWindow』を参照。**
属性
height
locationbarVisible
menubarVisible
name
resizable
scrollbarsVisible
statusbarVisible
toolbarVisible
url
width

<blox:editableCellStyle> ネスト・タグ **277 ページの『editableCellStyle』を参照。**
属性
background
font
foreground

<blox:editedCellStyle> ネスト・タグ **278 ページの『editedCellStyle』を参照。**
属性
background
font
foreground

<blox:formatMask> ネスト・タグ **280 ページの『formatMask』を参照。**
属性
index
mask

<blox:formatName> ネスト・タグ **281 ページの『formatName』を参照。**
属性
index
name

使用法

各カスタム・タグには 1 つ以上の属性を含めることができ、それぞれを 1 つ以上のスペースまたは改行文字で区切ります。余分のスペースまたは改行文字は無視されます。読みやすくするため、同じインデントでそれぞれ別々の行に属性を並べることができます。

ネストされたタグ (<blox:cellAlert> または <blox:cellStyle> タグなど) がない場合は、終了 </blox:grid> タグを省略表現で置換し、以下のようにタグを属性リストの末尾で終了することができます。

```
width="650" />
```

ネストされたタグがある場合は、省略表現は無効となり、終了タグが必要となります。

例

```
<blox:grid id="myGrid"
     height="400"
     width="500"
     bandingEnabled="true" />

<blox:grid id="anotherGrid"
     height="300"
     width="500"
     bandingEnabled="true">
     <blox:cellAlert index="1"
          condition="any"
          background="cyan" />
</blox:grid>
```

カテゴリー別の GridBlox タグ属性

このセクションでは、GridBlox に固有のタグ属性をリストします。複数の Blox に共通するタグ属性については、35 ページの『カテゴリー別の共通の Blox タグ属性』を参照してください。 GridBlox でサポートされるタグ属性は、以下のように相互参照として編成されています。

- 251 ページの『グリッドの外観』
- 251 ページの『数値のフォーマット』
- 252 ページの『セル・アラート』
- 252 ページの『リレーショナル詳細データへのドリル』
- 252 ページの『印刷』
- 252 ページの『書き戻しおよびコメント用のグリッド UI』
- 252 ページの『ポップアウトのプロパティー』

GridBlox メソッドについては、Javadoc 内にある com.alphablox.blox パッケージの GridBlox クラスを参照してください。DHTML クライアントのクライアント・サイド API については、「*開発者用ガイド*」を参照してください。

グリッドの外観

以下のタグ属性は、グリッドがページに表示される方法に影響を与えます。

- bandingEnabled
- cellLink
- columnHeadersWrapped
- columnWidths
- expandCollapseMode
- gridLinesVisible
- informationWindowName
- menubarVisible
- missingValueString
- noAccessValueString
- relationalRowNumbersOn
- removeAction
- rightClickMenuEnabled
- rowHeadersWrapped
- rowHeadingsVisible
- rowHeadingWidths
- rowHeight
- rowIndentation
- showColumnDataGeneration
- showColumnHeaderGeneration
- showRowDataGeneration
- showRowHeaderGeneration

数値のフォーマット

以下のタグ属性は、数値がグリッド・データ域に表示される方法を定義します。使用可能なフォーマットについての詳細は、以下を参照してください。

http://java.sun.com/j2se/1.4.2/docs/api/java/text/DecimalFormat.html

- cellFormat
- defaultCellFormat
- formatMask
- formatName
- localeCode

セル・アラート

以下の表には、セル・アラートに関連するタグ属性が表示されています。

- cellAlert

リレーショナル詳細データへのドリル

ドリルスルー操作は、IBM DB2 OLAP Server、IBM DB2 OLAP Server Deployment Services、Hyperion Essbase、または Essbase Deployment Services のデータ・ソースでサポートされています。これらのデータ・ソースには、Essbase Integration Services (EIS) によって設定されたドリルスルー・レポートがあります。この機能は、Microsoft Analysis Services でもサポートされています。

- drillThroughEnabled
- drillThroughWindow

印刷

印刷タグ属性は、グリッドが配信用に印刷形式でレンダリングされるときにグリッドに適用されます。

- defaultCellFormat
- headingsEnabled

書き戻しおよびコメント用のグリッド UI

書き戻し UI タグ属性は、ユーザーがデータ・セル値を変更するのを許可する条件を GridBlox データ域に設定します。コメント UI タグ属性 (commentsEnabled) は、 1) コメントを追加および表示するためのメニュー項目をグリッド・セルの右クリック・メニューに表示するかどうか、2) セル・コメントが使用可能な場合にコメント標識を右上隅に表示するかどうかを指定します。これらのプロパティーは、253 ページの『GridBlox タグ属性』 と共に使用されます。

- cellEditor
- commentsEnabled
- editableCellStyle
- editedCellStyle
- writebackEnabled

ポップアウトのプロパティー

以下の表は、GridBlox を別のポップアウト・ブラウザー・ウィンドウに表示することに関連したタグ属性をリストしています。

- enablePoppedOut
- poppedOut
- poppedOutHeight
- poppedOutTitle
- poppedOutWidth

GridBlox タグ属性

このセクションでは、GridBlox によってサポートされるタグ属性をアルファベット順に説明します。GridBlox メソッドの参照情報については、Javadoc の com.alphablox.blox パッケージの GridBlox クラスを参照してください。 GridBlox で使用可能な共通 Blox プロパティーについては、リストされていますが説明はありません。共通 Blox タグ属性の詳しい説明は、36 ページの『複数の Blox に共通するタグ属性』を参照してください。

id

これは共通の Blox タグ属性です。詳しい説明は、43 ページの『id』を参照してください。

applyPropertiesAfterBookmark

これは共通の Blox プロパティーです。詳細記述は、36 ページの『applyPropertiesAfterBookmark』を参照してください。

autosizeEnabled

最大のデータ値が収まるよう列を自動的にサイズ変更するかどうかを指定します。

データ・ソース

すべて

構文

```
autosizeEnabled="autosize"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
autosize	true	列および行を columnWidths、rowHeadingWidths、および rowHeight の設定に基づいてレンダリングするには、false を指定します。

例

```
autosizeEnabled = "true"
```

bandingEnabled

グリッド行の代替背景色を使用可能にするかどうかを指定します。

データ・ソース

すべて

構文

```
bandingEnabled="enable"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
enabled	true	セル・バンディングを使用可能にするには true、使用不可にするには false を指定します。

使用法

アプリケーションのデフォルト・レンダリング・モードが DHTML に設定されている場合、デフォルトは true です。

例

```
bandingEnabled = "true"
```

bloxEnabled

これは共通の Blox プロパティーです。詳しい説明は、38 ページの『bloxEnabled』を参照してください。

bloxName

これは共通の Blox プロパティーです。詳しい説明は、39 ページの『bloxName』を参照してください。

bookmarkFilter

これは共通の Blox プロパティーです。詳しい説明は、37 ページの『bookmarkFilter』を参照してください。

cellAlert

数値データ・セル内の値を強調表示するための規則を指定します。

データ・ソース

すべて

構文

```
<blox:cellAlert
      index="cellAlertNumber"
      apply="row|column|cell"
      background="background"
      condition="condition"
      description="description"
      enabled="enabled"
      font="font"
      foreground="foreground"
      format="formatmask"
      group="groupName"
      image="image"
      image_align="left|right|center"
      link="link"
      scope="scope"
      value="value1"
      value2="value2"
/>
```

ここで、それぞれ以下のとおりです。

属性	必須か?	説明
apply	いいえ	強調表示するグリッド対象の領域です。可能な値は以下のとおりです。 • ROW: 行内で条件を満たすセルがあれば行全体を強調表示します。最初の行適用アラートは、他の行適用アラートに優先します。 • COLUMN: 列内で条件を満たすセルがあれば列全体を強調表示します。最初の列適用アラートは、他の列適用アラートに優先します。 • CELL: 条件を満たす特定のセルだけを強調表示します (デフォルト)。セル適用アラートは、行適用アラートまたは列適用アラートをオーバーライドします。
background	いいえ	セルの背景色です。色の名前または 16 進値を使用します。
condition	はい	セル値に適用する条件です。条件を満たすセルはアラートをアクティブにします。可能な値は以下のとおりです。 • ANY: 任意の値のセルを受け入れます。 • MISSING: 値が欠落しているセルだけを受け入れます。 • NA: 値が使用不可のセルだけを受け入れます。 以下の可能な値は、value および value2 属性と連携して機能します。 • LT または < • GT または > • EQ または = • LTEQ または <= • GTEQ または >= • BETWEEN (*value*, *value2*): 値は *value* 以上で、*value2* 以下です。
description	いいえ	セル・アラートの説明です。この説明は、ユーザーがマウス・ポインターをセルの上に移動したときにツール・ヒントとして表示されます。
enabled	いいえ	セル・アラートをアクティブにするかどうかを指定します。セル・アラートを使用不可にするには、false に設定します。デフォルトは true です。
font	いいえ	セルのテキストに使用する *font name:style:point* です。 • *font name*: 受け入れ可能なフォント名の値は、ブラウザーおよびクライアント・マシンによってさまざまに異なります。一般に受け入れられるフォント名には Arial、Courier、Helvetica、TimesRoman、SansSerif、 Serif、Monospace などがあります。 • *style*: 有効なフォント・スタイルは plain、italic、bold、および bolditalic です。 • *Point*: ポイント・サイズの整数 (通常は 8 から 36)。 3 つの属性のうちのいずれかが指定されていない場合、デフォルトの、または現行の継承されたフォント値が適用されます。ただし、属性を分離するコロンを含める必要があります。例えば、以下のようにします。 `font="Arial:bolditalic:12"` `font=":Bold:12"`
foreground	いいえ	セルのテキストの色です。色の名前または 16 進値を使用します。
format	いいえ	セル・データに適用するフォーマット・マスクです。詳しくは、272 ページの『defaultCellFormat』を参照してください。
group	いいえ	トラフィック・ライト・グループとして扱うセル・アラートのグループ名です。グループ名が同じセル・アラートは、トラフィック・ライトの作成および管理ユーザー・インターフェースでトラフィック・ライトの 1 セットとして表示されます。

属性	必須か?	説明
image	いいえ	定義されたリンクを指すカスタム・イメージです。 リンクを定義してもカスタム・イメージを指定しない場合は、セルの内容がリンクとして表示されます。ただし、カスタム・イメージを指定してリンクを定義していない場合は、イメージが表示されます。 イメージの URL は、絶対パスと相対パスのどちらでも構いません。 • 絶対 URL の場合、ストリングは「http://」で開始する必要があります。 • 相対 URL の場合: – ストリングをスラッシュ (/) で始めると、URL がサーバー・ルートに対して相対であることを示します。アプリケーション・コンテキストを URL に含める必要があるということに注意してください。 – ストリングをスラッシュ (/) なしで始めると、URL が現行の文書に対して相対であることを示します。
image_align	いいえ	image 属性によって指定されるカスタム・イメージの位置を設定します。有効な値は以下のとおりです。 • LEFT: イメージをセル・コンテンツの前に配置します (デフォルト) • RIGHT: イメージをセル・コンテンツの後ろに配置します。
link	いいえ	HTML レンダリングのハイパーリンクまたは JavaScript メソッドに対する呼び出しです。 URL にリンクしている場合は、常に新規ウィンドウが開かれます。 リンク値の以下のエンティティーは、リンクの生成時に示されている値で置き換えられます。エンティティーはアンパーサンド (&) で始まり、セミコロン (;) で終了します。 • &Description; - セル・アラートの説明 • &Value; - セル値 • &ColHeader; - アンパーサンドで区切られたディメンション/メンバー値の組 • &RowHeader; - アンパーサンドで区切られたディメンション/メンバー値の組 • &ColIndex; - グリッド内の列の表示インデックス (0 ベース) • &RowIndex; - グリッド内の行の表示インデックス (0 ベース) 以下の例では、セル・アラートのリンクは次のように設定されています。 `link="decoderequest.jsp?row=&RowHeader;&column=` `&ColHeader;&value=&Value;&rowIndex=&RowIndex;&col` `Index=&ColIndex;"` Time ディメンションの 3 番目の行の 1234.55 という値のセルと、Product ディメンションの 4 番目の列がクリックされると、パススルーされる URL は以下のようになります。 `decoderequest.jsp?row=Time=Q3&column=Product=Chocolate%20` `Nuts&value=1234.55&rowIndex=2&colIndex=3`

属性	必須か?	説明
scope	いいえ	アラートを適用するセルで、一連のディメンションおよびメンバー・セットを中括弧で囲んで指定します。固有の名前を使用して、正しいメンバーが検出されるようにしてください。Essbase では、DataBlox の useAliases が false に設定されている場合は (ユーザーはユーザー・インターフェースを介してこれを設定できる)、表示名を使用しても機能しません。MSAS および Alphablox Cube データ・ソースでは、固有の名前を使用して、正しいメンバーが正しいレベルで検出されるようにしてください。SAP BW の場合には、必ず固有の名前を使用します。表示名を使用すると、データ例外またはランダム・データになります。 有効範囲は、行軸および列軸だけでなく結果セットのすべての軸に適用されます。有効範囲は、以下のように指定します。 `scope="{d0:m00[,m01,... m0n]} {d1:m10[,m11,...m1n]}..."` ここで、d0 はディメンション、m00 はそのディメンション内のメンバーを表します。たとえば、Essbase データ・ソースでは、以下のようにします。 `scope={Product:Coke} {Scenario: Actual, Budget}` MSAS、Alphablox Cube、および SAP BW データ・ソースの場合は、以下のように固有の名前を使用します。 `scope="{[My Cube].[Product]:[Code]} {[My Cube].[Scenario]: [Actual], [Budget]}"` ディメンション名は大括弧で囲み、さらにキューブ名 (MSAS、Alphablox Cube) または照会名 (SAP BW) を含める必要があります。ディメンションおよびメンバー名の指定方法について詳しくは、以下の『使用法』セクション内の注記を参照してください。 セル・アラートを適用するメンバーのレベルを指定する場合は、以下の検索関数が使用可能です。 • Leaf(): 指定されたメンバーのリーフ・レベルの子孫を検索します。関数に指定できるメンバーは 1 つだけです。例: scope="{Market:leaf(East)}" (Essbase) • Child(): 指定されたメンバーの子を検索します。関数に指定できるメンバーは 1 つだけです。例: scope="{Market:child(East)}" (Essbase) • Descendants(): 指定のメンバーの子孫すべて。関数に指定できるメンバーは 1 つだけです。例: scope="{[My Cube].[Market]:descendants([East])}" (MSAS、Alphablox Cube、SAP BW) • Gen(): 指定された世代のすべてのメンバーを検索します。例: scope="{Market:gen(2)}" (Essbase) • Not(): セル・アラートを適用しないメンバーを検索します。複数のメンバーをコンマで分離して指定できます。例: scope="{[My Cube].[Market]:not([East], [West])}" (MSAS、Alphablox Cube、SAP BW) 関数名は大文字小文字を区別しません。scope ステートメントでは、関数を結合できます。
value	いいえ	condition が LT、GT、EQ、LTEQ、GTEQ の場合は比較の対象となる値で、condition が BETWEEN の場合は小さい方の値です。
value2	いいえ	condition が BETWEEN の場合は大きいほうの値です。
index	いいえ	定義する cellAlert の番号です。この属性を指定しない場合は、次に使用可能な cellAlert 番号が使用されます。例えば、cellAlerts 1 から 4 が定義済みの場合は、cellAlert 5 が使用されます。

使用法

セル・アラートのフォーマットは、内容が数値の場合にのみ適用されます。セル・アラートの番号は、セル・アラートが評価される順番を指示します。各データ・セル値は、最初の cellAlert (index="1") から定義されている最も大きいアラート番号まで、定義されているすべてのアラートに対して評価されます。データ・セルの値および有効範囲と一致する最初のセル・アラートだけが、そのセルに適用されます。オーバーラップがある場合は、セル・アラートを適切な順序で定義してください。

- 特定の属性を指定しない場合は、デフォルト値がセルに適用されます。例えば、デフォルトのセル背景が白で、背景属性を指定しない場合は、セルの背景は白のままです。
- 編集可能セルの場合、グリッドが最初に表示されるときに、セル・アラートのフォーマットがセル・エディターのフォーマットに優先します。セルを編集すると、セル・エディターの色設定がセル・アラートによって指定される色設定に優先します。
- 有効範囲に指定するディメンションおよびメンバー名ストリングには、固有の名前 (IBM DB2 OLAP Server または Hyperion Essbase のベース名) または表示名を使用してください。固有の名前を使用すると、同じ表示名を持つ異なるメンバーまたはディメンションを区別できます。 IBM DB2 OLAP Server または Hyperion Essbase では、別名表に関係なく、ベース名を使用することによりメンバーを指定できます。
- MSAS および Alphablox Cube データ・ソースの場合、有効範囲を指定する際、以下のようにしてください。
  - ディメンション名を大括弧で囲む場合、キューブ名を含めてください。たとえば、″[My cube].[Market]″ となります。別の方法としては、ディメンション名を大括弧なしで指定します。たとえば、″Market″ となります。
  - メンバー名を大括弧で囲むか、キューブから始めるメンバーの完全修飾された固有名を使用してください。たとえば、″[California]″ または ″[My Cube].[Market].[West].[California]″ となります。メンバー名を大括弧で囲まないと、名前はすべて大文字のメンバーとして扱われます。

例

- 値を表示する前に値を 1000 で除算します。

```
<blox:cellAlert index="1"
                condition="any"
                format="#,###/1000;[red](#,###/1000)"
                background="#3333FF"
                scope="{Scenario: Budget}" />
```

  MSAS、Alphablox Cube、および SAP BW の場合、scope は次のようにします。

```
scope="{[My Cube].[Scenario]: [Budget]}
```

- リテラル文字として記号 (この場合は % 記号) を使用します。

```
<blox:cellAlert index="3"
                condition="any"
                format="#'%';[red](#'%')"
                background="#9999FF"
                scope="{Scenario: Variance %}" />
```

MSAS、Alphablox Cube、および SAP BW の場合、scope は次のようにします。scope="{[My Cube].[Scenario]: [Variance%]}

- 値を表示する前に値に 100 を乗算し、フォントは Times New Roman、太字、14 ポイントのサイズを使用します。

```
<blox:cellAlert index="4"
                condition="any"
                format="#.##*100%;[red](#.##*100%)"
                background="#CCCCFF"
                scope="{Scenario: Variance %}"
                font="Times New Roman:bold:14" />
```

- 以下の例は、PresentBlox に表示されるグリッドにセル・アラート 2 を設定します。このアラートは、Market ディメンションの Central メンバーの値が 1000 より大きいかどうかをテストします。データ値がこの基準を満たす場合は、以下のようになります。
  - 前景 (フォント) の色が緑になる。
  - 背景色が白になる。
  - 値が右に位置合わせされる。

```
PresentBlox.getGridBlox().setCellAlert(2, "condition=GT, value=1000,
scope={Market:Central}, foreground=green, background=white, align=right");
```

MSAS および SAP BW の場合、scope は次のようにします。scope="{[My Cube].[Market]:[Central]}

- 以下の例は、すべての数値内容をシアンの背景で強調表示します。

```
<blox:cellAlert index="1"
                condition="any"
                background="cyan" />
```

これは、数値データを含むすべてのセルにシアンの背景を適用します。セルにデータが含まれていない場合は、この背景色は適用されません。

- 一連の関連したセル・アラートにより、増加するセルの量が異なる色で表示されます。

```
<blox:cellAlert index="1"
                 condition="Between" value="1000" value2="3000"
                 format="00.00"
                 foreground="Orange"
                 scope="{[My Cube].[Market]:[East],[West],[South],[Central]}" />

<blox:cellAlert index="2"
                 condition="Between" value="3001" value2="5000"
                 scope="{[My Cube].[Year]:[Qtr1],[Qtr2]}"
                 format="00.00"
                 foreground="Blue" />

<blox:cellAlert index="3"
                 condition="GT"
                 value="5000"
                 format="00.00"
                 foreground="Magenta" />

<blox:cellAlert index="4"
                condition="LT"
                value="1000"
                format="(00.00)"
                foreground="Red" />
```

その結果、以下のようなグリッドが表示されます。

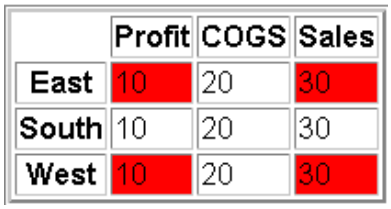
Year	Product	East	West	South	Central	Market
Qtr1	Colas	2747.00	1042.00	1051.00	2208.00	7048.00
	Root Beer	(562.00)	2325.00	1465.00	2369.00	6721.00
	Cream Soda	(591.00)	2363.00	(561.00)	2414.00	5929.00
	Fruit Soda	1480.00	1407.00		2118.00	5005.00
	Diet Drinks	(555.00)	2025.00	1146.00	3291.00	7017.00
	Product	5380.00	7137.00	3077.00	9109.00	24703.00
Qtr2	Colas	3352.00	(849.00)	1198.00	2473.00	7872.00
	Root Beer	(610.00)	2423.00	1540.00	2457.00	7030.00
	Cream Soda	(922.00)	2739.00	(529.00)	2579.00	6769.00
	Fruit Soda	1615.00	1504.00		2317.00	5436.00
	Diet Drinks	(652.00)	1975.00	1289.00	3420.00	7336.00
	Product	6499.00	7515.00	3267.00	9826.00	27107.00

有効範囲の例

以下の表の例では `background=red` および `condition=any` という前提で、異なる有効範囲を使用して結果を示しています。

注: `scope` 属性は、行軸および列軸だけでなく結果セットのすべての軸に適用されます。例えば、グリッドが accounts ディメンションの profit メンバーでフィルタリングされる場合 (profit メンバーが選択された状態で accounts ディメンションがページ・フィルターに配置される)、グリッドは以下の最初の例に示されているように表示されます。

scope の例	**結果**
以下の scope は、accounts ディメンションの profit メンバーにのみ適用されます。 `scope="{Accounts: Profit}"`	<table><tr><th></th><th>Profit</th><th>COGS</th><th>Sales</th></tr><tr><td>East</td><td>10</td><td>20</td><td>30</td></tr><tr><td>South</td><td>10</td><td>20</td><td>30</td></tr><tr><td>West</td><td>10</td><td>20</td><td>30</td></tr></table>
以下の scope は、accounts ディメンションの profit および sales メンバーに適用されます。 `scope="{Accounts: Profit, Sales}"` これは、profit OR sales という意味であることに注意してください。この scope は、以下のように指定することもできます。 `scope="{Accounts: not(COGS)}"`	<table><tr><th></th><th>Profit</th><th>COGS</th><th>Sales</th></tr><tr><td>East</td><td>10</td><td>20</td><td>30</td></tr><tr><td>South</td><td>10</td><td>20</td><td>30</td></tr><tr><td>West</td><td>10</td><td>20</td><td>30</td></tr></table>

scope の例	結果
以下の scope は、accounts ディメンションの profit および sales メンバーが、 market ディメンションの east および west メンバーと交差する場合に適用されます。これは AND 演算です。 `scope="{Accounts: Profit, Sales}, {Market: East, West}"` または `scope="{Accounts: not(COGS)}, {Market: not(South)}"`	

関連項目

261 ページの『cellEditor』、264 ページの『cellFormat』、267 ページの『cellLink』、Blox API Javadoc 内の `listCellAlertIds()` メソッド、252 ページの『セル・アラート』

cellEditor

データ・セルの編集可能域を定義および強調表示するための規則を指定します。

データ・ソース

SAP BW 以外すべて

構文

```
<blox:cellEditor
     index="cellEditorNumber"
     scope="scope" >
</blox:cellEditor>
```

ここで、それぞれ以下のとおりです。

属性	必須か?	説明
scope	はい	エディターを適用するセルで、一連のディメンションおよびメンバー・セットを中括弧で囲んで指定します。固有の名前を使用して、正しいメンバーが検出されるようにしてください。Essbase では、DataBlox の useAliases が false に設定されている場合は (ユーザーはユーザー・インターフェースを介してこれを設定できる)、表示名を使用しても機能しません。MSAS および Alphablox Cube データ・ソースでは、固有の名前を使用して、正しいメンバーが正しいレベルで検出されるようにしてください。 有効範囲は、行軸および列軸だけでなく結果セットのすべての軸に適用されます。有効範囲は、以下のように指定します。 `scope="{d0:m00[,m01,... m0n]} {d1:m10[,m11,...m1n]}..."` ここで、d0 はディメンション、m00 はそのディメンション内のメンバーを表します。たとえば、 Essbase では、以下のようにします。 `scope="{Product:Coke} {Scenario: Actual, Budget}"` MSAS および Alphablox Cube では、以下のようになります。 `scope="{[My Cube].[Product]:[Code]}` `{[My Cube].[Scenario]:[Actual],[Budget]}"` • ディメンション名を大括弧で囲む場合、キューブ名を含める必要があります。別の方法としては、ディメンション名を大括弧なしで指定します。たとえば、″Market″ または ″[My Cube].[Market]″ となります。 • メンバー名を大括弧で囲むか、キューブから始めるメンバーの完全修飾された固有名を使用してください。たとえば、″[California]″ または ″[My Cube].[Market].[West].[California]″ となります。メンバー名を大括弧で囲まないと、名前はすべて大文字のメンバーとして扱われます。 エディターを適用するメンバーのレベルを指定する場合は、以下のメンバー検索関数が使用可能です。 • Leaf(): 指定されたメンバーのリーフ・レベルの子孫を検索します。関数に指定できるメンバーは 1 つだけです。例: `scope="{Market:leaf(East)}"` (Essbase) • Child(): 指定されたメンバーの子を検索します。関数に指定できるメンバーは 1 つだけです。例: `scope="{Market:child(East)}"` (Essbase) • Descendants(): 指定のメンバーの子孫すべて。関数に指定できるメンバーは 1 つだけです。例: `scope="{[My Cube].[Market]:descendants([East])}"` (MSAS) • Gen(): 指定された世代のすべてのメンバーを検索します。例: `scope="{Market:gen(2)}"` (Essbase) • Not(): セル・エディターを適用しないメンバーを検索します。複数のメンバーをコンマで分離して指定できます。例: `scope="{[My Cube].[Market]:not([East], [West])}"` (MSAS) 関数名は大文字小文字を区別しません。scope ステートメントでは、関数を結合できます。別の例については、260 ページの 『有効範囲の例』を参照してください。

属性	必須か?	説明
index	いいえ	注: この属性は cellEditor タグ属性としてのみ有効です。setCellEditor Java メソッドの場合は、代わりに id 引数を使用してください。 定義するセル・エディターの番号です。この属性を指定しない場合は、次に使用可能なセル・エディター番号が使用されます。例えば、セル・エディター 1 から 4 が定義済みの場合は、セル・エディター 5 が使用されます。

使用法

scope 属性を使用して、グリッド内の編集可能セルの領域を定義します。

セル・エディターを活動化するには、writebackEnabled を true に設定する必要があります。

グリッド・ユーザー・インターフェースで、非数値セルを編集可能にできます。ただし、非数値の値は、値が欠落しているものとして書き戻されます。IBM DB2 OLAP Server または Hyperion Essbase (数値の値だけが書き戻される) とは異なり、Microsoft Analysis Services では、どのデータ・タイプでも書き戻されます。有効範囲を指定する場合は、非数値のセルが "#MISSING" ストリングで上書きされないよう注意する必要があります。リレーショナル・データ・ソースについては、DB2 Alphablox はデータの書き戻しを実行しません。変更されたセルのリストおよびセルの新規の値をプログラマチックに取得し、JDBC を使用して値を書き戻す必要があります。このアプローチの利点は、非数値データを書き戻せることです。

グリッドが最初に表示されるときに、セル・アラートのフォーマットがセル・エディターのフォーマットに優先します。セルを編集すると、セル・エディターの色設定がセル・アラートによって指定される色設定に優先します。

例

```
<blox:cellEditor
   index="3"
     scope="scope={[My Cube].[Market]: [East]}" />
```

以下の例は、PresentBlox に表示されるグリッドにセル・エディター 2 を設定します。 Central メンバーの値以外の Market ディメンション内の任意の値が編集可能です。

```
<blox:present id="myPresent" >
   <blox:data bloxRef="aPreDefinedDataBlox" />
   <blox:grid>
      <blox:cellEditor
         index="2"
         scope="scope={[My Cube].[Market]:not([Central])" />
   </blox:grid>
</blox:present>
```

関連項目

254 ページの『cellAlert』、264 ページの『cellFormat』、267 ページの『cellLink』、Javadoc 内にある GridBlox clearCellEditors() および listCellEditorIds() メソッド

cellFormat

グリッドのセル内のデータ値のフォーマットを指定します。

データ・ソース

すべて

構文

```
<blox:cellFormat
      index="cellFormatNumber"
      background="background"
      font="font"
      foreground="foreground"
      format="formatmask"
      group="group"
      scope="scope" >
</blox:cellFormat>
```

ここで、それぞれ以下のとおりです。

属性	必須か?	説明
background	いいえ	セルの背景色です。色の名前または 16 進値を使用します。
font	いいえ	セルのテキストに使用する font name:style:point です。 • *font name*: 受け入れ可能なフォント名の値は、ブラウザーおよびクライアント・マシンによってさまざまに異なります。一般に受け入れられるフォント名には Arial、Courier、Helvetica、TimesRoman、SansSerif、Serif、Monospace などがあります。 • *style*: 有効なフォント・スタイルは plain、italic、bold、および bolditalic です。 • *Point*: ポイント・サイズの整数 (通常は 8 から 36)。 3 つの属性のうちのいずれかが指定されていない場合、デフォルトの、または現行の継承されたフォント値が適用されます。ただし、属性を分離するコロンを含める必要があります。例えば、以下のようにします。 font="Arial:bolditalic:12" font=":Bold:12"
foreground	いいえ	セルのテキストの色です。色の名前または 16 進値を使用します。
format	はい	セル・データに適用するフォーマット・マスクです。詳しくは、272 ページの『defaultCellFormat』を参照してください。
group	いいえ	セットと同じ名前のセル・フォーマットをグループ化するネーム・スペースです。

属性	必須か?	説明
scope	はい	フォーマットを適用するセルで、一連のディメンションおよびメンバー・セットを中括弧で囲んで指定します。固有の名前を使用して、正しいメンバーが検出されるようにしてください。Essbase では、DataBlox の useAliases が false に設定されている場合は (ユーザーはユーザー・インターフェースを介してこれを設定できる)、表示名を使用しても機能しません。MSAS および Alphablox Cube データ・ソースでは、固有の名前を使用して、正しいメンバーが正しいレベルで検出されるようにしてください。SAP BW の場合には、必ず固有の名前を使用します。表示名を使用すると、データ例外またはランダム・データになります。 有効範囲は、行軸および列軸だけでなく結果セットのすべての軸に適用されます。有効範囲は、以下のように指定します。 `scope="{d0:m00[,m01,... m0n]} {d1:m10[,m11,...m1n]}..."` ここで、d0 はディメンション、m00 はそのディメンション内のメンバーを表します。たとえば、 Essbase では、以下のようにします。 `scope="{Product:Coke} {Scenario: Actual, Budget}"` MSAS、Alphablox Cube、および SAP BW の場合、以下のようになります。 `scope="{[My Cube].[Product]:[Code]}` `{[My Cube].[Scenario]: [Actual], [Budget]}"` ディメンション名は大括弧で囲み、さらにキューブ名 (MSAS、Alphablox Cube) または照会名 (SAP BW) を含める必要があります。ディメンションおよびメンバー名の指定方法について詳しくは、以下の『使用法』セクション内の注記を参照してください。 フォーマットを適用するメンバーのレベルを指定する場合は、以下のメンバー検索関数が使用可能です。 • Leaf(): 指定されたメンバーのリーフ・レベルの子孫を検索します。関数に指定できるメンバーは 1 つだけです。例: `scope="{Market:leaf(East)}"` (Essbase) • Child(): 指定されたメンバーの子を検索します。関数に指定できるメンバーは 1 つだけです。例: `scope="{Market:child(East)}"` (Essbase) • Descendants(): 指定のメンバーの子孫すべて。関数に指定できるメンバーは 1 つだけです。例: `scope="{[My Cube].[Market]:descendants([East])}"` (MSAS) • Gen(): 指定された世代のすべてのメンバーを検索します。例: `scope="{Market:gen(2)}"` (Essbase) • Not(): セル・フォーマットを適用しないメンバーを検索します。複数のメンバーをコンマで分離して指定できます。例: `scope="{[My Cube].[Market]:not([East], [West])}"` (MSAS) 関数名は大文字小文字を区別しません。scope ステートメントでは、関数を結合できます。別の例については、260 ページの 『有効範囲の例』を参照してください。

属性	必須か?	説明
index	いいえ	注: この属性は cellFormat タグ属性としてのみ有効です。setCellFormat Java メソッドの場合は、代わりに id 引数を使用してください。 定義するセル・フォーマットの番号です。この属性を指定しない場合は、次に使用可能なセル・フォーマット番号が使用されます。例えば、セル・フォーマット 1 から 4 が定義済みの場合は、セル・フォーマット 5 が使用されます。

使用法

cellFormat タグ属性は、セル・アラートと組み合わせて使用できます。

セル・フォーマットに関しては、以下の点に注意してください。

- 特定のプロパティーを指定しない場合は、デフォルト値がセルに適用されます。
- フォーマット・ストリングにバックスラッシュ・エスケープ文字 (\) を使用しないでください。
- % 記号などの記号を表示する場合は単一引用符を使用してください。表示する記号が二重引用符の場合は、バックスラッシュ・エスケープ文字 (\) を先頭に置きます。
- セル・フォーマット番号 (*N*) は、フォーマット・マスクを評価する順序を指示します。各データ・セル値は、cellFormat1 から開始し、定義されているすべてのマスクに対して順番に評価されます。後の cellFormat が前のものとオーバーラップする場合は、後の方が適用されます。つまり、最も大きい id を持つセル・フォーマット (Java メソッド) または index (JSP タグ) が優先するということです。オーバーラップがある場合は、セル・フォーマット・マスクを正しい順序で定義してください。
- scope に指定するディメンションおよびメンバー名ストリングには、固有の名前 (IBM DB2 OLAP Server または Hyperion Essbase のベース名) または表示名を使用できます。固有の名前を使用すると、同じ表示名を持つ異なるメンバーまたはディメンションを区別できます。 IBM DB2 OLAP Server または Hyperion Essbase では、別名表に関係なく、ベース名を使用することによりメンバーを指定できます。
- MSAS および Alphablox Cube データ・ソースの場合、有効範囲を指定する際、以下のようにしてください。
  - ディメンション名を大括弧で囲む場合、キューブ名を含めてください。たとえば、″[My cube].[Market]″ となります。別の方法としては、ディメンション名を大括弧なしで指定します。たとえば、″Market″ となります。
  - メンバー名を大括弧で囲むか、キューブから始めるメンバーの完全修飾された固有名を使用してください。たとえば、″[California]″ または ″[My Cube].[Market].[West].[California]″ となります。メンバー名を大括弧で囲まないと、名前はすべて大文字のメンバーとして扱われます。
- IBM DB2 OLAP Server または Hyperion Essbase データ・ソースの場合は、{DECIMAL} レポート・スクリプト・コマンドではなく、defaultCellFormat または cellFormat を使用して、データ値のフォーマットを制御します。

例

- Accounts ディメンションの Profit メンバー、および Product ディメンションの TV および Video メンバーのすべてのセル値に、小数点以下 2 桁で百と千の位がコンマで区切られたフォーマットを適用します。セル値が 999.99 以下の場合は、セル値を赤で表示します。

```
<blox:cellFormat
                  index="1"
                  format="#,###.##; [red]###.##"
                   scope="{Accounts:Profit}{Product:TV, Video}" />
```

- Accounts ディメンションの COGS メンバーを除くすべてのセル値に、千の位を区切るフォーマットを適用します。

```
<blox:cellFormat
                  index="2"
                  format="#,###K"
                  scope="{Accounts:not(COGS)}" />
```

- Accounts ディメンションの Total メンバーのすべてのセル値に、ドル記号を含む整数のドルに丸められ、セルに特定の背景色およびフォント・スタイルを指定するフォーマットを適用します。

```
<blox:cellFormat
                  index="4"
                  format="$#,##0"
                  scope="{Accounts:Total}"
                  font="Arial:Bold:20"
                  background="#CCCCFF" />
```

- Accounts ディメンションの COGS および Total メンバーのすべてのセル値を小数点以下 2 桁のパーセントとしてフォーマットし、パーセントが 1% より小さい場合は、数値を .55% ではなく 0.55 % のように表示します。

```
<blox:cellFormat
                  index="5"
                  format="0.##'%'"
                  scope="{Accounts:COGS, Total}" />
```

関連項目

272 ページの『defaultCellFormat』、254 ページの『cellAlert』、261 ページの『cellEditor』、267 ページの『cellLink』、GridBlox `listCellFormatIds()` メソッド

cellLink

リンクを含むセルを定義するための規則を指定します。

データ・ソース

すべて

構文

```
<blox:cellLink
      index="cellLinkNumber"
      description="description"
      image="image"
      image_align="left|right|center"
      link="link"
      scope="scope" >
</blox:cellLink>
```

ここで、それぞれ以下のとおりです。

属性	必須か?	説明
description	いいえ	セル・リンクの説明です。
image	いいえ	定義されたリンクを指すカスタム・イメージです。リンクを定義してもカスタム・イメージを指定しない場合は、セルの内容がリンクとして表示されます。ただし、カスタム・イメージを指定してリンクを定義していない場合は、イメージが表示されます。 イメージの URL は、絶対パスと相対パスのどちらでも構いません。 • 絶対 URL の場合、ストリングは「http://」で開始する必要があります。 • 相対 URL の場合: – ストリングをスラッシュ (/) で始めると、URL がサーバー・ルートに対して相対であることを示します。アプリケーション・コンテキストを URL に含める必要があるということに注意してください。 – ストリングをスラッシュ (/) なしで始めると、URL が現行の文書に対して相対であることを示します。
image_align	いいえ	image 属性によって指定されるカスタム・イメージの位置を設定します。有効な値は以下のとおりです。 • LEFT: イメージをセル・コンテンツの前に配置します (デフォルト) • RIGHT: イメージをセル・コンテンツの後ろに配置します。
link	はい	HTML レンダリングのハイパーリンクまたは JavaScript メソッドに対する呼び出しです。 URL にリンクしている場合は、常に新規ウィンドウが開かれます。 **注:** 絶対 URL と相対 URL のどちらでも構いません。 • 絶対 URL の場合、ストリングは「http://」で開始する必要があります。 • 相対 URL の場合: – ストリングをスラッシュ (/) で始めると、URL がサーバー・ルートに対して相対であることを示します。アプリケーション・コンテキストを URL に含める必要があるということに注意してください。 – ストリングをスラッシュ (/) なしで始めると、URL が現行の文書に対して相対であることを示します。 リンク値の以下のエンティティーは、リンクの生成時に示されている値で置き換えられます。エンティティーはアンパーサンド (&) で始まり、セミコロン (;) で終了します。 • &Description; - セル・アラートの説明 • &Value; - セル値 • &ColHeader; - アンパーサンドで区切られたディメンション/メンバー値の組 • &RowHeader; - アンパーサンドで区切られたディメンション/メンバー値の組 • &ColIndex; - グリッド内の列の表示インデックス (0 ベース) • &RowIndex; - グリッド内の行の表示インデックス (0 ベース) 置換変数が処理される方法の例については、このプロパティーの例のセクションを参照してください。

属性	必須か?	説明
scope	いいえ	リンクを適用するセルで、一連のディメンションおよびメンバー・セットを中括弧で囲んで指定します。固有の名前を使用して、正しいメンバーが検出されるようにしてください。Essbase では、DataBlox の useAliases が false に設定されている場合は (ユーザーはユーザー・インターフェースを介してこれを設定できる)、表示名を使用しても機能しません。MSAS および Alphablox Cube データ・ソースでは、固有の名前を使用して、正しいメンバーが正しいレベルで検出されるようにしてください。 SAP BW の場合には、必ず固有の名前を使用します。表示名を使用すると、データ例外またはランダム・データになります。 有効範囲は、行軸および列軸だけでなく結果セットのすべての軸に適用されます。有効範囲は、以下のように指定します。 `scope="{d0:m00[,m01,... m0n]} {d1:m10[,m11,...m1n]}..."` ここで、d0 はディメンション、m00 はそのディメンション内のメンバーを表します。たとえば、 Essbase では、以下のようにします。 `scope="{Product:Coke} {Scenario: Actual, Budget}"` MSAS、Alphablox Cube、および SAP BW の場合、以下のようになります。 `scope="{[My Cube].[Product]:[Code]}` `{[My Cube].[Scenario]: [Actual], [Budget]}"` ディメンション名は大括弧で囲み、さらにキューブ名 (MSAS、Alphablox Cube) または照会名 (SAP BW) を含める必要があります。ディメンションおよびメンバー名の指定方法について詳しくは、以下の『使用法』セクション内の注記を参照してください。 リンクを適用するメンバーのレベルを指定する場合は、以下のメンバー検索関数が使用可能です。 • Leaf(): 指定されたメンバーのリーフ・レベルの子孫を検索します。関数に指定できるメンバーは 1 つだけです。例: `scope="{Market:leaf(East)}"` (Essbase) • Child(): 指定されたメンバーの子を検索します。関数に指定できるメンバーは 1 つだけです。例: `scope="{Market:child(East)}"` (Essbase) • Descendants(): 指定のメンバーの子孫すべて。関数に指定できるメンバーは 1 つだけです。例: `scope="{[My Cube].[Market]:descendants([East])}"` (MSAS) • Gen(): 指定された世代のすべてのメンバーを検索します。例: `scope="{Market:gen(2)}"` (Essbase) • Not(): セル・リンクを適用しないメンバーを検索します。複数のメンバーをコンマで分離して指定できます。例: `scope="{[My Cube].[Market]:not([East], [West])}"` (MSAS) 関数名は大文字小文字を区別しません。scope ステートメントでは、関数を結合できます。別の例については、260 ページの 『有効範囲の例』を参照してください。
index	いいえ	注: この属性は cellLink タグ属性としてのみ有効です。setCellLink Java メソッドを使用する場合は、代わりに id 引数を使用してください。 定義するセル・リンクの番号です。この属性を指定しない場合は、次に使用可能なセル・リンク番号が使用されます。例えば、セル・リンク 1 から 4 が定義済みの場合は、セル・リンク 5 が使用されます。

使用法

cellLink プロパティーでは、HTML レンダリングの定義済みのハイパーリンクまたは JavaScriptメソッドに対する呼び出しをセルが指す条件を指定できます。セル・リンクの番号は、それが評価される順番を指示します (最初の cellLink (index="1") から開始します)。セルの条件および有効範囲と一致する最初に定義されたセル・リンクだけが、そのセルに適用されます。セル・リンクを定義するときには、オーバーラップの可能性を忘れずに考慮してください。

- 特定のプロパティーを指定しない場合は、デフォルト値がセルに適用されます。
- cellLink を使用して定義されたリンクは、cellEditor を使用して定義された編集可能セルに表示されます。
- cellAlert を使用して定義されたリンクは、cellLink で定義されたリンクに優先します。特定のセルで、リンクを含む cellAlert と cellLink の両方が定義されていて、しかも両方のパラメーターの条件が true の場合は、cellAlert リンクが使用されます。cellAlert の条件が true でない場合は、cellLink が使用されます。
- scope に指定するディメンションおよびメンバー名ストリングには、固有の名前 (IBM DB2 OLAP Server または Hyperion Essbase のベース名) または表示名を使用できます。固有の名前を使用すると、同じ表示名を持つ異なるメンバーまたはディメンションを区別できます。 IBM DB2 OLAP Server または Hyperion Essbase では、別名表に関係なく、ベース名を使用することによりメンバーを指定できます。
- MSAS および Alphablox Cube データ・ソースの場合、有効範囲を指定する際、以下のようにしてください。
  - ディメンション名を大括弧で囲む場合、キューブ名を含めてください。たとえば、″[My cube].[Market]″ となります。別の方法としては、ディメンション名を大括弧なしで指定します。たとえば、″Market″ となります。
  - メンバー名を大括弧で囲むか、キューブから始めるメンバーの完全修飾された固有名を使用してください。たとえば、″[California]″ または ″[My Cube].[Market].[West].[California]″ となります。メンバー名を大括弧で囲まないと、名前はすべて大文字のメンバーとして扱われます。

例

以下の例は、セル・リンクを {Market:Central} のすべてのセルにセル値とは関係なく追加します。セル・リンク標識は、セルの内容の左側に表示されます。ユーザーがリンクをクリックすると、ページ「www.ibm.com」が新規ブラウザー・ウィンドウに表示されます。

```
<blox:cellLink
   index="3"
   scope="{[Cube].[Market]:[Central]}"
   description="Cells with the DB2 Alphablox link"
   link="http://www.ibm.com"
   image="myIcon.gif"
   image_align="left" />
```

以下の例では、セル・アラートのリンクは次のように設定されています。

```
link="decoderequest.jsp?row=&RowHeader;&column=&ColHeader;&value=&Value;
&rowIndex=&RowIndex;&colIndex=&ColIndex;"
```

ユーザーが以下のグリッドの Q3、John Bob のセルをクリックすると、

Time	Hank	Jack	Mary Lou	John Bob
Q1		*(i)***115551.471**	14025.051	82578.896
Q2	13135.487	*(i)***117395.421**	34878.844	39445.495
Q3	*(i)***191617.066**	93620.337	51401.572	*(i)***111110.831**
Q4	24777.37	84440.596	#No Access	44903.466

パススルーされる URL は、以下のとおりです。

```
decoderequest.jsp?row=Time=Q3&column=Customer=John%20Bob&value=111110.831&
rowIndex=2&colIndex=3
```

関連項目

254 ページの『cellAlert』、261 ページの『cellEditor』、264 ページの『cellFormat』

columnHeadersWrapped

グリッド列見出しの長い見出しを複数行に折り返すかどうかを指定します。

データ・ソース

すべて

構文

```
columnHeadersWrapped="wrapped"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
wrapped	false	列見出しの折り返しを使用可能にするには true を指定します。

使用法

このタグ属性を false (デフォルト) に設定すると、列の幅は折り返しなしで列見出しテキスト全体が収まるサイズに設定されます。このプロパティーを true に設定すると、列見出しテキストは折り返され、列の幅が小さくなります。列の幅は、見出しの最も長い語およびデータ・セルの最も長いデータが収まるよう自動的に決定されます。

関連項目

287 ページの『rowHeadersWrapped』

columnWidths

グリッドの列の幅を指定します。

データ・ソース

すべて

構文

```
columnWidths="widths"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
widths	ヌル	列の幅をピクセル単位で定義するコンマで区切られた整数のストリングです。

使用法

このプロパティーを使用するには、autosizeEnabled プロパティーを false に設定する必要があります。ブラウザーは、最も長いデータ値が収まる列の幅を自動的に決定します。列の幅を明示的に設定する場合は、列に表示する値が定義された幅より大きい場合、その値は無視されます。

関連項目

253 ページの『autosizeEnabled』

commentsEnabled

1) コメントを追加および表示するためのメニュー項目をグリッドの右クリック・メニューに表示するかどうか、2) セル・コメントが使用可能な場合にコメント標識を右上隅に表示するかどうかを指定します。

データ・ソース

マルチディメンション

構文

```
commentsEnabled = "boolean"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
enabled	false	「コメント」メニュー項目と、サブメニュー項目の「コメントの追加」および「コメントの表示」をグリッド・セルの右クリック・メニューでユーザーに対して使用可能にするかどうかを指定します。false に設定すると、これらの選択項目は表示されず、コメントがセルに関連付けられている場合でもセルの右上隅のコメント標識 (赤の三角形) は表示されません。

関連項目

159 ページの『第 8 章 CommentsBlox タグ・リファレンス』

defaultCellFormat

グリッドのすべてのデータ値のデフォルト・フォーマット・マスクを指定します。

データ・ソース

すべて

構文

```
defaultCellFormat="mask"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
mask	空ストリング	セル・フォーマット属性を定義するストリングです。フォーマット・ストリングについて詳しくは、以下の例を参照してください。

使用法

defaultCellFormat によって指定されるフォーマットは、他のスタイル・フォーマットが存在しない場合 (cellFormat プロパティーとユーザー・インターフェースのいずれによっても指定されていない場合) に適用されます。

defaultCellFormat プロパティーに関しては、以下の点に注意してください。

- フォーマット・ストリングにバックスラッシュ・エスケープ文字 (\) を使用しないでください。
- % 記号などの記号を表示する場合は単一引用符を使用してください。表示する記号が二重引用符の場合は、バックスラッシュ・エスケープ文字 (\) を先頭に置きます。
- 一部の仮想マシン (Sun 1.1.6 など) では、番号マスク (#,###.00) は正のマスクと負のマスクで同じでなければなりません。同じでない場合は、負のマスクは正しく機能しません。
- IBM DB2 OLAP Server または Hyperion Essbase データ・ソースの場合は、{DECIMAL} レポート・スクリプト・コマンドではなく、defaultCellFormat または cellFormat を使用して、データ値のフォーマットを制御します。

例

- 赤の負の数: #,###.00;[red]-#,###.00

  ```
  defaultCellFormat="#,###.00;[red]-#,###.00"
  ```
- 括弧で囲んだ赤の負の数: #,###.00;[red](#,###.00)

  ```
  defaultCellFormat="#,###.00;[red](#,###.00)"
  ```
- 百万単位: #,###M

  ```
  defaultCellFormat="#,###M"
  ```
- 千単位: #,###K

  ```
  defaultCellFormat="#,###K"
  ```
- 小数点以下 2 桁のパーセント: ###.00'%'

  ```
  defaultCellFormat="###.00'%'"
  ```
- 6 桁のゼロを埋め込んで整数を表示: 000000

  ```
  defaultCellFormat="000000"
  ```
- 小数点以下 2 桁を表示 (基礎となる値の精度とは無関係): #,###.00

  ```
  defaultCellFormat="#,###.00"
  ```

関連項目

264 ページの『cellFormat』、280 ページの『formatMask』

drillThroughEnabled

GridBlox ユーザー・インターフェースでドリルスルー操作を使用可能にするかどうかを指定します。

データ・ソース

IBM DB2 OLAP Server、Hyperion Essbase、Microsoft Analysis Services

構文

```
drillThroughEnabled="drillThroughEnabled"
```

使用法

IBM DB2 OLAP Server、IBM DB2 OLAP Server Deployment Services、Hyperion Essbase、または Essbase Deployment Services の場合、このメソッドは IBM DB2 OLAP Server Integration Services または Essbase Integration Services によって設定されたドリルスルー・レポートを持つデータ・ソース用です。

GridBlox で `drillThroughEnabled` が `true` に設定されている場合は、DB2 Alphablox はデフォルトで、ドリルスルー操作が開始されたセルの座標を RDBResultSetDataBlox に送信し、ReportBlox を使用して関連詳細データをレンダリングします。レポートが別個のサイズ変更可能ブラウザー・ウィンドウに表示されます。ユーザーが別のセルを右クリックし、右クリック・メニューから「ドリルスルー」を選択する場合は、関連詳細データを表示した別のブラウザー・ウィンドウがポップアップします。これによって、ユーザーは別のセルの詳細データを比較できるようになります。

ドリルスルー操作がトリガーされたセルが階層の最も低いレベルにある場合は、戻される行セットは 1 つだけです。それ以外の場合は、そのセルのソース・データを構成するすべての行セットが戻されます。MSAS の場合、戻される最大行数は、DB2 Alphablox ホーム・ページの「管理」タブの「データ・ソース (Data Sources)」リンクで指定される「ドリルスルーの最大行数 (Maximum DrillThrough Rows)」設定によって決まります。IBM DB2 OLAP Server または Hyperion Essbase の場合、これは Essbase 管理者によって EIS に設定されます。

関連詳細データを表示するための独自のウィンドウ・プロパティーを定義するか、カスタム JSP を起動するには、`drillThroughWindow` タグを使用します。このタグが機能する方法についての詳細は、「*開発者用ガイド*」の『Microsoft Analysis Services のためのドリルスルー・サポート』のセクションを参照してください。実際の例は、Blox Sampler の『Retrieving Data』セクションにあります。

関連項目

275 ページの『drillThroughWindow』 RDBResultSetDataBlox および ReportBlox についての詳細は、「*Relational Reporting 開発者用ガイド*」を参照してください。

drillThroughWindow

GridBlox ユーザー・インターフェースでドリルスルー操作がトリガーされた場合にポップアップするブラウザー・ウィンドウのプロパティーを指定します。

データ・ソース

IBM DB2 OLAP Server、Hyperion Essbase、Microsoft Analysis Services

構文

```
drillThroughWindow="drillThroughWindowProperties"
```

または

```
<blox:drillThroughWindow
    url=""
    name=""
    height=""
    width=""
    resizable=""
    locationbarVisible=""
    menubarVisible=""
    scrollbarsVisible=""
    statusbarVisible=""
    toolbarVisible=""
>
</blox:drillThroughWindow>
```

ここで、それぞれ以下のとおりです。

引数	説明
drillThroughWindowProperties	ウィンドウのプロパティーを指定する、コンマで区切られた名前と値の組を含むストリングです。ストリングの有効な値は、drillThroughWindow タグの以下のタグ属性です。 • url: ポップアップ・ウィンドウにロードする JSP の URL を含むストリング • name: ポップアップ・ウィンドウの名前を含むストリング • height: ポップアップ・ウィンドウの高さ (ピクセル単位) • width: ポップアップ・ウィンドウの幅 (ピクセル単位) • resizable: true または false。ポップアップ・ウィンドウのサイズ変更が可能かどうか。デフォルトは true です。 • locationbarVisible: true または false。ポップアップ・ウィンドウにロケーション・バーを表示するかどうか。デフォルトは true です。 • menubarVisible: true または false。ポップアップ・ウィンドウにメニュー・バーを表示するかどうか。デフォルトは true です。

- scrollbarsVisible: true または false。ポップアップ・ウィンドウにスクロール・バーを表示するかどうか。デフォルトは true です。
- statusbarVisible: true または false。ポップアップ・ウィンドウにステータス・バーを表示するかどうか。デフォルトは true です。
- toolbarVisible: true または false。ポップアップ・ウィンドウにツールバー (ブラウザーのツールバー) を表示するかどうか。デフォルトは true です。

使用法

IBM DB2 OLAP Server、IBM DB2 OLAP Server Deployment Services、Hyperion Essbase、または Essbase Deployment Services の場合、このメソッドは IBM DB2 OLAP Server Integration Services または Essbase Integration Services によって設定されたドリルスルー・レポートを持つデータ・ソース用です。

GridBlox で drillThroughEnabled が true に設定されている場合は、DB2 Alphablox はデフォルトで、ドリルスルー操作が開始されたセルの座標を RDBResultSetDataBlox に送信し、ReportBlox を使用して関連詳細データをレンダリングします。レポートがポップアップ・ブラウザー・ウィンドウに表示されます。このポップアップ・ブラウザー・ウィンドウは、ツールバー、スクロール・バー、メニュー・バー、ステータス・バー、およびロケーション・バーと共にデフォルトでサイズ変更可能です。

ポップアップ・ブラウザー・ウィンドウに独自のウィンドウ・プロパティーを指定したい場合は、ドリルスルー・ウィンドウの URL、名前、機能を表すコンマで区切られた名前と値の組みのストリングを指定します。ウィンドウのプロパティーは、JavaScript ウィンドウ・オブジェクトに有効なプロパティーと類似しています。

以下のいずれかの形式の URL を指定できます。

- 絶対 URL の場合、ストリングは「http://」で開始する必要があります。
- 相対 URL の場合:
  - ストリングをスラッシュ (/) で始めると、URL がサーバー・ルートに対して相対であることを示します。アプリケーション・コンテキストを URL に含める必要があるということに注意してください。
  - ストリングをスラッシュ (/) なしで始めると、URL が現行の文書に対して相対であることを示します。

例

```
drillThroughWindow =
"url=myDrillThroughPage.jsp,name=myDrillThroughWindowName,height=600,
width=800,statusbarVisible=false, locationbarVisible=false"
```

関連項目

274 ページの『drillThroughEnabled』

editableCellStyle

編集可能セルの前景色および背景色を指定します。

データ・ソース

SAP BW 以外すべて

構文

```
editableCellStyle="style"
```

または

```
<blox:editableCellStyle
    background=""
    font=""
    foreground="" >
</blox:editableCellStyle>
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
style	background=white, foreground=blue	以下を指定するコンマで区切られたストリングです。 foreground: セルのテキストの色 background: セルの背景色 font: 使用する *font name:style:point* 色の名前または 16 進値を使用します。

使用法

フォントには、以下の構文を使用してフォント名、使用するスタイル、およびポイント・サイズを指定できます。

font name:style:point

- *font name*: 受け入れ可能なフォント名の値は、ブラウザーおよびクライアント・マシンによってさまざまに異なります。一般に受け入れられるフォント名には Arial、Courier、Helvetica、TimesRoman、SansSerif、 Serif、Monospace などがあります。
- *style*: 有効なフォント・スタイルは plain、italic、bold、および bolditalic です。
- *Point*: ポイント・サイズの整数 (通常は 8 から 36)。

3 つの属性のうちのいずれかが指定されていない場合、デフォルトの、または現行の継承されたフォント値が適用されます。ただし、属性を分離するコロンを含める必要があります。以下の例は、JSP タグを使用してフォントを指定する方法を示しています。

```
font="Arial:bolditalic:12"
font=":Bold:12"
```

例

```
editableCellStyle="background=red, foreground=green, font=Arial:bold:12"
```

関連項目

278 ページの『editedCellStyle』、261 ページの『cellEditor』

editedCellStyle

編集されたセルの前景色および背景色を指定します。

データ・ソース

SAP BW 以外すべて

構文

```
editedCellStyle="style"
```

または

```
<blox:editedCellStyle
    background=""
    font=""
    foreground="">
</blox:editedCellStyle>
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
style	background=white, foreground=blue	以下を指定するコンマで区切られたストリングです。 foreground: セルのテキストの色 background: セルの背景色 font: 使用する font name:style:point 色の名前または 16 進値を使用します。

使用法

このプロパティーは、ユーザーが値を変更した後の編集可能セルの色を指定します。変更されたセルに別の色を指定することにより、多くのセルを編集するユーザーに視覚的合図を提供できます。

フォントには、以下の構文を使用してフォント名、使用するスタイル、およびポイント・サイズを指定できます。

font name:style:point

- *font name*: 受け入れ可能なフォント名の値は、ブラウザーおよびクライアント・マシンによってさまざまに異なります。一般に受け入れられるフォント名には Arial、Courier、Helvetica、TimesRoman、SansSerif、 Serif、Monospace などがあります。

- *style*: 有効なフォント・スタイルは `plain`、`italic`、`bold`、および `bolditalic` です。
- *Point*: ポイント・サイズの整数 (通常は 8 から 36)。

3 つの属性のうちのいずれかが指定されていない場合、デフォルトの、または現行の継承されたフォント値が適用されます。ただし、属性を分離するコロンを含める必要があります。以下の例は、JSP タグを使用してフォントを指定する方法を示しています。

```
font="Arial:bolditalic:12"
font=":Bold:12"
```

例

```
editedCellStyle="background=gray, foreground=orange,
font=Helvetica:plain:12"
```

関連項目

277 ページの『editableCellStyle』、261 ページの『cellEditor』

enablePoppedOut

これは共通の Blox プロパティーです。 GridBlox が PresentBlox 内でネストされる場合は、以下のようになります。

- `poppedOut` プロパティーおよびそれに関連したプロパティーが PresentBlox で指定されている場合、PresentBlox の設定が使用されます。
- `poppedOut` プロパティーおよび関連プロパティーが PresentBlox に指定されていない場合は、ネストされた GridBlox のポップアウト設定が PresentBlox に適用されます。

詳しい説明は、175 ページの『enablePoppedOut』を参照してください。

expandCollapseMode

グリッドで展開および縮小 (プラスおよびマイナス) 記号をメンバーに表示するかどうかを指定します。

データ・ソース

すべて

構文

```
expandCollapseMode="expandCollapseMode"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
`expandCollapseMode`	`false`	展開および縮小を使用可能にするには `true`、展開および縮小を使用不可にするには `false` に設定します。

使用法

expandCollapseMode を true に設定すると、GridBlox のユーザー・インターフェースでのドリルアップまたはドリルダウン操作を示すために、正符号 (+) と負符号 (-) が表示されます。親メンバーが子の前に表示されるようにするには、照会を使用するのではなく、DataBlox parentFirst プロパティーを true に設定する必要があります。これは、展開/縮小モードで、結果セットが正しく検索されるようにし、ベース・メンバーおよび共用メンバーを判別できるようにするためです。

例

```
expandCollapseMode="true"
```

formatMask

フォーマット・マスク・ユーザー・インターフェースを使用する場合に、セルの定義済みフォーマット・マスクを指定します。

データ・ソース

すべて

構文

```
<blox:formatMask
       index="maskNumber"
       mask="mask"
/>
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
maskNumber	ヌル	定義または検索するマスクの索引番号です。1 から 15 までの整数でなければなりません。
mask	空ストリング	フォーマット属性を定義するストリングです。

使用法

defaultCellFormat または cellFormat とは異なり、このプロパティーはグリッド自体に影響を与えることはなく、「フォーマット・マスクの適用」ダイアログに表示される内容にのみ影響を与えます。

以下の表は、定義済みマスク値および各マスクの関連したフォーマット名をリストしています。フォーマット名およびマスクは、英語以外の言語バージョンでは異なる場合があります。定義済みマスクに加えて、12 から始まる独自の番号のマスクを作成できます。

フォーマット・マスク番号	マスク	フォーマット名
formatMask1		マスクなし (No mask)
formatMask2	#,##0.00;[red]-#,##0.00	負の数は赤 (Negative red)
formatMask3	#,##0.00;[red](#,##0.00)	負の数は赤で括弧 (Negative red parenthesis)
formatMask4	#,##0.00;(#,##0.00)	括弧 (Parenthesis)

フォーマット・マスク番号	マスク	フォーマット名
formatMask5	#,###K	千単位 (Thousands)
formatMask6	#,###M	百万単位 (Millions)
formatMask7	##0.00'%'	パーセント (Percentage)
formatMask8	$,##0	ドル (Dollars)
formatMask9	#,##0.00;(#,##0.00)	ユーロ (Euros)
formatMask10	#,##0.00	小数点以下 2 桁 (2 decimal places)
formatMask11	0	整数 (Integer)

以下の点に気を付けてください。
- 値ストリングにバックスラッシュ・エスケープ文字 (\) を使用しないでください。
- スラッシュ (/) 文字を使用して、ある値を別の値で除算できます (例えば、$#,###/1000)。
- このプロパティーは、formatName1-15 プロパティーと組み合わせて使用します。以下の例、および281 ページの『formatName』を参照してください。

例

以下のコードによって、ユーザーはフォーマット・マスク・ユーザー・インターフェースから、#,##0.00;[red]-#,##0.00 という関連した番号マスクで「Negative red」という名前のフォーマット名を選択できます。

```
<blox:formatMask
               index="2"
                mask="#,##0.00;[red]-#,##0.00" />

<blox:formatName
               index="2"
                name="Negative red" />
```

関連項目

281 ページの『formatName』

formatName

フォーマット・マスク・ユーザー・インターフェースを使用する場合に、セルの定義済みフォーマット名を指定します。

データ・ソース

すべて

構文

```
<blox:formatName
       index="formatNumber"
       name="name"
/>
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
formatNumber	ヌル	定義または検索するフォーマット名の索引番号です。1 から 15 までの整数でなければなりません。
name	空ストリング	指定されたフォーマット・マスクの名前を定義するストリングです。

使用法

すべての定義済み formatMask プロパティーには、定義済みの formatName が割り当てられています。プロパティー名は、formatName1 から formatName15 のいずれかです。

以下の表は、フォーマット名のプロパティー、定義済みの名前、および関連したフォーマット・マスクをリストしています。フォーマット名および番号マスク構文は、英語以外の言語バージョンでは異なる場合があります。

フォーマット名のプロパティー	フォーマット名	フォーマット・マスク
formatName1	マスクなし (No mask)	
formatName2	負の数は赤 (Negative red)	#,##0.00;[red]-#,##0.00
formatName3	負の数は赤で括弧 (Negative red parenthesis)	#,##0.00;[red](#,##0.00)
formatName4	括弧 (Parenthesis)	#,##0.00;(#,##0.00)
formatName5	千単位 (Thousands)	#,###K
formatName6	百万単位 (Millions)	#,###M
formatName7	パーセント (Percentage)	##0.00'%'
formatName8	ドル (Dollars)	$,##0
formatName9	ユーロ (Euros)	#,##0.00;(#,##0.00)
formatName10	小数点以下 2 桁 (2 decimal places)	#,##0.00
formatName11	整数 (Integer)	0

例

以下のコードによって、ユーザーはフォーマット・マスク・ユーザー・インターフェースから、#,##0.00;[red]-#,##0.00 という関連した番号マスクで「Negative red」という名前のフォーマット名を選択できます。

```
<blox:formatMask
               index="2"
                mask="#,##0.00;[red]-#,##0.00" />

<blox:formatName
               index="2"
                name="Negative red" />
```

関連項目

280 ページの『formatMask』

gridLinesVisible

グリッドの各セルの間に線を表示するかどうかを指定します。

データ・ソース
すべて

構文
```
gridLinesVisible="visible"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
visible	true	グリッド線を表示する場合は true、グリッド線を隠す場合は false を指定します。

例
```
gridLinesVisible="false"
```

headingsEnabled
グリッドの印刷時に行見出しおよび列見出しを表示するかどうかを指定します。

データ・ソース
すべて

構文
```
headingsEnabled="enable"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
enable	true	見出しを表示する場合は true、見出しを隠す場合は false を指定します。

例
```
headingsEnabled="false"
```

height
これは共通の Blox プロパティーです。詳しい説明は、42 ページの『height』を参照してください。

helpTargetFrame
これは共通の Blox プロパティーです。詳しい説明は、42 ページの『helpTargetFrame』を参照してください。

htmlGridScrolling
グリッドにスクロール・バーを表示するかどうかを指定します。

データ・ソース

すべて

構文

```
htmlGridScrolling="scroll"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
scroll	true	必要なときにスクロール・バーを表示する場合は true、スクロール・バーを表示しない場合は false を指定します。

使用法

このプロパティーを true に設定すると、スクロール・バーは必要なときにのみ表示されます。表示域にすべての要求されたデータが収まる場合、または値が false の場合、スクロール・バーは表示されません。

htmlShowFullTable

グリッド内のすべての行および列を表示するかどうかを指定します (定義された Blox 域および htmlGridScrolling プロパティーの設定は無視されます)。スクロール・バーは、ディスプレイの一部ではありません。グリッドの内容に応じて、データが画面の表示可能域を超えて広がる場合があります。この場合は、HTML ページのスクロール・バーを使用することによって、ユーザーはスクリーン外のデータまでスクロールしてそれを表示できます。デフォルトは false で、ディスプレイは Blox 境界内にとどまり、表全体が表示されることはありません。

データ・ソース

すべて

構文

```
htmlShowFullTable="show"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
show	false	すべての行および列をグリッドに表示する場合は true、すべての行および列がスペースに収まらない場合にスクロール・バーを使用する場合は false を指定します。

informationWindowName

特定のアプリケーション用のヘッダー・リンクに定義されている HTML ページを表示するために使用されるウィンドウの名前を指定します。

データ・ソース
すべて

構文
```
informationWindowName="name"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
name	"Information"	ウィンドウ名を表すストリングです。

使用法
このプロパティーを使用してウィンドウ名を定義することにより、URL ごとに新規ウィンドウが開かれるのではなく、すべてのヘッダー・リンク URL が定義済みウィンドウで開かれるようになります。

localeCode
これは共通の Blox プロパティーです。詳しい説明は、43 ページの『localeCode』を参照してください。

maximumUndoSteps
これは共通の Blox プロパティーです。詳しい説明は、44 ページの『maximumUndoSteps』を参照してください。

menubarVisible
これは共通の Blox プロパティーです。詳しい説明は、45 ページの『menubarVisible』を参照してください。

missingValueString
データベースにデータがないセルに表示するストリングを指定します。

データ・ソース
すべて

構文
```
missingValueString="value"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
value	空ストリング	任意のストリング

使用法
リレーショナル・データ・ソースへのアクセス中に、セルの値がヌルの場合にメッセージが表示されます。

例

```
missingValueString="Data is missing"
```

関連項目

286 ページの『noAccessValueString』

noAccessValueString

データ・アクセスがユーザーに付与されていないグリッド・セルに表示するストリングを指定します。

データ・ソース

マルチディメンション

構文

```
noAccessValueString="value"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
value	#NoAccess	任意のストリング

例

```
noAccessValueString="Access denied"
```

関連項目

missingValueString

noDataMessage

これは共通の Blox プロパティーです。詳しい説明は、45 ページの『noDataMessage』を参照してください。

poppedOut

これは、ContainerBlox から継承されたプロパティーです。GridBlox が PresentBlox 内でネストされる場合は、以下のようになります。

- poppedOut プロパティーおよびそれに関連したプロパティーが PresentBlox で指定されている場合、PresentBlox の設定が使用されます。
- poppedOut プロパティーおよび関連プロパティーが PresentBlox に指定されていない場合は、ネストされた GridBlox のポップアウト設定が PresentBlox に適用されます。

詳しい説明は、176 ページの『poppedOut』を参照してください。

poppedOutHeight

これは、ContainerBlox から継承されたプロパティーです。詳しい説明は、176 ページの『poppedOutHeight』を参照してください。

poppedOutTitle

これは、ContainerBlox から継承されたプロパティーです。詳しい説明は、177 ページの『poppedOutTitle』を参照してください。

poppedOutWidth

これは、ContainerBlox から継承されたプロパティーです。詳しい説明は、178 ページの『poppedOutWidth』を参照してください。

relationalRowNumbersOn

リレーショナル・データ・ソースの行番号を表示するかどうかを指定します。

データ・ソース

リレーショナル

構文

```
relationalRowNumbersOn="enable"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
enable	false	行番号を表示する場合は true、行番号を隠す場合は false を指定します。

例

```
relationalRowNumbersOn="true"
```

関連項目

288 ページの『rowHeadingsVisible』

removeAction

これは共通の Blox プロパティーです。詳しい説明は、46 ページの『removeAction』を参照してください。

render

これは共通の Blox プロパティーです。詳しい説明は、47 ページの『render』を参照してください。

rightClickMenuEnabled

これは共通の Blox プロパティーです。詳しい説明は、48 ページの『rightClickMenuEnabled』を参照してください。

rowHeadersWrapped

グリッド行見出しの長い見出しを複数行に折り返すかどうかを指定します。

データ・ソース

すべて

構文

```
rowHeadersWrapped="wrapped"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
wrapped	false	行見出しの折り返しを使用可能にするには true を指定します。

使用法

このプロパティーを false (デフォルト) に設定すると、行の幅は折り返しなしで行見出しテキスト全体が収まるサイズに設定されます。このプロパティーを true に設定すると、行見出しテキストは折り返され、行見出し列の幅が小さくなります。

関連項目

271 ページの『columnHeadersWrapped』

rowHeadingsVisible

データ値の左側の行見出しをグリッドに表示するかどうかを指定します。

データ・ソース

マルチディメンション

構文

```
rowHeadingsVisible="visible"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
visible	true	行見出しを表示する場合は true、行見出しを隠す場合は false を指定します。

例

```
rowHeadingsVisible="false"
```

関連項目

287 ページの『relationalRowNumbersOn』

rowHeadingWidths

グリッド上の行見出しの幅を指定します。

データ・ソース
すべて

構文
```
rowHeadingWidths="widths"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
widths	なし	行見出しをピクセル単位で表した整数のコンマで区切られたリストです。

使用法
このプロパティーを使用するには、autosizeEnabled プロパティーを false に設定する必要があります。行見出しの幅を指定しない場合は、幅は見出し全体が収まるよう自動的に計算されます。指定された幅より見出しが長い場合は、グリッドが正しく表示されないことがあります。

関連項目
253 ページの『autosizeEnabled』

rowHeight
各行の高さ (ピクセル単位) を指定します。

データ・ソース
すべて

構文
```
rowHeight="height"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
height	-1	行の高さをピクセル単位で表す整数です。

使用法
autosizeEnabled プロパティーを false に設定する必要があります。デフォルト (-1) は、行の高さを選択されたフォントに適切な値に設定します。

例
```
rowHeight="15"
```

関連項目
253 ページの『autosizeEnabled』

rowIndentation

行見出しをインデントするかどうか (およびインデントする方法) を指定します。

データ・ソース

マルチディメンション

構文

```
rowIndentation="strType"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
strType	parentLeft	可能な値は以下のとおりです (大/小文字の区別があります): parentRight: 最も低い世代の子が最も左側に表示され、各親は右に少しインデントされます。 parentLeft: 最も低い世代の子が最も右側に表示され、各親はその少し左に表示されます。 none: インデントされません。

使用法

行見出しのインデントは、ディメンション階層を示すのに役立ちます。

例

```
rowIndentation="none"
```

関連項目

288 ページの『rowHeadingsVisible』

showColumnDataGeneration

列に対して世代スタイルを使用できるようにします。

データ・ソース

マルチディメンション

構文

```
showColumnDataGeneration="show"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
show	false	世代スタイルを使用する場合は true、世代スタイルを使用しない場合は false を指定します。

使用法

世代スタイルをデータ・セルに適用するには、showColumnDataGeneration および showRowDataGeneration プロパティーを true に設定する必要があります。

スタイルはすべて、現在使用中のテーマから設定します。したがって、テーマでスタイル設定し、行データの世代スタイルを制御する必要があります (csClmnDtGnrtn0、csClmnDtGnrtn1、... csClmnDtGnrtn*N* クラス)。サポートされるテーマのスタイルシートは、<alphablox>/repository/theme/{*themeName*} にあります。また、行と列が交差するセルでは、列スタイルが行スタイルをオーバーライドします。

例

```
showColumnDataGeneration="true"
```

関連項目

291 ページの『showRowDataGeneration』

showColumnHeaderGeneration

列見出しに対して世代スタイルを使用できるようにします。

データ・ソース

マルチディメンション

構文

```
showColumnHeaderGeneration="show"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
show	false	世代スタイルを使用する場合は true、世代スタイルを使用しない場合は false を指定します。

使用法

値を true に設定すると、データ世代ごとの定義済みスタイルが見出しテキストに適用されます。スタイルをカスタマイズするには、基礎となるテーマで csClmnHdrGnrtn0、csClmnHdrGnrtn1、... csClmnHdrGnrtn*N* クラスを変更します。サポートされるテーマのスタイルシートは、<alphablox>/repository/theme/{*themeName*} にあります。

関連項目

292 ページの『showRowHeaderGeneration』

showRowDataGeneration

行に対して世代スタイルを使用できるようにします。

データ・ソース
マルチディメンション

構文
```
showRowDataGeneration="show"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
show	false	世代スタイルを使用する場合は true、世代スタイルを使用しない場合は false を指定します。

使用法
世代スタイルをデータ・セルに適用するには、showRowDataGeneration および showColumnDataGeneration プロパティーを true に設定する必要があります。

スタイルはすべて、現在使用中のテーマから設定します。したがって、基礎となるテーマでスタイルを設定し、行データの世代スタイルを制御する必要があります (csRwDtGnrtn0、csRwDtGnrtn1、... csRwDtGnrtn*N* スタイル・クラス)。サポートされるテーマのスタイルシートは、<alphablox>/repository/theme/{*themeName*} にあります。また、行と列が交差するセルでは、列スタイルが行スタイルをオーバーライドします。

例
```
showRowDataGeneration="true"
```

関連項目
290 ページの『showColumnDataGeneration』

showRowHeaderGeneration
行見出しに対して世代スタイルを使用できるようにします。

データ・ソース
マルチディメンション

構文
```
showRowHeaderGeneration=show
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
show	false	世代スタイルを使用する場合は true、世代スタイルを使用しない場合は false を指定します。

使用法

値を true に設定すると、データ世代ごとの定義済みスタイルが見出しテキストに適用されます。スタイルをカスタマイズするには、DHTML クライアントで、基礎となるテーマの csRwHdrGnrtn0、csRwHdrGnrtn1、... csRwHdrGnrtn*N* クラスを変更します。サポートされるテーマのスタイルシートは、<alphablox>/repository/theme/{*themeName*} にあります。

関連項目

291 ページの『showColumnHeaderGeneration』

toolbarVisible

ツールバーを表示するかどうかを指定します。

データ・ソース

すべて

構文

JSP タグ属性

```
toolbarVisible="visible"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
visible	true	true: ツールバーが表示されます。 false: ツールバーは表示されません。

使用法

デフォルトで、ツールバーは独立型 GridBlox で表示されます。ネストされた <blox:toolbar> タグが追加された場合、その設定値はこの属性の値を上書きします。たとえば、次のコーディングではツールバーが表示されます。

```
<blox:grid id="myGrid" toolbarVisible="false" ....>
   <blox:toolbar visible="true" />
   <blox:data bloxRef="myDataBlox"/>
</blox:grid>
```

ヒント: toolbarVisible は単にタグ属性であり、GridBlox プロパティーではないので、関連する getter または setter メソッドはありません。

visible

これは共通の Blox プロパティーです。詳しい説明は、49 ページの『visible』を参照してください。

width

これは共通の Blox プロパティーです。詳しい説明は、49 ページの『width』を参照してください。

writebackEnabled

ユーザーがグリッドのセルを編集するのを許可します。

データ・ソース

SAP BW 以外すべて

構文

```
writebackEnabled="enabled"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
enable	false	書き戻しを使用可能にするには true、書き戻しを使用不可にするには false を指定します。

使用法

関連したグリッドの書き戻しプロパティーおよびメソッドを有効にするには、このプロパティーを GridBlox に指定する必要があります。

例

```
writebackEnabled="true"
```

関連項目

261 ページの『cellEditor』

第 13 章 MemberFilterBlox タグ・リファレンス

この章には、MemberFilterBlox タグ属性の参照資料が含まれています。 Blox についての一般的な参照情報は、21 ページの『第 3 章 一般 Blox リファレンス情報』を参照してください。このリファレンスの使用方法については、1 ページの『第 1 章 このリファレンスの使用法』を参照してください。

MemberFilterBlox の概説

MemberFilterBlox を使用すると、ユーザーがメンバーを選択するための「メンバー・フィルター」ダイアログを提示できます。メンバー・フィルターは、Blox ユーザー・インターフェースに組み込まれ、ユーザーは以下の場合にこれを使用できます。

- 「メンバー・フィルター...」を GridBlox の右クリック・メニューから選択する場合
- 「メンバー・フィルター...」を「データ・レイアウト」パネルのドロップダウン・リストから選択する場合
- 「続く...」を「ページ」パネルから選択する場合

MemberFilterBlox を使用すると、DataBlox の使用を指定し、その後ページに「メンバー・フィルター」ダイアログを置くことができ、ユーザーはそれを使用して選択可能なすべてのディメンションまたは指定したディメンションからメンバーを選択することができます。このディメンション選択ドロップ・リストには、基礎となる DataBlox のデータ照会に基づいてデータが取り込まれます。

基礎となる DataBlox が同じページの PresentBlox で使用される場合、PresentBlox は行われる選択を自動的に反映します。

デフォルトでは、`dimensionSelectionEnabled` プロパティーは `true` に設定されています。これにより、データ照会の結果として使用可能なすべてのディメンションがリストに表示されます。これらのディメンションはアルファベット順にリストされます。他のものを指定しない限り、リストの最初のディメンションが初期選択ディメンションとなり、左の「ディメンション階層」パネルに現れます。初期選択ディメンションを指定するには、selectedDimension プロパティーを使用します。ドロップ・リストに現れるディメンションを制限するには、selectableDimensions プロパティーを使用してディメンションのリストを指定できます。

MemberFilterBlox JSP カスタム・タグ構文

DB2 Alphablox タグ・ライブラリーは、それぞれの Blox を作成するために JSP ページで使用するカスタム・タグを提供します。このセクションでは、カスタム・タグを作成して MemberFilterBlox を作成する方法を説明します。すべての属性を含むタグのコピー・アンド・ペースト・バージョンについては、527 ページの『MemberFilterBlox JSP カスタム・タグ』を参照してください。

構文

```
<blox:memberFilter
    [attribute="value"] >
    <blox:data bloxRef="" />
</blox:memberFilter>
```

ここで、それぞれ以下のとおりです。

attribute 属性表にリストされている属性の 1 つです。

value 属性の有効な値です。

属性は以下のいずれかになります。

属性
id
applyButtonEnabled
bloxEnabled
bloxName
dimensionSelectionEnabled
height
selectableDimensions
selectedDimension
visible
width

使用法

各カスタム・タグには 1 つ以上の属性を含めることができ、それぞれを 1 つ以上のスペースまたは改行文字で区切ります。余分のスペースまたは改行文字は無視されます。読みやすくするため、同じインデントでそれぞれ別々の行に属性を並べることができます。

終了タグ </blox:memberFilter> は、終了スラッシュで置き換えられます。ただし、タグの最後の属性と終了文字 ″>″ の間に置く必要があります。たとえば、最後の属性が width の場合、タグの最後は以下のようになります。

```
selectedDimension="All Products" />
```

例

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

<blox:memberFilter id="myMemberFilter">
    <blox:data bloxRef="myDataBlox" />
</blox:memberFilter>
```

MemberFilterBlox の例

このセクションでは、MemberFilterBlox をユーティリティーとして使用して、PresentBlox 内の選択可能なすべてのディメンション、指定したディメンションのみ、1 つのディメンションのみのそれぞれでメンバーをフィルターに掛ける方法を示す例を提供します。

例 1: 選択可能なすべてのディメンションでメンバーをフィルターに掛ける

これは、同じ DataBlox を使用して PresentBlox と同じページで MemberFilterBlox を追加する例です。

1. MemberFilterBlox は `<blox:memberFilter>` タグを使用してページに追加されます。
2. ディメンション選択ドロップ・リストは使用可能になっています。`selectableDimensions` が指定されていないため、DataBlox から選択可能なすべてのディメンションがドロップ・リストに現れます。
3. ドロップ・リストの初期選択ディメンションは All Time Periods に設定されています。
4. `bloxRef` タグ属性を使用して基礎となる DataBlox を指定しています。
5. PresentBlox で同じ DataBlox が使用されています。ユーザーが MemberFilterBlox で行う選択は自動的に PresentBlox に反映されます。

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

      <blox:data id="myDataBlox"
          dataSourceName="QCC-Essbase"
          useAliases="true"
          query="<ROW (¥"All Products¥") <ICHILD ¥"All Products¥"
                 <COLUMN (¥"All Time Periods¥" Measures Scenario)
                 <CHILD ¥"All Time Periods¥" !"
          selectableSlicerDimensions="All Locations" />

      <html>
      <head>
          <blox:header />
      </head>
      <body>
(1)   <blox:memberFilter id="memberFilterBlox"
(2)       dimensionSelectionEnabled = "true"
(3)       selectedDimension="All Time Periods">
(4)       <blox:data bloxRef="myDataBlox" />
      </blox:memberFilter>
      <br>
      <blox:present id="myPresentBlox" width="600" height="400">
(5)       <blox:data bloxRef="myDataBlox" />
      </blox:present>
      </body>
      </html>
```

例 2: 指定されたディメンションのみでメンバーをフィルターに掛ける

これは、selectableDimensions タグ属性を使用して、ディメンション選択ドロップ・リストに All Products ディメンションと All Time Periods ディメンションのみを持つ MemberFilterBlox を追加する例です。

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

<blox:data id="myDataBlox"
   dataSourceName="QCC-Essbase"
   useAliases="true"
   query="<ROW (¥"All Products¥") <ICHILD ¥"All Products¥"
      <COLUMN (¥"All Time Periods¥" Measures Scenario)
      <CHILD ¥"All Time Periods¥" !" />
...

<blox:memberFilter id="memberFilterBlox"
   dimensionSelectionEnabled="true"
   selectableDimensions="All Products, All Time Periods">
   <blox:data bloxRef="myDataBlox" />
</blox:memberFilter>
...
```

例 3: 1 つのディメンションのみでメンバーをフィルターに掛ける

これは、ディメンション選択ドロップ・リストを持たない (dimensionSelectionEnabled = "false") MemberFilterBlox を追加する例です。selectedDimension を All Products に設定すると、左の「ディメンション階層」パネルに All Products が表示されます。ユーザーはこのディメンションからのみメンバーを選択できます。

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

<blox:data id="myDataBlox"
```

```
   dataSourceName="QCC-Essbase"
   useAliases="true"
   query="<ROW (¥"All Products¥") <ICHILD ¥"All Products¥"
      <COLUMN (¥"All Time Periods¥" Measures Scenario)
      <CHILD ¥"All Time Periods¥" !" />
...

<blox:memberFilter id="memberFilterLocked"
   dimensionSelectionEnabled="false"
   selectedDimension="All Products">
   <blox:data bloxRef="myDataBlox" />
</blox:memberFilter>
...
```

固有の MemberFilterBlox タグ属性

以下は、固有の MemberFilterBlox タグ属性です。複数の Blox に共通するタグ属性のリストについては、35 ページの『カテゴリー別の共通の Blox タグ属性』を参照してください。

- applyButtonEnabled
- dimensionSelectionEnabled
- selectableDimensions
- selectedDimension

MemberFilterBlox タグ属性

このセクションでは、MemberFilterBlox によってサポートされるタグ属性をアルファベット順で説明します。使用可能な共通 Blox タグ属性については、リストされていますが説明はありません。共通 Blox プロパティーの詳しい説明は、36 ページの『複数の Blox に共通するタグ属性』を参照してください。 MemberFilterBlox メソッドの参照情報については、Javadoc の com.alphablox.blox パッケージの MemberFilterBlox クラスを参照してください。

id

これは共通の Blox タグ属性です。詳しい説明は、43 ページの『id』を参照してください。

applyButtonEnabled

「メンバー・フィルター」ユーザー・インターフェースに「適用」ボタンを表示します。

データ・ソース

マルチディメンション

構文

```
applyButtonEnabled = "applyButtonEnabled"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
applyButtonEnabled	true	true:「適用」ボタンを表示します。false:「適用」ボタンを非表示にします。

使用法

メンバー・フィルターの「適用」ボタンが不要な場合もあります。たとえば、MemberFilterBlox で参照される DataBlox が同じページのユーザー・インターフェース Blox で使用されず、個別の照会を構成したり計算を実行したりする目的でユーザーが関係するメンバーを指定するために MemberFilterBlox のみが使用されるような場合です。「適用」ボタンを非表示にするには、このプロパティーを false に設定してください。

bloxEnabled

これは共通の Blox プロパティーです。詳しい説明は、38 ページの『bloxEnabled』を参照してください。

bloxName

これは共通の Blox プロパティーです。詳しい説明は、39 ページの『bloxName』を参照してください。

dimensionSelectionEnabled

ディメンション選択ドロップ・リストの表示/非表示を指定します。

データ・ソース

マルチディメンション

構文

```
dimensionSelectionEnabled = "dimensionSelectionEnabled"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
dimensionSelectionEnabled	true	true: ディメンション選択ドロップ・リストを表示します。 false: ディメンション選択ドロップ・リストを非表示にします。

使用法

dimensionSelectionEnabled が true に設定されると、データ照会の結果として使用可能なすべてのディメンションがリストに表示されます。ディメンションはアルファベット順にリストされます。デフォルトでは、リストの最初のディメンションが初期選択ディメンションとなり、左の「ディメンション階層」パネルに現れま

す。初期選択と異なるディメンションを指定するには、selectedDimensionを参照してください。ドロップ・リストのディメンションを制限するには、selectableDimensionsを使用します。

関連項目

302 ページの『selectedDimension』、301 ページの『selectableDimensions』

height

これは共通の Blox プロパティーです。詳しい説明は、42 ページの『height』を参照してください。

メンバー・フィルターには、最適なレイアウトのためのデフォルトの幅と高さがあります。実際に特定のサイズが必要ない場合は、幅または高さを指定しないでください。指定するサイズが小さすぎる場合 (高さ 325 ピクセル、幅 600 ピクセル未満)、サイズは 325 X 600 に設定されます。

selectableDimensions

ディメンション選択ドロップ・リストに現れるディメンションを指定します。

データ・ソース

マルチディメンション

構文

```
selectableDimensions = "selectableDimensions"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
selectableDimensions	結果セット内の選択可能なすべてのディメンション	コンマで区切られたディメンション・リスト。

使用法

dimensionSelectionEnabled が true に設定されると (デフォルト)、selectableDimensions で他の指定を行わない限り、データ結果セット内の選択可能なすべてのディメンションがディメンション選択ドロップ・リストに現れます。指定したディメンションは、指定した順序に関係なく常にアルファベット順にドロップ・リストに現れます。 selectedDimension を使用して他の指定を行わない限り、最初のディメンションが初期選択ディメンション (「ディメンション階層」パネルに現れるディメンション) となります。

例

```
<blox:memberFilter id="myMemberFilter"
   dimensionSelectionEnabled = "true"
   selectableDimensions = "Year, Scenario, Products">
   <blox:data bloxRef = "myDataBlox" />
</blox:memberFilter>
```

関連項目

300 ページの『dimensionSelectionEnabled』、302 ページの『selectedDimension』

selectedDimension

初期選択ディメンションを指定します。

データ・ソース

マルチディメンション

構文

```
seletedDimension = "selectedDimension"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
selectedDimension	データ照会から選択可能なディメンションの中の最初のディメンション	初期選択ディメンションになるディメンションの名前。

使用法

1 つのディメンションからのメンバーしか「ディメンション階層」パネルに表示できないため、初期選択ディメンションを指定する必要があります。これが指定されないと、アルファベット順の最初のディメンションが選択ディメンションとなります。

visible

これは共通の Blox プロパティーです。詳しい説明は、49 ページの『visible』を参照してください。

width

これは共通の Blox プロパティーです。詳しい説明は、49 ページの『width』を参照してください。

メンバー・フィルターには、最適なレイアウトのためのデフォルトの幅と高さがあります。実際に特定のサイズが必要ない場合は、幅または高さを指定しないでください。指定するサイズが小さすぎる場合 (高さ 325 ピクセル、幅 600 ピクセル未満)、サイズは 325 X 600 に設定されます。

第 14 章 PageBlox タグ・リファレンス

この章には、PageBlox タグ属性の参照資料が含まれています。 Blox についての一般的な参照情報は、21 ページの『第 3 章 一般 Blox リファレンス情報』を参照してください。このリファレンスの使用方法については、1 ページの『第 1 章 このリファレンスの使用法』を参照してください。

PageBlox の概説

PageBlox を使用すると、ユーザーはグリッドまたはチャートに現れるデータをフィルターに掛けることができます。「ページ」軸にある現行の結果セットの各ディメンションは、ページ・フィルターのドロップ・リストとして現れます。ユーザーがドロップ・リストからディメンション・メンバーを選択すると、そのメンバーはグリッドまたはチャートに現れるデータをフィルターに掛けるために使用されます。

PageBlox JSP カスタム・タグ構文

DB2 Alphablox タグ・ライブラリーは、それぞれの Blox を作成するために JSP ページで使用するカスタム・タグを提供します。このセクションでは、カスタム・タグを作成して PageBlox を作成する方法を説明します。すべての属性を含むタグのコピー・アンド・ペースト・バージョンについては、527 ページの『PageBlox JSP カスタム・タグ』を参照してください。

構文

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

ここで、それぞれ以下のとおりです。

attribute	属性表にリストされている属性の 1 つです。
value	属性の有効な値です。

属性は以下のいずれかになります。

属性
id
applyPropertiesAfterBookmark
bloxEnabled
bloxName

属性
bookmarkFilter
fixedChoiceLists
height
helpTargetFrame
labelPlacement
localeCode
maximumUndoSteps
moreChoicesEnabled
moreChoicesEnabledDefault
noDataMessage
render
visible
width

使用法

各カスタム・タグには 1 つ以上の属性を含めることができ、それぞれを 1 つ以上のスペースまたは改行文字で区切ります。余分のスペースまたは改行文字は無視されます。読みやすくするため、同じインデントでそれぞれ別々の行に属性を並べることができます。

終了タグ </blox:page> は、終了スラッシュで置き換えられます。ただし、タグの最後の属性と終了文字 ″>″ の間に置く必要があります。たとえば、最後の属性が width の場合、タグの最後は以下のようになります。

```
width="650" />
```

例

```
<blox:page
     fixedChoiceLists="Year:Qtr1,Qtr2;Market:East"
     >
</blox:page>
```

カテゴリー別の PageBlox タグ属性

以下の表は、固有の PageBlox タグ属性をリストしたものです。複数の Blox に共通するタグ属性のリストについては、35 ページの『カテゴリー別の共通の Blox タグ属性』を参照してください。 PageBlox メソッドの参照情報については、Javadoc の com.alphablox.blox パッケージの PageBlox クラスを参照してください。

PageBlox によってサポートされるタグ属性は、以下のように相互参照に編成されています。

- 305 ページの『選択リスト』
- 305 ページの『パネルのタイプと外観』

選択リスト

以下は、PageBlox 選択リストに関連したタグ属性です。

- fixedChoiceLists
- moreChoicesEnabledDefault
- moreChoicesEnabledDefault

パネルのタイプと外観

以下は、ページ・パネルのタイプを設定するためのタグ属性です。この設定は PageBlox の外観にも影響を与えます。

- labelPlacement

PageBlox タグ属性

このセクションでは、PageBlox によってサポートされるタグ属性をアルファベット順で説明します。共通 Blox タグ属性については、リストされていますが説明はありません。共通 Blox タグ属性の詳しい説明は、36 ページの『複数の Blox に共通するタグ属性』を参照してください。 PageBlox メソッドの参照情報については、Javadoc の com.alphablox.blox パッケージの PageBlox クラスを参照してください。

id

これは共通の Blox プロパティーです。詳しい説明は、43 ページの『id』を参照してください。

applyPropertiesAfterBookmark

これは共通の Blox プロパティーです。詳しい説明は、36 ページの『applyPropertiesAfterBookmark』を参照してください。

bloxEnabled

これは共通の Blox プロパティーです。詳しい説明は、38 ページの『bloxEnabled』を参照してください。

bloxName

これは共通の Blox プロパティーです。詳しい説明は、39 ページの『bloxName』を参照してください。

bookmarkFilter

これは共通の Blox プロパティーです。詳しい説明は、37 ページの『bookmarkFilter』を参照してください。

fixedChoiceLists

名前付きディメンションおよびメンバーをドロップ・リストに置いてユーザーがそれにアクセスできるようにします。

データ・ソース

マルチディメンション

構文

```
fixedChoiceLists="dimensionMemberList"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
dimensionMemberList	空ストリング	指定されたディメンション (セミコロンで区切られる) のメンバー名のコンマ区切りストリングの対: *dimension1:member1,member2; dimension2:member1,member2* ディメンション名またはメンバー名に含まれない限り、スペースはどこにも含めません。

使用法

デフォルトでは、ユーザーはすべてのディメンションおよびメンバーにアクセスできます。

fixedChoiceLists プロパティーで指定したディメンションはすべて、DataBlox selectableSlicerDimensions プロパティーにも現れなければなりません。

初期表示の照会には、fixedChoiseLists プロパティーで指定された各ディメンションのメンバー 1 つのみ含める必要があります。これを行わないと、ディメンションのルート・レベルのメンバーが最初に固定選択リストに現れます。照会で指定されたメンバーは、固定選択リストのデフォルト・メンバーになります。たとえば、メンバー Q1 および Q2 を持つ Year ディメンション上の固定選択リストがあり、Q1 と Q2 のどちらも照会で指定されない場合、固定選択リストには最初に Year、Q1、および Q2 が表示されます。ページ・フィルターで Q1 または Q2 を選択した後、リストから Year が除去されます。照会で複数のメンバーが指定される場合、固定選択リストは PageBlox に現れません。

PageBlox が別の Blox (たとえば PresentBlox) 内でネストされ、ユーザーが (たとえば) PageBlox から行または列の軸にメンバーを移動する場合、固定選択リストはその行または列の軸には適用されず、PageBlox にのみ適用されます。これを回避したい場合は、ネストされた PageBlox の代わりにスタンドアロンの PageBlox を使用してください。

プロパティーの値に指定されるディメンション名およびメンバー名には、固有の名前 (IBM DB2 OLAP Server または Hyperion Essbase のベース名) または表示名を使用できます。これにより、アセンブラーは同じ表示名を持つ異なるメンバーまたはディメンションを区別できます。

IBM DB2 OLAP Server または Hyperion Essbase では、別名表に関係なく、ベース名を使用することによりメンバーを指定できます。

例

```
fixedChoiceLists="Year:Qtr1,Qtr2;Market:East"
```

関連項目

307 ページの『moreChoicesEnabled』、308 ページの『moreChoicesEnabledDefault』、229 ページの『selectableSlicerDimensions』

height

これは共通の Blox プロパティーです。詳しい説明は、42 ページの『height』を参照してください。

helpTargetFrame

これは共通の Blox プロパティーです。詳しい説明は、42 ページの『helpTargetFrame』を参照してください。

labelPlacement

PageBlox ラベルの PageBlox ドロップダウン・リストに対する相対位置を設定します。有効な `labelPlacement` 値は `left`、`top`、および `none` です。デフォルト値は `left` です。

データ・ソース

マルチディメンション

構文

```
labelPlacement="placement"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
placement	left	PageBlox ラベルのドロップダウン・リストに対する相対位置を示すストリング。有効な値は `left`、`top`、および `none` です。

例

```
labelPlacement="top"
```

localeCode

これは共通の Blox プロパティーです。詳しい説明は、43 ページの『localeCode』を参照してください。

maximumUndoSteps

これは共通の Blox プロパティーです。詳しい説明は、44 ページの『maximumUndoSteps』を参照してください。

moreChoicesEnabled

ユーザーがメンバー・フィルターを使用して他の選択を表示できるよう、名前付きディメンションの PageBlox ドロップ・リストの「続く...」オプションを選択可能にするかどうかを指定します。

データ・ソース

マルチディメンション

構文

```
moreChoicesEnabled="choices"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
choices	空ストリング	セミコロン (";") で区切られた *dimension:enabled* の対のストリング。ディメンション名の中のスペースではない限り、このリストにはスペースを含めるべきではありません。 • *dimension*: ディメンション名 • *enabled*: true または false

使用法

デフォルトでは、「**続く...**」オプションは、ページ軸に配置されているすべてのディメンションで選択できます。このプロパティーを使用すると、ユーザーが他の選択を行うためにメンバー・フィルターにアクセスすることができないよう、名前付きディメンションの「**続く...**」オプションを非表示にできます。

例

以下の行を使用すると、Product ディメンションの PageBlox ドロップ・リストで「続く...」オプションを選択でき、Market ディメンションではできないようようにすることができます。ディメンション名にスペースが含まれていない限り、この中には空白を含めません。

```
moreChoicesEnabled="Product:true;Market:false"
```

関連項目

305 ページの『fixedChoiceLists』、308 ページの『moreChoicesEnabledDefault』

moreChoicesEnabledDefault

ユーザーがメンバー・フィルターを使用して他の選択を表示できるよう、名前付きディメンションの PageBlox ドロップ・リストの「続く...」オプションを選択可能にするかどうかに関するデフォルトを指定します。

データ・ソース

マルチディメンション

構文

```
moreChoicesEnabledDefault="boolean"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
enabledDefault	true	有効な値は true および false です。

例

```
moreChoicesEnabledDefault="false"
```

関連項目

305 ページの『fixedChoiceLists』、307 ページの『moreChoicesEnabled』

noDataMessage

これは共通の Blox プロパティーです。詳しい説明は、45 ページの『noDataMessage』を参照してください。

render

これは共通の Blox プロパティーです。詳しい説明は、47 ページの『render』を参照してください。

visible

これは共通の Blox プロパティーです。詳しい説明は、49 ページの『visible』を参照してください。

width

これは共通の Blox プロパティーです。詳しい説明は、49 ページの『width』を参照してください。

第 15 章 PresentBlox タグ・リファレンス

この章には、PresentBlox の参照資料が含まれています。Blox についての一般的な参照情報は、21 ページの『第 3 章 一般 Blox リファレンス情報』を参照してください。このリファレンスの使用方法については、1 ページの『第 1 章 このリファレンスの使用法』を参照してください。

PresentBlox の概説

PresentBlox は、単一のプレゼンテーションの中に ChartBlox、 GridBlox、 PageBlox、 ToolbarBlox、および DataLayoutBlox をネストできるグラフィカル・ユーザー・インターフェースを提供します。アプリケーション・アセンブラーは PresentBlox プロパティーを使用して、これらの Blox の表示方法を調整できます。

9 ページの『ネストされた Blox』に示されているように、PresentBlox は、多数の Blox 修飾子を使用します。ネストされたそれぞれの Blox について詳しくは、以下のページのいずれかを参照してください。

- 77 ページの『ChartBlox の概説』
- 239 ページの『DataLayoutBlox の概説』
- 245 ページの『GridBlox の概説』
- 303 ページの『PageBlox の概説』
- 339 ページの『ToolbarBlox の概説』

PresentBlox は複数の Blox を 1 つに結合して、同じウィンドウ領域に同じデータのチャート・ビューおよびグリッド・ビューを同時に表示します。

注: ユーザーがツールバーの「Grid (グリッド)」、「Chart (チャート)」、「Page Filter (ページ・フィルター)」、および「Data Layout Panel (データ・レイアウト・パネル)」ボタンをクリックすると、これらのコンポーネントを PresentBlox 内で表示または非表示にできます。さらにユーザーは、グリッドとチャートの間にあるスライダー・バーを移動して、各ビューに割り振られたスペースの量を変更できます。

PresentBlox JSP カスタム・タグ構文

DB2 Alphablox タグ・ライブラリーは、それぞれの Blox を作成するために JSP ページで使用するカスタム・タグを提供します。このセクションでは、PresentBlox を作成するためのカスタム・タグの作成方法について説明します。すべての属性を含むタグのコピー・アンド・ペースト・バージョンについては、528 ページの『PresentBlox JSP カスタム・タグ』を参照してください。

構文

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

ここで、それぞれ以下のとおりです。

attribute 属性表にリストされている属性の 1 つです。

value 属性の有効な値です。

属性は以下のいずれかになります。

属性
id
applyPropertiesAfterBookmark
bloxEnabled
bloxName
chartAvailable
chartFirst
dataLayoutAvailable
dividerLocation
enablePoppedOut
gridAvailable
height
helpTargetFrame
localeCode
maximumUndoSteps
menubarVisible
noDataMessage
pageAvailable
poppedOut
poppedOutHeight
poppedOutTitle
poppedOutWidth
render
splitPane
splitPaneOrientation
toolbarVisible
visible
width

使用法

各カスタム・タグには 1 つ以上の属性を含めることができ、それぞれを 1 つ以上のスペースまたは改行文字で区切ります。余分のスペースまたは改行文字は無視されます。読みやすくするため、同じインデントでそれぞれ別々の行に属性を並べることができます。

終了タグ </blox:present> は、終了スラッシュで置き換えられます。ただし、タグの最後の属性と終了文字 ″>″ の間に置く必要があります。たとえば、最後の属性が width の場合、タグの最後は以下のようになります。

```
width="650" />
```

例

```
<blox:present
   id="myPresent1"
   width="650"
   height="600"
   >
   <blox:data
      dataSourceName="TBC"
      query="<SYM <ROW(Product) <ICHILD Product <COLUMN(Year,
         Scenario) Qtr1 Qtr2 <CHILD Scenario Sales !"
   />
</blox:present>
```

カテゴリー別の PresentBlox タグ属性

以下は、機能のカテゴリー別に編成された PresentBlox タグ属性です。複数の Blox に共通するタグ属性のリストについては、35 ページの『カテゴリー別の共通の Blox タグ属性』を参照してください。 PresentBlox によってサポートされるタグ属性は、以下のように相互参照に編成されています。

- 313 ページの『データ域』
- 313 ページの『チャートの外観』
- 314 ページの『データ・レイアウトの外観』
- 314 ページの『グリッドの外観』
- 314 ページの『ページの外観』
- 314 ページの『メニュー・バーの外観』
- 314 ページの『ツールバーの外観』
- 314 ページの『ポップアウトのプロパティー』

データ域

以下のタグ属性は、PresentBlox のデータ域に影響を与えます。

- dividerLocation
- splitPane
- splitPaneOrientation

チャートの外観

以下のタグ属性は、PresentBlox 内の ChartBlox の外観に影響を与えます。

- chartAvailable
- chartFirst

データ・レイアウトの外観

以下のタグ属性は、PresentBlox 上の DataLayoutBlox の外観に影響を与えます。

- dataLayoutAvailable

グリッドの外観

以下のタグ属性は、PresentBlox 上の GridBlox の外観に影響を与えます。

- gridAvailable

ページの外観

以下のタグ属性は、PresentBlox 上の PageBlox の外観に影響を与えます。

- pageAvailable

メニュー・バーの外観

以下のタグ属性は、PresentBlox 上のメニュー・バーの外観に影響を与えます。

- menubarVisible

ツールバーの外観

以下のタグ属性は、PresentBlox 上のツールバーの外観に影響を与えます。

- toolbarVisible

ポップアウトのプロパティー

PresentBlox を別個のポップアウト・ブラウザー・ウィンドウで表示することに関連したタグ属性を以下にリストします。

- enablePoppedOut
- poppedOut
- poppedOutHeight
- poppedOutTitle
- poppedOutWidth

PresentBlox タグ属性

このセクションでは、PresentBlox によってサポートされるタグ属性をアルファベット順で説明します。共通 Blox タグ属性については、リストされていますが説明はありません。共通 Blox タグ属性の詳しい説明は、36 ページの『複数の Blox に共通するタグ属性』を参照してください。 PresentBlox メソッドの参照情報については、Javadoc の com.alphablox.blox パッケージの PresentBlox クラスを参照してください。

id

これは共通の Blox タグ属性です。詳しい説明は、43 ページの『id』を参照してください。

applyPropertiesAfterBookmark

これは共通の Blox プロパティーです。詳しい説明は、36 ページの『applyPropertiesAfterBookmark』を参照してください。

bloxEnabled

これは共通の Blox プロパティーです。詳しい説明は、38 ページの『bloxEnabled』を参照してください。

bloxName

これは共通の Blox プロパティーです。詳しい説明は、39 ページの『bloxName』を参照してください。

chartAvailable

PresentBlox 内でユーザーがチャートを使用できるかどうかを指定します。

データ・ソース

すべて

構文

```
chartAvailable="available"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	可能な値
available	true	チャートを使用可能にするには、true を指定します。使用不可にするには、false を指定します。

使用法

デフォルトは true です。 false に設定した場合、ユーザーはチャートをまったく表示できません。ユーザーが呼び出すまでチャートの出現を抑制するには、dividerLocation プロパティーを使用します。

例

```
chartAvailable="false"
```

関連項目

315 ページの『chartFirst』、317 ページの『dividerLocation』

chartFirst

両方が PresentBlox 表示域に表示されるとき、チャートがグリッドよりも前に表示されるかどうかを設定します。

データ・ソース

すべて

構文

```
chartFirst="first"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	可能な値
first	false	チャートを最初に表示するには、true を指定します。チャートをグリッドの後に表示するには、false を指定します。

使用法

splitPaneOrientation プロパティーに指定した値に応じて、“before” はグリッドの「左側」または「上側」のいずれかになります。

例

```
chartFirst="true"
```

関連項目

315 ページの『chartAvailable』、315 ページの『chartFirst』、317 ページの『dividerLocation』

dataLayoutAvailable

PresentBlox 内でデータ・レイアウト・パネルを使用できるかどうかを指定します。

データ・ソース

すべて

構文

```
dataLayoutAvailable="available"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	可能な値
available	true	データ・レイアウト・パネルを使用可能にするには、true を指定します。使用不可にするには、false を指定します。

使用法

dataLayoutAvailable プロパティーについては、以下の事柄に注意してください。

- このプロパティーが false に設定されていると、ユーザーはデータ・レイアウト・パネルを呼び出せません。「レイアウト」ボタンは、ツールバーに表示されません。
- このプロパティーが true に設定されて、visible プロパティーが false に設定されていると、データ・レイアウト・パネルはユーザーがツールバーの「レイアウト」ボタンをクリックしたときにだけ表示されます。

例

```
dataLayoutAvailable="false"
```

dividerLocation

チャートおよびグリッドを表示するために、使用可能な領域をペインに分割する位置を指定します。

データ・ソース

すべて

構文

```
dividerLocation="location"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	可能な値
location	.5	0 から 1 までの数値 (double 型)。

使用法

値の 1 は、左側 (または上側、splitPaneOrientation プロパティーの値による) の表示だけが現れることを意味します。値の 0 は、右側 (または下側) の表示だけが現れることを意味します。値の .5 は、領域が 2 つの表示で均等に分割されることを示します。

例

```
dividerLocation=".7"
```

関連項目

315 ページの『chartAvailable』、317 ページの『gridAvailable』、320 ページの『splitPaneOrientation』

enablePoppedOut

これは、ContainerBlox から継承されたプロパティーです。詳しい説明は、175 ページの『enablePoppedOut』を参照してください。

gridAvailable

PresentBlox 内でグリッドを使用できるかどうかを指定します。

データ・ソース

すべて

構文

```
gridAvailable="available"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	可能な値
available	true	グリッドを使用可能にするには、true を指定します。使用不可にするには、false を指定します。

例

```
gridAvailable="false"
```

関連項目

317 ページの『dividerLocation』

height

これは共通の Blox プロパティーです。詳しい説明は、42 ページの『height』を参照してください。

helpTargetFrame

これは共通の Blox プロパティーです。詳しい説明は、42 ページの『helpTargetFrame』を参照してください。

localeCode

これは共通の Blox プロパティーです。詳しい説明は、43 ページの『localeCode』を参照してください。

maximumUndoSteps

これは共通の Blox プロパティーです。詳しい説明は、44 ページの『maximumUndoSteps』を参照してください。

menubarVisible

これは共通の Blox プロパティーです。詳しい説明は、45 ページの『menubarVisible』を参照してください。

noDataMessage

これは共通の Blox プロパティーです。詳しい説明は、45 ページの『noDataMessage』を参照してください。

pageAvailable

PresentBlox 内でページ・パネルを使用できるかどうかを指定します。

データ・ソース

すべて

構文

```
pageAvailable="available"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	可能な値
available	true	この値が false に設定された場合、ユーザーが項目を「データ・レイアウト」パネル上の「ページ」軸にドラッグしてもページ・フィルターは表示されません。

例

```
pageAvailable="false"
```

poppedOut

これは、ContainerBlox から継承されたプロパティーです。詳しい説明は、176 ページの『poppedOut』を参照してください。

poppedOutHeight

これは、ContainerBlox から継承されたプロパティーです。詳しい説明は、176 ページの『poppedOutHeight』を参照してください。

poppedOutTitle

これは、ContainerBlox から継承されたプロパティーです。詳しい説明は、177 ページの『poppedOutTitle』を参照してください。

poppedOutWidth

これは、ContainerBlox から継承されたプロパティーです。詳しい説明は、178 ページの『poppedOutWidth』を参照してください。

render

これは共通の Blox プロパティーです。詳しい説明は、47 ページの『render』を参照してください。

splitPane

PresentBlox がインスタンス化されるとデータ表示領域がペインに分割されるかどうかを指定します。

データ・ソース

すべて

構文

```
splitPane="split"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	可能な値
split	true	ペインに分割するには、true を指定します。チャートとグリッドで共有する単一のデータ表示領域をレンダリングするには、false を指定します。

使用法

ユーザーは「グリッド」および「チャート」ツールバー・ボタンを使用して、これら 2 つのデータ・プレゼンテーションを切り替えます。

関連項目

317 ページの『dividerLocation』、320 ページの『splitPaneOrientation』

splitPaneOrientation

使用可能な領域をチャート用およびグリッド用のペインに分割する方法を指定します。

データ・ソース

すべて

構文

```
splitPaneOrientation="orientation"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	可能な値
orientation	vertical	Blox を (ポートレイトのように) 横並びに表示するには、vertical を指定します。または、Blox を (ランドスケープのように) 上下に重ねて表示するには、horizontal を指定します。

使用法

chartFirst プロパティーの値は、 ChartBlox または GridBlox のどちらが「最初に」(上側または左側に) 表示されるかを決めます。

例

```
splitPaneOrientation="horizontal"
```

関連項目

315 ページの『chartFirst』、319 ページの『splitPane』

toolbarVisible

ツールバーを表示するかどうかを指定します。

データ・ソース

すべて

構文

```
toolbarVisible="visible"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
visible	true	true: ツールバーが表示されます。 false: ツールバーは表示されません。

使用法

デフォルトでは、ツールバーは PresentBlox 内で表示されます。ネストされた <blox:toolbar> タグが追加された場合、その設定値はこの属性の値を上書きします。たとえば、次のコーディングではツールバーが表示されます。

```
<blox:present id="myPresent" toolbarVisible="false" ....>
   <blox:toolbar visible="true" />
   <blox:data bloxRef="myDataBlox"/>
</blox:chart>
```

ヒント: toolbarVisible は単にタグ属性であり、PresentBlox プロパティーではありません。

visible

これは共通の Blox プロパティーです。詳しい説明は、49 ページの『visible』を参照してください。

width

これは共通の Blox プロパティーです。詳しい説明は、49 ページの『width』を参照してください。

第 16 章 RepositoryBlox タグ・リファレンス

この章には、RepositoryBlox の参照資料が含まれています。Blox についての一般的な参照情報は、21 ページの『第 3 章 一般 Blox リファレンス情報』を参照してください。このリファレンスの使用方法については、1 ページの『第 1 章 このリファレンスの使用法』を参照してください。

RepositoryBlox の概説

RepositoryBlox は、DB2 Alphablox Repository 内に保管されているアプリケーション・プロパティーおよびさまざまなオブジェクトを、保管および検索する手段を開発者に提供します。この機能は、個別設定されたアプリケーションを作成するために重要です。RepositoryBlox のメソッドは、以下の 3 種類に分類されます。

- 複数のアプリケーション状態を保管および保守するためのもの
- アプリケーション、ユーザー、およびグループ・プロパティーへのアクセスを提供するもの
- DB2 Alphablox リポジトリーに格納されるオブジェクトの保管、およびそのオブジェクトへのアクセスを行うもの

複数のアプリケーション状態がリポジトリー内に存在する場合、ユーザーは必要なインスタンスを DB2 Alphablox ホーム・ページのアプリケーション・ページから選択できます。

ユーザー、アプリケーション状態、およびグループ・プロパティーを保管および検索するメソッドに加えて、RepositoryBlox はさまざまなタイプの Java オブジェクトを保管および検索するメソッドも提供します。これらのタイプは、`TYPE_BINARY`、`TYPE_TEXT`、 `TYPE_CONTAINER` (ディレクトリー内のサブフォルダー)、`TYPE_HASHTABLE` (オブジェクトの配列)、および `TYPE_XMLDOCUMENT` などの定数として表現されます。これにより、DB2 Alphablox リポジトリーを使用して保管および検索できるデータの種類の柔軟性および可能性が大幅に向上します。

注: パフォーマンスを向上させるために、グループ名はリポジトリーに保管されるときにすべて小文字に変換されます。

RepositoryBlox には、グラフィカル・ユーザー・インターフェースはありません。サーバー・サイド RepositoryBlox メソッドを呼び出すには、497 ページの『クライアント・サイド API 概説』で説明されている DHTML Client API を使用できます。

RepositoryBlox の JSP カスタム・タグ構文

DB2 Alphablox タグ・ライブラリーは、それぞれの Blox を作成するために JSP ページで使用するカスタム・タグを提供します。このセクションでは、RepositoryBlox を作成するためのカスタム・タグの作成方法について説明します。すべての属性を含むタグのコピー・アンド・ペースト・バージョンについては、528 ページの『RepositoryBlox JSP カスタム・タグ』を参照してください。

構文

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

ここで、それぞれ以下のとおりです。

attribute 属性表にリストされている属性の 1 つです。

value 属性の有効な値です。

属性は以下のいずれかになります。

属性
id
bloxName
render

使用法

各カスタム・タグには 1 つ以上の属性を含めることができ、それぞれを 1 つ以上のスペースまたは改行文字で区切ります。余分のスペースまたは改行文字は無視されます。読みやすくするため、同じインデントでそれぞれ別々の行に属性を並べることができます。終了部分の `</blox:repository>` タグは、省略表現を使用して置き換えることができます。これは、属性リストの末尾のタグを以下のように終了させるものです。

```
id="myRepositoryBlox" />
```

例

```
<blox:repository id="myRepository" />
```

RepositoryBlox タグ属性

このセクションでは、RepositoryBlox によってサポートされるタグ属性を説明します。すべての RepositoryBlox プロパティーは複数の Blox に共通で、このセクションでリストされていますが説明されてはいません。共通 Blox プロパティーの詳しい説明は、36 ページの『複数の Blox に共通するタグ属性』を参照してください。RepositoryBlox メソッドについては、Javadoc の com.alphablox.blox パッケージの RepositoryBlox クラスを参照してください。

id

これは共通の Blox タグ属性です。詳しい説明は、43 ページの『id』を参照してください。

bloxName

これは共通の Blox タグ属性です。詳しい説明は、39 ページの『bloxName』を参照してください。

render

これは共通の Blox プロパティーです。詳しい説明は、47 ページの『render』を参照してください。

第 17 章 ResultSetBlox タグ・リファレンス

この章には、ResultSetBlox の参照資料が含まれています。Blox についての一般的な参照情報は、21 ページの『第 3 章 一般 Blox リファレンス情報』を参照してください。このリファレンスの使用方法については、1 ページの『第 1 章 このリファレンスの使用法』を参照してください。

ResultSetBlox の概説

ResultSetBlox を DataBlox に付加すると、JDBC データ・ソースに関連した通常機能を拡張できます。ResultSetBlox を使用して、カスタム ResultSet を任意にプッシュして DataBlox に入れることができます。または、メソッドを Blox に付加して関連する DataBlox 内での照会をインターセプトすること、および任意の ResultSet オブジェクトを DataBlox に戻すことができます。

ResultSetBlox には、以下のように結果セットハンドラー・クラスの指定を可能にする `resultSetHandler` プロパティーがあります。

```
<blox:data id="rsData"
   dataSourceName="canned"
   connectOnStartup="false"
/>
...
<blox:resultSet id="rset1"
   dataBlox="<%=rsData%>"
   resultSetHandler="<%=new TupleResultSet()%>"
/>
```

ここで、`rsData` は事前に定義された DataBlox で、TupleResultSet は結果セット・ハンドラーです。このハンドラーは、java.sql.ResultSet オブジェクトを戻す `executeQuery()` メソッド、および結果セットからのデータが取り出されたときに接続を終了する `fetchComplete()` メソッドを提供する、 IResultSetHandler インターフェースをインプリメントする必要があります。

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

<%!
 public class TupleResultSet implements IResultSetHandler {
    ...
    // Store your custom query into the String myQuery

    Connection conn = null;
    public ResultSet executeQuery(String myQuery) throws Exception {
       //code here to get connected
    }

    public void fetchComplete() throws Exception {
       //close the connection
```

```
        conn.close();
        conn = null;
    }
%>
```

以下の点に気を付けてください。

- 上の例の DataBlox は、最初はデータ・ソースに接続しません (`connectionOnStartup = "false"`)。これは、初期の結果セットを取得したくない場合、およびユーザーが選択を行うことにより照会ストリングを動的に作成して、その照会を JDBC 接続を使用して発行したい場合に、有用な手法です。
- java.sql.* のインポート・ステートメントを追加する必要があります。
- 結果セットから取得するデータのタイプに応じて、 java.sql.ResultSet 上に最低限インプリメントしなければならない API があります。これらの API は、次のセクションにリストされています。

完全な実例については、Blox Sampler の『Retrieving Data』セクションを参照してください。

ResultSet にインプリメントする最小の API

結果セットから取得するデータのタイプに応じて、 java.sql.ResultSet 上にインプリメントしなければならない最低限の API のリストを以下に示します。

- `next()` : void
- 以下のように、結果セットが戻すデータ・タイプに応じてインプリメントする getter メソッド。
  - `getInt(int)`: Integer
  - `getBoolean(int)`: Boolean
  - `getBigDecimal(int)`: BigDecimal
  - `getFloat(int)`: Float
  - `getDouble(int)`: Double
  - `getString(int)`: String
  - `getDate(int)`: Date
  - `getObject(int)`: Object
  - `getMetaData()`: java.sql.ResultSetMetaData

戻される結果セットのタイプが java.sql.ResultSetMetaData である場合、以下のメソッドをインプリメントしてください。

- `getColumnCount()` : int
- `getColumnType(int)` : int
- `getScale(int)` : int
- `getPrecision(int)` : int
- `getColumnName(int)` : String
- `getColumnLabel(int)`: String
- `getColumnTypeName(int)` : String
- `getColumnType(int)` : int

ResultSetBlox の JSP カスタム・タグ構文

DB2 Alphablox タグ・ライブラリーは、それぞれの Blox を作成するために JSP ページで使用するカスタム・タグを提供します。このセクションでは、ResultSetBlox を作成するためのカスタム・タグの作成方法について説明します。すべての属性を含むタグのコピー・アンド・ペースト・バージョンについては、 529 ページの『ResultSetBlox JSP カスタム・タグ』を参照してください。

構文

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

ここで、それぞれ以下のとおりです。

attribute 属性表にリストされている属性の 1 つです。

value 属性の有効な値です。

属性は以下のいずれかになります。

属性
id
bloxName
dataBlox
resultSetHandler

使用法

各カスタム・タグには 1 つ以上の属性を含めることができ、それぞれを 1 つ以上のスペースまたは改行文字で区切ります。余分のスペースまたは改行文字は無視されます。読みやすくするため、同じインデントでそれぞれ別々の行に属性を並べることができます。終了部分の `</blox:resultSet>` タグは、省略表現を使用して置き換えることができます。これは、属性リストの末尾のタグを以下のように終了させるものです。

```
id="myResultSet" />
```

例

```
<blox:resultSet id="myResultSet" />
```

ResultSetBlox タグ属性

このセクションでは、ResultSetBlox によってサポートされるタグ属性を説明します。複数の Blox に共通するタグ属性は、このセクションでリストされていますが説明されてはいません。共通 Blox プロパティーの詳しい説明は、36 ページの『複数の Blox に共通するタグ属性』を参照してください。 ResultSetBlox メソッドについては、Javadoc の com.alphablox.blox パッケージの ResultSetBlox クラスを参照してください。

id

これは共通の Blox タグ属性です。詳しい説明は、43 ページの『id』を参照してください。

bloxName

これは共通の Blox タグ属性です。詳しい説明は、39 ページの『bloxName』を参照してください。

dataBlox

この ResultSetBlox に関連した DataBlox。

データ・ソース

リレーショナル

構文

```
dataBlox="dataBlox"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
dataBlox		この ResultSetBlox が接続された DataBlox。

resultSetHandler

照会を取得して結果セットを戻す `executeQuery()` メソッドをインプリメントするハンドラー。

データ・ソース

リレーショナル

構文

```
resultSetHandler="handler"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
handler		照会を実行するためにインプリメントするハンドラー。

使用法

このハンドラーは、メソッドを IResultSetHandler にインプリメントする必要があります。Javadoc の com.alphablox.blox パッケージの IResultSetHandler インターフェースを参照してください。

第 18 章 StoredProceduresBlox タグ・リファレンス

この章には、ストアード・プロシージャーを使用するための StoredProceduresBlox の参照資料が含まれています。

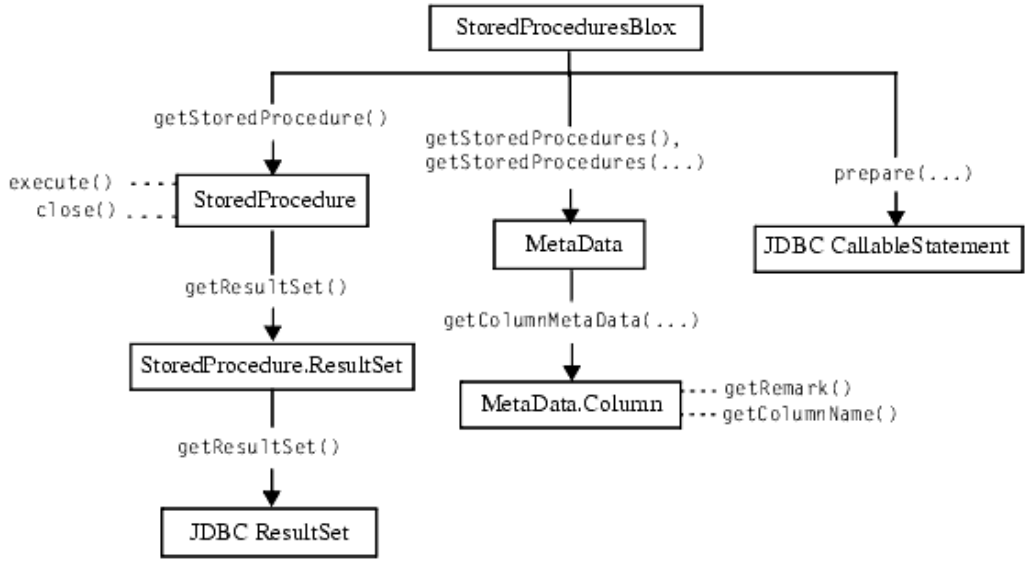
StoredProceduresBlox の概説

StoredProceduresBlox は、リレーショナル・データベースのストアード・プロシージャーを使用するための開始点となります。これにより、データベースへの接続を作成し、ストアード・プロシージャー・ステートメントを準備できます。適正な DB2 Alphablox データ・ソースおよび他の接続パラメーターが設定された後、以下のことが可能になります。

- `prepare(...)` メソッドを使用して JDBC CallableStatement オブジェクトを戻すこと。このオブジェクトは、ストアード・プロシージャーの実行に必要なストアード・プロシージャー・パラメーターのセットアップに使用できます。
- `getStoredProcedure()` メソッドを使用して現行の StoredProcedure オブジェクトにアクセスすること。その後、ストアード・プロシージャーの実行、実行したストアード・プロシージャーの ResultSet の取得、または JDBC ResultSet へのアクセスが可能になります。
- `getStoredProcedures()` または `getStoredProcedures(...)` メソッドを使用して、開発者に個別のパラメーターへのアクセスを付与する 1 つ以上の MetaData オブジェクトを戻すこと。

StoredProcedure オブジェクトと MetaData オブジェクトは、`com.alphablox.blox.data.rdb.storedprocedure` パッケージ内の別々のクラスです。StoredProcedure と MetaData とで StoredProceduresBlox からの別々のオブジェクトを使用することにより、ストアード・プロシージャーを 1 回準備した後にそれを複数回実行することが可能になります。ストアード・プロシージャー・パラメーターは実行ごとに変更することができますが、毎回の実行でストアード・プロシージャーを準備しないようにすれば、パフォーマンスを向上させることができます。

以下の図は、ストアード・プロシージャーに関連したオブジェクトのオブジェクト階層を示しています。



StoredProcedure オブジェクトと MetaData オブジェクトとは別々のパッケージに含まれているため、これらのオブジェクト内の API を使用するためには、 JSP ファイルの最初で以下の JSP インポート・ステートメントを使用する必要があります。

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

注: JDBC ストアード・プロシージャーは、IBM DB2 UDB、Sybase、 Oracle、および Microsoft SQL Server データベースでサポートされています。

StoredProcedure オブジェクトを使用して準備済みストアード・プロシージャーを実行する際には、以下の事柄に注意してください。

- DataBlox を使用してストアード・プロシージャーの情報を表示する場合、DataBlox は StoredProceduresBlox と同じデータ・ソースに対して別個に接続されている必要があります。
- DataBlox を使用してストアード・プロシージャーの情報を表示する場合で、そのストアード・プロシージャーにも出力パラメーターがあるときには、出力パラメーターを取得する前にまず結果セットを使用しなければなりません。これは JDBC の制約事項です。
- ストアード・プロシージャーに入力パラメーターおよび出力パラメーターがある場合、 `StoredProceduresBlox.prepare(...)` を使用して JDBC CallableStatement オブジェクトを取得してください。このオブジェクトにより、ストアード・プロシージャー上の入力パラメーターおよび出力パラメーターを取得して設定することが可能になります。
- ストアード・プロシージャーが実行されて、出力パラメーターまたは結果セットが使用された後、 `StoredProceduresBlox.disconnect()` を呼び出してリソースを切断および解放する必要があります。データベースへの接続を開いたままにしたい場合、 `StoredProceduresBlox.close()` を呼び出して、使用されたリソースを解放してください。
- DataException がスローされた場合、`DataException.getNestedException()` を調べると、追加の情報を SQLException として入手できることがあります。

ストアード・プロシージャーが実行された後、それは StoredProcedure.ResultSet オブジェクトを戻します。このオブジェクトは開発者に JDBC ResultSet オブジェクトへのアクセスを付与します。JDBC ResultSet オブジェクトを直接使用する必要がある場合、ResultSet.getResultSet() メソッドを使用してこのオブジェクトを取得してください。

ストアード・プロシージャーを取り扱うときには、java.sql パッケージもインポートすることが推奨されています。そのため、JSP ファイルは以下の 2 つのパッケージをインポートすることになります。

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

StoredProceduresBlox JSP カスタム・タグ構文

DB2 Alphablox タグ・ライブラリーは、それぞれの Blox を作成するために JSP ページで使用するカスタム・タグを提供します。このセクションでは、StoredProceduresBlox を作成するためのカスタム・タグの作成方法について説明します。

構文

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

ここで、それぞれ以下のとおりです。

attribute 属性表にリストされている属性の 1 つです。

value 属性の有効な値です。

属性は以下のいずれかになります。

属性
id
bloxName

使用法

各カスタム・タグには 1 つ以上の属性を含めることができ、それぞれを 1 つ以上のスペースまたは改行文字で区切ります。余分のスペースまたは改行文字は無視されます。読みやすくするため、同じインデントでそれぞれ別々の行に属性を並べることができます。

終了部分の </blox:storedProcedures> タグは、省略表現を使用して置き換えることができます。これは、属性リストの末尾のタグを以下のように終了させるものです。

```
id="myBlox" />
```

例

```
<blox:storedProcedures
     id="namedStoredProceduresBlox" />
```

StoredProceduresBlox の例

このセクションには、StoredProceduresBlox とその関連オブジェクトの使用方法を例示する 6 つの例があります。その他の例については、Javadoc を参照してください。

- 334 ページの『例 1: DataBlox のないデータ・ソースに接続する』
- 334 ページの『例 2: StoredProceduresBlox を使用して DataBlox と共に使用するデータ・ソースに接続する』
- 335 ページの『例 3: 名前が指定のパターンと一致するストアード・プロシージャーのリストを取得する』
- 335 ページの『例 4: ストアード・プロシージャーごとのすべてのパラメーターのリストを取得する』
- 336 ページの『例 5: 1 つの入力パラメーターと 2 つの出力パラメーターがあるストアード・プロシージャーを実行する』
- 337 ページの『例 6: DataBlox へのストアード・プロシージャー結果セットを設定する』

例 1: DataBlox のないデータ・ソースに接続する

この例では、DataBlox のないデータ・ソースに接続する方法を示します。この方法は、DataBlox を必要しないパラメーターを取得したり INSERT SQL ストアード・プロシージャーを実行する際に使用します。

```
<%@ 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();
%>
```

例 2: StoredProceduresBlox を使用して DataBlox と共に使用するデータ・ソースに接続する

この例では、ストアード・プロシージャーの情報の表示に使用される DataBlox を、 StoredProceduresBlox と同じデータ・ソースに別個に接続する方法を示します。

```
<%@ 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();
%>
```

例 3: 名前が指定のパターンと一致するストアード・プロシージャーのリストを取得する

この例では、getStoredProcedures(...) メソッドを使用して、名前が ″procedure″ で始まるストアード・プロシージャーのリストを取得する方法を示します。このメソッドは、MetaData オブジェクトの配列を戻します。MetaData オブジェクトには、各ストアード・プロシージャーのパラメーターに関する情報が含まれます。

```
<%@ 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> <%
} %>
```

MetaData オブジェクトによって、指定されたストアード・プロシージャーの個別のパラメーターにアクセスできます。

例 4: ストアード・プロシージャーごとのすべてのパラメーターのリストを取得する

この例は、MetaData オブジェクトを使用して各ストアード・プロシージャーおよび各ストアード・プロシージャーのパラメーターを取得する方法を示しています。この例では、前の例で示されているように、開発者が MetaData オブジェクトの戻りをすでに取得していると想定します。

```
MetaData procedures[] =
        mySP.getStoredProcedures("procedure%");
```

ここで、各ストアード・プロシージャーおよびそのカタログ、スキーマ、名前、および注釈情報を 1 つの表にリストします。

```
<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>
```

さらに、ストアード・プロシージャーごとの各パラメーターの詳細も取得できます。

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

例 5: 1 つの入力パラメーターと 2 つの出力パラメーターがあるストアード・プロシージャーを実行する

この例では、prepare() メソッドを使用して、入力パラメーターと出力パラメーターのあるストアード・プロシージャーの実行に使用できる JDBC CallableStatement オブジェクトを戻す方法を示します。

```
<%@ 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();
%>
```

例 6: DataBlox へのストアード・プロシージャー結果セットを設定する

この例では、DataBlox へのストアード・プロシージャー結果セットを取得する方法を示します。

```
<%@ 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);
%>
```

StoredProceduresBlox タグ属性

このセクションでは、StoredProceduresBlox によってサポートされるタグ属性を説明します。 StoredProceduresBlox メソッドについては、Javadoc の com.alphablox.blox パッケージの StoredProceduresBlox クラスを参照してください。

id

これは共通の Blox タグ属性です。詳しい説明は、43 ページの『id』を参照してください。

bloxName

これは共通の Blox タグ属性です。詳しい説明は、39 ページの『bloxName』を参照してください。

第 19 章 ToolbarBlox タグ・リファレンス

この章には、ToolbarBlox の参照資料が含まれています。Blox についての一般的な参照情報は、21 ページの『第 3 章 一般 Blox リファレンス情報』を参照してください。このリファレンスの使用方法については、1 ページの『第 1 章 このリファレンスの使用法』を参照してください。

ToolbarBlox の概説

ToolbarBlox は、カスタマイズされた Blox ツールバーを提示します。これは次の 2 つの方法で追加されます。

- PresentBlox、ChartBlox、または GridBlox の中で、ネストされた `<blox:toolbar>` タグを使用する。
- `<blox:present>`、`<blox:chart>`、または `<blox:grid>` タグで、`toolbarVisible` タグ属性を `true` に設定する。

DHTML クライアントでは、スタンドアロンの ToolbarBlox を使用できません。デフォルトで、ツールバーは PresentBlox、スタンドアロンの GridBlox、およびスタンドアロンの ChartBlox 内での使用および表示が可能です。

グラフィカル・ユーザー・インターフェース

ToolbarBlox は DHTML クライアント内に、標準ツールバーおよびナビゲーション・ツールバーの 2 つのツールバーと共に表示されます。デフォルトで、これら 2 つのツールバーには以下のボタンが含まれています。

- ポップアウト
- コピー
- 再実行
- 取り消し
- ブックマークのロード
- PDF にエクスポート
- Excel にエクスポート
- ヘルプ
- データ・ナビゲーション
- ソート
- メンバー・フィルター
- グリッド
- チャート

- ページ・フィルター
- データ・レイアウト・パネル

ボタンおよびツールバーは、完全にカスタマイズ可能です。Blox UI Tag Library には、ツールバーまたはツールバー・ボタンを追加、編集、または除去するためのタグが備わっています。詳しくは、417 ページの『第 23 章 Blox UI タグ・リファレンス』を参照してください。

ToolbarBlox ユーザー・インターフェースの使用方法については、オンライン・ユーザー・ヘルプを参照してください。ユーザー・ヘルプにアクセスするには、Blox ユーザー・インターフェースのツールバーにある「ヘルプ」ボタンをクリックします。

ToolbarBlox JSP カスタム・タグ構文

DB2 Alphablox タグ・ライブラリーは、それぞれの Blox を作成するために JSP ページで使用するカスタム・タグを提供します。このセクションでは、 PresentBlox、GridBlox、または ChartBlox 内でツールバーを作成するためのカスタム・タグの作成方法について説明します。すべての属性を含むタグのコピー・アンド・ペースト・バージョンについては、530 ページの『blox.tld 内のその他のタグ』を参照してください。

パラメーター

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

ここで、それぞれ以下のとおりです。

attribute	属性表にリストされている属性の 1 つです。
value	属性の有効な値です。

属性は以下のいずれかになります。

属性
id
applyPropertiesAfterBookmark
bloxEnabled
bloxName
bookmarkFilter
helpTargetFrame
localeCode
removeAction
removeButton
rolloverEnabled
textVisible
toolTipsVisible

属性
visible

使用法

各カスタム・タグには 1 つ以上の属性を含めることができ、それぞれを 1 つ以上のスペースまたは改行文字で区切ります。余分のスペースまたは改行文字は無視されます。読みやすくするため、同じインデントでそれぞれ別々の行に属性を並べることができます。

終了タグ </blox:toolbar> は、終了スラッシュで置き換えられます。ただし、タグの最後の属性と終了文字 ″>″ の間に置く必要があります。たとえば、最後の属性が width の場合、タグの最後は以下のようになります。

```
width="650" />
```

例

```
<blox:toolbar
   id="myToolbar1"
   toolTipsVisible="false">
</blox:toolbar>
```

カテゴリー別の ToolbarBlox タグ属性

以下は、固有の ToolbarBlox タグ属性です。複数の Blox に共通するタグ属性については、35 ページの『カテゴリー別の共通の Blox タグ属性』を参照してください。 ToolbarBlox によってサポートされるタグ属性は、以下のように相互参照に編成されています。

- 341 ページの『外観』
- 341 ページの『内容』

外観

以下のタグ属性は、ToolbarBlox の外観に影響を与えます。

- rolloverEnabled
- textVisible
- toolTipsVisible

内容

以下のタグ属性は、ToolbarBlox の内容に影響を与えます。

- removeButton

ToolbarBlox タグ属性

このセクションでは、ToolbarBlox によってサポートされるタグ属性を説明します。ToolbarBlox から使用可能な共通 Blox プロパティーについては、リストしますが解説はしません。共通 Blox プロパティーの詳しい説明は、36 ページの『複数の Blox に共通するタグ属性』を参照してください。

id

これは共通の Blox プロパティーです。詳しい説明は、43 ページの『id』を参照してください。

applyPropertiesAfterBookmark

これは共通の Blox プロパティーです。詳しい説明は、36 ページの『applyPropertiesAfterBookmark』を参照してください。

bloxEnabled

これは共通の Blox プロパティーです。詳しい説明は、38 ページの『bloxEnabled』を参照してください。

bloxName

これは共通の Blox プロパティーです。詳しい説明は、39 ページの『bloxName』を参照してください。

bookmarkFilter

これは共通の Blox プロパティーです。詳しい説明は、37 ページの『bookmarkFilter』を参照してください。

helpTargetFrame

これは共通の Blox プロパティーです。詳しい説明は、42 ページの『helpTargetFrame』を参照してください。

localeCode

これは共通の Blox プロパティーです。詳しい説明は、43 ページの『localeCode』を参照してください。

removeAction

これは共通の Blox プロパティーです。詳しい説明は、46 ページの『removeAction』を参照してください。

removeButton

ToolbarBlox から (ユーザーに表示される前に) 除去するボタンを識別します。

データ・ソース

すべて

構文

```
removeButton = "removeButton"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
removeButton	"Save,Load"	コンマで区切られたボタン名のストリング。

使用法

値は引用符で囲まれてコンマで区切られたボタンのリストで、たとえば “Save,Load” などです。この場合、それらのボタンがツールバーから除去されて、アプリケーション状態の保管/ロード機能へのユーザー・アクセスが除去されます。ボタン名として使用可能な値は、 Chart、Layout、Grid、Swap、Bookmark、Help、Load、および Save です。

ヒント: setRemoveButton() メソッドに指定するリストは、デフォルトを上書きします。「保管」および「ロード」ボタンを表示したくない場合、除去するボタンのリストにそれらを必ず含めてください。ここにリストされていないボタンを除去するには、Blox UI タグを使用します。454 ページの『カスタム・ツールバーのタグ』を参照してください。

ヒント: 「保管」および「ロード」ボタンによって、ユーザーはアプリケーションの状態をプライベートまたはパブリックとして保管できます。これはブックマーク機能と似ています。異なる点は、ネストされていない複数のプレゼンテーション Blox があるとき、「保管」および「ロード」ボタンはページ上のすべての Blox の状態を自動的に保管するので、ブックマーク機能の場合のように状態を保管したい Blox を指定する必要がないことです。保管されたアプリケーション状態はブックマークとは別に管理されるので、ブックマークまたはアプリケーション状態の保管/ロード機能のどちらかだけを提供することにより、混乱を回避できます。

例

```
removeButton="Chart,Save,Load"
```

rolloverEnabled

マウスがボタンの上に移動したとき、ツールバー・ボタンの色がグレースケールからカラーに変化するようにするかどうかを指定します。

データ・ソース

すべて

構文

```
rolloverEnabled = "boolean"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
enable	false	ブール引数。値の true は、ロールオーバー効果を示すためにマウスオーバーでツールバー・ボタンのイメージが変化することを示します。値の false は、ツールバー・ボタンのイメージが変化しないことを示します。

使用法

このプロパティー値が true に設定されている場合、マウスオーバーで、ツールバー・ボタンにはロールオーバー効果が示されます。ツールバー・ボタンを <bloxui:toolbarButton> タグによって追加した場合、このプロパティーを true に設定したときには、マウスオーバー用のイメージを "_active" サフィックスを付けて供給する必要があります。詳しくは、454 ページの『カスタム・ツールバーのタグ』を参照してください。

例

```
rolloverEnabled="false"
```

textVisible

ツールバー・ボタンにあるアイコンの下にテキスト・ラベルを表示するかどうかを指定します。

データ・ソース

すべて

構文

```
textVisible = "boolean"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
visible	false	ブール引数。値の true は、テキスト・ラベルを表示することを指定します。値の false は、表示しないことを指定します。

使用法

値が true に設定されていると、ツールバー・ボタンにあるアイコンの下にテキスト・ラベルが表示されます。

例

```
textVisible="false"
```

toolTipsVisible

ユーザーがマウスをツールバー・ボタンの上に移動したときに、説明テキストが表示されるようにするかどうかを指定します。

データ・ソース

すべて

構文

```
toolTipsVisible = "boolean"
```

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
visible	true	ブール値。値の true は、ツールバーにマウスオーバーしたときにツールチップを表示することを指定します。値の false は、ツールチップを表示しないことを指定します。

使用法

値が true に設定されていると、ユーザーがツールバー・ボタンにマウスオーバーしたときに、説明テキストが表示されます。

例

343 ページの『rolloverEnabled』、345 ページの『toolTipsVisible』

visible

これは共通の Blox プロパティーです。詳しい説明は、49 ページの『visible』を参照してください。

第 20 章 Blox Form タグ・リファレンス

さまざまな FormBlox を使用して、ページをリフレッシュしなくても、HTML フォームに似たユーザー・インターフェースを JSP に追加して、フォーム・エレメントをページ上のサーバー・サイド・コンポーネントまたは他のフォーム・コンポーネントにリンクできます。これらの FormBlox を追加するタグは、Blox Form Tag Library (bloxform.tld) に含まれています。この章には、このライブラリー内のタグに関する参照資料が含まれています。 API の詳細なリストは、Javadoc 内の com.alphablox.blox.form パッケージを参照してください。

FormBlox の概説

FormBlox およびビジネス・ロジック Blox (389 ページの『第 21 章 ビジネス・ロジック Blox および TimeSchema DTD リファレンス』 で説明されている) は、データ認識ビジネス・ロジックの必要、および状態を維持する必要という、分析的なアプリケーション開発の際に一般的に生じる 2 つの問題を解決するために設計されています。一連の専門化された FormBlox により、 Blox Form Tag Library (bloxform.tld) を使用するだけで、時間枠、データ・ソース、キューブ、ディメンション、およびメンバー選択リストを作成できます。

- これらの Blox を使用して、ラジオ・ボタン、チェック・ボックス、編集フィールドなどの、標準の HTML フォーム・エレメントに似たユーザー・インターフェースを作成できます。
- 汎用 HTML フォーム・エレメントとは異なり、FormBlox はセッション中にページが再ロードした後、自動的に状態を保守します。

- FormBlox は、指定のデータ・ソース、キューブ、ディメンション、またはタイム・スキーマに基づいて、選択リストに自動的にデータを追加することができます。

そのため、複雑な時系列計算の実行、ユーザー選択リストにデータを追加するためのメタデータの検索、またはフォーム・エレメントの状態の管理を行うために、コードを記述する必要はありません。

FormBlox のバリエーション

さまざまなバリエーションの FormBlox があり、それぞれは他の FormBlox またはサーバー・サイド・コンポーネントにリンクできる、特定のユーザー・インターフェースを追加するように設計されています。以下の表は、すべての FormBlox をリストして、それぞれの目的を説明してます。

FormBlox	説明
汎用 HTML フォーム・エレメント用の FormBlox	
CheckBoxFormBlox	HTML <input type="checkbox"...> タグの FormBlox インプリメンテーション。 ただし、ボックスにチェックが入れられていない場合には通知されたときに値を送信しない HTML フォームとは異なり、 CheckBoxFormBlox は常にフォーム・ポスト上の値を戻します。
EditFormBlox	HTML 編集フィールド (<input type="text" ...> または <textarea ...> のいずれか) の FormBlox インプリメンテーション。
RadioButtonFormBlox	HTML ラジオ・ボタン・セット (<input type="radio" ... >) の FormBlox インプリメンテーション。
SelectFormBlox	HTML <select> エレメントの FormBlox インプリメンテーション。単一選択と複数選択の両方をサポートします。
MetaData 選択リスト用の FormBlox	
DataSourceSelectFormBlox	使用可能なデータ・ソースを表示する、HTML <select> リストの FormBlox インプリメンテーション。マルチディメンションまたはリレーショナル・データ・ソースに限定できます。
CubeSelectFormBlox	指定のマルチディメンション・データ・ソースで使用可能なキューブを表示する、HTML <select> リストの FormBlox インプリメンテーション。
DimensionSelectFormBlox	指定のマルチディメンション・キューブで使用可能なディメンションを表示する、HTML <select> リストの FormBlox インプリメンテーション。
MemberSelectFormBlox	指定のマルチディメンション・データ・ソース・ディメンションからメンバーを表示する、 HTML <select> エレメントの専門化されたインプリメンテーション。 DataBlox、キューブ (必要であれば)、およびディメンションが指定されると、ディメンション内のメンバーを含む選択リストを表示します。
TimeSchema 関連の選択リスト用の FormBlox	

FormBlox	説明
TimePeriodSelectFormBlox	タイム・スキーマで使用可能な TimeSeries を表示する、HTML <select> エレメントの FormBlox インプリメンテーション。TimeSeries は、最近 2 カ月、最近の 1 四半期、最近の 2 四半期、現在までの月、現在までの四半期、現在の月、および現在の週などの、時間の期間です。
TimeUnitSelectFormBlox	指定のタイム・スキーマからの期間タイプを表示する、HTML <select> エレメントの専門化されたインプリメンテーション。時間単位 (PeriodType) は、年、半年期、四半期、月、週、日などの、タイム・スキーマで使用される単位です。
ナビゲーション・ツリー用の FormBlox	
TreeFormBlox	Blox UI ツリー・コントロールの FormBlox カプセル化。DHTML ツリー・コントロールは、ページにレンダリングされます。

FormBlox 関連のすべてのクラスは、com.alphablox.blox.form パッケージの下にあります。それらのタグは、bloxform.tld タグ・ライブラリーで入手可能です。

共通の FormBlox プロパティーおよび属性

FormBlox クラスは、すべての FormBlox の基本クラスです。そのため、すべての FormBlox は共通のプロパティー、メソッド、タグ、および振る舞いを共有します。

- それらは、同じイベント・モデル FormEventListener を使用します。これが、すべての FormBlox イベントが取り扱われる方法です。
- それらはフォーム POST を使用して、値を通知します (TreeFormBlox を除く)。
- それらにはすべて、以下のタグ属性があります。

共通 FormBlox 属性	説明
id	このページにレンダリングされるオブジェクトの id。bloxName 属性が指定されている場合を除いて、これは bloxName でもあります。
bloxName	サーバー (ピア) 上のオブジェクトの名前。
formElementName	フォーム POST で使用される、レンダリングされるページ・エレメント名およびパラメーター名。
themeClass	テーマ・クラスの名前。
visible	オブジェクトが所定の場所にレンダリングされる場合は true。デフォルトは true です。

FormBlox のイベント

com.alphablox.blox.form 内の FormEventListener インターフェースは、すべての FormBlox のためのイベント・ハンドラーです。 addFormEventListener() および removeFormEventListener() メソッドを使用して、 FormEventListener を追加/除去することにより、イベント処理を使用可能/使用不可にすることができます。イベント処理の際に、FormBlox が変更される (たとえば、チェック・ボックスがチェック

される/チェックを外される、ラジオ・ボタンがクリックされる、選択リストで選択が行われる、など) と常に FormEventListener.valueChanged() メソッドが呼び出されます。

getChangedProperty および setChangedProperty タグは、基本的なイベント処理をサポートしています。そして多くの場合、イベント・ハンドラーの記述を不要にすることができます。 getChangedProperty タグおよび setChangedProperty タグで十分となるのは、 1 つのオブジェクト上のプロパティーが他の Java Bean 上の対応するプロパティーを常に変更する場合です。

setChangedProperty タグ

FormBlox は、任意の Java Bean 上のプロパティーを設定できます。<bloxform:setChangedProperty> タグにより、 FormBlox が変更されたときに、どのプロパティーを選択された新しい値に変更するかを指定できます。グリッドに対して代替の行カラーを使用可能にするかどうかをユーザーが選択できる、チェック・ボックスを検討してください。そのチェック・ボックスは、以下のように CheckBoxFormBlox を使用して追加されます。

```
<bloxform:checkBox id="bandingCheckBox"
   checked="false"
   checkedValue="true"
   uncheckedValue="false">
   <bloxform:setChangedProperty
      targetRef="myGridBlox"
      targetProperty="bandingEnabled" />
</bloxform:checkBox> Enable alternate row banding
```

このチェック・ボックスは、ページにレンダリングされるときにチェックが外されています。ユーザーがボックスにチェックを入れると、チェックされた値 (checkedValue) は myGridBlox (targetRef) の bandingEnabled プロパティー (targetProperty) に設定されます。

getChangedProperty タグ

<bloxform:getChangedProperty> タグは、FormBlox 用のタグ内でネストされていて、 FormBlox 上のプロパティーが別の FormBlox 上にある対応するプロパティーの変更時に常に変更されるように設定します。たとえば、このタグを使用して複数の FormBlox をリンクすることにより、 1 つの FormBlox での選択が別の FormBlox での選択可能項目を設定するようにすることができます。一般的なシナリオは、いわゆる「カスケード・メニュー」です。カスケード・メニューでは、最初のメニューのオプションから選択を行うと、それに応じて次のメニューでの選択可能なオプションが決まります。

たとえば、ユーザーが都市を選択して売上データを表示するためのメニューのセットがあるレポートを考えます。カスケード・メニューは、ゾーン・メニューから開始します。ゾーンを選択すると、それに応じて 2 番目のメニューで選択可能になるエリアが決まります。エリアを選択すると、それに応じて後続のメニューで選択可能になる都市が決まります。以下の例では、 All Locations ディメンション内の世代 2 のメンバーを取得することにより、ゾーン・メニューを作成します。エリア・メニューは、選択したメンバーをゾーン・メニューから取得することにより作成します。コード断片を以下に示します。

```
<%--The zone menu: displaying all generation 2 members of
    the All Locations dimension. Note that for MSAS data sources
    the member name should be enclosed in square brackets (unique
    names). --%>
<bloxform:memberSelect id="zone"
   dataBloxRef="myData"
   dimensionName="All Locations"
   filterOperator="=="
   filterGeneration="2">
</bloxform:memberSelect>

<%--The area menu: displaying all generation 3 members of
    the selected member from the zone menu. --%>
<bloxform:memberSelect id="area"
   dataBloxRef="myData"
   dimensionName="All Locations"
   filterOperator="=="
   filterGeneration="3">
   <bloxform:getChangedProperty
      formBloxRef="zone"
      formProperty="selectedMembers"
      property="rootMembers" />
</bloxform:memberSelect>
```

FormPropertyLink オブジェクト

com.alphablox.blox.form パッケージ内の FormPropertyLink クラスは、getChangedProperty タグおよび setChangedProperty タグの背後にあるオブジェクトです。これを使用して FormBlox を相互にリンクし、基本プロパティーを相互間で転送します。

FormBlox 上でプロパティーが変更されると常に、FormPropertyLink がターゲット Bean に新規の値を設定します。これは通常の Java Bean イントロスペクションを使用しているので、ターゲットは FormBlox だけではなく任意の Java Bean であることができます。必要であれば、プロパティーの変更後に Bean 上の 1 つの追加メソッドを呼び出すことができます。これにより、リンクが DataBlox 上の query プロパティーに対する変更などのケースを取り扱うことが可能になります。この場合、変更を完了するためには、照会プロパティーを変更した後に DataBlox の updateResultSet() メソッドを呼び出す必要があります。

以下のように、タグを使用して 2 つの異なるプロパティーをリンクすると、FormPropertyLink は異なるデータ・タイプの変換を自動的に行います。

- 呼び出し元の引数と呼び出し先の予想パラメーターとが同じタイプである場合、引数は現状のまま呼び出し先に渡されます。
- 呼び出し元の引数が配列であり、期待の予想パラメーターが配列ではない場合、渡される配列の最初のエレメントが呼び出し先に渡されます。
- 呼び出し元の引数が配列ではなく、呼び出し先が配列を予想している場合、引数は長さ 1 の配列に変換されて呼び出し先に渡されます。
- 呼び出し元の引数が String であり、呼び出し先がブールを予想している場合、ストリングはブールに変換されます。たとえば、ストリング "true" はブール true に変換されて、呼び出し先に渡されます。
- 呼び出し元の引数が非プリミティブ Java オブジェクトであり、呼び出し先がストリングを予想している場合、引数は toString() を使用して String に変換されてから呼び出し先に渡されます。

スタイル設定の FormBlox

すべての FormBlox には、コンポーネントのテーマ・クラスを指定するための、themeClass プロパティーおよび themeClass タグ属性があります。以下の TreeFormBlox は、 myMenuTree と呼ばれるスタイル・クラスを使用してメニュー項目テキストのスタイルを設定します。

```
<!--some code ommitted here...>
<head>
   <blox:header/>
   <style>
   .myMenuTree { background-color: #FFFF80; }
   </style>
</head>
<body>
<bloxform:tree id="myMenu" rootVisible="false" themeClass="myMenuTree">
   <bloxform:folder> <%--root folder--%>
      <bloxform:folder label="Sales Analysis">
         <bloxform:item label="Sales Trend by Region"
            href="salesByRegion.jsp"
            target="mainFrame" />
         <bloxform:item label="Sales by Store"
            href="salesByStore.jsp"
            target="mainFrame" />
         <bloxform:item label="Units Sold by Product"
            href="unitsSoldByProduct.jsp"
            target="mainFrame" />
      </bloxform:folder>
   </bloxform:folder>
<!--more code omitted here-->...
```

<alphablox_dir>/repository/theme/<themeName> 内の <themeName>_dhtml.css ファイルで定義された DB2 Alphablox テーマ・クラスを使用することもできます。これにより、アプリケーション全体で一貫性のあるルック・アンド・フィールを提供できます。以下の例は、csSlctBg と呼ばれる定義済みテーマ・クラスを使用する SelectFormBlox を示しています。

```
<!--some code omitted here-->...
<b>Select Chart Type:</b><br>
<bloxform:select id="ChartSelection" size="4"
   themeClass="csSlctBg">
   <bloxform:option label="Bar" value="Bar" selected="true"/>
   <bloxform:option label="Pie" value="Pie" />
   <bloxform:option label="Line" value="Line" />
   <bloxform:option label="3D Bar" value="3D Bar" />
   <bloxform:setChangedProperty
      targetRef="myChart"
      targetProperty="chartType" />
</bloxform:select>
<!--more code omitted here-->......
```

CSS テーマがサポートされて使用される方法についての詳細は、「*開発者用ガイド*」の『データの提示』の章を参照してください。そこには、DB2 Alphablox テーマでサポートされるスタイル・クラスのリストも含まれています。

選択リストを作成する FormBlox

多くの FormBlox は、選択リストを作成します。さまざまなバリエーションの SelectFormBlox はすべて、単一選択リストの場合に選択オプションを明示的に設定しなければ、リストの最初のオプションがデフォルト選択オプションとなる点で、類似の振る舞いをします。 DataSourceSelectFormBlox および

TimePeriodSelectFormBlox を除いて、これらの Blox には multipleSelect タグ属性および size タグ属性があります。 選択リストの size が 1 よりも大きいとき、または複数選択が供されているときには、少なくとも 1 つのオプションを初期選択として設定しないとエラーが生じます。これらの Blox のほとんどが DataBlox と結合していて、それらのインスタンス化にはデータ・ソースへの照会が含まれるので、初期選択を設定する必要があります。

注: DataBlox と結合した FormBlox を使用すると、選択リストで選択が行われるたびに、FormEventListener.valueChanged() メソッドが呼び出されて、照会が発行されます。大きな結果セットまたは複雑な照会を取り扱うとき、遅延またはパフォーマンス問題が生じることがあります。

カテゴリー別の Blox Form Tag Library リファレンス

以下の FormBlox タグを使用するには、次の taglib ディレクティブを JSP ファイルの先頭に含めます。

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

FormBlox メソッドについて詳しくは、Javadoc 内の com.alphablox.blox.form パッケージを参照してください。

Blox Form Tag Library には、以下のフォーム・タグが含まれます。

汎用 HTML フォーム・エレメント用の FormBlox

- 355 ページの『<bloxform:checkBox> タグ』
- 365 ページの『<bloxform:edit> タグ』
- 370 ページの『<bloxform:radioButton> タグ』
  - 370 ページの『ネストされた <bloxform:button> タグ』
- 373 ページの『<bloxform:select> タグ』
  - 374 ページの『ネストされた <bloxform:option> タグ』

データに関連した選択リスト用の FormBlox

- 357 ページの『<bloxform:cubeSelect> タグ』
- 360 ページの『<bloxform:dataSourceSelect> タグ』
- 362 ページの『<bloxform:dimensionSelect> タグ』
- 368 ページの『<bloxform:memberSelect> タグ』

TimeSchema 関連の選択リスト用の FormBlox

- 377 ページの『<bloxform:timePeriodSelect> タグ』
  - 378 ページの『ネストされた <bloxform:timeSeries> タグ』
- 381 ページの『<bloxform:timeUnitSelect> タグ』

TreeFormBlox

- 383 ページの『<bloxform:tree> タグ』
  - 384 ページの『ネストされた <bloxform:folder> タグ』
  - 385 ページの『ネストされた <bloxform:item> タグ』

FormBlox に接続してアクションを指定するためのネストされたタグ

- 386 ページの『<bloxform:getChangedProperty> タグ・リファレンス』
- 387 ページの『<bloxform:setChangedProperty> タグ・リファレンス』

以下のセクションでは、それぞれの FormBlox のプロパティー、タグ、および属性について説明し、その使用法および構文を例示するための例を挙げます。

CheckBoxFormBlox リファレンス

追加するチェック・ボックスごとに、レンダリングされるときにそのチェック・ボックスにチェックを入れておくかどうか、およびボックスにチェックが入れられたときまたは入れられないときにどのような値を渡すかを指定できます。ページ・レイアウトを改善するために、各 CheckBoxFormBlox を表セルの中に入れて、その隣にテキストを表示することができます。ユーザーがチェック・ボックスをクリックすると、 FormEventListener 上の `valueChanged()` メソッドが呼び出されて、新規の値が即時に設定されることに注意してください。

CheckBoxFormBlox のプロパティー

`<bloxform:getChangedProperty>` タグおよび `<bloxform:setChangedProperty>` タグを使用して FormBlox をリンクするとき、ターゲットの FormBlox 上で取得または変更したいプロパティーの名前を指定しなければならないことがあります。このセクションでは、CheckBoxFormBlox のプロパティーをすべてリストします。関連したメソッドについては、com.alphablox.blox.form パッケージの下にある FormBlox Javadoc を参照してください。

プロパティー	タイプ	説明
`checked`	`boolean`	`true` であれば、チェック・ボックスがレンダリングされるときに、チェック・ボックスがチェックされて `checkedValue` が設定されます。デフォルトは `false` です。
`checkedValue`	`String`	チェック・ボックスがチェックされたときに戻される値。これは、ネストされた `<setChangedProperty>` タグを使用してターゲット・オブジェクト上の指定されたプロパティーを設定する値です。
`formElementName`	`String`	フォーム POST で使用される、レンダリングされるページ・エレメント名およびパラメーター名。
`formValue`	`String`	コンポーネントの通知される値。
`formValues`	`String[]`	コンポーネントの通知される複数の値。
`renderHook`	`FormBloxRenderHook`	Blox がレンダリングされているかどうかを示します。 FormBlox のカスタム・レンダラーを提供するために使用されます。
`themeClass`	`String`	このエレメントに設定するテーマ・クラス名 (複数も可) を含む String。複数のクラス名は、スペースで区切ってください。
`title`	`String`	チェック・ボックスの後に表示するテキストを含むストリング。
`uncheckedValue`	`String`	チェック・ボックスのチェックが外されたときに戻される値。これは、ネストされた `<setChangedProperty>` タグを使用してターゲット・オブジェクト上の指定されたプロパティーを設定する値です。

<bloxform:checkBox> タグ

以下の表は、 <bloxform:checkBox> タグの属性をすべてリストしています。

属性	デフォルト	説明
id		このページにレンダリングされるオブジェクトの id。 bloxName 属性が指定されている場合を除いて、これは bloxName でもあります。
bloxName	デフォルトは id	サーバー (ピア) 上のオブジェクトの名前。
checked	false	true であれば、チェック・ボックスがレンダリングされるときに、チェック・ボックスがチェックされて checkedValue が設定されます。この属性が指定されていない場合、チェック・ボックスからチェックが外されて、チェック・ボックスがレンダリングされるときに uncheckedValue が設定されます。
checkedValue	true	チェック・ボックスがチェックされたときに戻される値。これは、ネストされた <setChangedProperty> タグを使用してターゲット・オブジェクト上の指定されたプロパティーを設定する値です。
formElementName		フォーム POST で使用される、レンダリングされるページ・エレメント名およびパラメーター名。
themeClass		このエレメントに設定するテーマ・クラスの名前 (複数も可)。複数のクラスを指定する場合、それらの名前をスペースで区切ってください。
title		チェック・ボックスの後に表示するテキストを含むストリング。
uncheckedValue	false	チェック・ボックスのチェックが外されたときに戻される値。これは、ネストされた <setChangedProperty> タグを使用してターゲット・オブジェクト上の指定されたプロパティーを設定する値です。
visible	true	オブジェクトが所定の場所にレンダリングされる場合は true。

CheckBoxFormBlox の例

この例は、ユーザーが GridBlox 内の代替行バンディングをオン/オフに切り換えることを可能にする方法を示しています。

- GridBlox は追加されていますが、レンダリングされていません (visible="false")。
- CheckBoxFormBlox は title 属性に定義された後、表示するテキストとともに追加されます。
- checked 属性は true に設定されています。そのため、チェック・ボックスはページにレンダリングされるときにチェックが入れられて、 myGridBlox の bandingEnabled プロパティーに checkedValue が設定されます。 checkedValue 属性が指定されていないので、デフォルト値の "true" が使用されることに注意してください。
- ネストされた <bloxform:setChangedProperty> タグを使用して、変更するターゲット・オブジェクトおよびオブジェクトのプロパティーを指定します。
- GridBlox は <blox:display> タグを使用してレンダリングされ、 GridBlox は代替行バンディングが使用可能になって表示されます。

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

<blox:grid id="myGridBlox"
   visible="false"
   width="600"
   height="350">
   <blox:data
```

```
      dataSourceName="QCC-Essbase"
      query="<SYM <ROW (¥"All Products¥") <CHILD ¥"All Products¥"
        <COL (¥"All Time Periods¥") <CHILD ¥"All Time Periods¥"
         <PAGE(Measures) Sales !" />
</blox:grid>

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

<body>
<bloxform:checkBox id="bandingCheckBox"
   title="Enable Alternate Row Banding"
   checked="true">
   <bloxform:setChangedProperty
      targetRef="myGridBlox"
      targetProperty="bandingEnabled" />
   </bloxform:checkBox>

<blox:display bloxRef="myGridBlox" />

</body>
</html>
```

CubeSelectFormBlox リファレンス

この FormBlox は、指定のマルチディメンション・データ・ソースで使用可能なキューブの選択リストを追加します。

CubeSelectFormBlox のプロパティー

<bloxform:getChangedProperty> タグおよび <bloxform:setChangedProperty> タグを使用して FormBlox をリンクするとき、ターゲットの FormBlox 上で取得または変更したいプロパティーの名前を指定しなければならないことがあります。このセクションでは、CubeSelectFormBlox のプロパティーをすべてリストします。関連したメソッドについては、com.alphablox.blox.form パッケージの下にある FormBlox Javadoc を参照してください。

プロパティー	タイプ	説明
dataBlox	DataBlox	キューブのリストを取得する DataBlox。
formElementName	String	フォーム POST で使用される、レンダリングされるページ・エレメント名およびパラメーター名。
formValue	String	選択が行われたときに戻される値。これは、ネストされた <setChangedProperty> タグを使用してターゲット・オブジェクト上の指定されたプロパティーを設定する値です。
formValues	String[]	選択が行われたときに戻される複数の値。これらは、ネストされた <setChangedProperty> タグを使用してターゲット・オブジェクト上の指定されたプロパティーを設定する値です。
minimumWidth	String	ピクセル数による、エレメントの最小幅。
multipleSelect	boolean	複数選択が可能な場合は true。デフォルトは false です。

プロパティー	タイプ	説明
renderHook	FormBloxRenderHook	Blox がレンダリングされているかどうかを示します。FormBlox のカスタム・レンダラーを提供するために使用されます。
selectedCube	Cube	選択した Cube オブジェクト。
selectedCubeNames	String[]	選択したキューブの名前。
selectedCubes	Cube[]	選択された複数の Cube オブジェクト。
size	int	リスト内に表示される行数。デフォルトは 1 です。multipleSelect プロパティーが true の場合、 size は 1 より大きいことが必要です。それ以外の場合には、ブラウザーのサイズがデフォルトの 4 になります。
themeClass	String	このエレメントに設定するテーマ・クラス名 (複数も可) を含む String。複数のクラス名は、スペースで区切ってください。

注: 選択リストを作成するほとんどの FormBlox (CubeSelectFormBlox、DimensionSelectFormBlox、MemberSelectFormBlox、SelectFormBlox、および TimeUnitSelectFormBlox) には、選択リストの size が 1 のとき (ドロップダウン・リスト)、この selectedCube/Dimension/Member/Series 属性が明示的に指定されていなければ、最初のオプションが初期選択として自動的に設定されるという、同じ振る舞いがあります。選択リストの size が 1 よりも大きいとき、または複数選択が供されているときには、少なくとも 1 つのオプションを初期選択として設定しないとエラーが生じます。

<bloxform:cubeSelect> タグ

以下の表は、 <bloxform:cubeSelect> タグの属性をすべてリストしています。

属性	デフォルト	説明
id		このページにレンダリングされるオブジェクトの id。bloxName 属性が指定されている場合を除いて、これは bloxName でもあります。
bloxName	デフォルトは id	サーバー (ピア) 上のオブジェクトの名前。
dataBlox		DataBlox。たとえば、次のようにします。 dataBlox="<%=myDataBlox %>"
dataBloxRef		ページですでにインスタンス化されている DataBlox の名前。
formElementName		フォーム POST で使用される、レンダリングされるページ・エレメント名およびパラメーター名。
minimumWidth	リスト内の最長のオプションに適合する幅。	ピクセル数による、選択リストの最小幅。選択リストにオプションが含まれない場合、とても幅の狭いリストとして表示されます。この属性を使用して、空の選択リストの見栄えを改善することができます。
multipleSelect	false	複数選択が可能な場合は true。
selectedCube	メタデータ内の最初の Cube	リスト内で最初に選択された Cube オブジェクト。
selectedCubeName		リスト内で最初に選択されたキューブの名前。

属性	デフォルト	説明
size	1	リスト内に表示される項目数。multipleSelect が true の場合、size は 1 より大きくなくてはなりません。それ以外の場合には、ブラウザーのサイズがデフォルトの 4 になります。
themeClass		このエレメントに設定するテーマ・クラスの名前 (複数も可)。複数のクラスを指定する場合、それらの名前をスペースで区切ってください。
visible	true	オブジェクトが所定の場所にレンダリングされる場合は true。

CubeSelectFormBlox の例

以下の例は、指定のマルチディメンション・データ・ソースで使用可能なすべてのキューブを選択リストに取り込む方法を示しています。

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

<blox:data id="myDataBlox"
   useAliases="true"
   dataSourceName="Durico"
   connectOnStartup="false"
/>
...
<bloxform:cubeSelect id="cubes"
      dataBloxRef="myDataBlox"
      visible="true" />
...
```

DataSourceSelectFormBlox リファレンス

この FormBlox は、DB2 Alphablox に定義されたデータ・ソースの選択リストを追加します。データ・ソース・タイプ (MDB、RDB、または ALL)、または IBM DB2 JDBC Driver、IBM DB2 OLAP Server、 Hyperion Essbase Adapter、Oracle Driver などの特定のデータ・アダプターを指定できます。

DataSourceSelectFormBlox のプロパティー

<bloxform:getChangedProperty> タグおよび <bloxform:setChangedProperty> タグを使用して FormBlox をリンクするとき、ターゲットの FormBlox 上で取得または変更したいプロパティーの名前を指定しなければならないことがあります。このセクションでは、DataSourceSelectFormBlox のプロパティーをすべてリストします。関連したメソッドについては、com.alphablox.blox.form パッケージの下にある FormBlox Javadoc を参照してください。

プロパティー	タイプ	説明
adapterNameFilter	String	アダプターに基づく、リストに表示するための特定のタイプのデータ・ソース。有効な値は、以下の定数です。 • DB2Driver • DB2OLAPDeploymentServicesAdapter • DB2OLAPServerAdapter • EssbaseAdapter • EssbaseDeploymentServicesAdapter • JDBC_ODBCBridgeforMSVM • JDBC_ODBCBridgeforSunVM • MSOLAPAdapter • MSSQLDriver • OracleDriver • SybaseDriver • CannedDataAdapter
adminBlox	AdminBlox	AdminBlox。
formElementName	String	フォーム POST で使用される、レンダリングされるページ・エレメント名およびパラメーター名。
formValue	String	選択が行われたときに戻される値。これは、ネストされた <setChangedProperty> タグを使用してターゲット・オブジェクト上の指定されたプロパティーを設定する値です。
formValues	String[]	選択が行われたときに戻される複数の値。これらは、ネストされた <setChangedProperty> タグを使用してターゲット・オブジェクト上の指定されたプロパティーを設定する値です。
minimumWidth	String	ピクセル数による、エレメントの最小幅。
nullDataSourceLabel	String	選択リストで最初に表示されるラベル。これは、データ・ソースが選択されていないことも示します。通常、このラベルはユーザーにデータ・ソースの選択を指示するために設定されます。 361 ページの『DataSourceSelectFormBlox の例』を参照してください。
renderHook	FormBloxRenderHook	Blox がレンダリングされているかどうかを示します。 FormBlox のカスタム・レンダラーを提供するために使用されます。
selectedDataSource	DataSource	選択した DataSource オブジェクト (com.alphablox.blox.repository.DataSource)
selectedDataSourceName	String	選択したデータ・ソースの名前。
themeClass	String	このエレメントに設定するテーマ・クラス名 (複数も可) を含む String。複数のクラス名は、スペースで区切ってください。

プロパティー	タイプ	説明
typeFilter	String	リストに表示するデータ・ソースのタイプ。有効な値は、 MDB、RDB、または ALL です。デフォルト値は、ALL です。

<bloxform:dataSourceSelect> タグ

以下の表は、 <bloxform:dataSourceSelect> タグの属性をすべてリストしています。

属性	デフォルト	説明
id		このページにレンダリングされるオブジェクトの id。bloxName 属性が指定されている場合を除いて、これは bloxName でもあります。
bloxName	デフォルトは id	サーバー (ピア) 上のオブジェクトの名前。
adapter		アダプターに基づく、リストに表示するための特定のタイプのデータ・ソース。有効な値は以下のとおりです。 • IBM DB2 JDBC Driver • IBM DB2 OLAP Server Deployment Services • IBM DB2 OLAP Server • Hyperion Essbase Adapter • Hyperion Essbase Deployment Services • Generic JDBC-ODBC Bridge for MS VM • Generic JDBC-ODBC Bridge for Sun VM • Microsoft SQL Server Driver • Sybase SQL Server Driver • OLEDB for OLAP • Oracle Driver • Canned Data Adapter
adminBloxRef		ページですでにインスタンス化されている AdminBlox。
formElementName		フォーム POST で使用される、レンダリングされるページ・エレメント名およびパラメーター名。
minimumWidth	リスト内の最長のオプションに適合する幅。	ピクセル数による、選択リストの最小幅。選択リストにオプションが含まれない場合、とても幅の狭いリストとして表示されます。この属性を使用して、空の選択リストの見栄えを改善することができます。
nullDataSourceLabel		設定すると、追加のメニュー項目が先頭に追加されてデフォルトとなり、他のものが (selectedDataSourceName を介して) 設定されなければデータ・ソースが選択されていないことを示します。通常、このラベルはユーザーにデータ・ソースの選択を指示するために設定されます。 361 ページの『DataSourceSelectFormBlox の例』を参照してください。
selectedDataSourceName	リストの最初の項目	選択したデータ・ソースの名前。
themeClass		このエレメントに設定するテーマ・クラスの名前 (複数も可)。複数のクラスを指定する場合、それらの名前をスペースで区切ってください。

属性	デフォルト	説明
type	ALL	リストに表示するデータ・ソースのタイプ。有効な値は、MDB、RDB、または ALL です。
visible	true	オブジェクトが所定の場所にレンダリングされる場合は true。

DataSourceSelectFormBlox の例

以下の例は、すべてのマルチディメンション・データ・ソースを含むドロップダウン選択リストを作成します。選択リストは、「データ・ソースを選択してください (Select a Data Source)」を最初の項目として表示されます。選択を DataBlox に接続する方法を示す完全な例は、 『FormBlox』セクションの下の『DataSourceSelectFormBlox を使用する随時分析』を参照してください。

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxformtld" prefix="bloxform"%>
<html>
<head>
   <blox:header />
</head>
<body>
<bloxform:dataSourceSelect id="dataSourceName"
   type="MDB"
   nullDataSourceLabel="Select a Data Source">
</bloxform:dataSourceSelect>
</body>
</html>
```

DimensionSelectFormBlox リファレンス

この FormBlox は、指定のマルチディメンション・データ・ソース内の指定のキューブにあるディメンションの選択リストを追加します。

DimensionSelectFormBlox のプロパティー

<bloxform:getChangedProperty> タグおよび <bloxform:setChangedProperty> タグを使用して FormBlox をリンクするとき、ターゲットの FormBlox 上で取得または変更したいプロパティーの名前を指定しなければならないことがあります。このセクションでは、DimensionSelectFormBlox のプロパティーをすべてリストします。関連したメソッドについては、com.alphablox.blox.form パッケージの下にある FormBlox Javadoc を参照してください。

プロパティー	タイプ	説明
cube	Cube	ディメンションを取得する Cube オブジェクト。
cubeName	String	ディメンションを取得するキューブの名前。
dataBlox	DataBlox	メタデータを取得する DataBlox。
formElementName	String	フォーム POST で使用される、レンダリングされるページ・エレメント名およびパラメーター名。

プロパティー	タイプ	説明
formValue	String	選択が行われたときに戻される値。これは、ネストされた <setChangedProperty> タグを使用してターゲット・オブジェクト上の指定されたプロパティーを設定する値です。
formValues	String[]	選択が行われたときに戻される複数の値。これらは、ネストされた <setChangedProperty> タグを使用してターゲット・オブジェクト上の指定されたプロパティーを設定する値です。
minimumWidth	String	ピクセル数による、エレメントの最小幅。
multipleSelect	boolean	複数選択が可能な場合は true。デフォルトは false です。
renderHook	FormBloxRenderHook	Blox がレンダリングされているかどうかを示します。 FormBlox のカスタム・レンダラーを提供するために使用されます。
selectedDimension	Dimension	選択された Dimension オブジェクト。デフォルトは、単一の選択リストでリストの最初の項目になります。
selectedDimensions	Dimension[]	複数選択がサポートされるとき、選択された Dimension オブジェクト。デフォルトは、NULL または複数選択リストでの空の配列です。
selectedUniqueName	String	選択したディメンションの固有の名前。デフォルトは、単一の選択リストでリストの最初の項目になります。
selectedUniqueNames	String[]	複数選択がサポートされているときに、選択されたディメンションの固有の名前。デフォルトは、NULL または複数選択リストでの空の配列です。
size	int	リスト内に表示される行数。デフォルトは 1 です。 multipleSelect プロパティーが true の場合、 size は 1 より大きいことが必要です。それ以外の場合には、ブラウザーのサイズがデフォルトの 4 になります。
themeClass	String	このエレメントに設定するテーマ・クラス名 (複数も可) を含む String。複数のクラス名は、スペースで区切ってください。

<bloxform:dimensionSelect> タグ

以下の表は、 <bloxform:dimensionSelect> タグの属性をすべてリストしています。

属性	デフォルト	説明
id		このページにレンダリングされるオブジェクトの id。 bloxName 属性が指定されている場合を除いて、これは bloxName でもあります。
bloxName	デフォルトは id	サーバー (ピア) 上のオブジェクトの名前。
cube		ディメンションを取得する Cube オブジェクト。
cubeName		ディメンションを取得するキューブの名前。

属性	デフォルト	説明
dataBlox		メタデータを取得する DataBlox。
dataBloxRef		ページですでにインスタンス化されている DataBlox の名前。
formElementName		フォーム POST で使用される、レンダリングされるページ・エレメント名およびパラメーター名。
minimumWidth	リスト内の最長のオプションに適合する幅。	ピクセル数による、選択リストの最小幅。選択リストにオプションが含まれない場合、とても幅の狭いリストとして表示されます。この属性を使用して、空の選択リストの見栄えを改善することができます。
multipleSelect	false	複数選択が可能な場合は true。
selectedDimension	デフォルトは、リストの最初の項目です。	リスト内で最初に選択された Dimension オブジェクト。
selectedDimensionName	デフォルトは、リストの最初の項目です。	リスト内で最初に選択されたディメンションの名前。MSAS および SAP BW データ・ソースでは、ディメンション名は大括弧 ([ ]) で囲んでください。
size	1	リスト内に表示される項目数。multipleSelect が true の場合、size は 1 より大きくなくてはなりません。それ以外の場合には、ブラウザーのサイズがデフォルトの 4 になります。
themeClass		このエレメントに設定するテーマ・クラスの名前 (複数も可)。複数のクラスを指定する場合、それらの名前をスペースで区切ってください。
visible	true	オブジェクトが所定の場所にレンダリングされる場合は true。

注: 選択リストを作成するほとんどの FormBlox (CubeSelectFormBlox、DimensionSelectFormBlox、MemberSelectFormBlox、SelectFormBlox、および TimeUnitSelectFormBlox) には、選択リストの size が 1 のとき (ドロップダウン・リスト)、この selected*Cube/Dimension/Member/Series* 属性が明示的に指定されていなければ、最初のオプションが初期選択として自動的に設定されるという、同じ振る舞いがあります。選択リストの size が 1 よりも大きいとき、または複数選択が供されているときには、少なくとも 1 つのオプションを初期選択として設定しないとエラーが生じます。

DimensionSelectFormBlox の例

以下の例は、指定の DataBlox 内の指定のキューブに含まれるすべてのディメンションを取り込んだドロップダウン・リストを作成します。

```
<bloxform:dimensionSelect id="allDimensions"
       dataBloxRef="dataBlox"
   cubeName="Cube1"
   visible="true" />
```

以下の例は、指定の DataBlox 内のすべのキューブを取り込んだドロップダウン・リストを作成します。ディメンション・ドロップ・リストに表示するディメンションは、キューブの選択によって決まります。

```
<table>
<tr>
<td width="100">Select a cube:</td>
```

```
<td width="140">Select a dimension:</td>
</tr>
<tr>
<td><bloxform:cubeSelect id="cubes"
       dataBloxRef="dataBlox"
      visible="true" /></td>
<td><bloxform:dimensionSelect id="dimensions"
       dataBloxRef="dataBlox"
       visible="true">
       <bloxform:getChangedProperty formBloxRef="cubes"
          formProperty="selectedCube"
          property="cube"/>
    </bloxform:dimensionSelect></td>
</tr>
</table>
```

EditFormBlox リファレンス

EditFormBlox は、<text> または <textarea> のいずれかのタグをレンダリングされるページに追加します。lines 属性が指定されていないか、または 1 に設定されている場合、<text> タグが挿入されます。ページ・レイアウトを改善するために、EditFormBlox を表セルの中に入れて、その隣にテキストを表示することができます。

ユーザーが (情報を入力するために) テキスト・フィールド内をクリックすると、入力フォーカスは EditFormBlox になります。ユーザーがページ上の他の場所をクリックして、入力フォーカスがリセットされるとすぐに、FormEventListener は valueChanged() メソッドを呼び出して新規の値が設定されます。Enter キーを押しても、フォーム POST はトリガーされないことに注意してください。

EditFormBlox のプロパティー

<bloxform:getChangedProperty> タグおよび <bloxform:setChangedProperty> タグを使用して FormBlox をリンクするとき、ターゲットの FormBlox 上で取得または変更したいプロパティーの名前を指定しなければならないことがあります。このセクションでは、EditFormBlox のプロパティーをすべてリストします。関連したメソッドについては、com.alphablox.blox.form パッケージの下にある FormBlox Javadoc を参照してください。

プロパティー	タイプ	説明
charactersPerLine	int	テキスト・フィールドで、行ごとに許可される文字数。デフォルトの charactersPerLine は 20 です。
focus	boolean	このコンポーネントがレンダリングされる場合、キーボード・フォーカスをこのコンポーネントに設定するには、true。
formElementName	String	フォーム POST で使用される、レンダリングされるページ・エレメント名およびパラメーター名。
formValue	String	選択が行われたときに戻される値。これは、ネストされた <setChangedProperty> タグを使用してターゲット・オブジェクト上の指定されたプロパティーを設定する値です。

プロパティー	タイプ	説明
formValues	String[]	選択が行われたときに戻される複数の値。これらは、ネストされた <setChangedProperty> タグを使用してターゲット・オブジェクト上の指定されたプロパティーを設定する値です。
lines	int	テキスト・フィールドにレンダリングされる行数。デフォルトは 1 です。この属性が 1 よりも大きな値に設定された場合、テキスト域がレンダリングされます。
maskInput	boolean	文字をマスクする場合 (アスタリスクとして表示される)、true。
maxCharacters	int	テキスト・フィールドで許可される文字数。
renderHook	FormBloxRenderHook	Blox がレンダリングされているかどうかを示します。FormBlox のカスタム・レンダラーを提供するために使用されます。
themeClass	String	このエレメントに設定するテーマ・クラス名 (複数も可) を含む String。複数のクラス名は、スペースで区切ってください。
value	String	テキスト・フィールドに入力される値。

<bloxform:edit> タグ

以下の表は、 <bloxform:edit> タグの属性をすべてリストしています。

属性	デフォルト	説明
id		このページにレンダリングされるオブジェクトの id。 bloxName 属性が指定されている場合を除いて、これは bloxName でもあります。
bloxName	デフォルトは id	サーバー (ピア) 上のオブジェクトの名前。
charactersPerLine	20	テキスト・フィールドで、行ごとに許可される文字数。
focus	true	このコンポーネントがレンダリングされる場合、キーボード・フォーカスをこのコンポーネントに設定するには、true。
formElementName		フォーム POST で使用される、レンダリングされるページ・エレメント名およびパラメーター名。
lines	1	テキスト・フィールドにレンダリングされる行数。この属性が 1 よりも大きな値に設定された場合、テキスト域がレンダリングされます。
maskInput	false	文字をマスクする場合 (アスタリスクとして表示される)、true。
maxCharacters		テキスト・フィールドで許可される文字数。
themeClass		このエレメントに設定するテーマ・クラスの名前 (複数も可)。複数のクラスを指定する場合、それらの名前をスペースで区切ってください。
visible	true	オブジェクトが所定の場所にレンダリングされる場合は true。

EditFormBlox の例

この例は、EditFormBlox を使用して、ユーザーが ChartBlox のタイトルを指定できるようにする方法を示しています。

- EditFormBlox はセル表の中に追加されて、同じ表の行にある別のセルに含まれるテキストが、その前に表示されます。

- maxCharacters 属性および charactersPerLine 属性は、どちらも 30 に設定されます。
- ネストされた <bloxform:setChangedProperty> タグを使用して、変更するターゲット・オブジェクトおよびオブジェクトのプロパティーを指定します。
- 入力フォーカスがページ上の他の場所に設定されるとすぐに、 ChartBlox の title プロパティーはテキスト・フィールドに入力された値に設定されます。

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

<blox:chart id="myChartBlox"
   visible="false"
   width="500"
   height="500">
   <blox:data
      dataSourceName="QCC-Essbase"
      query="<SYM <ROW ('All Products') <CHILD 'All Products'
        <COL ('All Time Periods') <CHILD 'All Time Periods'
         <PAGE(Measures) Sales !" />
</blox:chart>

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

<body>
<table>
<tr>
<td>Title for this chart:</td>
<td>
   <bloxform:edit id="titleEdit"
      charactersPerLine="30"
      maxCharacters="30">
      <bloxform:setChangedPBroperty
         targetRef="myChartBlox"
         targetProperty="title" />
   </bloxform:edit>
</td>
</tr>
</table>
<font size="-1">(When you are done, click anywhere else on the page to set the
title.)</font><p>
<blox:display bloxRef="myChartBlox" />
</body>
</html>
```

MemberSelectFormBlox リファレンス

この FormBlox は、指定の DataBlox の指定のキューブ (必要な場合) の指定のディメンションにあるメンバーの選択リストを追加します。設定できるのはルート・メンバーだけなので、ルート・メンバーおよびその子孫だけがリストに表示されます。指定の世代と等しい、より小さい、またはより大きいメンバーのいずれに限定して表示するかを指定して、リストをフィルターに掛けることもできます。

MemberSelectFormBlox のプロパティー

<bloxform:getChangedProperty> タグおよび <bloxform:setChangedProperty> タグを使用して FormBlox をリンクするとき、ターゲットの FormBlox 上で取得または変更したいプロパティーの名前を指定しなければならないことがあります。このセ

クションでは、MemberSelectFormBlox のプロパティーをすべてリストします。関連したメソッドについては、com.alphablox.blox.form パッケージの下にある FormBlox Javadoc を参照してください。

プロパティー	タイプ	説明
dataBlox	DataBlox	メタデータを取得する DataBlox。
dimension	Dimension	メンバーをリストする Dimension オブジェクト。
dimensionName	String	メンバーをリストする固有のディメンション名。 MSAS および SAP BW データ・ソースでは、メンバー名は大括弧 ([ ]) で囲んでください。
filterGeneration	int	リストに列挙する世代。下記の filterOperator プロパティーを参照してください。デフォルトは 0 です。
filterOperator	String	filterGeneration に適用される演算子。有効な値は、==、<、または > です。
filterType	int	filterGeneration に適用される演算子。有効な値は、以下の定数です。 • MemberSelectFormBlox.EQUALS • MemberSelectFormBlox.GREATERTHAN • MemberSelectFormBlox.LESSTHAN
formElementName	String	フォーム POST で使用される、レンダリングされるページ・エレメント名およびパラメーター名。
formValue	String	選択が行われたときに戻される値。これは、ネストされた <setChangedProperty> タグを使用してターゲット・オブジェクト上の指定されたプロパティーを設定する値です。
formValues	String[]	選択が行われたときに戻される複数の値。これらは、ネストされた <setChangedProperty> タグを使用してターゲット・オブジェクト上の指定されたプロパティーを設定する値です。
minimumWidth	String	ピクセル数による、エレメントの最小幅。
multipleSelect	boolean	複数選択が可能な場合は true。デフォルトは false です。
renderHook	FormBloxRenderHook	Blox がレンダリングされているかどうかを示します。FormBlox のカスタム・レンダラーを提供するために使用されます。
rootMembers	Member[]	この選択リスト内のルート・メンバー。MSAS および SAP BW データ・ソースでは、メンバー名は大括弧 ([ ]) で囲んでください。
rootUniqueNames	String[]	ルート・メンバーの固有の名前。
selectedDisplayName	String	選択したメンバーの表示名。
selectedDisplayNames	String[]	選択したメンバーの複数の表示名。
selectedMembers	Member[]	複数選択がサポートされるとき、選択された Member オブジェクト。
selectedUniqueName	String	選択したメンバーの固有の名前。
selectedUniqueNames	String[]	複数選択がサポートされているときに、選択されたメンバーの固有の名前。
size	int	リスト内に表示される項目数。
themeClass	String	このエレメントに設定するテーマ・クラス名 (複数も可) を含む String。複数のクラス名は、スペースで区切ってください。

<bloxform:memberSelect> タグ

以下の表は、 <bloxform:memberSelect> タグの属性をすべてリストしています。

属性	デフォルト	説明
id		このページにレンダリングされるオブジェクトの id。 bloxName 属性が指定されている場合を除いて、これは bloxName でもあります。
bloxName	デフォルトは id	サーバー (ピア) 上のオブジェクトの名前。
dataBlox		メタデータを取得する DataBlox。
dataBloxRef		ページですでにインスタンス化されている DataBlox の名前。
dimension		メンバーをリストする Dimension オブジェクト。
dimensionName		メンバーをリストする固有のディメンション名。 MSAS および SAP BW データ・ソースでは、メンバー名は大括弧 ([ ]) で囲んでください。
filterGeneration	0	リストに列挙する世代。下記の filterOperator 属性を参照してください。
filterOperator		filterGeneration に適用される演算子。有効な値は、==、<、または > です。 以下の例は、世代 2 に含まれるメンバーだけをリストします。 `filterGeneration="2"` `filterOperator="=="`
formElementName		フォーム POST で使用される、レンダリングされるページ・エレメント名およびパラメーター名。
minimumWidth	リスト内の最長のオプションに適合する幅。	ピクセル数による、選択リストの最小幅。選択リストにオプションが含まれない場合、とても幅の狭いリストとして表示されます。この属性を使用して、空の選択リストの見栄えを改善することができます。
multipleSelect	false	リストが複数選択をサポートしている場合、true。
rootMemberName	デフォルトは、世代 1 ルート	ルート・メンバーの固有の名前。MSAS および SAP BW データ・ソースでは、メンバー名は大括弧 ([ ]) で囲んでください。
rootMemberNames	デフォルトは、世代 1 ルート	ルート・メンバーの固有の名前。たとえば、以下のようにします。 `rootMemberNames= "<%=new String[]` `{"[Location].[All Locations]","[Product].[Category]"} %>"`
rootMembers		この選択リスト内のルート・メンバー。たとえば、以下のようにします。 `rootMembers="<%= mbrs%>"` ここで、mbrs は Member[] オブジェクトです。
selectedMember	デフォルトは、単一の選択リストでリスト内の最初の項目になります。	選択した Member オブジェクト。
selectedMemberName	デフォルトは、単一の選択リストでリスト内の最初の項目になります。	選択したメンバーの固有の名前。MSAS および SAP BW データ・ソースでは、メンバー名は大括弧 ([ ]) で囲んでください。
size	1	リスト内に表示される項目数。この値が 1 の場合、リストはドロップダウン・リストとしてレンダリングされます。

属性	デフォルト	説明
themeClass		このエレメントに設定するテーマ・クラスの名前 (複数も可)。複数のクラスを指定する場合、それらの名前をスペースで区切ってください。
visible	true	オブジェクトが所定の場所にレンダリングされる場合は true。

注: rootMemberName、rootMemberNames、および selectedMemberName タグ属性は、すべて表示名としてではなく固有のメンバー名として機能します。表示名だけがあるときにタグ属性を使用するには、まず MDBMetaData.resolveMember(*memberName*, true) メソッドを使用して表示名からメンバー名を解決します。表示名からメンバー名を解決する方法について詳しくは、 com.alphablox.blox.data.mdb パッケージの MDBMetaData インターフェースの resolveMember() メソッドを参照してください。

注: 選択リストを作成するほとんどの FormBlox (CubeSelectFormBlox、DimensionSelectFormBlox、MemberSelectFormBlox、SelectFormBlox、および TimeUnitSelectFormBlox) には、選択リストの size が 1 のとき (ドロップダウン・リスト)、この *selectedCube/Dimension/Member/Series* 属性が明示的に指定されていなければ、最初のオプションが初期選択として自動的に設定されるという、同じ振る舞いがあります。選択リストの size が 1 よりも大きいとき、または複数選択が供されているときには、少なくとも 1 つのオプションを初期選択として設定しないとエラーが生じます。

MemberSelectFormBlox の例

350 ページの『getChangedProperty タグ』 および 399 ページの『MemberSecurityBlox のタグ』を参照してください。

RadioButtonFormBlox リファレンス

追加するラジオ・ボタンのセットごとに、そのグループ内のボタンを縦または横のどちらの方向に位置合わせするか、およびボタンのセットの周囲に枠を描画するかどうかを指定できます。グループ内のボタンは、相互に排他的です。つまり、1 つを選択すると、グループ内の他のすべてのボタンは選択解除されます。ユーザーがラジオ・ボタンを選択すると、FormEventListener 上の valueChanged() メソッドが呼び出されて、新規の値が即時に設定されることに注意してください。

RadioButtonFormBlox プロパティー

<bloxform:getChangedProperty> タグおよび <bloxform:setChangedProperty> タグを使用して FormBlox をリンクするとき、ターゲットの FormBlox 上で取得または変更したいプロパティーの名前を指定しなければならないことがあります。このセクションでは、RadioButtonFormBlox のプロパティーをすべてリストします。関連したメソッドについては、com.alphablox.blox.form パッケージの下にある FormBlox Javadoc を参照してください。

プロパティー	タイプ	説明
borderEnabled	boolean	ラジオ・ボタンのセットの周囲に枠を描画する場合は、true。デフォルトは、true。

プロパティー	タイプ	説明
buttons	String[]	セット内のすべてのボタンの値。
formElementName	String	フォーム POST で使用される、レンダリングされるページ・エレメント名およびパラメーター名。
formValue	String	選択したラジオ・ボタンの値。
formValues	String[]	選択したラジオ・ボタンの値。 RadioButtonFormBlox では、配列内で複数の値が渡される場合、最初の値が使用されます。
layout	Layout	適用するレイアウト: HorizontalLayout または VerticalLayout (com.alphablox.blox.unimodel.core にある)。
renderHook	FormBloxRenderHook	Blox がレンダリングされているかどうかを示します。 FormBlox のカスタム・レンダラーを提供するために使用されます。
selectedButton	String	選択したボタンの値。
selectedObject	Object	選択したユーザー Object。関連したユーザー・オブジェクトがない場合、これは選択したボタンの値となります。
themeClass	String	このエレメントに設定するテーマ・クラス名 (複数も可) を含む String。複数のクラス名は、スペースで区切ってください。

<bloxform:radioButton> タグ

<bloxform:radioButton> タグを使用して、ラジオ・ボタン・セットを追加します。個別のラジオ・ボタンを追加するには、<bloxform:button> タグを使用します。以下の表は、<bloxform:radioButton> タグの属性をすべてリストしています。

属性	デフォルト	説明
id		このページにレンダリングされるオブジェクトの id。 bloxName 属性が指定されている場合を除いて、これは bloxName でもあります。
bloxName	デフォルトは id	サーバー (ピア) 上のオブジェクトの名前。
align	horizontal	horizontal または vertical。デフォルトは horizontal です。
borderEnabled	true	ラジオ・ボタンのセットの周囲に枠を描画する場合は、true。デフォルトは、true。
formElementName		フォーム POST で使用される、レンダリングされるページ・エレメント名およびパラメーター名。
themeClass		このエレメントに設定するテーマ・クラスの名前 (複数も可)。複数のクラスを指定する場合、それらの名前をスペースで区切ってください。
visible	true	オブジェクトが所定の場所にレンダリングされる場合は true。

ネストされた <bloxform:button> タグ

<bloxform:button> タグは、<bloxform:radioButton> タグ内にネストされて、ボタンを追加します。

属性	説明
label	ラジオ・ボタンにレンダリングされるテキスト。
object	ラジオ・ボタンに関連したオブジェクト値。この属性は、value 属性の使用を除外します。
selected	この項目をラジオ・ボタンのグループ内での最初の選択として設定する場合には、true。
value	ラジオ・ボタンに関連したストリング値。この属性は、object 属性の使用を除外します。

ターゲット・オブジェクト、およびチェック・ボックスがチェックされたかチェックを外されたかに応じて設定するプロパティーを指定するには、ネストされた <bloxform:setChangedProperty> タグを使用します。 387 ページの『<bloxform:setChangedProperty> タグ・リファレンス』を参照してください。

RadioButtonFormBlox の例

この例は、RadioButtonFormBlox を使用して、ユーザーが 2 つのレポートから選択できるようにする方法を示しています。各レポートは、DataBlox 上に異なる照会を設定するので、 1 つのレポートを選択すると、他のレポートは選択解除されてデータ照会はリセットされます。

- RadioButtonFormBlox が 2 つのボタンと共に追加されます。
- align 属性が vertical に設定されて、ボタンが縦方向に積み重なるようにします。 borderEnabled 属性が false に設定されます。
- <blox:data> タグにはデータ照会が指定されないことに注意してください。最初のラジオ・ボタン「Sales By Product」が初期選択として設定されているので (selected="true")、GridBlox がページにレンダリングされるとき、その値が DataBlox の照会を設定するために使用されます。
- プロパティーを暗黙的な DataBlox (GridBlox 内でネストされている) に設定する必要があるので、以下のスクリプトレットを使用して、ネストされた <bloxform:setChangedProperty> タグ内で参照可能なセッション変数 ToggleData を作成します。

```
<%
   session.setAttribute("ToggleData",ToggleGridBlox.getDataBlox());
%>
```

- ユーザーがラジオ・ボタンをクリックするとすぐに、 ToggleData の query プロパティーがそのボタンに関連した値に設定されます。その後、updateResultSet() メソッドを呼び出して結果セットを更新します。

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

<blox:grid id="ToggleGridBlox"
   visible="false"
   width="450"
   height="250">
   <blox:data
      dataSourceName="QCC-Essbase"
      useAliases="true" />
</blox:grid>
<%
   session.setAttribute("ToggleData",ToggleGridBlox.getDataBlox());
%>
```

```
<html>
<head>
   <blox:header />
</head>
<body>
<b>Display Report:</b>
<bloxform:radioButton id="ReportSelection"
   borderEnabled="false"
   align="vertical" >
   <bloxform:button label="Sales By Product"
      value="<SYM <ROW('All Products') <CHILD 'All Products'
        <COLUMN('All Time Periods') <CHILD 'All Time Periods' Sales !"
      selected="true" />
   <bloxform:button label="Sales By Market"
      value="<SYM <PAGE(Measures) Sales <ROW(¥"All Locations¥") <CHILD
         ¥"All Locations¥" <COLUMN(¥"All Time Periods¥") <CHILD
         ¥"All Time Periods¥" !" />
   <bloxform:setChangedProperty
      targetRef="ToggleData"
      targetProperty="query"
      callAfterChange="updateResultSet" />
</bloxform:radioButton>

<blox:display bloxRef="ToggleGridBlox" />

</body>
</html>
```

SelectFormBlox リファレンス

SelectFormBlox によって、HTML <select> エレメントをページに追加できます。HTML フォーム <select> エレメントと同様に、複数選択が許可されるかどうか、およびリスト内に表示されるオプション数を指定できます。ユーザーが選択すると、FormEventListener 上の valueChanged() メソッドが呼び出されて、新規の値が即時に設定されることに注意してください。

SelectFormBlox プロパティー

<bloxform:getChangedProperty> タグおよび <bloxform:setChangedProperty> タグを使用して FormBlox をリンクするとき、ターゲットの FormBlox 上で取得または変更したいプロパティーの名前を指定しなければならないことがあります。このセクションでは、SelectFormBlox のプロパティーをすべてリストします。関連したメソッドについては、com.alphablox.blox.form パッケージの下にある FormBlox Javadoc を参照してください。

プロパティー	タイプ	説明
formElementName	String	フォーム POST で使用される、レンダリングされるページ・エレメント名およびパラメーター名。
formValue	String	選択が行われたときに戻される値。これは、ネストされた <setChangedProperty> タグを使用してターゲット・オブジェクト上の指定されたプロパティーを設定する値です。
formValues	String[]	選択が行われたときに戻される複数の値。これらは、ネストされた <setChangedProperty> タグを使用してターゲット・オブジェクト上の指定されたプロパティーを設定する値です。

プロパティー	タイプ	説明
items	String[]	選択リスト内のすべてのラベルの配列。
minimumWidth	String	ピクセル数による、エレメントの最小幅。
multipleSelect	boolean	複数選択が可能な場合は true。デフォルトは false です。 multipleSelect が true の場合、size は 1 より大きくなくてはなりません。
renderHook	FormBloxRenderHook	Blox がレンダリングされているかどうかを示します。FormBlox のカスタム・レンダラーを提供するために使用されます。
selectedIndexes	int[]	選択したメニュー項目のインデックス。リストの size が 1 で、空の配列が渡されたときは、最初の項目がデフォルトで選択されます。
selectedItem	String	選択した項目のラベル。
selectedItems	String[]	複数選択がサポートされているときに、選択された項目のラベル。
selectedObject	Object	リスト内の最初に選択した項目に関連したユーザー Object。
selectedObjects	Objects[]	リスト内の選択した項目に関連した、ユーザー Objects。
size	int	リスト内に表示される項目数。デフォルトは 1 です。multipleSelect プロパティーが true の場合、 size は 1 より大きいことが必要です。それ以外の場合には、ブラウザーのサイズがデフォルトの 4 になります。
themeClass	String	このエレメントに設定するテーマ・クラス名 (複数も可) を含む String。複数のクラス名は、スペースで区切ってください。

<bloxform:select> タグ

<bloxform:select> タグを使用して、選択リストを追加します。タグに個々のオプションを追加するには、<bloxform:option> タグを使用します。以下の表は、<bloxform:select> タグの属性をすべてリストしています。

属性	デフォルト	説明
id		このページにレンダリングされるオブジェクトの id。 bloxName 属性が指定されている場合を除いて、これは bloxName でもあります。
bloxName	デフォルトは id	サーバー (ピア) 上のオブジェクトの名前。
formElementName		フォーム POST で使用される、レンダリングされるページ・エレメント名およびパラメーター名。
minimumWidth	リスト内の最長のオプションに適合する幅。	ピクセル数による、選択リストの最小幅。選択リストにオプションが含まれない場合、とても幅の狭いリストとして表示されます。この属性を使用して、空の選択リストの見栄えを改善することができます。
multipleSelect	false	複数選択が可能な場合は true。

属性	デフォルト	説明
size	1	リスト内に表示されるオプション数。この値が 1 の場合、リストはドロップダウン・リストとしてレンダリングされます。 <bloxform:option> タグを使用して追加したオプションで selected が true に設定されたものがなければ、リストの最初のオプションが初期選択として設定されます。 multipleSelect が true の場合、size は 1 より大きくなくてはなりません。それ以外の場合には、ブラウザーのサイズがデフォルトの 4 になります。
themeClass		このエレメントに設定するテーマ・クラスの名前 (複数も可)。複数のクラスを指定する場合、それらの名前をスペースで区切ってください。
visible	true	オブジェクトが所定の場所にレンダリングされる場合は true。

ネストされた <bloxform:option> タグ

このタグは、選択リストにオプションを追加します。これには、以下の属性があります。

属性	デフォルト	説明
label		選択リストにオプションとしてレンダリングされるテキスト。
object		オプションに関連した Object 値。この属性は、value 属性の使用を除外します。
selected	リストの最初の項目	この項目を選択リストの最初の選択として設定する場合には、true。
value		オプションに関連した値。この属性は、object 属性の使用を除外します。

注: 選択リストを作成するほとんどの FormBlox (CubeSelectFormBlox、DimensionSelectFormBlox、MemberSelectFormBlox、SelectFormBlox、および TimeUnitSelectFormBlox) には、選択リストの size が 1 のとき (ドロップダウン・リスト)、この selected*Cube/Dimension/Member/Series* 属性が明示的に指定されていなければ、最初のオプションが初期選択として自動的に設定されるという、同じ振る舞いがあります。選択リストの size が 1 よりも大きいとき、または複数選択が供されているときには、少なくとも 1 つのオプションを初期選択として設定しないとエラーが生じます。

SelectFormBlox の例

以下の例は、選択リストを使用して、ユーザーがチャート・タイプを選択できるようにする方法を示しています。

- 4 つのオプションのある選択リストがページに追加されます。これは、 4 つの項目すべてがリストに表示される (size="4")、単一の選択リスト (デフォルトで multipleSelect は false) です。
- 最初のオプションは、初期選択として選択されます (selected="true")。
- ネストされた ChartBlox のプロパティーを設定する必要があるので、以下のように、後に <bloxform:setChangedProperty> タグで参照できるセッション変数 myChart を作成します。

```
<%
   session.setAttribute("myChart",myPresentBlox.getChartBlox());
%>
```

- 選択が行われた後、myChart の chartType プロパティーは選択されたオプションの値に設定されます。

完全なコードは以下のとおりです。

```
<%@ taglib uri="bloxtld" prefix="blox" %>
<%@ taglib uri="bloxformtld" prefix="bloxform" %>
<blox:present id="myPresentBlox"
   visible="false"
   width="560"
   height="450">
   <blox:data
      dataSourceName="QCC-Essbase"
      query="<SYM <ROW('All Products') <CHILD 'All Products'
        <COLUMN('All Time Periods') <CHILD 'All Time Periods' Sales !"
      useAliases="true" />
</blox:present>

<%
   session.setAttribute("myChart",myPresentBlox.getChartBlox());
%>

<html>
<head>
    <blox:header/>
</head>
<body>
<b>Select Chart Type:</b><br>
<bloxform:select id="ChartSelection" size="4">
   <bloxform:option label="Bar" value="Bar" selected="true"/>
   <bloxform:option label="Pie" value="Pie" />
   <bloxform:option label="Line" value="Line" />
   <bloxform:option label="3D Bar" value="3D Bar" />
   <bloxform:setChangedProperty
      targetRef="myChart"
      targetProperty="chartType" />
</bloxform:select>
<blox:display bloxRef="myPresentBlox" />
</body>
</html>
```

TimePeriodSelectFormBlox リファレンス

TimePeriodSelectFormBlox は、TimeSchemaBlox で使用可能な TimeSeries を表示する選択リストを作成します。デフォルトで、以下の TimeSeries 項目が表示されます。

- 最近 1 カ月
- 最近 2 カ月
- 最近 3 カ月
- 最近 6 カ月
- 最近 12 カ月
- 最近の 1 四半期
- 最近の 2 四半期
- 最近の 4 四半期
- 最近 1 年

- 最近 2 年
- 現行までの月
- 現在までの四半期
- 現在までの年
- 現在の月
- 現在の週

タイム・スキーマに指定の期間タイプが含まれない場合、その期間タイプに依存する項目はデフォルトから自動的に除去されることに注意してください。追加のカスタム項目をプログラマチックにコントロールに追加することも可能です。 401 ページの『TimeSchemaBlox タグ』を参照してください。

TimeSeries

TimeSeries オブジェクトは、その名前が示すように、以下のプロパティーを持つ期間を表します。

- `baseInterval`: 月、週、四半期、年などの、基本的な期間タイプ。これは日付範囲を決めるために使用されます。
- `rollups`: ロールアップに含めるさまざまなタイプの時間単位。たとえば、TimeSeries が先月の場合、ロールアップ単位は月、週、または日に指定できます。
- `start`: 開始期間。現在の時間枠からのオフセット。現在の時間枠は 0、直前の時間枠は -1、直前の 2 単位の時間枠は -2、次の時間枠は 1、以下同様となります。
- `count`: 含める期間の数。
- `toDate`: この TimeSeries が、現在までの期間 (`TODATE`) または一連の期間 (`SEQUENCE`) のどちらを表すかを指定します。たとえば、 `TODATE(Month)(Week)` は、週をロールアップの時間単位とする、過去 1 カ月間を示します。`SEQUENCE(Month,-12,12)(Month,Quarter)` は、月と四半期をロールアップの単位とする、最近の 12 カ月を示します。

TimeSeries オブジェクトは、com.alphablox.blox.logic パッケージの一部です。TimePeriodSelectFormBlox のネストされた `<bloxform:timeSeries>` タグを使用して、時系列を選択リストに含めるように指定できます。

TimePeriodSelectFormBlox のプロパティー

`<bloxform:getChangedProperty>` タグおよび `<bloxform:setChangedProperty>` タグを使用して FormBlox をリンクするとき、ターゲットの FormBlox 上で取得または変更したいプロパティーの名前を指定しなければならないことがあります。このセクションでは、TimePeriodSelectFormBlox のプロパティーをすべてリストします。関連したメソッドについては、com.alphablox.blox.form パッケージの下にある FormBlox Javadoc を参照してください。

プロパティー	タイプ	説明
defaultSeriesVisible	boolean	デフォルトの時系列メニュー項目を表示する場合、true。デフォルトは true です。すべてのデフォルト項目のリストは、375 ページの『TimePeriodSelectFormBlox リファレンス』を参照してください。
formElementName	String	フォーム POST で使用される、レンダリングされるページ・エレメント名およびパラメーター名。
formValue	String	選択が行われたときに戻される値。これは、ネストされた <setChangedProperty> タグを使用してターゲット・オブジェクト上の指定されたプロパティーを設定する値です。
formValues	String[]	選択が行われたときに戻される複数の値。これらは、ネストされた <setChangedProperty> タグを使用してターゲット・オブジェクト上の指定されたプロパティーを設定する値です。
minimumWidth	String	ピクセル数による、エレメントの最小幅。
renderHook	FormBloxRenderHook	Blox がレンダリングされているかどうかを示します。 FormBlox のカスタム・レンダラーを提供するために使用されます。
selectedSeries	TimeSeries	現在選択されている TimeSeries。
selectedSeriesName	String	現行選択されている TimeSeries のラベル。
selectedSeriesString	String	系列ストリングを使用した、現在選択されている時系列。時系列の表現ストリングについて詳しくは、378 ページの『ネストされた <bloxform:timeSeries> タグ』を参照してください。
themeClass	String	このエレメントに設定するテーマ・クラス名 (複数も可) を含む String。複数のクラス名は、スペースで区切ってください。
timeSchema	TimeSchemaBlox	このページでインスタンス化されている TimeSchemaBlox。
tuples	Member[][]	時系列に対応するタプル。

<bloxform:timePeriodSelect> タグ

<bloxform:timePeriodSelect> タグには以下の属性があります。

属性	デフォルト	説明
id		このページにレンダリングされるオブジェクトの id。bloxName 属性が指定されている場合を除いて、これは bloxName でもあります。
bloxName	デフォルトは id	サーバー (ピア) 上のオブジェクトの名前。
defaultSeriesVisible	true	デフォルトの時系列メニュー項目を表示する場合、true。すべてのデフォルト項目のリストは、375 ページの『TimePeriodSelectFormBlox リファレンス』を参照してください。
formElementName		フォーム POST で使用される、レンダリングされるページ・エレメント名およびパラメーター名。

属性	デフォルト	説明
minimumWidth	リスト内の最長のオプションに適合する幅。	ピクセル数による、選択リストの最小幅。選択リストにオプションが含まれない場合、とても幅の狭いリストとして表示されます。この属性を使用して、空の選択リストの見栄えを改善することができます。
selectedSeries	リストの最初の項目	現在選択されている TimeSeries。
selectedSeriesString	リストの最初の項目	系列ストリングを使用した、現在選択されている時系列。時系列の表現ストリングについて詳しくは、378 ページの『ネストされた <bloxform:timeSeries> タグ』を参照してください。
themeClass		このエレメントに設定するテーマ・クラスの名前 (複数も可)。複数のクラスを指定する場合、それらの名前をスペースで区切ってください。
timeSchemaBloxRef		このページでインスタンス化されている TimeSchemaBlox。
visible	true	オブジェクトが所定の場所にレンダリングされる場合は true。

ネストされた <bloxform:timeSeries> タグ

<bloxform:timeSeries> タグは、 <bloxform:timePeriodSelect> タグの中でネストされています。これには、以下の属性があります。

属性	説明
expression	TimeSeries を組み立てるための式。TimeSeries は、SEQUENCE または TODATE のいずれかになります。 • SEQUENCE の場合、期間タイプ、開始、カウント、およびロールアップの時間単位を指定します。 • TODATE の場合、期間タイプおよびロールアップの時間単位を指定します。

たとえば、次のようにします。

- expression="SEQUENCE(MONTH,-12,12)(MONTH)" は、最近の 12 カ月 (12 カ月前から開始して 12 カ月間続く) を示し、月が時間単位になります。
- expression="SEQUENCE(QUARTER,-1,1)(QUARTER)" は、最近の四半期 (先月から開始して 1 カ月間続く) を示し、四半期が時間単位になります。
- expression="SEQUENCE(MONTH,-1,1)(WEEK)" は先月を示し、週が時間単位になります。
- expression="TODATE(MONTH)(WEEK)" は過去 1 カ月間を示し、週がロールアップの時間単位になります。
- expression="TODATE(QUARTER)(MONTH)" は現在までの四半期を示し、月がロールアップの時間単位になります。

有効な期間タイプは、年、半年、四半期、 月、週、および 日 です。

name	TimeSeries の表示されるラベル。

TimePeriodSelectFormBlox の例

以下の例は、MDBQueryBlox および TimePeriodSelectFormBlox を使用してユーザーが行軸のメンバーを選択できるようにする方法を示しています。

- DataBlox が最初に照会なしで作成されます。
- 列軸のタプルが最初に <bloxlogic:tupleList> タグを使用して指定されます。この TupleList (id="histTuples") は、列軸の TupleList となります。これは <bloxlogic:mdbQuery> タグの外部で定義されるので、列軸でメンバーの選択が行われたときにこのオブジェクトのプロパティーを設定するための id を持つことができます。
- TimePeriodSelectFormBlox が追加されて、使用可能なすべてのデフォルトの時間枠が表示されます。初期の選択済みメンバーは最近の 6 カ月に設定されて、月がロールアップ単位になります (selectedSeriesString="SEQUENCE(MONTH,-6,6)(MONTH)")。
- 照会の行軸および列軸が定義された MDBQueryBlox が追加されます。行軸については、Chocolate Blocks、Chocolate Nuts、および Specialties だけを表示します。列軸については、histTuples が参照されます。
- ユーザーによる選択の後、histTuples の listFromMetadataMembers プロパティーが TimePeriodSelectFormBlox の選択済みメンバーによって更新されます。changed() メソッドが呼び出されて、基礎となる DataBlox が更新されます。

```
<%@ page import="java.util.Date"%>
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxformtld" prefix="bloxform"%>
<%@ taglib uri="bloxlogictld" prefix="bloxlogic"%>

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

<bloxlogic:timeSchema id="timeSchema"
   name="QCC-MSAS" dataBloxRef="dataBlox" />
<%
// Since QCC-MSAS only has data up to 2002, we are setting a fixed date
// for today so the example will run. In real applications, you do not need
   // to set today's date.
   setToday(timeSchema);
%>
   <bloxlogic:tupleList id="histTuples">
      <bloxlogic:dimension list="<%=timeSchema.getDimensions()%>">
   </bloxlogic:dimension>
</bloxlogic:tupleList>

<bloxform:timePeriodSelect id="historySelector"
   timeSchemaBloxRef="timeSchema"
   selectedSeriesString="SEQUENCE(MONTH,-6,6)(MONTH)"
   visible="false">
   <bloxform:setChangedProperty formProperty="tuples"
      targetRef="histTuples"
      debugEnabled="true"
      targetProperty="listFromMetadataTuples"
      callAfterChange="changed"/>
</bloxform:timePeriodSelect>

<bloxlogic:mdbQuery id="query" dataBloxRef="dataBlox"
   cubeName="[QCC]">
   <bloxlogic:axis type="rows"
      queryFragment="{[Chocolate Blocks],[Chocolate Nuts],[Specialties]} ON
ROWS " />
```

```
   <bloxlogic:axis type="columns">
      <bloxlogic:tupleList tuplesRef="histTuples" />
   </bloxlogic:axis>
</bloxlogic:mdbQuery>

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

<b>Select a time period: </b>
<blox:display bloxRef="historySelector" />
<blox:grid id="myBlox" width="90%" height="75%"
   toolbarVisible="false" menubarVisible="false">
   <blox:data bloxRef="dataBlox" />
</blox:grid>
</body>
</html>
<%!
   // Set today to a "fixed" date since the sample QCC-MSAS database
   // only has data up to 2002. In real applications, you do not need
   // to set today's date.
   public void setToday(com.alphablox.blox.logic.timeschema.TimeSchemaBlox
timeSchema) throws Exception {
      com.alphablox.blox.logic.timeschema.PeriodType small =
      com.alphablox.blox.logic.timeschema.PeriodType.getSmallest(timeSchema
.getPeriods());
      long end = timeSchema.last(small).getEndDate().getTime();
      timeSchema.setToday(new Date(end));
   }
%>
```

TimeUnitSelectFormBlox リファレンス

TimePeriodSelectFormBlox は、TimeSchemaBlox で使用可能な時間単位を表示する選択リストを作成します。デフォルトで、以下の時間単位の項目が表示されます。

- 年
- 四半期
- 月
- 週

TimeUnitSelectFormBlox のプロパティー

<bloxform:getChangedProperty> タグおよび <bloxform:setChangedProperty> タグを使用して FormBlox をリンクするとき、ターゲットの FormBlox 上で取得または変更したいプロパティーの名前を指定しなければならないことがあります。このセクションでは、TimeUnitSelectFormBlox のプロパティーをすべてリストします。関連したメソッドについては、com.alphablox.blox.form パッケージの下にある FormBlox Javadoc を参照してください。

プロパティー	タイプ	説明
formElementName	String	フォーム POST で使用される、レンダリングされるページ・エレメント名およびパラメーター名。

プロパティー	タイプ	説明
formValue	String	選択が行われたときに戻される値。これは、ネストされた <setChangedProperty> タグを使用してターゲット・オブジェクト上の指定されたプロパティーを設定する値です。
formValues	String[]	選択が行われたときに戻される複数の値。これらは、ネストされた <setChangedProperty> タグを使用してターゲット・オブジェクト上の指定されたプロパティーを設定する値です。
items	String[]	選択リスト内のすべてのラベルの配列。
minimumWidth	String	ピクセル数による、エレメントの最小幅。
multipleSelect	boolean	複数選択が可能な場合は true。デフォルトは false です。
renderHook	FormBloxRenderHook	Blox がレンダリングされているかどうかを示します。 FormBlox のカスタム・レンダラーを提供するために使用されます。
selectedPeriodTypes	PeriodType	現在選択されている PeriodType。
size	int	リスト内に表示される項目数。multipleSelect プロパティーが true の場合、size は 1 より大きいことが必要です。それ以外の場合には、ブラウザーのサイズがデフォルトの 4 になります。
themeClass	String	このエレメントに設定するテーマ・クラス名 (複数も可) を含む String。複数のクラス名は、スペースで区切ってください。
timeSchema	TimeSchemaBlox	このページでインスタンス化されている TimeSchemaBlox。

<bloxform:timeUnitSelect> タグ

<bloxform:timeUnitSelect> タグには以下の属性があります。

属性	デフォルト	説明
id		このページにレンダリングされるオブジェクトの id。 bloxName 属性が指定されている場合を除いて、これは bloxName でもあります。
bloxName	デフォルトは id	サーバー (ピア) 上のオブジェクトの名前。
formElementName		フォーム POST で使用される、レンダリングされるページ・エレメント名およびパラメーター名。
minimumWidth	リスト内の最長のオプションに適合する幅。	ピクセル数による、選択リストの最小幅。選択リストにオプションが含まれない場合、とても幅の狭いリストとして表示されます。この属性を使用して、空の選択リストの見栄えを改善することができます。
multipleSelect	false	複数選択が可能な場合は true。
selectedTimeUnit	リストの最初の項目	PeriodType ストリングを使用した、現在選択されている時間単位。393 ページの『PeriodType』を参照してください。
size	1	リスト内に表示される項目数。multipleSelect が true の場合、size は 1 より大きくなくてはなりません。それ以外の場合には、ブラウザーのサイズがデフォルトの 4 になります。

属性	デフォルト	説明
themeClass		このエレメントに設定するテーマ・クラスの名前 (複数も可)。複数のクラスを指定する場合、それらの名前をスペースで区切ってください。
timeSchemaBloxRef		このページでインスタンス化されている TimeSchemaBlox。
visible	true	オブジェクトが所定の場所にレンダリングされる場合は true。

注: 選択リストを作成するほとんどの FormBlox (CubeSelectFormBlox、DimensionSelectFormBlox、MemberSelectFormBlox、SelectFormBlox、および TimeUnitSelectFormBlox) には、選択リストの size が 1 のとき (ドロップダウン・リスト)、この selected*Cube/Dimension/Member/Series* 属性が明示的に指定されていなければ、最初のオプションが初期選択として自動的に設定されるという、同じ振る舞いがあります。選択リストの size が 1 よりも大きいとき、または複数選択が供されているときには、少なくとも 1 つのオプションを初期選択として設定しないとエラーが生じます。

TreeFormBlox リファレンス

TreeFormBlox は、Blox UI モデル内のツリー・コントロールに基づくフォルダーおよび項目のある、ナビゲーション・ツリーを追加します。TreeFormBlox ごとに、ツリー内の項目がドラッグ可能でユーザーは項目を移動および再配列するできるかどうか、ならびに、ウィンドウまたはフレームが狭すぎる場合にフォルダーと項目ラベルを折り返すかどうかを指定できます。

各 TreeFormBlox には、正確に 1 つのルート・フォルダーが必要です。設計に応じて、rootVisible 属性を true または false に設定することにより、ルート・フォルダーを表示または非表示にすることができます。

メニュー項目ごとに、HTML href タグ属性で指定する場合と同様の方法で、 href 属性の値を指定できます。項目がクリックされたときに新規ページをロードする場合、ターゲット・ウィンドウも指定できます。フォルダーおよび項目の両方にリンクが存在することがあります。<bloxform:tree>、 <bloxform:folder>、および <bloxform:item> の 3 つのタグが、ツリーを作成するために必要です。

TreeFormBlox のプロパティー

このセクションでは、TreeFormBlox のプロパティーをすべてリストします。関連したメソッドと内部クラス (TreeFormBlox.Folder、TreeFormBlox.Item、TreeFormBlox.ItemDraggedEvent、および TreeFormBlox.ItemDraggedEventListener) については、com.alphablox.blox.form パッケージの下にある FormBlox Javadoc を参照してください。

プロパティー	タイプ	説明
draggingEnabled	boolean	ツリー内の項目を他のフォルダーにドラッグできるか、またはフォルダー内の項目を再配列できるかどうかを指定します。
folderStyle	Style	ツリー内の各フォルダー・ラベルのために使用される Style オブジェクト。

プロパティー	タイプ	説明
formElementName	String	フォーム POST で使用される、レンダリングされるページ・エレメント名およびパラメーター名。
formValue	String	フォーム POST で使用される、レンダリングされるページ・エレメント名およびパラメーター名。
formValues	String[]	フォーム POST で使用される、レンダリングされるページ・エレメント名およびパラメーター名。 RadioButtonFormBlox では、配列内で複数の値が渡される場合、最初の値が使用されます。
itemPositioningEnabled	boolean	フォルダー内での項目の位置を保持するかどうかを指定します。 draggingEnabled が true に設定されていて、 itemPositioningEnabled も true に設定されているとき、ユーザーは項目を別のフォルダーにドラッグできますが、フォルダー内の項目を再配列することはできません。
labelStyle	Style	ツリー内のすべての項目ラベルのために使用される Style オブジェクト。
renderHook	FormBloxRenderHook	Blox がレンダリングされているかどうかを示します。 FormBlox のカスタム・レンダラーを提供するために使用されます。
root	TreeFormBlox.Folder	このツリーのルート・フォルダー。
rootVisible	boolean	ルート・フォルダーを表示する場合は true。デフォルトは true です。
selected	String	選択した Item の名前。
selectedItem	TreeFormBlox.Item	選択した Item オブジェクト。
textWrapped	boolean	ラベルがブラウザー・ウィンドウまたはフレームの幅よりも長いとき、この TreeFormBlox 内のすべてのフォルダーおよび項目のラベルを折り返す場合は、 true となります。デフォルトは true です。
themeClass	String	このエレメントに設定するテーマ・クラス名 (複数も可) を含む String。複数のクラス名は、スペースで区切ってください。

<bloxform:tree> タグ

このタグは、TreeFormBlox をページ上に追加します。このタグには、以下の属性があります。

属性	デフォルト	説明
id		このページにレンダリングされるオブジェクトの id。 bloxName 属性が指定されている場合を除いて、これは bloxName でもあります。
bloxName	デフォルトは id	サーバー (ピア) 上のオブジェクトの名前。

属性	デフォルト	説明
draggingEnabled		ツリー内の項目を他のフォルダーにドラッグできるか、またはフォルダー内の項目を再配列できるかどうかを指定します。
itemPositoningEnabled		フォルダー内での項目の位置を保持するかどうかを指定します。 draggingEnabled が true に設定されていて、itemPositioningEnabled も true に設定されているとき、ユーザーは項目を別のフォルダーにドラッグできますが、フォルダー内の項目を再配列することはできません。
rootVisible	true	ルート・フォルダーを表示する場合は true。この属性が true のとき、ルート・フォルダーが表示されます。 この属性が false のとき、ルート・フォルダーは表示されないで、そのサブフォルダーが最上位のフォルダーとして表示されます。
textWrapped	true	ラベルがブラウザー・ウィンドウまたはフレームの幅よりも長いとき、この TreeFormBlox 内のすべてのフォルダーおよび項目のラベルを折り返す場合は、 true となります。
themeClass		このエレメントに設定するテーマ・クラスの名前 (複数も可)。複数のクラスを指定する場合、それらの名前をスペースで区切ってください。
visible	true	オブジェクトが所定の場所にレンダリングされる場合は true。デフォルトは true です。

フォルダーをツリーに追加するには、ネストされた <bloxform:folder> タグを使用します。

ネストされた <bloxform:folder> タグ

このタグを <bloxform:tree> タグの内側に追加して、フォルダーを追加します。各ツリーには、正確に 1 つだけのルート・フォルダーが必要であることに注意してください。そのため、通常はツリー内に少なくとも 2 レベルのフォルダーが必要です。詳しくは、386 ページの『TreeFormBlox の例』を参照してください。

項目をフォルダーに追加するには、ネストされた <bloxform:item> タグを使用します。 <bloxform:folder> タグには以下の属性があります。

属性	説明
draggable	フォルダーをドラッグ可能にするかどうかを指定します。
expanded	フォルダーがレンダリングされるとき、それを拡張するかどうかを指定します。デフォルトは false です。
href	フォルダーがクリックされたときにロードする URI。これは、リンクまたは JavaScript 関数とすることができます。
imageURL	使用するカスタム・イメージの URL。themeBasedImage を true に設定すると、テーマのイメージが DB2 Alphablox リポジトリーに保管されているディレクトリーに、カスタム・イメージを

	入れておく必要があります。 URL の指定方法についての詳細は、 448 ページの『ネストされた <bloxui:menuItem> タグ属性』にある同じプロパティーとタグ属性についての説明を参照してください。
label	フォルダーの隣にレンダリングされるテキスト。
name	フォルダー・オブジェクトの名前。
object	このフォルダーに関連したユーザー・オブジェクト。
target	href で指定した URI をロードする、ターゲット・ウィンドウまたはフレーム。デフォルトで、指定した URI は同じウィンドウまたはフレームにロードされます。
themeBasedImage	true に設定すると、テーマ・ベースのイメージを追加します。テーマ・ベースのイメージは、各テーマのイメージがリポジトリーで保管されるディレクトリーに入れておく必要があります。
tooltip	マウスをフォルダー上に移動したときに表示されるテキスト。

ネストされた <bloxform:item> タグ

このタグは、個別のメニュー項目をフォルダーに追加します。これには、以下の属性があります。

属性	**説明**
draggable	項目をドラッグ可能にするかどうかを指定します。
href	項目がクリックされたときにロードする URI。これは、リンクまたは JavaScript 関数とすることができます。
imageURL	使用するカスタム・イメージの URL。themeBasedImage を true に設定すると、テーマのイメージが DB2 Alphablox リポジトリーに保管されているディレクトリーに、カスタム・イメージを入れておく必要があります。 URL の指定方法についての詳細は、 448 ページの『ネストされた <bloxui:menuItem> タグ属性』にある同じプロパティーとタグ属性についての説明を参照してください。
label	項目の隣にレンダリングされるテキスト。
name	項目オブジェクトの名前。
object	この項目に関連したユーザー・オブジェクト。
target	href で指定した URI をロードする、ターゲット・

	ウィンドウまたはフレーム。デフォルトで、指定した URI は同じウィンドウまたはフレームにロードされます。
themeBasedImage	true に設定すると、テーマ・ベースのイメージを追加します。テーマ・ベースのイメージは、各テーマのイメージがリポジトリーで保管されるディレクトリーに入れておく必要があります。
tooltip	マウスを項目上に移動したときに表示されるテキスト。

TreeFormBlox の例

以下の例は、2 つのフォルダーのある、ドラッグ可能ではないメニュー・ツリーを作成します。ルート・フォルダーは表示されません。メニュー・ツリーは 1 つのフレーム内にあり、ユーザーがメニュー項目をクリックすると、新しいページが別のターゲット・フレーム内にロードされることを想定しています。

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

<html>
<head>
    <blox:header/>
</head>
<body>
<bloxform:tree id="myMenu" rootVisible="false" >
   <bloxform:folder> <%--root folder--%>
      <bloxform:folder label="Sales Analysis">
         <bloxform:item label="Sales Trend by Region"
            href="salesByRegion.jsp"
            target="mainFrame" />
         <bloxform:item label="Sales by Store"
            href="salesByStore.jsp"
            target="mainFrame" />
         <bloxform:item label="Units Sold by Product"
            href="unitsSoldByProduct.jsp"
            target="mainFrame" />
      </bloxform:folder>

      <bloxform:folder label="Variance Analysis" expanded="false">
         <bloxform:item label="Sales Variance"
            href="varianceSales.jsp"
            target="mainFrame" />
         <%--Pop up an alert window as the report is not available.--%>
         <bloxform:item label="Ad-Hoc Variance Analysis"
            href="javascript:alert(¥"Currently unavailable.¥")" />
      </bloxform:folder>
      </bloxform:folder>
</bloxform:tree>
</body>
</html>
```

注: JavaScript 呼び出しの中で、href 属性のためにエスケープさせた二重引用符を使用してください。単一引用符を使用すると、JavaScript エラーが生じます。

<bloxform:getChangedProperty> タグ・リファレンス

このタグを使用して FormBlox をリンクして、1 つの FormBlox が別の FormBlox の選択されたプロパティー値を持つことができるようにします。

属性	説明
debugEnabled	`true` に設定すると、プロパティー変更に対するデバッグ・ロギングがオンになります。
formBlox	ソース FormBlox。これはプロパティー値の元となる FormBlox です。たとえば、以下のようにします。 `<bloxform:select id="mySelectFormBlox" ...>` `  <bloxform:getChangedProperty` `  formBlox="<%= anotherFormBloxn%>"` `  ..../>` `</bloxform:select>` この属性または `formBloxRef` 属性のどちらかを指定する必要があります。
formBloxRef	ページですでにインスタンス化されているソース FormBlox の名前。これは値の元となる FormBlox です。この属性または `formBlox` 属性のどちらかを指定する必要があります。
formProperty	変更が通知を生じさせるソース FormBlox 上のプロパティーの名前。
property	ターゲット FormBlox 上のプロパティーの名前。

このタグの例は、350 ページの『getChangedProperty タグ』を参照してください。

<bloxform:setChangedProperty> タグ・リファレンス

このタグを使用して FormBlox をリンクして、1 つの FormBlox での選択が別の FormBlox の選択されたプロパティー値を設定できるようにします。これは通常の Java Bean イントロスペクションを使用しているので、任意の Java Bean 上のプロパティーを設定できます。 351 ページの『FormPropertyLink オブジェクト』 の説明を参照してください。

属性	説明
callAfterChange	ターゲット上でプロパティーが変更された後に呼び出すメソッド。このメソッドには、パラメーターは指定できません。 一般的なシナリオは、DataBlox プロパティーが変更されたときに、プロパティーの変更に続いて `updateResultSet()` メソッドを呼び出す必要がある場合です。別の例は、TupleList、CrossJoin、または Axis オブジェクトのプロパティーが更新されたとき、その `changed()` メソッドを呼び出して、値が変更された照会に通知する必要がある場合です。
debugEnabled	true に設定すると、プロパティー変更に対するデバッグ・ロギングがオンになります。
formProperty	変更の際に伝搬するプロパティーの名前。
target	ターゲット・オブジェクト。これは任意の Java

	Bean とすることができます。これは、プロパティーが変更されるターゲットです。この属性または targetRef 属性のどちらかを設定する必要があります。
targetRef	ページですでにインスタンス化されている Bean の名前。これは、プロパティーが変更されるターゲットです。この属性または target 属性のどちらかを設定する必要があります。
targetProperty	ターゲット Bean 上にある、変更するプロパティーの名前。

このタグの例は、以下を参照してください。

- 355 ページの『CheckBoxFormBlox の例』
- 365 ページの『EditFormBlox の例』
- 371 ページの『RadioButtonFormBlox の例』
- 374 ページの『SelectFormBlox の例』
- 379 ページの『TimePeriodSelectFormBlox の例』
- 399 ページの『MemberSecurityBlox のタグ』

第 21 章 ビジネス・ロジック Blox および TimeSchema DTD リファレンス

この章には、3 つのビジネス・ロジック -TimeSchemaBlox、MDBQueryBlox、および MemberSecurityBlox- についての参照資料が含まれています。 TimeSchema XML を作成するためのデータ・タイプ定義 (DTD) についても説明されます。

Blox Logic タグの概説

DB2 Alphablox は、分析アプリケーションで共通に必要なビジネス・ロジックを追加するために役立つ、次の 3 つのビジネス・ロジック Blox を提供します - TimeSchemaBlox、MDBQueryBlox、および MemberSecurityBlox。これらのビジネス・ロジック Blox および FormBlox (347 ページの『第 20 章 Blox Form タグ・リファレンス』 で説明されている) は、データ認識ビジネス・ロジックの必要性および状態を維持する必要性という、分析アプリケーションを開発する際に一般的に生じる 2 つの問題を解決するために設計されています。

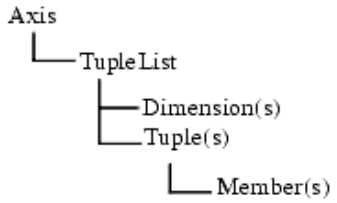
- 3 つの Blox はすべて、IBM DB2 OLAP Server、Essbase、Microsoft Analysis Service、および SAP BW データ・ソースをサポートします。
- これらの Blox および関連オブジェクトは、com.alphablox.blox.logic および com.alphablox.blox.logic.timeschema パッケージ内にあります。
- これらのビジネス・ロジック Blox のタグは、Blox Logic Tag Library に備わっています。 Blox Logic タグを使用するには、以下の taglib インポート・ステートメントをページに含める必要があります。

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

MDBQueryBlox

MDBQueryBlox は、マルチディメンション・データ・クエリーをオブジェクトで表したものです。これにより、データ・ソースに関連した照会言語を使用しないで MDB クエリーを操作できます。 `<bloxlogic:mdbQuery>` タグまたはその API を使用して、軸のタプルのパーツを変更するなど、クエリーのパーツを操作できます。MDBQueryBlox が (その `changed()` メソッドが呼び出されて) 変更されると、そのソース DataBlox はデータ・クエリーが再実行されて自動的に更新されます。

MDBQueryBlox には、行、列、およびスライサー (または「ページ」軸) の 3 つの軸があります。それぞれは、照会の特定の部分を表します。それぞれの軸は、複数のタプルから構成される Axis オブジェクトなので、TupleList です。各 TupleList は、ディメンションおよび Tuple によって定義されます。



Tuple は、1 つまたは複数のディメンションからの、メンバーのリストを含んでいます。以下の例は、“All Products” ディメンションのメンバー “All Products” (ルート・メンバー) で構成される 1 つのタプルが行軸にあり、同じディメンションからの 2 つのメンバーで構成される 2 つのタプルが列軸にある、GridBlox を示しています。

All Products	Qtr 1 01	Qtr 2 01
All Products	1770633.39	2826604.715

以下の例は、列軸のための 2 つのタプルがある GridBlox を示しています。それぞれのタプルは、次の 2 つのディメンションからのメンバーで形成されています - All Time Periods および Scenario。

	Qtr 1 01	Qtr 2 01
All Products	Actual	Actual
All Products	1770633.39	2826604.715

Axis オブジェクトは、1 つ以上の CrossJoin オブジェクトから形成されることもあります。CrossJoin は、それが結合するタプルの「相互製品」を生成します。たとえば、`tuples1 = {"Jan", "Feb"}` で `tuples2 = {"Colas", "Root Beer"}` の場合、`CrossJoin.getTuples()` は `{{"Jan", "Colas"}, {"Jan", "Root Beer"}, {"Feb", "Colas"},{"Feb", "Root Beer"}}.` を戻します。以下の例は、次の相互結合の結果として生じる列軸上の 4 つのタプルを示しています。

- “All Time Periods” ディメンションからの “Qtr 1 01” および “Qtr 2 01”
- “Scenario” ディメンションからの “Actual” および “Forecast”

	Qtr 1 01		Qtr 2 01	
All Products	Actual	Forecast	Actual	Forecast
All Products	1770633.39	1780642.53	2826604.715	2809041.365

TupleList は、Axis または CrossJoin の一部であることがある、タプルのセットを表します。 MDBQueryBlox のタグには、通常、以下のネスト関係があります。

```
<bloxlogic:mdbQuery>
   <bloxlogic:axis>
      <bloxlogic:tupleList>
```

あるいは、

```
<bloxlogic:mdbQuery>
   <bloxlogic:axis>
      <bloxlogic:crossJoin>
         <bloxlogic:tupleList>
```

MDBQueryBlox 内の行、列、またはページ軸のディメンションおよびメンバーを設定することにより、照会の一部を変更できます。たとえば、 MemberSelectFormBlox を使用して、ユーザーが任意のメンバーを選択して行軸に表示できるようにするメンバー選択リストを作成できます。その後、選択したメンバーを使用して TupleList の `listFromMetaDataMembers` または `ListFromMetadataTuples` プロパティーの値を設定できます。これにより、`TupleList.changed()` メソッドが呼び出された後に DataBlox が更新されます。例については、398 ページの『MDBQueryBlox の例』を参照してください。

軸ごとに TupleList を指定する

MDBQueryBlox を使用してデータ照会を定義するには、軸のタイプ (`rows`、`columns`、または `pages`) を指定してから、軸を形成する TupleList (複数可) を指定します。

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

<bloxlogic:mdbQuery id="query" dataBloxRef="myDataBlox">
   <bloxlogic:axis type="rows">
      <bloxlogic:tupleList>

       ... ディメンションとタプルを定義します

      </bloxlogic:tupleList>
   </bloxlogic:axis>

... 行と列のタプルのリストを同様に指定します

</bloxlogic:mdbQuery>
```

例 1: 簡単な照会: この例は、行軸と列軸とに 1 つずつのタプルがある簡単な照会を定義する方法を示しています。それぞれのタプルは、ディメンションからのルート・メンバーで構成されます。レンダリングされた GridBlox は、次のようになります。

All Products	All Time Periods
All Products	14072286.895

この照会を指定するタグは、以下のとおりです。

```
<%@ taglib uri="bloxlogictld" prefix="bloxlogic"%>
<bloxlogic:mdbQuery id="myQuery" dataBloxRef="someDataBlox">
   <bloxlogic:axis type="rows">
      <bloxlogic:tupleList>
         <bloxlogic:dimension>All Products</bloxlogic:dimension>
         <bloxlogic:tuple>
            <bloxlogic:member>All Products</bloxlogic:member>
         </bloxlogic:tuple>
      </bloxlogic:tupleList>
   </bloxlogic:axis>

   <bloxlogic:axis type="columns">
      <bloxlogic:tupleList>
         <bloxlogic:dimension>All Time Periods</bloxlogic:dimension>
         <bloxlogic:tuple>
            <bloxlogic:member>All Time Periods</bloxlogic:member>
```

```
        </bloxlogic:tuple>
      </bloxlogic:tupleList>
   </bloxlogic:axis>
</bloxlogic:mdbQuery>
```

例 2: 軸の上に 2 つのディメンション: この例は、異なるディメンションからのメンバーで形成されるタプルのある照会を定義する方法を示しています。以下の GridBlox には、次のものがあります。

- 行軸の All Products ディメンションからの Chocolate Blocks および Chocolate Nuts
- 列軸の All Time Periods からの Qtr 1 01 および Scenario からの Actual で形成されたタプル
- 列軸の All Time Periods からの Qtr 2 01 および Scenario からの Actual で形成された、別のタプル

	Qtr 1 01	Qtr 2 01
All Products	Actual	Actual
Chocolate Blocks	347784.61	428594.33
Chocolate Nuts	660425.345	1294959.57

行軸および列軸のメンバーを指定するタグは、以下のとおりです。

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

<bloxlogic:mdbQuery>
   <bloxlogic:axis type="rows">
      <bloxlogic:tupleList>
         <bloxlogic:dimension>All Products</bloxlogic:dimension>
         <bloxlogic:tuple>
            <bloxlogic:member>Chocolate Blocks</bloxlogic:member>
         </bloxlogic:tuple>
         <bloxlogic:tuple>
            <bloxlogic:member>Chocolate Nuts</bloxlogic:member>
         </bloxlogic:tuple>
      </bloxlogic:tupleList>
   </bloxlogic:axis>

   <bloxlogic:axis type="columns">
      <bloxlogic:tupleList>
         <bloxlogic:dimension>All Time Periods</bloxlogic:dimension>
         <bloxlogic:dimension>Scenario</bloxlogic:dimension>
         <bloxlogic:tuple>
            <bloxlogic:member>Qtr 1 01</bloxlogic:member>
            <bloxlogic:member>Actual</bloxlogic:member>
         </bloxlogic:tuple>
         <bloxlogic:tuple>
            <bloxlogic:member>Qtr 2 01</bloxlogic:member>
            <bloxlogic:member>Actual</bloxlogic:member>
         </bloxlogic:tuple>
      </bloxlogic:tupleList>
   </bloxlogic:axis>
</bloxlogic:mdbQuery>
```

MemberSecurityBlox

MemberSecurityBlox は、指定のディメンションでユーザーがアクセス可能なメンバーのリストを提供します。このリストは、指定の MemberSecurityFilter に基づいて DataBlox 上で suppressNoAccess を実行することにより構成されます。

MemberSecurityFilter を設定するには、その addMemeber() または setMember() メソッドを使用して、ディメンションおよびそのディメンション内のメンバー (複数も可) を指定します。

TimeSchemaBlox

TimeSchemaBlox は、ユーザーによる TimeSchema の定義に基づいて、指定のデータ・ソースのタイム・テーブルを作成します。 TimeSchema データ・タイプ定義 (DTD) を使用して、以下を指定することにより、Time ディメンションが構成される方法を定義できます。

- 時間ディメンション (複数も可) の名前
- 年、四半期、月、および週の世代レベル
- キューブの時間枠の開始日
- 通常の暦時間または週時間のどちらを適用するか
- 年の長さが例外的なもの (48 週の年など) かどうか

TimeSchema の定義を含む XML ファイルの名前は timeschema.xml として、アプリケーションの WEB-INF/ ディレクトリーに保管してください。 timeschema.dtd ファイルも、同じ場所に保管する必要があります。 TimeSchema XML の定義に使用されるデータ・タイプ定義 (DTD) は、402 ページの『TimeSchema XML DTD』に記述されます。

タイム・スキーマが構成された後、TimeSchemaBlox およびその関連オブジェクトが指定の日付または時間枠にマップされたメンバーのセットの判別を取り扱います。タイム・スキーマは、基本的な日付計算を行うことができ、日付と日付との間に一連のメンバーを生成する機能を持っています。TimeSchemaBlox により、タイム・スキーマを TimePeriodSelectFormBlox および TimeUnitSelectFormBlox が使用して、ユーザーが任意の時間枠および単位を選択するための選択リストを作成できるようになります。また、TimeSchemaBlox API によって、時間ディメンションの名前、現在の月、四半期、年、または過去 2 カ月、2 四半期、2 年、その他の情報を検索できます。 TimePeriodSelectFormBlox および TimeUnitSelectFormBlox については、347 ページの『第 20 章 Blox Form タグ・リファレンス』を参照してください。

com.alphablox.blox.logic.timeschema パッケージ内の TimeSchemaManager オブジェクトは、 TimeSchema オブジェクトへのアクセスを提供するグローバル・マネージャーです。 TimeSchema へは、TimeSchemaManager の getTimeSchema() メソッドを使用してアクセスできます。より便利な方法は、<bloxlogic:timeSchema> タグにその仕事を行わせることです。

PeriodType

PeriodType は、TimeSeries の期間タイプについて説明します。有効な期間タイプは、以下の定数です。

- PeriodType.YEAR
- PeriodType.HALFYEAR
- PeriodType.QUARTER
- PeriodType.MONTH
- PeriodType.WEEK

- PeriodType.DAY.

詳しくは、375 ページの『TimePeriodSelectFormBlox リファレンス』で TimeSeries についての説明を参照してください。

TimeMember

TimeMember は、TimeSchema のスライスを表すインターフェースです。TimeMember を使用して、このタイム・テーブルのスライスがどこで開始するか、どこで終了するか、どのタプルが日付に関連しているか、またはどの Member オブジェクトがスライスに関連しているかを調べることができます。

TimeSeries

TimeSeries は期間の系列を表し、以下のプロパティーを持ちます。

- baseInterval: 月、週、四半期、および年などの、基本的な期間タイプ。これは日付範囲を決めるために使用されます。
- rollups: ロールアップに含めるさまざまなタイプの時間単位。
- start: 開始期間。現在の時間枠からのオフセット。現在の時間枠は 0、直前の時間枠は -1、直前の 2 単位の時間枠は -2、次の時間枠は 1、以下同様となります。
- count: 含める期間の数。
- toDate: この TimeSeries が、現在までの期間 (TODATE) または一連の期間 (SEQUENCE) のどちらを表すかを指定します。たとえば、 TODATE(Month)(Week) は、週をロールアップの時間単位とする、過去 1 カ月間を示します。SEQUENCE(Month,-12,12)(Month,Quarter) は、月と四半期をロールアップの単位とする、最近の 12 カ月を示します。

時系列は、SEQUENCE(QUARTER, 0, 1)(WEEK) などのストリングで表現できます。この例は、この四半期 (0) から開始して、1 四半期の長さであり、ロールアップの単位が週であるシーケンスを示しています。定義済みのタイム・スキーマによって、TimeSeries Bean を使用して時系列を構成できます。以下の例は、最近の四半期の TimeSeries で、ロールアップの単位が月であるものを構成する方法を示しています。

```
<%
TimeSeries lastQuarter = TimeSeries.parseString("SEQUENCE(Quarter, -1, 1)
(MONTH)");
%>
```

TimePeriodSelectFormBlox を使用して指定の期間から選択リストを作成するときも、時系列を指定できます。詳しくは、375 ページの『TimePeriodSelectFormBlox リファレンス』を参照してください。

MDBQueryBlox のタグ

このセクションでは、MDBQueryBlox のタグ構文について説明します。

- 396 ページの『<bloxlogic:tupleList> タグ』
- 397 ページの『ネストされた <bloxlogic:tuple> タグ』
- 397 ページの『ネストされた <bloxlogic:dimension> タグ』
- 397 ページの『ネストされた <bloxlogic:member> タグ』
- 398 ページの『MDBQueryBlox の例』

<bloxlogic:mdbQuery> タグ属性

<bloxlogic:mdbQuery> タグには以下の属性があります。

属性	説明
id	この MDBQueryBlox の固有の id。
dataBloxRef	このページですでにインスタンス化されている DataBlox。
cubeName	指定の DataBlox のキューブの名前。

汎用タグ構文

MDBQueryBlox に関連したタグには、以下のネスト関係があります。

```
<bloxlogic:mdbQuery>
   <bloxlogic:axis>
      <bloxlogic:tupleList>
```

あるいは、

```
<bloxlogic:mdbQuery>
   <bloxlogic:axis>
      <bloxlogic:crossjoin>
         <bloxlogic:tupleList>
```

<bloxlogic:tupleList> タグは、<bloxlogic:mdbQuery> タグの外側で独立して使用することもできます。詳しくは、396 ページの『<bloxlogic:tupleList> タグ』を参照してください。

ネストされた <bloxlogic:axis> タグ

このタグは、<bloxlogic:mdbQuery> タグの内側にネストされている必要があります。これには、以下の属性があります。

属性	説明
mutable	関連した DataBlox の変更に応じて軸が変更される場合は、true です。デフォルトは false です。プレゼンテーション Blox 内でユーザー対話が可能な場合、行軸と列軸の両方で mutable を true に設定して、ユーザーのデータ・ナビゲーションによって生じる変更が正しく反映されるようにしてください。データ・ドリルを許可しないで (たとえば、Toolbar およびメニュー・バーを除去して、コンポーネントの clickable 属性を false に設定するなどして)、プレゼンテーション Blox が事前定義され

	た選択に基づいてデータを表示するだけにする場合、 mutable を true に設定する必要のないこともあります。
queryFragment	軸を表す照会フラグメント。
type	軸のタイプ。有効な値は、rows、columns、または pages です。

ネストされた <bloxlogic:crossJoin> タグ

これは <bloxlogic:axis> タグ内にネストされたタグです。これには属性がありません。<bloxlogic:crossJoin> タグ内で、結合する TupleLists を指定します。以下の例を参照してください。

CrossJoin の例

以下の例は、[Time].[Calendar] からのものと [Scenario].[All Scenario].[Actual] からのものの、2 つの TupleList を列軸で結合させる方法を示しています。

```
<bloxlogic:timeSchema id="timeSchema"
   name="QCC-MSAS" dataBloxRef="myDataBlox" />

<bloxlogic:mdbQuery>
   <bloxlogic:axis type="columns" mutable="true">
      <bloxlogic:crossJoin>

         <bloxlogic:tupleList>
            <bloxlogic:dimension>
               [Time.Calendar]
            </bloxlogic:dimension>
            <bloxlogic:tuple>
      <bloxlogic:member>
                  [Time.Calendar].[2000]
               </bloxlogic:member>
            </bloxlogic:tuple>
         </bloxlogic:tupleList>

         <bloxlogic:tupleList>
            <bloxlogic:dimension>
               [Scenario]
   </bloxlogic:dimension>
            <bloxlogic:tuple>
               <bloxlogic:member>
                  [Scenario].[All Scenario].[Actual]
               </bloxlogic:member>
            </bloxlogic:tuple>
         </bloxlogic:tupleList>

      </bloxlogic:crossJoin>
   </bloxlogic:axis>
</bloxlogic:mdbQuery>
```

<bloxlogic:tupleList> タグ

<bloxlogic:tupleList> タグは、<bloxlogic:axis> タグ内にネストされたタグです。このタグは、ネストさせないで独立して使用することもできます。これにより、TupleList の id を指定して、後に <bloxform:setChangeProperty> タグの中などで参照可能にすることができます。

属性	説明
id	オブジェクトの id。

tuplesRef	このページですでにインスタンス化されている TupleList。

<bloxlogic:tupleList> タグには、以下の 2 つのネストされたタグがあります。

- 397 ページの『ネストされた <bloxlogic:dimension> タグ』
- 397 ページの『ネストされた <bloxlogic:tuple> タグ』

ネストされた <bloxlogic:dimension> タグ

軸の中にディメンションを指定するには、 <bloxlogic:dimension> タグ内でそのディメンションを指名します。たとえば、次のようにします。

```
<bloxlogic:dimension>[Scenario]</bloxlogic:dimension>
```

ディメンションのリストを指定するには、以下の属性を使用します。

属性	説明
list	ディメンション名の配列。 役立つシナリオは、TimeSchema などから、ディメンションのリストを動的に取得する必要がある場合です。 `list="<%=myTimeSchema.getDimensions()%>"` これにより、id が myTimeSchema の TimeSchemaBlox がすでに作成されているという前提で、アプリケーションの timeshema.xml ファイルで定義された TimeSchema ディメンションのリストを取得できます。 以下のようにリストを構成することもできます。 `list="<%= new String[] {"dim1", "dime2"} %>"`

ネストされた <bloxlogic:tuple> タグ

複数の <bloxlogic:tuple> タグを <bloxlogic:tupleList> タグの中で使用することができます。それぞれの <bloxlogic:tuple> タグは、 1 つ以上のネストされた <bloxlogic:member> タグを持つことができます。タプルのリストを指定するには、以下の属性を使用します。

属性	説明
list	ストリング配列または Member 配列の配列。

ネストされた <bloxlogic:member> タグ

このタグには属性がありません。指定のディメンションおよびタプルが使用可能なメンバーは、開始タグと終了タグとの間に追加してください。たとえば、次のようにします。

```
<bloxlogic:tuple>
<bloxlogic:member>
   [Locations].[All Locations]
      </bloxlogic:member>
```

```
   <bloxlogic:member>
      [Products].[All Products]
         </bloxlogic:member>
</bloxlogic:tuple>
```

MDBQueryBlox の例

以下の例は、MDBQueryBlox および MemberSelectFormBlox を使用してユーザーが行軸のメンバーを選択できるようにする方法を示しています。

- DataBlox が最初に照会なしで作成されます。
- 行のタプルが最初に <bloxlogic:tupleList> タグを使用して指定されます。この TupleList (id="rowTuples") は、行軸の TupleList となります。これは <bloxlogic:mdbQuery> タグの外部で定義されるので、行軸でメンバーの選択が行われたときにこのオブジェクトのプロパティーを設定するための id を持つことができます。この TupleList は、[Locations].[All Locations] の下にあるすべてのメンバーを取得します。固有のメンバー名が必要なことに注意してください。
- 照会の行軸および列軸が定義された MDBQueryBlox が追加されます。列軸に関しては、Measures ディメンション内に Sales、COGS、Gross Margin だけを表示しています。行軸に関しては、rowTuples が参照されます。
- [Locations] の下のメンバーを表示するために、MemberSelectFormBlox が追加されます。初期選択メンバーは [Locations].[All Locations] に設定されます。この設定は、 rowTuples での設定と同じであることに注意してください。
- ユーザーによる選択の後、rowTuples の listFromMetadataMembers プロパティーが更新されます。changed() メソッドが呼び出されて、基礎となる DataBlox が更新されます。

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxformtld" prefix="bloxform"%>
<%@ taglib uri="bloxlogictld" prefix="bloxlogic"%>
<html>
<head>
   <blox:header />
</html>
<body>
<blox:data id="myDataBlox"
   dataSourceName="QCC-MSAS"/>

<bloxlogic:tupleList id="rowTuples">
   <bloxlogic:dimension>[Locations]</bloxlogic:dimension>
   <bloxlogic:tuple>
      <bloxlogic:member>
         [Locations].[All Locations]
      </bloxlogic:member>
   </bloxlogic:tuple>
</bloxlogic:tupleList>

<bloxlogic:mdbQuery id="myQuery" dataBloxRef="myDataBlox" cubeName="[QCC]">
   <bloxlogic:axis type="rows">
      <bloxlogic:tupleList tuplesRef="rowTuples" />
   </bloxlogic:axis>
   <bloxlogic:axis type="columns" mutable="true">
      <bloxlogic:tupleList>
         <bloxlogic:dimension>[Measures]</bloxlogic:dimension>
            <bloxlogic:tuple>
               <bloxlogic:member>[Measures].[Sales]</bloxlogic:member>
            </bloxlogic:tuple>

            <bloxlogic:tuple>
               <bloxlogic:member>[Measures].[COGS]</bloxlogic:member>
```

```
            </bloxlogic:tuple>

            <bloxlogic:tuple>
               <bloxlogic:member>[Measures].[Gross Margin %]
            </bloxlogic:member>
         </bloxlogic:tuple>
      </bloxlogic:tupleList>
   </bloxlogic:axis>
</bloxlogic:mdbQuery>

<bloxform:memberSelect id="locationSelector"
   dataBloxRef="myDataBlox"
   dimensionName="[Locations]"
   selectedMemberName="[Locations].[All Locations]"
   multipleSelect="true" visible="false">
   <bloxform:setChangedProperty formProperty="selectedMembers"
      targetRef="rowTuples"
      targetProperty="listFromMetadataMembers"
      callAfterChange="changed"/>
</bloxform:memberSelect>

<b>Select Locations for Row Axis:</b>
<blox:display bloxRef="locationSelector" />
<blox:grid id="myGridBlox" width="100%" height="100%">
   <blox:data bloxRef="myDataBlox" />
</blox:grid>
</body>
</html>
```

MemberSecurityBlox のタグ

このセクションでは、MemberSecurityBlox のタグ構文について説明します。そのメソッドについては、Javadoc の com.alphablox.blox.logic パッケージの MemberSecurityBlox クラスを参照してください。

<bloxlogic:memberSecurity>

<bloxlogic:memberSecurity> タグには以下の属性があります。

属性	説明
id	この MemberSecurityBlox の固有の id。
cubeName	suppressNoAccess をオンに設定するためのキューブの名前。
dataBlox	DataBlox。
dataBloxRef	ページですでにインスタンス化されている DataBlox の名前。
dimensionName	suppressNoAccess をオンに設定するためのディメンションの名前。

<bloxlogic:memberSecurityFilter>

<bloxlogic:memberSecurityFilter> タグには以下の属性があります。

属性	説明
dimensionName	ディメンションの名前。
memberName	メンバーの名前。

MemberSecurityBlox の例

以下の例は、複数のタグおよびそれらのネストされた関係を使用する方法を示しています。

```
<bloxlogic:memberSecurity id="memberSecurity"
   dataBloxRef="dataBlox"
   dimensionName="Market">

   <bloxlogic:memberSecurityFilter
      dimensionName="Measures"
      memberName="Profit" />
   <bloxlogic:memberSecurityFilter
      dimensionName="Measures"
      memberName="Inventory" />

</bloxlogic:memberSecurity>
```

Market 別の Profit および Inventory に関するレポートを作成する場合で、以下の条件があることを想定します。

- 選択リストに Market ディメンションからのメンバーを取り込んで、ユーザーが任意のマーケットを選択できるようにする。
- ユーザーが Profit および Inventory 上のデータにアクセスできる Market ディメンションのメンバーだけをリストに含める。

これを行うには、以下のことが必要です。

- MemberSecurityBlox の dimensionName 属性を Market に設定する。
- メンバー・セキュリティー・フィルターを Measures ディメンションの Profit および Inventory に設定する。

完全なコードは以下のとおりです。

```
<%@ page import="com.alphablox.blox.logic.MemberSecurityFilter"%>
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxformtld" prefix="bloxform"%>
<%@ taglib uri="bloxlogictld" prefix="bloxlogic"%>

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

<blox:data id="dataBlox" query="!"
   dataSourceName="essbaseFilter"/>

<bloxlogic:memberSecurity id="memberSecurity"
   dataBloxRef="dataBlox"
   dimensionName="Market">

   <bloxlogic:memberSecurityFilter
      dimensionName="Measures"
      memberName="Profit" />
   <bloxlogic:memberSecurityFilter
      dimensionName="Measures"
      memberName="Inventory" />

</bloxlogic:memberSecurity>

<bloxform:select id="members"
   visible="false"
   multipleSelect="true"
   size="5" >
```

```
   <%
      members.setItems(memberSecurity.getDisplayMemberNames());
   %>
</bloxform:select>

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

以下のように、ログイン・ユーザーに対するデータベース管理者のアクセスが限定されている場合:

Market	Profit	Inventory
New York	#No Access	8723
Massachusetts	#No Access	3699
Florida	#No Access	5632
Connecticut	#No Access	4852
New Hampshire	#No Access	2838
East	#No Access	25744
California	#No Access	#No Access
Oregon	#No Access	#No Access
Washington	#No Access	#No Access
Utah	#No Access	#No Access
Nevada	#No Access	#No Access
West	29861	#No Access
Texas	#No Access	#No Access
Oklahoma	#No Access	#No Access
Louisiana	#No Access	#No Access
New Mexico	#No Access	#No Access
South	#No Access	#No Access
Illinois	#No Access	#No Access
Ohio	#No Access	#No Access
Wisconsin	#No Access	#No Access
Missouri	#No Access	#No Access
Iowa	#No Access	#No Access
Colorado	#No Access	#No Access
Central	#No Access	#No Access
Market	#No Access	#No Access

以下のメンバーが結果として戻されます。

- New York
- Massachusetts
- Florida
- Connecticut
- New Hampshire
- East
- West

TimeSchemaBlox タグ

このセクションでは、TimeSchemaBlox のタグ構文について説明します。関連メソッドについては、Javadoc の com.alphablox.blox.logic.timescheme パッケージを参照してください。

<bloxlogic:timeSchema>

<bloxlogic:timeSchema> タグには以下の属性があります。

属性	説明
id	この TimeSchemaBlox の固有 ID。
dataBloxRef	ページですでにインスタンス化されている DataBlox の名前。
name	TimeSchemaBlox の名前。この名前は、timeschema.xml ファイルで指定された名前と一致する必要があります (<timeSchema> エレメントの name 属性)。
today	現在の日付。この日付は、"mm/dd/yyyy" の形式で指定する必要があります。たとえば、次のようにします。 today = ″05/01/2003″

TimeSchemaBlox の例

<bloxlogic:timeSchema> タグは、時間枠の選択リストを作成したりデータ・クエリーを取り扱うために、 TimePeriodSelectFormBlox、TimeUnitSelectFormBlox、または MDBQueryBlox によって参照される、 TimeSchemaBlox を作成します。以下のコード断片は、TimePeriodSelectFormBlox によって使用される TimeSchemaBlox を示しています。デフォルトで、TimePeriodSelectFormBlox はユーザーに選択可能な時間枠のリストを示します。選択が行われると、changed() メソッドが呼び出されるときに、 histTuples' listFromMetadataTuples プロパティーはそれに応じて変更されます。詳しい例は、379 ページの『TimePeriodSelectFormBlox の例』を参照してください。

```
<blox:data id="dataBlox" dataSourceName="MSAS" />
<bloxlogic:timeSchema id="timeSchema" name="MSAS"
   dataBloxRef="dataBlox" />
<bloxlogic:tupleList id="histTuples">
   <bloxlogic:dimension list="<%=timeSchema.getDimensions()%>">
   </bloxlogic:dimension>
</bloxlogic:tupleList>
<bloxform:timePeriodSelect id="historySelector"
   timeSchemaBloxRef="timeSchema"
   selectedSeriesString="SEQUENCE(QUARTER,-1,1)(QUARTER)"
   visible="false">
   <bloxform:setChangedProperty formProperty="tuples"
      targetRef="histTuples"
      targetProperty="listFromMetadataTuples"
      callAfterChange="changed"/>
   </bloxform:timePeriodSelect>
```

TimeSchema XML DTD

TimeSchema の定義は、Web アプリケーションの WEB-INF/ ディレクトリーにある timeschema.xml ファイルに格納してください。このファイルは、変更されるたびに再ロードされます。timeschema.xml ファイルを作成する最良の方法は、FastForward アプリケーション内にあるファイルをコピーして、それを変更することです。 FastForward アプリケーション・ディレクトリーは、次の場所にあります。

<alphablox_dir>/system/ApplicationStudio/FastForward/

それぞれの timeschema.xml ファイルは、アプリケーションに必要なデータ・ソースごとに 1 つずつの、複数の TimeSchema を含むことができます。このセクションには、以下のトピックが含まれています。

- 403 ページの『timeschema.xml の構造』
- 403 ページの『IBM DB2 OLAP Server または Hyperion Essbase データ・ソースのサンプル TimeSchema』
- 404 ページの『Microsoft Analysis Services データ・ソースのサンプル TimeSchema』
- 405 ページの『DTD のエレメントおよび属性』

timeschema.xml の構造

このファイルには、以下の一般的な構造があります。

```
<timeSchemas>
   <timeSchema dataSource="QCC-MSAS" name="QCC-MSAS" type="Weekly1D"
cube="qcc">
      ...
   </timeSchema>

   <timeSchema dataSource="TBC" name="tbc" type="Normal1D">
      ...
   </timeSchema>
</timeSchemas>
```

- timeSchemas は最も外側のエレメントです。
- アプリケーションで必要なデータ・ソースごとに、timeSchema エレメントを使用してください。

以下の 2 つの例は、一般的な構造を示しています。

IBM DB2 OLAP Server または Hyperion Essbase データ・ソースのサンプル TimeSchema

IBM DB2 OLAP Server または Hyperion Essbase データ・ソースを使用する例を、以下に示します。

```
<timeSchema dataSource="TBC" name="tbc" type="Normal1D">
   <calculation startDate="01/01/1998"/>
   <dimension name="Year" rootMember="Year">
      <level type="years" generation="1" match="Year"/>
      <level type="quarters" generation="2" match="Qtr{0}"/>
      <level type="months" generation="3" match="{MMM}"/>
   </dimension>
</timeSchema>
```

- この TimeSchema は、TBC と呼ばれるデータ・ソースに関連しています。
- この TimesShema の名前は tbc です。これは TimeSchema を検索するための名前です。
- この TimeSchema のタイプは、″Normal1D″ です。この type パラメーターは、年の長さを計算する方法 (Normal または Weekly)、および年のメンバーがカレンダーに関連した他のメンバーと同じディメンション内にあるかどうか (1D) を示します。この場合、年は ″Normal″ のカレンダー年と等しく、年のメンバーは他のメンバーと同じディメンション内にあります。

- 項目内の <calculation> エレメントは、タイム・テーブルが 1998 年 1 月 1 日から開始することを示しています。
- 項目内の <dimension> エレメントは、メンバーが Year ディメンション内に存在すること、およびルート・メンバーが Year であることを示しています。
- <dimension> エレメント内には、3 つの <level> エレメントがあります。
  - ″years″ レベルは Year ディメンションの世代 1 にあり、そのメンバーはパターン ″Year″ と一致するはずです。
  - ″quarters″ レベルは Year ディメンションの世代 2 にあり、そのメンバーはパターン "Qtr{0}" (Qtr1 や Qtr2 など) と一致するはずです。
  - ″months″ レベルは Year ディメンションの世代 3 にあり、そのメンバーはパターン ″{MMM}″ (ローカライズされた 3 文字の月の省略形。 Jan、Feb、Mar など) と一致するはずです。

Microsoft Analysis Services データ・ソースのサンプル TimeSchema

Microsoft Analysis Services データ・ソースを使用する例を、以下に示します。この項目は、DB2 Alphablox に同梱されている QCC-MSAS データ・ソースのためのものです。

```
<timeSchema dataSource="QCC-MSAS"
   name="QCC-MSAS"
   type="Weekly1D"
   cube="qcc">
   <calculation startDate="01/30/2000">
      <exceptionYear lengthWeeks="48">2000</exceptionYear>
   </calculation>
   <dimension name="[Time].[Fiscal]">
      <level type="years" generation="2" match="[Time].[Fiscal].[All
       Time Periods].[FY{0000}]"/>
      <level type="quarters" generation="3"
       match="[Time].[Fiscal].[All Time Periods].[FY{0000}].[Qtr {0}
       FY{00}]"/>
      <level type="months" generation="4"
           match="[Time].[Fiscal].[All Time Periods].[FY{0000}].
       [Qtr {0} FY{00}].[{MMM} FY{00}]"/>
      <level type="weeks" generation="5" match="[Time].[Fiscal].[All
       Time Periods].[FY{0000}].[Qtr {0} FY{00}].[{MMM} FY{00}].[{00}-
       {00}-{0000}]"/>
   </dimension>
</timeSchema>
```

- この場合は、MSAS データ・ソースは複数のキューブを持つことがあるので、cube 属性が必要です。
- これは 48 週だけの年なので、type 属性は Weekly1D に設定します。<exceptionYear> エレメントの lengthWeeks 属性は 48 に設定して、 2000 年 1 月 30 日から開始するタイム・テーブルを作成する必要があります。
- <level> エレメントには、match 属性があります。 TimeSchema は Microsoft Analysis Services データ・ソース内の固有のメンバー名に対してメタデータ検索だけを行うので、 match 属性を指定すると、″[Time].[Fiscal].[All Time Periods].[FY{0000}]″ などのパターンを指定できます。それぞれの上位世代のメンバー名には下位世代の名前が組み入れられているので、残りのパターンは単にそこに追加されます。

DTD のエレメントおよび属性

このセクションは、TimeSchema XML DTD 内のエレメントおよびそれらの属性について説明します。

<timeSchemas>

これは最も外側のエレメントです。これには属性がありません。<timeSchemas> の内側に、各データ・ソースにつき 1 つずつの、複数の timeSchema エレメントを持つことができます。

<timeSchema>

アプリケーションで必要な各データ・ソースのタイム・スキーマは、 timeSchema エレメントの内側で定義される必要があります。これには、以下の属性があります。

属性	必須か?	説明
cube	はい (MSAS および SAP BW の場合)	キューブの名前。Microsoft Analysis Services および SAP BW データ・ソースに必要です。
dataSource	はい	データ・ソースの名前。
name	はい	このタイム・スキーマの名前。この名前を使用して、timeSchema タグによる検索を行います。
type	はい	次の 4 つの有効なタイプがあります。 • Normal1D • Normal2D • Weekly1D • Weekly2D 「通常」の TimeSchema では、年は標準の (グレゴリオ) カレンダー年に対応します。うるう年でなければ、365 日あります。週次の TimeSchema では、特に注記がなければ年は 52 週に対応します。この週次カレンダーは、一般的な会計計画カレンダーです。 1D は「1 ディメンション」、つまりすべてのメンバーが MDB キューブの 1 つのディメンション内にあることを示します。 2D カレンダー・タイプは「2 ディメンション」であり、年のメンバーが一方のディメンションに含まれていて、他のメンバーは他方のディメンションに含まれています。ディメンションをこのように分割することは、IBM DB2 OLAP Server または Hyperion Essbase キューブのインプリメンテーションでは一般的な方法です。
useAliases	いいえ	別名を使用するときは、true です。IBM DB2 OLAP Server または Hyperion Essbase 専用です。デフォルトは false です。 useAliases を true に設定するときは、match 属性 (<level> エレメント内にある) に指定するパターンが、必ず別名を使用するようにしてください。

calculation

このエレメントには、以下の属性があります。

属性	説明
startDate	mm/dd/yyyy 形式のタイム・テーブルを計算するための開始日付。たとえば、次のようにします。 startDate="01/30/2000"

<exceptionYear>

最も一般的には、週ベースのタイム・スキーマを使用するときに、 exceptionYear エレメントを使用して 53 週の年を指定します。各年は 52 週よりも長いために、約 5 年ごとに 53 週の年を混在させる必要があります。さらに、データが不足しているときに年を短くするためにも使用できます。

属性	必須か?	説明
lengthDays	いいえ	年に含まれる Day の数。
lengthWeeks	いいえ	年に含まれる Week の数。

<dimension>

TimeSchema 内には、最大で 2 つの <dimension> エレメントが存在することがあります。 TimeSchema が 1 ディメンション (Normal1D または Weekly1D のタイプ) である場合、1 つの <dimension> エレメントだけがあるはずです。

属性	必須か?	説明
name	はい	TimeSchema 内で使用されるメンバーが存在するディメンションの名前。
rootMember	いいえ	ルート・メンバー名。

<level>

<level> エレメントは <dimension> エレメント内にネストされていて、ディメンション内の指定の世代のメンバーが各レベル type とどのように一致する必要があるかを指定します。これには、以下の属性があります。

属性	必須か?	説明
generation	はい	指定の type を表すディメンション内の世代。このエレメントのタイプ属性を参照してください。

属性	必須か?	説明
match	はい	一致させる固有の名前のパターン。パターンは、以下の 3 つの特殊文字を使用して、中括弧で囲んで指定します。 • 0: 0 から 9 までの数字。 • #: オプショナルで、0 から 9 までの数字。 • M: 英字 ([A-Z][a-z])。 たとえば、次のようにします。 match="[Time].[Fiscal].[All Time Periods].[FY{0000}].[Qtr {0} FY{00}].{[MMM} FY {00}]" これは、[Time].[Fiscal].[All Time Periods].[FY2002].[Qtr 1 FY02].[Jan FY02]. などのメンバーと一致します。 **注:** useAliases を true に設定するとき (<timeSchema> エレメント内で) は、match 属性 (<level> エレメント内にある) に指定するパターンが、必ず別名を使用するようにしてください。
order	いいえ	有効な値は、ascending または descending です。デフォルトは、ascending です。これは、アウトライン上でメンバーが時間の昇順に配列することを意味します。昇順では、年の 1990 は 1991 の前に配されて、月の Jan は Feb の前に配されます。何かの理由で、アウトライン上でメンバーを逆に配列する場合には、descending を使用します。
startMember	いいえ	TimeSchema が開始するためのメンバー。このメンバーは、パターンと一致していなければなりません。
stopMember	いいえ	TimeSchema が停止するためのメンバー。このメンバーは、パターンと一致していなければなりません。
type	はい	有効な値は以下のとおりです。 • years • quarters • months • weeks

第 22 章 Blox Portlet タグ・リファレンス

Blox Portlet タグ・ライブラリー (bloxPortlet.tld) には、ポートレット・リンクおよびアクション・リンクの作成を簡単にするカスタム JSP タグが含まれています。これらのタグを使用して、クリック時にアクションまたはポートレット・リンクをトリガーする Blox または Blox UI コンポーネントへのリンクを付加することができます。この章には、タグ・ライブラリーの概説およびこのライブラリー内のタグに関する参照資料が含まれています。

Blox Portlet タグ・ライブラリーの概説

Blox UI モデルの ClientLink オブジェクトを使用して、特定の Blox UI コンポーネントへのリンクを付加することができます。この URL ベースのリンクは、コンポーネントがクリックされた場合にブラウザーによって処理されます。ただし、ポータル環境では、リンクは初回のみ機能します。リンクがページ再ロードをトリガーした後、ポータル・サーバーの各ページ要求の扱い方により、ポートレット・リンクに不整合が生じます。その後リンクをクリックしても、実際のアクションをサブミットしません。 Blox Portlet タグ・ライブラリーを使用すると、PortletLinkDefinition または ActionLinkDefinition を追加することができます。これらは以下の機能を提供します。

- PortletLinkDefinition が追加されると、その使用時に PortletLink オブジェクトが作成されます。 PortletLink オブジェクトは、指定されたパラメーター値を持つ URI を呼び出す実リンクの定義に使用されます。 URL はページが更新されるたびにポートレットによって再エンコードされ、これにより不整合の発生が防止されます。
- ActionLinkDefinition が追加されると、その使用時に PortletLink が作成されます。また、アクション名を BloxResponse.getActionURL() に渡すことによりこのリンクのポートレット URI を入手するためにも使用できます。

PortletLinkDefinition または ActionLinkDefinition が Blox と結合されると、Blox に定義が割り当てられ、Blox UI モデル内で使用する ClientLink を生成するために PortletLink または ActionLink が使用されます。 Blox Portlet タグは、どのデータ・プレゼンテーション Blox、FormBlox、ReportBlox、または Blox UI コンポーネント内でも使用できます。 PortletLinkDefinition と ActionLinkDefinition のどちらも PortletLink の作成に使用できますが、ActionLinkDefinition に関してはこのリンク定義のアクション名を設定し、PortletURI を作成するために使用することもできます。ただし、PortletLink が設定された後、BloxResponse.getActionURL() に渡されるようにアクション名を設定することはできません。アクションは Portlet API の Property Broker がソース・ポートレットによって提供されるプロパティー値をターゲット・ポートレットに配信するために使用する情報をカプセル化するため、ほとんどの場合、アクション・リンクを使用することになります。

PortletLinkDefinition と ActionLinkDefinition のどちらも、最上位の Blox に追加する必要があります。 PresentBlox がある場合、リンクは、ネストされた GridBlox ではなく、PresentBlox に追加します。

Blox Portlet タグ・ライブラリーの例

このセクションには、GridBlox、Button (Blox UI コンポーネント)、ReportBlox、および TreeFormBlox 内での Blox Portlet タグの使用例が含まれています。それぞれの例は、アクション・リンクまたはポートレット・リンクを追加するための基本的なアプローチを示しています。

1. Blox または UI コンポーネント内に <bloxportlet:actionLinkDefinition> タグを追加してこのリンク定義を付加し、action 属性を使用してアクションの名前を指定します。
2. ネストされた <bloxportlet:parameter> タグを使用して、パラメーターの名前とその値を指定します。

これで PortletLink にアクセスし、スクリプトレットでリンク情報またはパラメーター値を設定することができます。 API について詳しくは、Javadoc の com.alphablox.blox.portlet パッケージを参照してください。

GridBlox へのリンクの追加

以下の例は、独立型 GridBlox の ″setTimeMember″ という名前のアクションを定義しています。 UI モデル・コンポーネントの再ビルド時に通知イベントが送信されるたびに、アクション・パラメーター ″memberName″ がセルの値に設定されます。

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

<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxportlettld" prefix="bloxportlet" %>
<%@ taglib uri="/WEB-INF/tld/portlet.tld" prefix="portletAPI" %>
<%@ page contentType="text/html"
import="com.alphablox.blox.*,
        com.alphablox.blox.portlet.*,
        com.alphablox.blox.uimodel.*,
        com.alphablox.blox.uimodel.core.*,
        org.apache.jetspeed.portlet.*,
        java.lang.reflect.Method,
        com.alphablox.blox.uimodel.core.event.IEventHandler,
        com.alphablox.blox.uimodel.core.event.ComponentRebuiltNotify,
        com.alphablox.blox.uimodel.core.grid.GridCell,
        com.alphablox.blox.uimodel.core.grid.Grid"%>

<portletAPI:init/>

<%
    String gridBloxName = portletResponse.encodeNamespace("linkedGridBlox");
%>

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

<blox:grid id="gridBlox" bloxName="<%= gridBloxName %>" height="200"
   menubarVisible="false">
   <blox:toolbar visible="false" />
   <blox:data dataSourceName="Canned" />

   <bloxportlet:actionLinkDefinition action="setTimeMember">
      <bloxportlet:parameter name="memberName" />
   </bloxportlet:actionLinkDefinition>

<%
   BloxModel model = gridBlox.getBloxModel();
      Controller controller = model.getController();
```

```
      GridEventHandler eventHandler =
         new GridEventHandler(gridBlox.getPortletLink("setTimeMember"));

      controller.addEventHandler(eventHandler);
  %>
</blox:container>

<%!
public class GridEventHandler implements IEventHandler {
    private PortletLink portletLink;

    public GridEventHandler(PortletLink portalLink) {
        this.portletLink = portalLink;
    }

 public boolean handleComponentRebuiltNotify(ComponentRebuiltNotify event)
throws Exception {
        Component component = event.getComponent();
        if (component instanceof GridCell) {
            GridCell cell = (GridCell)component;
            if (cell.isRowHeader() && !cell.isColumnHeader()) {
                String cellValue = cell.getValue();

                portletLink.setParameterValue("memberName", cellValue);

                Component text = cell.get(0);
                text.setClientLink(portletLink.getClientLink());
            }
        }
  return false;
   }
    }
%
```

この情報を使用するには追加コードが必要です。以下のコード・サンプルは、このパラメーター値を処理し、使用する actionPerformed() メソッドを示しています。

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

        if (actionString.equals("setTimeMember")) {
            String timeMember = request.getParameter("timeMember");
            // ... use the time member accordingly ...
        }
  }
%>
```

Button へのリンクの追加

以下の例は、3 つのパラメーターを持つ Button の ″showData″ という名前のアクションを定義しています。 PortletLink の ClientLink がボタンと連結されているため、ボタンがクリックされると、この PortletLink の 3 つのパラメーターのうちの 2 つの値が設定されます。この情報を使用するには、別のポートレットなどの追加コードが必要です。この例は、リンクの設定方法を示しているにすぎません。

```
<%@ 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>
```

ReportBlox へのリンクの追加

以下の例は、ReportBlox の selectProductCode という名前のアクション・リンクを定義しています。リンクは Product 列に接続されます。レポート内の製品名がクリックされると、リンクはパラメーター "code" の値を製品のコードに設定します。この情報を使用するには、別のポートレットなどの追加コードが必要です。この例は、リンクの設定方法を示しているにすぎません。

```
<%@ 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>
```

TreeFormBlox へのリンクの追加

以下の例は、TreeFormBlox の selectItem および selectFolder という名前の 2 つのアクション・リンクを定義しています。 PortletLink の ClientLink が TreeFormBlox と連結されているため、項目またはフォルダーがクリックされると、パラメーター値はこの PortletLink の項目名またはフォルダー名に設定されます。この情報を使用するには、別のポートレットなどの追加コードが必要です。この例は、リンクの設定方法を示しているにすぎません。

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

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

<portletAPI:init/>

<%
  String treeName = portletResponse.encodeNamespace("tree2");
%>

<head>
  <blox:header />
</head>
<%!
  private String getFolder(PortletLink folderLink, String label) {
    folderLink.setParameterValue("folder", label);
    return folderLink.getLinkHRef();
  }

  private String getItem(PortletLink itemLink, String label) {
    itemLink.setParameterValue("item", label);
    return itemLink.getLinkHRef();
  }
%>

<bloxform:tree id="tree" bloxName="<%= treeName %>"
   draggingEnabled="true">
   <bloxportlet:actionLinkDefinition action="selectItem">
      <bloxportlet:parameter name="item" />
   </bloxportlet:actionLinkDefinition>

   <bloxportlet:actionLinkDefinition action="selectFolder">
      <bloxportlet:parameter name="folder" />
   </bloxportlet:actionLinkDefinition>

   <%
    PortletLink itemLink = tree.getPortletLink("selectItem");
    PortletLink folderLink = tree.getPortletLink("selectFolder");
   %>

   <bloxform:folder name="root" label="Root Folder"
      href="<%= getFolder(folderLink, ¥"Root Folder¥") %>">
      <bloxform:item label="Item1" href="<%= getItem(itemLink, ¥"Item1¥") %>" />
```

```
    <bloxform:folder label="Folder1"
       href="<%= getFolder(folderLink, ¥"Folder1¥") %>" >
       <bloxform:item label="Item2" href="<%= getItem(itemLink, ¥"Item2¥") %>" />
       <bloxform:item label="Item3" href="<%= getItem(itemLink, ¥"Item3¥") %>" />
    </bloxform:folder>

    <bloxform:folder label="Folder2"
       href="<%= getFolder(folderLink, ¥"Folder2¥") %>" >
       <bloxform:item label="Item4" href="<%= getItem(itemLink, ¥"Item4¥") %>" />
       <bloxform:item label="Item5" href="<%= getItem(itemLink, ¥"Item5¥") %>" />
    </bloxform:folder>

    <bloxform:folder name="Folder3" label="Grow Tree"
       href="<%= getFolder(folderLink, ¥"Folder3¥") %>" />

    <%
      tree.addFormEventListener(new TreeEventListener());
     %>
</bloxform:treer>
```

<bloxportlet:actionLinkDefinition> タグ

このタグは ActionLinkDefinition を追加します。これはデータ・プレゼンテーション Blox、Blox UI コンポーネント、FormBlox、または ReportBlox 内にネストする必要があります。 1 つ以上の <bloxportlet:parameter> タグをネストして、1 つ以上のパラメーター名を渡すことができます。これには、以下の属性があります。

属性	必要	説明
action	はい	アクション・リンクの名前。

<bloxportlet:actionLink> タグ

このタグは、以前に定義されたアクション・リンク名の指定に使用します。 1 つ以上の <bloxportlet:parameter> タグをネストして、名前付きアクションの 1 つ以上のパラメーター値を渡すことができます。これには、以下の属性があります。

属性	必要	説明
action	はい	アクションの名前。このアクション名は <bloxportlet:actionLinkDefinition> タグで定義されている必要があります。

<bloxportlet:parameter> タグ

このタグは、名前付きパラメーターの値の指定に使用します。これには、以下のタグ属性があります。

属性	必要	説明
name	はい	値タグ属性、または Java コードの PortletLink.setParameterValue(String *name*, String *value*) メソッドのいずれかを介してその値が設定されるパラメーターの名前。

属性	必要	説明
value	いいえ	名前付きパラメーターの値。

<bloxportlet:portletLinkDefinition> タグ

このタグは PortletLinkDefinition を追加します。これはデータ・プレゼンテーション Blox、Blox UI コンポーネント、FormBlox、または ReportBlox 内にネストする必要があります。 1 つ以上の <bloxportlet:parameter> タグをネストして、1 つ以上のパラメーター名を渡すことができます。これには、以下の属性があります。

属性	必要	説明
name	はい	PortletLink の名前。

<bloxportlet:portletLink> タグ

このタグは、以前に定義された PortletLink の指定に使用します。 1 つ以上の <bloxportlet:parameter> タグをネストして、名前付き PortletLink の名前付きパラメーターの値を指定することができます。これには、以下のタグ属性があります。

属性	必要	説明
name	はい	<bloxportlet:portletLinkDefinition> タグを使用して以前定義された PortletLink の名前。

第 23 章 Blox UI タグ・リファレンス

この章には、bloxui.tld タグ・ライブラリーにある Blox UI 修飾子タグの参照資料が含まれています。これらのタグを使用すると、DHTML クライアントで、効果的に Blox ユーザー・インターフェース、データ・レイアウトの変更およびカスタマイズを実行できます。

Blox UI タグの概説

DB2 Alphablox タグ・ライブラリー は、各 Blox の作成用に JSP ページで使用するカスタム・タグを提供しています。またこれには、すべてタグを使用して行う、Blox UI の変更や、カスタム分析機能の追加のための Blox UI タグ・ライブラリーも含まれています。これらのタグは、DHTML ユーザー・インターフェース・モデルで直接作用し、他のクライアントには、影響を与えません。

blox.tld タグ・ライブラリーを使用すると、Blox をページに作成および追加できます。bloxui.tld タグ・ライブラリーを使用すると、Blox タグを介して設定できる Blox プロパティーに加えて、Blox の外観と振る舞いをカスタマイズできます。Blox UI タグは通常、このような Blox の外観や振る舞いをカスタマイズするプレゼンテーション Blox タグ内にネストされています。

可能な場合はいつでも、Blox タグを使用して、データのプロパティー、一般的なユーザー・インターフェース編成 (chartFirst、menubarVisible 、splitPaneOrientation など)、および一般的な Blox 機能 (全クライアントで使用可能なセル・アラートや書き戻しなど) を設定するべきです。Blox UI タグは、DHTML クライアントを使用しており、Blox プロパティーで提供したよりも高いレベルで UI をカスタマイズする必要がある場合のみ使用してください。これらのタグは、DHTML クライアントで使用されるテーマ・ベースの Cascading Stylesheet のクラス設定値をオーバーライドするスタイルを使用します。

Blox UI タグにはいくつかのタイプがあります。

- コンポーネント・カスタマイズ・タグ: メニューやツールバーのカスタマイズを行う、UI コンポーネント・カスタマイズ用のタグ。Blox UI モデルが使用する共通コンポーネント名はすべて定数です。Javadoc の com.alphablox.blox.uimodel パ

ッケージにあるModelConstants インターフェースでこの定数をすべて検出できます。コンポーネント・カスタマイズ・タグを使用すると、名前によってこのコンポーネントを識別し、その後、位置、可視性、およびスタイルといった属性値を指定できます。

- カスタム・レイアウト・タグ: バタフライ・レイアウトを適用したり、データ列/行間にスペースを追加したりして、グリッド・レイアウトを主にカスタマイズできるタグ。
- 分析タグ: アプリケーションにデータ分析機能を追加するタグ。
- ユーティリティー・タグ: アクションの処理を促す便利なタグ。
- アクセシビリティ・タグ: このタグは、身体障害を持つユーザーのユーザー・エクスペリエンスを拡張するために使用できます。

Blox UI 修飾子タグを使用するには、ページに以下の taglib インポート・ステートメントを含める必要があります。

```
<%@ taglib uri="bloxuitld" prefix="bloxui" %>
```

Blox UI タグ・ライブラリーの相互参照

Blox UI タグ・ライブラリーには以下のタグが含まれます。

コンポーネント・カスタマイズ・タグ

- 419 ページの『CalculationEditor タグ』
- 419 ページの『コンポーネント・タグ』
- 446 ページの『カスタム・メニュー・タグ』
- 454 ページの『カスタム・ツールバーのタグ』

カスタム分析タグ

- 424 ページの『<bloxui:bottomN> タグ』
- 427 ページの『<bloxui:customAnalysis> タグ』
- 431 ページの『<bloxui:topN> タグ』

カスタム・レイアウトのタグ

- 434 ページの『<bloxui:butterflyLayout> タグ』
- 436 ページの『<bloxui:compressLayout> タグ』
- 438 ページの『<bloxui:customLayout> タグ』
- 438 ページの『<bloxui:gridHighlight> タグ』
- 440 ページの『<bloxui:gridSpacer> タグ』
- 444 ページの『<bloxui:title> タグ』 (PresentBlox および ChartBlox にも適用)

ユーティリティー・タグ

- 463 ページの『<bloxui:actionFilter> タグ』
- 465 ページの『<bloxui:gridFilter> タグ』
- 468 ページの『<bloxui:clientLink> タグ』
- 469 ページの『<bloxui:setProperty> タグ』

462 ページの『アクセシビリティ・タグ』
- 462 ページの『<bloxui:accessibility> タグ』

上記のタグとその属性について、以下のセクションで説明します。

CalculationEditor タグ

このタグは、「計算エディター」オプションを右クリック・メニューとメニュー・バーの「データ」メニューに追加し、ツールバーに「計算エディター」アイコンを追加します。

計算エディターは、計算と計算式に関係のあるメンバーを指定して、ユーザーが新規メンバーを追加できるユーザー・インターフェースです。さまざまな算術計算および特別な計算関数が使用可能であり、ユーザーは、算出メンバーを配置する位置、どんな世代レベルにするか、欠落値の計算での取り扱い方法を指定できます。ユーザーが「計算エディター」オプションを選択すると、「計算エディター」がポップアップ表示されます。「計算エディター」の例を参照するには、DB2 Alphablox ホーム・ページの「アセンブリー (Assembly)」タブからクエリー・ビルダーを立ち上げます。クエリー・ビルダーでは、ツールバーに「計算エディター」アイコンが追加されています。

<bloxui:calculationEditor> タグは、プレゼンテーション Blox タグ内に以下のようにネストされているべきです。

```
<blox:present id="myPresentUI">
   <blox:data bloxRef="myDataBlox" />
   <bloxui:calculationEditor />
</blox:present>
```

このタグには属性がありません。

コンポーネント・タグ

コンポーネントは、UI モデル可視コンポーネントすべての基本クラスです。このクラスは、すべてのビジュアル・コンポーネントを通じて共通なデフォルトの振る舞いとプロパティーを提供します。 Javadoc の com.alphablox.blox.uimodel パッケージにある ModelConstants インターフェースのコンポーネントを表す定数がすべて検出できます。定数名はすべて大文字になります。その値を、Blox UI タグ属性で指定する場合、2 番目以降の語の先頭文字を大文字にして、それ以外のすべてを小文字にしなければなりません。読者の便宜のために、全定数のリストを 470 ページの『モデル定数とその値』でご利用になれます。

<bloxui:component> タグは、その Blox の UI コンポーネントを変更できるよう、プレゼンテーション Blox タグ内にネストされているべきです。

完全な <bloxui:component> タグは以下のようになっています。

```
<bloxui:component
   alignment=""
   bloxRef=""
   clickable=""
   disabled=""
   height=""
   name=""
```

```
   positionBefore=""
   style=""
   themeClass=""
   title=""
   tooltip=""
   valignment=""
   visible=""
   width="">

      <bloxui:clientLink
         features=""
         link=""
         target="" />

</bloxui:component>
```

このコンポーネント・タグは、全 UI モデル可視コンポーネントの基本となっているので、名前付き Menu、MenuItem、Toolbar、および ToolbarButton のカスタマイズに使用できます。

<bloxui:component> タグ属性

属性	必要	説明
alignment	いいえ	コンポーネントの水平位置合わせ設定。有効な値は、left、center、および right です。
bloxRef	いいえ	タグが Blox タグ以外で使用される場合に、このタグに適用される既存の Blox を参照します。Blox の UI コンポーネントを動的に設定できます。
clickable	いいえ	false に設定すると、対話が使用不可になる。コンポーネントは表示されますが、クリックはできません。デフォルトは true です。 この属性は、最も外側のユーザー・インターフェース Blox で設定する必要があります。ネストされたユーザー・インターフェース Blox はすべて、この属性を継承します。ネストされたユーザー・インターフェース Blox で clickable 属性を設定することはできません。
disabled	いいえ	true に設定すると、名前付きコンポーネントが使用不可になる。コンポーネントは、ぼかし表示されます。デフォルトは false です。
height	いいえ	このコンポーネントの高さ (ピクセル単位)。
name	はい	コンポーネントの名前。Blox ユーザー・インターフェースでデフォルトのコンポーネントをカスタマイズするには、UI コンポーネントの値を指定します。このコンポーネントは、Menu、MenuItem、Toolbar、ToolbarButton、Button になるか、または Component クラスから拡張されたコンポーネントになります。 UI コンポーネントを表している定数はすべて、com.alphablox.blox.uimodel パッケージの ModelConstants インターフェースにあります。 続く例では、メニュー・バーで「表示」メニューを識別する 2 つの方法を示しています。 `name="<%= ModelConstants.VIEW_MENU %>"` `name="viewMenu"`

属性	必要	説明
style	いいえ	コンポーネントに付加するスタイル。デフォルトのスタイルまたはコンポーネント用のテーマ・ベースのスタイルをオーバーライドします。たとえば、以下のスタイルでは、名前付きコンポーネントの背景色を黒 (枠なし) に指定します。 style="background-color: black; border-style:none;"
themeClass	いいえ	Cascading Stylesheet クラスの名前。
title	はい (カスタム・メニューの場合)	コンポーネントの表示タイトル。追加されるカスタム・コンポーネントには、タイトルがなければなりません。タイトルにスラッシュ ("/") を含めることはできません。
tooltip	いいえ	マウスオーバーで表示されるツールチップ。
valignment	いいえ	コンポーネントの垂直位置合わせ設定。有効な値は、top、center、および bottom です。デフォルトは center です。
visible	いいえ	コンポーネントの可視性。false に設定すると、このコンポーネントは表示されません。デフォルトは true です。
width	いいえ	このコンポーネントの幅 (ピクセル単位)。

ネストされた <bloxui:clientLink> タグ

複数の Blox UI タグに使用するネストされたタグです。詳しくは、468 ページの『<bloxui:clientLink> タグ』を参照してください。

コンポーネント・タグの例

例 1: メニュー項目のカスタマイズ

以下の例では、<bloxui:component> タグを使用して、既存の MenuItem (helpHelp、helpAbout、toolsGridOptions、および chartMenu) をカスタマイズする方法を例示しています。

- <bloxui:component> タグは、PresentBlox 内にネストされています。
- 「ヘルプ」メニュー (name = "helpHelp") の「ヘルプ...」メニュー項目は、可視性を false (visible="false") に設定すると、除去されます。
- 「Alphablox について (About Alphablox)」メニュー項目 (name = "helpAbout") は変更されて、「このアプリケーションについて... (About This App...)」(title="About This App...") になります。
- 「グリッド・オプション...」メニュー項目 (name = "toolsGridOptions") は、「ツール」メニューから使用不可です (disabled="true")。
- 「チャート」メニュー (name = "chartMenu") は、「データ」メニューよりも前に位置変更されます (positionBefore="dataMenu")。

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

<blox:data id="dataBlox" dataSourceName="TBC" useAliases="true"
   query="<SYM <ROW(Product) <ICHILD Product <COLUMN(Year, Scenario) Qtr1
Qtr2 <CHILD Scenario Sales !"/>
<html>
<head>
   <blox:header />
</head>
<body>
<blox:present id="myPresentBlox" width="700" height="500" >
   <blox:data bloxRef="dataBlox" />
```

```
   <bloxui:component name="helpHelp" visible="false" />
   <bloxui:component name="helpAbout" title="About This App..."
      tooltip="About this application" />
   <bloxui:component name="toolsGridOptions" disabled="true" />
   <bloxui:component name="chartMenu" positionBefore="dataMenu" />
</blox:present>
</body>
</html>
```

また、カスタマイズ・タスクは、<bloxui:menu> および <bloxui:menuItem> タグを使用しても実行できます。詳しくは、446 ページの『カスタム・メニュー・タグ』を参照してください。

例 2: bloxRef **属性** を使用し、動的に UI コンポーネントの可視性を設定する

以下の例では、ユーザーが「標準」ツールバーを対話式にオン/オフする方法を例示しています。

- 2 つの HTML ボタン「ツールバーの非表示 (Hide Toolbar)」および「ツールバーの表示 (Show Toolbar)」が作成されます。
- 初期ロードでは、choice パラメーターはヌルです。
- ユーザーがどちらかのボタンをクリックすると、action パラメーターの値を設定し、URL に追加された適切なアクションを指定して、ファイル (UITagBloxRef.jsp) を再ロードします。
- <bloxui:component> タグを使用して、「標準」ツールバーの可視性を設定します。

```
<%--UITagBloxRef.jsp --%>
<%@ taglib uri='bloxtld' prefix='blox'%>
<%@ taglib uri='bloxuitld' prefix='bloxui'%>

<%--Check the choice parameter. Upon initial load, the choice
    is null.
--%>

<%
   String choice = request.getParameter( "choice" );
   if ( choice != null ) {
      if ( "showToolbar".equals( choice ) ) {
%>
         <bloxui:component bloxRef="tagBloxRefBlox"
            name="standardToolbar"
            visible="true" />
<%
      }
      else if ( "hideToolbar".equals( choice ) ) {
%>
         <bloxui:component bloxRef="tagBloxRefBlox"
            name="standardToolbar"
            visible="false" />
<%
      }
   return;
   }
%>

<blox:data id="dataBlox" dataSourceName="qcc-essbase"
   useAliases="true" visible="false"
   query="<ROW (¥"All Locations¥", ¥"Measures¥") ¥"Central¥" ¥"East¥"
      ¥"West¥" ¥"All Locations¥" ¥"Gross Margin¥" <CHILD ¥"Ratios¥"
      <ASYM <COLUMN (¥"Scenario¥", ¥"All Time Periods¥") ¥"Actual¥"
      ¥"Actual¥" ¥"Forecast¥" ¥"Forecast¥" ¥"2000.Q3¥" ¥"2000.Q4¥"
```

```
      ¥"2001.Q1¥" ¥"2001.Q2¥" !" />

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

<body>
<!--Add two buttons to allow users to hide/show the toolbar
    When the button is clicked, reload the page with the
    choice parameter specified.
-->
<input type=button value="Hide Toolbar"
onclick="window.location.href='UITagBloxRef.jsp?render=dhtml&choice=hideToo
lbar'">

<input type=button value="Show Toolbar"
onclick="window.location.href='UITagBloxRef.jsp?render=dhtml&choice=showToo
lbar'">

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

例 3: PresentBlox をクリックできないように設定する

以下の例では、<bloxui:component タグの clickable 属性を使用して、ユーザー・インターフェース Blox を非対話式にする方法を例示しています。

- <bloxui:component> タグは、PresentBlox の名前を示すコンポーネントの name を指定して、PresentBlox にネストされています。
- UI Blox の一番外側でのみ clickable 属性を設定できます。この例では、clickable 属性は PresentBlox 上で設定されます。 PresentBlox 内のネストされた GridBlox または ChartBlox に clickable 属性を設定することはできません。

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxuitld" prefix="bloxui"%>
<blox:data id="dataBlox"
   dataSourceName="QCC-Essbase" useAliases="true"
   query="<SYM <ROW (¥"All Products¥") <CHILD ¥"All Products¥"
      <COL (¥"All Time Periods¥") <CHILD ¥"All Time Periods¥"
      <PAGE(Measures) Sales !" />
<html>
<head>
   <blox:header />
</head>
<body>
<blox:present id="notclickablePresentBlox"
   width="80%" height="70%" menubarVisible="false">
   <blox:toolbar visible="false" />
   <blox:data bloxRef="dataBlox" />
   <bloxui:component name="notclickablePresentBlox" clickable="false" />
</blox:present>
</body>
</html>
```

カスタム分析タグ

カスタム分析タグを使用すると、グリッドやチャートにカスタム分析機能を追加することができます。これに含まれているのは以下のタグです。

- <bloxui:bottomN> タグ
- <bloxui:customAnalysis> タグ
- <bloxui:eightyTwenty> タグ
- <bloxui:percentOfTotal> タグ
- <bloxui:topN> タグ

プレゼンテーション Blox タグ (PresentBlox、GridBlox、または ChartBlox) にこのタグを追加すると、このカスタム分析機能が、右クリック・メニューおよび「詳細 (Advanced)」オプションの下にあるメニュー・バーの「データ」メニューに表示されます。たとえば、以下のタグは、6 つの高機能なデータ分析オプションを追加します。

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxuitld" prefix="bloxui"%>
...
<html>
<head>
   <blox:header />
</head>
<body>
<blox:present ...>
   <bloxui:topN number="10" showRank="true" />
   <bloxui:topN number="5" showRank="true" />
   <bloxui:topN prompt="true" showRank="true"  number="20"/>
   <bloxui:bottomN number="10" showRank="true" />
   <bloxui:bottomN number="5" showRank="true" />
   <bloxui:bottomN prompt="true"  showRank="true"  number="7"/>
</blox:present>
</body>
</html>
```

追加されたこの 6 つのオプションは、右クリック・メニューとメニュー・バーの「データ」メニューに表示されます。

ヒント: 一度に有効な分析操作は 1 つだけです。ユーザーが上位 10 個を選択してから下位 5 個を選択するか、合計のパーセントを選択する場合、それぞれの操作は独立しているので、直前の操作の結果を保持し続けることはありません。

<bloxui:bottomN> タグ

以下に、 <bloxui:bottomN> タグのタグ属性すべてを示します。このタグは、PresentBlox または GridBlox のタグ内にネストされていなければなりません。

```
<bloxui:bottomN
   description=""
   hideOthers=""
   membersToAnalyze=""
   name=""
   number=""
   preserveGrouping=""
```

```
   prompt=""
   showOtherSummary=""
   showRank=""
/>
```

ここで、それぞれ以下のとおりです。

属性	必要	デフォルト	説明
description	いいえ	Bottom *N;* Bottom N...	「詳細」メニューのこのメニュー項目のテキストを指定します。デフォルトは、「Bottom *N*」で、*N* は number 属性の値です。prompt 属性が true に設定されると、デフォルトは「Bottom N....」です。
hideOthers	いいえ	all	残りの非ランク・メンバーと残りの計算に関係のないメンバーがビューに表示されるかどうかを指定します。有効な値は以下のとおりです。 all: デフォルト。非ランク・メンバーと、計算に関係のないメンバーを列軸と行軸の両方から非表示にします。 none: 非ランク・メンバーと計算に関係のないメンバーを、列軸と行軸の両方ですべて保持します。 unranked: 非ランク・メンバーのみ隠蔽します。ただし、計算に関係のないメンバーは反対の軸に保持します。
membersToAnalyze	いいえ	leaf	現在表示されているメンバーのみ、あるいはリーフ・メンバーすべてをランク付けします。有効な値は以下のとおりです。 • displayed: 現在表示されているメンバーだけをランク付けします。 • leaf: リーフ・メンバーすべてをランク付けします。
name	いいえ		メニュー・バーの「データ」>「詳細」メニュー・オプションの下の、この分析のメニュー ID。名前を割り当てると、カスタム・イベント・ハンドラーとアクションのメニューへプログラマチックにアクセスできます。
number	いいえ	10	表示する最小の値のメンバー数を指定する。数値が指定されていない場合、オプション「下位 10 (Bottom 10)」がメニューに表示されます。 prompt 属性が true に設定されると、この属性の値はポップアップ・ダイアログで表示されるデフォルトの数になり、表示するメンバー数を求めるプロンプトが出されます。

属性	必要	デフォルト	説明
preserveGrouping	いいえ	true	ランク付けする際に、グループを保持するかどうかを指定する。true に設定されると、軸に複数のディメンションがある場合、ランキングはグループごとに計算されます。426 ページの 例 1: 下位 10 分析 を参照。
prompt	いいえ	false	数を求めるプロンプトを出すかどうかを指定します。 prompt が true に設定されると、ダイアログがポップアップ表示され、ユーザーが参照したい最小の値のメンバー数を入力するように促します。プロンプト用にデフォルト値を指定するには、number 属性を使用します。
showOtherSummary	いいえ	false	残りのランクなしメンバーのサマリーとして、「その他」行/列を追加するかどうか指定します。
showRank	いいえ	true	追加した列/行でランキングを表示するかどうか指定します。 false に設定された場合、メンバーは、ランキングを表示する追加された列/行を使用しないランクによってソートされます。

bottomN タグの例

例 1: 下位 10 分析

以下のコードは、「下位 10 詳細分析 (Bottom 10 advanced analysis)」オプションを右クリック・メニューに追加します。

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxuitld" prefix="bloxui"%>
...
<blox:grid ...>
   <bloxui:bottomN
     number="10"
     showRank="true" />
</blox:grid>
```

ユーザーがこのオプションを選択すると、「[member name] Bottom 10」という列 (または行。これは、行ヘッダーまたは列ヘッダーのいずれを選択するかによって変わる) が、グリッドに追加されます。

例 2: プロンプトを伴う Bottom N 分析

デフォルトでは、列軸と行軸の両方にあるランクなしメンバーは、hideOthers が none または unranked に設定されないと非表示です。ランク付けしたいメンバー数と、現在表示されているメンバーだけ、またはリーフ・メンバーだけをランク付けするといったランキング・オプションをユーザーが指定できるようにするには、prompt 属性を true に設定します。

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxuitld" prefix="bloxui"%>
<blox:grid ...>
   <bloxui:bottomN
```

```
     prompt="true"
     number="7"
     showRank="true" />
</blox:grid>
```

ユーザーがこのオプションを選択すると、number で設定される値が、表示するメンバーのデフォルトの数になって、ダイアログがポップアップ表示されます。

例 3: 下位 5 とその他

以下のコードでは「下位 5 とその他 (Bottom 5 and Other)」メニュー・オプションを右クリック・メニューに追加します。

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxuitld" prefix="bloxui"%>
...
<blox:grid ...>
   <bloxui:bottomN
     description="Bottom 5 and Other"
     number="5"
     hideOthers="all"
     showOtherSummary="true"
     showRank="true" />
</blox:grid>
```

ユーザーがこのオプションを選択すると、「[member name] Bottom 5」という列 (または行。これは、行ヘッダーまたは列ヘッダーのいずれを選択するかによって変わる) がグリッドに追加されます。その際、追加の「その他」メンバーが反対側の軸に表示されて、残りのランク外メンバーのサマリー値が提供されます。
hideOthers を all (デフォルト) に設定するように注意してください。このようにすると、ランク外メンバーと、列軸および行軸の両方で計算に関係のないメンバーは表示されません。

hideOthers が unranked に設定されると、ランク外メンバーのみが非表示になります。計算に関係のない反対の軸にあるメンバーは、グリッドに留まります。

<bloxui:customAnalysis> タグ

以下に <bloxui:customAnalysis> タグのタグ属性すべてを示します。このタグは、PresentBlox または GridBlox のタグ内にネストされていなければなりません。

```
<bloxui:customAnalysis
   analysis=""
   name="" />
```

ここで、それぞれ以下のとおりです。

属性	必要	説明
analysis	はい	カスタム分析オブジェクトを指定します。たとえば、以下のようにします。 `<bloxui:customAnalysis` `analysis="<%= new TopN() %>" />` カスタム分析オブジェクト (この例では TopN) は、com.alphablox.blox.uimodel.tags.analysis パッケージ内の AbstractAnalysis をインプリメントしなければなりません。

属性	必要	説明
name	いいえ	メニュー・バーの「データ」>「詳細」メニュー・オプションの下の、この分析のメニュー ID。名前を割り当てると、カスタム・イベント・ハンドラーとアクションのメニューへプログラマチックにアクセスできます。

<bloxui:eightyTwenty> タグ

以下に、 <bloxui:eightyTwenty> タグのタグ属性すべてを示します。このタグは、PresentBlox および GridBlox のタグ内にネストされていなければなりません。

```
<bloxui:eightyTwenty
   description=""
   hideOthers=""
   membersToAnalyze=""
   name=""
   number=""
   preserveGrouping=""
   prompt="" />
```

ここで、それぞれ以下のとおりです。

属性	必要	デフォルト	説明
description	いいえ	80/20 分析	「詳細」メニューのこのメニュー項目のテキストを指定します。 prompt 属性が true に設定されると、デフォルトは「80/20 分析」です。
hideOthers	いいえ	all	残りの非ランク・メンバーと残りの計算に関係のないメンバーがビューに表示されるかどうかを指定します。有効な値は以下のとおりです。 all: デフォルト。非ランク・メンバーと、計算に関係のないメンバーを列軸と行軸の両方から非表示にします。 none: 非ランク・メンバーと計算に関係のないメンバーを、列軸と行軸の両方ですべて保持します。 unranked: 非ランク・メンバーのみ隠蔽します。ただし、計算に関係のないメンバーは反対の軸に保持します。
membersToAnalyze	いいえ	leaf	現在表示されているメンバーのみ、あるいは計算中のリーフ・メンバーすべてを含めるかどうか指定します。有効な値は以下のとおりです。 • displayed: 現在表示されているメンバーだけを含める。 • leaf: リーフ・メンバーすべてを含める。

属性	必要	デフォルト	説明
name	いいえ		メニュー・バーの「データ」>「詳細」メニュー・オプションの下の、この分析のメニュー ID。名前を割り当てると、カスタム・イベント・ハンドラーとアクションのメニューへプログラマチックにアクセスできます。
number	いいえ	80	表示するデータの上位パーセンテージを指定します。 prompt 属性が true に設定されていると、number 属性に別の値が設定されていなければ、デフォルト値 80 がポップアップ・ダイアログに表示されます。
preserveGrouping	いいえ	true	グループごと、またはグループに関係なく計算を行うかどうかを指定します。true が設定されると、軸に複数のディメンションがある場合、グループごとに計算を行います。
prompt	いいえ	false	数を求めるプロンプトを出すかどうかを指定します。 prompt が true に設定されると、ダイアログがポップアップ表示され、ユーザーがオプションを設定するように促します。

<bloxui:percentOfTotal> タグ

<bloxui:percentOfTotal> タグは、メンバーすべての合計とメンバーごとのパーセントを計算し、表示します。このタグは、PresentBlox または GridBlox のタグ内に追加する必要があります。これには、以下の属性があります。

```
<bloxui:percentOfTotal
   description=""
   hideOthers=""
   membersToAnalyze=""
   name=""
   number=""
   preserveGrouping=""
   prompt=""
/>
```

ここで、それぞれ以下のとおりです。

属性	必要	デフォルト	説明
description	いいえ	合計のパーセント	「詳細」メニューのこのメニュー項目のテキストを指定します。 prompt 属性が true に設定されると、デフォルトは「合計のパーセント...」です。

属性	必要	デフォルト	説明
hideOthers	いいえ	all	残りの非ランク・メンバーと残りの計算に関係のないメンバーがビューに表示されるかどうかを指定します。有効な値は以下のとおりです。 all: デフォルト。非ランク・メンバーと、計算に関係のないメンバーを列軸と行軸の両方から非表示にします。 none: 非ランク・メンバーと計算に関係のないメンバーを、列軸と行軸の両方ですべて保持します。 unranked: 非ランク・メンバーのみ隠蔽します。ただし、計算に関係のないメンバーは反対の軸に保持します。
membersToAnalyze	いいえ	leaf	現在表示されているメンバーのみ、あるいは計算中のリーフ・メンバーすべてを含めるかどうか指定します。有効な値は以下のとおりです。 • displayed: 現在表示されているメンバーだけを含める。 • leaf: リーフ・メンバーすべてを含める。
name	いいえ		メニュー・バーの「データ」>「詳細」メニュー・オプションの下の、この分析のメニュー ID。名前を割り当てると、カスタム・イベント・ハンドラーとアクションのメニューへプログラマチックにアクセスできます。
number	いいえ	100	表示するデータのパーセンテージを指定します。 prompt 属性が true に設定されていると、 number 属性に別の値が設定されていなければ、デフォルト値 100 がポップアップ・ダイアログに表示されます。
preserveGrouping	いいえ	true	グループごと、またはグループに関係なく計算を行うかどうかを指定します。true が設定されると、軸に複数のディメンションがある場合、グループごとに計算を行います。
prompt	いいえ	false	数を求めるプロンプトを出すかどうかを指定します。 prompt が true に設定されると、ダイアログがポップアップ表示され、ユーザーがオプションを設定するように促します。

percentOfTotal タグの例

以下の例では、「合計のパーセント」オプションを右クリック・メニューとメニュー・バーの「データ」メニューに追加します。

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxuitld" prefix="bloxui"%>
...
<blox:grid ...>
   <bloxui:percentOfTotal/>
</blox:grid>
```

<bloxui:topN> タグ

この <bloxui:topN> タグは、PresentBlox または GridBlox のタグ内にネストされていなければなりません。これには、以下のタグ属性があります。

```
<bloxui:topN
   description=""
   hideOthers=""
   membersToAnalyze=""
   name=""
   number=""
   preserveGrouping=""
   prompt=""
   showRank=""
   showOtherSummary=""
/>
```

ここで、それぞれ以下のとおりです。

属性	必要	デフォルト	説明
description	いいえ	Top *N;* Top N...	「詳細」メニューのこのメニュー項目のテキストを指定します。デフォルトは、「Top *N*」で、*N* は number 属性の値です。prompt 属性が true に設定されると、デフォルトは「Top N....」です。
hideOthers	いいえ	all	残りの非ランク・メンバーと残りの計算に関係のないメンバーがビューに表示されるかどうかを指定します。有効な値は以下のとおりです。 all: デフォルト。非ランク・メンバーと、計算に関係のないメンバーを列軸と行軸の両方から非表示にします。 none: 非ランク・メンバーと計算に関係のないメンバーを、列軸と行軸の両方ですべて保持します。 unranked: 非ランク・メンバーのみ隠蔽します。ただし、計算に関係のないメンバーは反対の軸に保持します。

属性	必要	デフォルト	説明
membersToAnalyze	いいえ	leaf	現在表示されているメンバーのみ、あるいはリーフ・メンバーすべてをランク付けします。有効な値は以下のとおりです。 • displayed: 現在表示されているメンバーだけをランク付けします。 • leaf: リーフ・メンバーすべてをランク付けします。
name	いいえ		メニュー・バーの「データ」>「詳細」メニュー・オプションの下の、この分析のメニュー ID。名前を割り当てると、カスタム・イベント・ハンドラーとアクションのメニューへプログラマチックにアクセスできます。
number	いいえ	10	表示する最大の値のメンバー数を指定します。数値が指定されていない場合、オプション「上位 10 (Top 10)」がメニューに表示されます。
preserveGrouping	いいえ	true	prompt 属性が true に設定されると、この属性の値はポップアップ・ダイアログで表示されるデフォルトの数になり、表示するメンバー数を求めるプロンプトが出されます。 グループごとにメンバーをランク付けするか、グループに関係なくランク付けするかを指定します。true が設定されると、軸に複数のディメンションがある場合、グループごとに計算を行います。
prompt	いいえ	false	数を求めるプロンプトを出すかどうかを指定します。 prompt が true に設定されると、ダイアログがポップアップ表示され、ユーザーが参照したい最大の値のメンバー数を入力するように促します。プロンプト用にデフォルト値を指定するには、number 属性を使用します。
showOtherSummary	いいえ	false	残りのランクなしメンバーのサマリーとして、「その他」行/列を追加するかどうか指定します。
showRank	いいえ	true	追加した列/行でランキングを表示するかどうか指定します。 false に設定された場合、メンバーは、ランキングを表示する追加された列/行を使用しないランクによってソートされます。

topN タグの例

以下のコードは、「上位 10 詳細分析 (Top 10 advanced analysis)」オプションを右クリック・メニューに追加します。

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxuitld" prefix="bloxui"%>
...
<blox:grid ...>
   <bloxui:topN
     description="Top 10 and Other"
     number="10"
     hideOthers="all"
     showOtherSummary="true"
     showRank="true" />
</blox:grid>
```

ユーザーがこのオプションを選択すると、「[member name] Top 10」という列 (または行。これは、行ヘッダーまたは列ヘッダーのいずれを選択するかによって変わる) がグリッドに追加されます。その際、追加の「その他」メンバーが反対側の軸に表示されて、残りのランク外メンバーのサマリー値が提供されます。 `hideOthers` が `all` (デフォルト) に設定されることに注意してください。すると、以下のようになります。

- ランクなしメンバーは追加された「その他」メンバーに加えて表示されることはありません。混乱する可能性があるからです。
- 計算に関係のないメンバー (シナリオ・ディメンションの予算および分散など) は表示されません。

ランクなしメンバーを表示しないで、上記の例にあるシナリオ・ディメンションの予算と分散をそのままにしておきたい場合のみ、`hideOthers` を `unranked` に設定します。

カスタム・レイアウトのタグ

カスタム・レイアウト・タグは、主に DHTML クライアントの GridBlox ユーザー・インターフェースで作動します。このタグを使用すると、行ヘッダーや列ヘッダーをグリッドの中央または中心に配置する、任意の行または列を強調表示する、空行や空列を追加するなどして、グリッドのレイアウトをカスタマイズできます。レイアウト・タグには以下のものがあります。

- 434 ページの『<bloxui:butterflyLayout> タグ』
- 436 ページの『<bloxui:compressLayout> タグ』
- 438 ページの『<bloxui:customLayout> タグ』
- 438 ページの『<bloxui:gridHighlight> タグ』
- 440 ページの『<bloxui:gridSpacer> タグ』
- 444 ページの『<bloxui:title> タグ』 (PresentBlox および ChartBlox にも適用)

`<bloxui:title>` タグ以外、これらのタグは、`<blox:present>` または独立型 `<blox:grid>` タグ内にネストされていなければなりません。

<bloxui:butterflyLayout> タグ

このタグを使用して、行ヘッダー列をグリッド内の指定した位置に配置します。これには、以下のタグ属性があります。このタグは、<blox:present> または <blox:grid> タグ内にネストしなければなりません。

```
<bloxui:butterflyLayout
   addSeparatorColumns=""
   applyLayout=""
   description=""
   name=""
   position=""
   scope=""
   separatorWidth=""
   showOnLayoutMenu="" />
```

ここで、それぞれ以下のとおりです。

属性	必要	説明
addSeparatorColumns	いいえ	true に設定すると、両側でヘッダー列とデータ列の間に空列を追加します。デフォルトは false です。
applyLayout	いいえ	true - ページがロードされた時、このレイアウトに適用される。false - ページがロードされた時、このレイアウトに適用されない。デフォルトは true です。 ユーザーが選択するまでは、レイアウトを適用したくないと思うこともあるでしょう。このような場合、showOnLayoutMenu 属性を true に設定し、プレゼンテーション Blox のメニュー・バーをオンにします。
description	いいえ	showOnLayoutMenu が true に設定される場合の表示レイアウト名。デフォルト値は、「Butterfly」です。
name	いいえ	メニュー・バーの「フォーマット」メニューの下の、このレイアウトのメニュー ID。名前を割り当てると、カスタム・イベント・ハンドラーとアクションのメニューへプログラマチックにアクセスできます。
position	いいえ	ヘッダー列を、指定した有効範囲の前または後ろに追加するかどうかを指定する。有効な値は before および after です。デフォルトは before です。
showOnLayoutMenu	いいえ	true に設定すると、「フォーマット」メニューをメニュー・バーに (既存でない場合)、「レイアウト」サブメニューの「バタフライ」メニュー・オプションを伴って追加します。デフォルトは false です。

属性	必要	説明
scope	はい	表示されるヘッダー列と関連のメンバーを定義します。以下の例では、「予測 (Forecast)」の前に配置されるヘッダー列があります。 `scope="Forecast"` `postion="before"` ヘッダー列をタプルの前に配置するには、以下の例で示しているように、有効範囲のメンバーをセミコロンで分離し、有効範囲全体を中括弧で囲みます。 `scope="{Forecast; Qtr 1 01}"` `position="before"` タプルを指定すると、タプルへのヘッダー列の相対位置が保存されます。上の例では、タプル {Forecast; Qtr 1 01} が指定されており、ユーザーが Qtr 1 01 でドリルダウンすると、{Forecast; Qtr 1 01} が行/列ヘッダーの右側に表示される一方、タプル {Forecast; Jan 01}、{Forecast; Feb 01}、および {Forecast; Mar 01} が、ヘッダーの左側に表示されます。
separatorWidth	いいえ	区切り文字列の幅をピクセル単位で設定します。デフォルトは 10 ピクセルです。
showOnLayoutMenu	いいえ	true に設定すると、「フォーマット」メニューをメニュー・バーに (既存でない場合)、「レイアウト」サブメニューの「バタフライ」メニュー・オプションを伴って追加します。デフォルトは false です。

データの変更を行うと、レイアウトが不適切になる場合があるので、データ・ナビゲーション機能を、バタフライ・レイアウトで表示されるグリッド内に制限しても良いでしょう。以下に例を示します。

- ユーザーが scope に指定されたタプルの非表示を選択すると、このレイアウトは適用できません。バタフライ・レイアウトの代わりに、左に行ヘッダー列のある通常グリッドが表示されます。
- <bloxui:butterflyLayout> タグがサポートするのは、中央に行ヘッダーがあり、左と右にデータがある列ベース・レイアウト (垂直バタフライ) だけです。ユーザーがピボット軸やスワップ軸を選択すると (水平バタフライではない)、フォーマットは失われる場合があります。
- 行ヘッダー列は常に、指定した有効範囲の相対位置に基づいて表示されます。Qtr 1 01 でドリルダウンする場合、ヘッダー列をタプル {Forecast; Qtr 1 01} の前に表示されるように指定すると、タプル {Forecast; Jan 01}、{Forecast; Feb 01}、および {Forecast; Mar 01} は行/列ヘッダーの左に表示されます。一方、{Forecast; Qtr 1 01} はヘッダーの右に表示されます。

このタグはグリッド UI が特定のレイアウトで表示されるように作動するので、グリッドと何度か対話した後、レイアウトが消滅すると、ユーザーは混乱してしまうかもしれません。ツールバーとメニュー・バーを非表示にすると、データ・ナビゲーションを利用しにくくすることができます。さらに、<bloxui:menu> または <bloxui:component> タグを使用して、データ・ナビゲーション機能が使用できない

ようにしたり、共通 Blox プロパティー removeAction を設定して、特定のアクションを除去できます。さらに、このイベントを見つけたら、このアクションがサポート未対応であることをユーザーに知らせるメッセージ・ボックスを表示できます。追加情報や例に関しては、「*開発者用ガイド*」を参照してください。

butterflyLayout タグ例

以下の例は、バタフライ・レイアウトをメンバー Forecast の前にあるヘッダー列があるグリッドに適用します。さらに、バタフライ・レイアウト・メニュー・オプションをメニュー・バーの「フォーマット」メニューにも追加します。

```
<%@ taglib uri="bloxtld" prefix="blox" %>
<%@ taglib uri="bloxuitld" prefix="bloxui" %>
...
<blox:present id="myPresent" width="600" height="500" >
   ...
   <bloxui:butterflyLayout scope="Forecast"
      showOnLayoutMenu="true"/>
...
</blox:present>
```

<bloxui:compressLayout> タグ

このタグを使用して、列軸または行軸に複数のディメンションがある場合、あるレベルに列ヘッダーおよび行ヘッダーを圧縮します。これには、以下のタグ属性があります。このタグは、<blox:present> または <blox:grid> タグ内にネストしなければなりません。

```
<bloxui:compressLayout
   applyLayout=""
   compressColumns=""
   compressRows=""
   description=""
   memberSeparator=""
   name=""
   showOnLayoutMenu="" />
```

ここで、それぞれ以下のとおりです。

属性	必要	説明
applyLayout	いいえ	true - ページがロードされた時、このレイアウトに適用される。false - ページがロードされた時、このレイアウトに適用されない。デフォルトは true です。 ユーザーが選択するまでは、レイアウトを適用したくないと思うこともあるでしょう。このような場合、showOnLayoutMenu 属性を true に設定し、プレゼンテーション Blox のメニュー・バーをオンにします。
compressColumns	いいえ	true - 列ヘッダーをあるレベルまで圧縮します。列軸に複数のディメンションがある場合、ヘッダーをあるレベルまで圧縮できます。 compressColumns のデフォルト値は false です。この属性を true に設定すると、デフォルト・メンバーの区切り文字は、垂直バー (″ | ″) です。 **注:** そのような場合、または、compressRows 属性が指定されていない場合、タグは何も機能しません。

属性	必要	説明
compressRows	いいえ	true - 列ヘッダーをあるレベルまで圧縮します。行軸に複数のディメンションがある場合、ヘッダーをあるレベルまで圧縮できます。 compressRows のデフォルト値は false です。この属性を true に設定すると、デフォルト・メンバーの区切り文字は、垂直バー (″ | ″) です。 **注:** そのような場合、または、compressColumns 属性が指定されていない場合、タグは何も機能しません。
description	いいえ	showOnLayoutMenu が true に設定される場合の表示レイアウト名。デフォルト値は、「Compressed Headers」です。
memberSeparator	いいえ	メンバーの分離に使用するテキスト。デフォルト値は、前後にスペースを入れた垂直バー (″ | ″) です。
name	いいえ	メニュー・バーの「フォーマット」メニューの下の、このレイアウトのメニュー ID。名前を割り当てると、カスタム・イベント・ハンドラーとアクションのメニューへプログラマチックにアクセスできます。
showOnLayoutMenu	いいえ	true に設定すると、「フォーマット」メニューを (既存でない場合) メニュー・バーに追加します。その際、「レイアウト」サブメニューにメニュー・オプションが追加され、description 属性の値の後にラベルが付けられます。デフォルトは false です。

compressLayout タグの例

以下の例は、「/」をメンバー区切り文字として使用して、グリッドで行ヘッダーおよび列ヘッダーを圧縮します。また、圧縮レイアウト・メニュー・オプションをメニュー・バーの「フォーマット」メニューに追加します。

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

<html>
<head>
   <blox:header />
</head>
...
<body>
<blox:present id="myPresent" width="600" height="500" >
   ...
   <bloxui:compressLayout
       compressRows="true"
       compressColumns="true"
       showOnLayoutMenu="true"
       memberSeparator = " / " />
...
</blox:present>
...
</body>
</html>
```

注: 列ヘッダーまたは行ヘッダーが圧縮されると、すべてのモデル・コンポーネントが単一グリッドのヘッダー・セルにコピーされます。たとえば、Actual と

Qtr 3 01 が区切り記号として「/」を使用して圧縮されると、3 つの静的コンポーネントが単一セル内に配置されます。

<bloxui:customLayout> タグ

このタグでグリッド・レイアウトをカスタマイズします。これには、以下のタグ属性があります。このタグは、<blox:present> または <blox:grid> タグ内にネストしなければなりません。

```
<bloxui:customLayout
   applyLayout=""
   layout=""
   name=""
   showOnLayoutMenu=""/>
```

ここで、それぞれ以下のとおりです。

属性	必要	説明
applyLayout	いいえ	true - ページがロードされた時、このレイアウトに適用される。false - ページがロードされた時、このレイアウトに適用されない。デフォルトは true です。 ユーザーが選択するまでは、レイアウトを適用したくないと思うこともあるでしょう。このような場合、showOnLayoutMenu 属性を true に設定し、プレゼンテーション Blox のメニュー・バーをオンにします。
layout	はい	レイアウト・オブジェクトを指定します。たとえば、以下のようにします。 `<bloxui:customLayout` `layout="<%= new ButterflyLayout() %>" />` カスタム・レイアウトを伴うクラス名。com.alphablox.blox.uimodel.tags.layout パッケージの AbstractLayout クラス、またはこのパッケージの既存のクラスにインプリメントするカスタム・クラスにできます。
name	いいえ	メニュー・バーの「フォーマット」メニューの下の、このレイアウトのメニュー ID。名前を割り当てると、カスタム・イベント・ハンドラーとアクションのメニューへプログラマチックにアクセスできます。
showOnLayoutMenu	いいえ	true に設定すると、「フォーマット」メニューを (既存でない場合) メニュー・バーに追加します。その際、「レイアウト」サブメニューに追加のメニュー・オプションを伴い、layout 属性値の後にラベルが付けられます。デフォルトは false です。

<bloxui:gridHighlight> タグ

このタグは、使用する有効範囲とスタイルを指定して、メンバー (単数または複数) を強調表示します。これには、以下のタグ属性があります。このタグは、<blox:present> または <blox:grid> タグ内にネストしなければなりません。

```
<bloxui:gridHighlight
   applyLayout=""
   description=""
```

```
includeData=""
includeHeaders=""
name=""
scope=""
selection=""
showOnLayoutMenu=""
style="" />
```

ここで、それぞれ以下のとおりです。

属性	必要	説明
applyLayout	いいえ	true - ページがロードされた時、このレイアウトに適用される。false - ページがロードされた時、このレイアウトに適用されない。デフォルトは true です。 ユーザーが選択するまでは、レイアウトを適用したくないと思うこともあるでしょう。このような場合、showOnLayoutMenu 属性を true に設定し、プレゼンテーション Blox のメニュー・バーをオンにします。
description	いいえ	showOnLayoutMenu が true に設定される場合の表示レイアウト名。
includeData	いいえ	true - 指定した有効範囲にデータ・セルを含める。 false - 指定した有効範囲からデータ・セルを除外する。デフォルトは true です。
includeHeaders	いいえ	true - 指定した有効範囲にヘッダー・セルを含める。 false - 指定した有効範囲からヘッダー・セルを除外する。デフォルトは true です。
name	いいえ	メニュー・バーの「フォーマット」メニューの下の、このグリッド強調表示レイアウトのメニューID。名前を割り当てると、カスタム・イベント・ハンドラーとアクションのメニューへプログラマチックにアクセスできます。
scope	はい。それ以外の場合、selection を指定します。そうしないと、タグは何も機能しません。	強調表示するメンバーを定義します。有効範囲のメンバーを、中括弧で囲み、セミコロンを使用して区切ります。以下の例にあるのは、強調表示する West 領域の粗利益です。 scope="{West;Gross Margin}"
selection	はい。それ以外の場合、scope を指定します。そうしないと、タグは何も機能しません。	強調表示する rowHeaders または columnHeaders のいずれかを指定します。こうして、すべての行ヘッダーまたは列ヘッダーのスタイルをカスタマイズできます。
showOnLayoutMenu	いいえ	true に設定すると、「フォーマット」メニューを(既存でない場合) メニュー・バーに追加します。その際、「レイアウト」サブメニューにメニュー・オプションが追加され、description 属性の値の後にラベルが付けられます。デフォルトは false です。

属性	必要	説明
style	はい。それ以外の場合、通常どおりテーマ・ベースのスタイルが適用され、タグは何の影響も与えません。	グリッドの強調表示に付加するスタイル。デフォルトのスタイルまたはコンポーネント用のテーマ・ベースのスタイルをオーバーライドします。たとえば、以下のスタイルは、強調表示されたセルの背景色を黒 (枠なし) に設定します。 `style="background-color: black; border-style:none;"`

gridHighlight タグの例

以下の例では、ページがロードされると、黄色の背景色に黒い文字のグリッドで列ヘッダーを強調表示します。West 地域の粗利益を強調表示する別のレイアウトは、ページ・ロードの際に適用されません (applyLayout="false")。ユーザーはメニュー・バーの「フォーマット」メニューで、このレイアウトを適用することを選択できます。

```
<%@ taglib uri="bloxtld" prefix="blox" %>
<%@ taglib uri="bloxuitld" prefix="bloxui" %>
...
<html>
<head>
   <blox:header />
</head>
<body>
...
<blox:present id="myPresent" width="600" height="500" >
   <blox:data dataSourceName="myData" query="<your query here>"... />
   <bloxui:gridHighlight
      description="Highlight Column Headers"
      selection="columnHeaders"
      style="color: black; background-color: yellow;"
      showOnLayoutMenu="true"/>

   <bloxui:gridHighlight
      description="Highlight West Gross Margin"
      scope="{west;gross margin}"
      style="font-weight:bold; color: teal; background-color: #FFFF99;"
      showOnLayoutMenu="true"
      applyLayout="false"/>
...
</blox:present>
</body>
</html>
```

<bloxui:gridSpacer> タグ

このタグで列間または行間にスペースを追加します。これには、以下の属性があります。このタグは、<blox:present> または <blox:grid> タグ内にネストしなければなりません。

```
<bloxui:gridSpacer
   applyLayout=""
   axis=""
   description=""
   height=""
   locked=""
   name=""
   position=""
```

```
   scope=""
   showOnLayoutMenu=""
   style=""
   width="" />
```

ここで、それぞれ以下のとおりです。

属性	必要	説明
applyLayout	いいえ	true - ページがロードされた時、このレイアウトに適用される。false - ページがロードされた時、このレイアウトに適用されない。デフォルトは true です。 ユーザーが選択するまでは、レイアウトを適用したくないと思うこともあるでしょう。このような場合、showOnLayoutMenu 属性を true に設定し、プレゼンテーション Blox のメニュー・バーをオンにします。
axis	はい	column または row を指定してスペーサーを追加します。
description	いいえ	showOnLayoutMenu が true に設定される場合の表示レイアウト名。
height	いいえ	スペーサーが行軸 (axis="row") に追加されたときの、スペーサーの高さのピクセル数。デフォルトは 10 ピクセルです。
locked	いいえ	true に設定すると、画面のスペーサー位置をロックし、表示領域外をスクロールしません。 追加スペースは実際には空行/空列なので、 locked が false にされると、この空行/空列は、通常のデータ行/列のようにスクロールを行います。 この設定は、スペーサーが before または after の位置に追加された場合だけ適用されます。デフォルトは false です。
name	いいえ	メニュー・バーの「フォーマット」メニューの下の、このグリッド・スペーサー・レイアウトのメニュー ID。名前を割り当てると、カスタム・イベント・ハンドラーとアクションのメニューへプログラマチックにアクセスできます。

属性	必要	説明
position	はい	スペーサーの位置を指定します。有効な値は以下のとおりです。 • after: 指名されたメンバーの後ろにスペーサーを追加します。 scope 属性を使用してメンバーを指定します。 • before: 指名されたメンバーの前にスペーサーを追加します。 scope 属性を使用してメンバーを指定します。 • bottom: グリッドの下部にスペーサーを追加します。属性 axis は row に設定されていること。 • interlace: 列間または行間にスペーサーを追加します。 • left: グリッドの左にスペーサーを追加します。属性 axis は column に設定されていること。 • memberchange: 指定したディメンションのメンバーが (scope 属性を介して) 変更されると、スペーサーを追加します。 • right: 指名された軸の右にスペーサーを追加します。属性 axis は column に設定されていること。 • top: グリッドの上部にスペーサーを追加します。属性 axis は row に設定されていること。 • 数値 (number): N 番目の列または行にスペーサーを追加します。たとえば、次のコードは、 3 番目の行 (0 が最上部になる) としてスペーサーを追加します。 `position="2" axis="row"` 0 の値は、スペーサーをグリッドの最上部または左に追加したものと同じです。 axis 属性を、この属性が機能するように設定する必要があります。
scope	はい	position が memberchange に設定されると、scope を指定しなければなりません。 position が memberchange にされた時、使用するディメンションが入る有効範囲を定義します。以下の例では、Forecast のメンバーが変更されると、スペーサーを追加するように指定します。 `scope="Forecast"` `position="memberchange"`
showOnLayoutMenu	いいえ	true に設定すると、「フォーマット」メニューを (既存でない場合) メニュー・バーに追加します。その際、「レイアウト」サブメニューにメニュー・オプションが追加され、description 属性の値の後にラベルが付けられます。デフォルトは false です。

属性	必要	説明
style	いいえ	スペーサーに付加するスタイル。デフォルトのスタイルまたはコンポーネント用のテーマ・ベースのスタイルをオーバーライドします。たとえば、以下のスタイルでは、スペーサーの背景色を黒 (枠なし) に指定します。 `style="background-color: black; border-style:none;"` スタイルが指定されないと、テーマ・ベースのスタイルが使用されます。
width	いいえ	スペーサーが列軸 (axis="column") に追加されたときの、スペーサーの幅のピクセル数。デフォルトは 10 ピクセルです。

gridSpacer タグの例

以下の例では、 6 つのスペーサー (上部枠、下部枠、左枠、右枠、列区切り文字、およびロケーション区切り文字) をグリッドに追加します。

- 上部枠および下部枠は、axis 属性を row に設定し、 position 属性を top および bottom に設定して、追加します。
- 左枠および右枠は、axis 属性を column に設定し、 position 属性を left および right に設定して、追加します。
- 列区切り文字は、axis 属性を column に設定し、position を interlace に設定して追加します。すると、2 列ごとにスペーサーがあることになります。
- 「全ロケーション」ディメンションのメンバーに変更があったときにスペーサーを追加することにより、ロケーション区切り文字を追加してグループ化効果を作成します。これを実行するには、axis を row に、position を memberchange に、そして scope を All Locations に設定します。

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

<blox:data id="gridSpacerData"
   dataSourceName="qcc-essbase" useAliases="true" visible="false"
   query="<ROW ('All Locations', 'Measures')  'Central'  'East'  'West'
'All Locations' 'Gross Margin' <CHILD 'Ratios' <ASYM <COLUMN ('Scenario',
'All Time Periods')     'Actual'  'Actual'  'Forecast'  'Forecast'
'2000.Q3'  '2000.Q4'  '2001.Q1'  '2001.Q2'!" />

<html>
<head>
   <blox:header />
</head>
<body>
<blox:grid id="gridSpacer" width="80%" height="500" visible="true">
   <blox:data bloxRef="gridSpacerData" />
   <bloxui:toolbar name="standardToolbar" visible="false" />

   <bloxui:gridSpacer
      axis="column"
      position="right"
      width="5"
      style="background-color: red;"
      description="Right Border"
```

```
      showOnLayoutMenu="true" />

   <bloxui:gridSpacer
      axis="column"
      position="left"
      width="5"
      style="background-color: red;"
      description="Left Border"
      showOnLayoutMenu="true" />

   <bloxui:gridSpacer
      axis="row"
      position="top"
      height="5"
      style="background-color: red;"
      description="Top Border"
      showOnLayoutMenu="true" />

   <bloxui:gridSpacer
      axis="row"
      position="bottom"
      height="5"
      style="background-color: red;"
      description="Bottom Border"
      showOnLayoutMenu="true" />

   <bloxui:gridSpacer
      axis="column"
      position="interlace"
      width="2"
      style="background-color: yellow;"
      description="Column Separators"
      showOnLayoutMenu="true" />

   <bloxui:gridSpacer
      axis="row"
      position="memberchange" scope="All Locations"
      description="Location Separators"
      height="5"
      showOnLayoutMenu="true" />
</blox:grid>
</body>
</html>
```

<bloxui:title> タグ

<bloxui:title> タグを使用すると、プレゼンテーション Blox (PresentBlox、GridBlox、および ChartBlox) の先頭にタイトルを追加できます。一般の HTML タグを使用することに比べて、このタグを使用してタイトルを追加することの利点は、Blox ユーザー・インターフェースにより良く統合されることです。タイトルは、プレゼンテーション Blox の一部になるので、Blox に適用されるスタイルを自動的に継承し、Blox がブラウザーでサイズ変更されるように折り返します。

<bloxui:title> タグは、プレゼンテーション Blox 内で追加しなければなりません。これには、以下のタグ属性が含まれています。

```
<bloxui:title
   title=""
   style=""
   alignment="" />
```

ここで、それぞれ以下のとおりです。

属性	説明
title	表示するタイトル。
style	タイトルに適用するスタイル。たとえば、以下のようにします。 `style="font-family: Arial; font-weight: bold; font-size: 14pt; color: black; background-color: #FFFFCC;"`
alignment	タイトルの位置合わせ。有効な値は、`left`、`center`、および `right` です。

タイトルに定義されるスタイルは、レンダリングされた表セル全体ではなく、タイトルのテキストにのみ適用されます。タイトルの背景色を指定したい場合、プレゼンテーション Blox の背景色も同じ色を使用していることを確認してください。

プレゼンテーション Blox の背景色を設定するには、`<bloxui:component>` タグを使用します。以下の例では、この点が示されています。

タイトル・タグの例

以下に、GridBlox のタイトルを設定する方法の例を示します。

- `<bloxui:title>` タグは、GridBlox にネストされています。
- 背景色、テキスト色、フォント・サイズおよびフォント・スタイルは、`style` 属性を使用して設定します。
- `<bloxui:component>` タグを使用して、GridBlox の背景色をタイトルの背景色と同じ色に設定します。コンポーネント名は GridBlox の名前に設定されます。

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

<blox:data id="myDataTest"
   dataSourceName="qcc-essbase"
   useAliases="true" visible="false"
   query="<ROW ('All Locations', 'Measures') 'Central' 'East'
      'West' 'All Locations' 'Gross Margin' <CHILD 'Ratios'
       <ASYM <COLUMN ('Scenario', 'All Time Periods') 'Actual'
      'Actual'  'Forecast' 'Forecast' '2000.Q3' '2000.Q4' '2001.Q1'
      '2001.Q2'!" />

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

<blox:grid id="myGridTest"
   width="80%"
   height="80%"
   visible="true"
   menubarVisible="false"
   bandingEnabled="true"
   gridLinesVisible="false">
   <blox:data bloxRef="myDataTest" />
   <bloxui:component name="navigationToolbar" visible="false"/>
   <bloxui:component name="standardToolbar" visible="false"/>

   <bloxui:component name="myGridTest"
      style="background-color: #FFFFCC; border-style:none;" />
```

```
    <bloxui:title title="Sales and Gross Margin By Location - FY'02"
       style="font-family: Arial; font-weight: bold;
          font-size: 14pt; color: black; background-color: #FFFFCC;"
       alignment="center" />

  </blox:grid>
  </body>
  </html>
```

カスタム・メニュー・タグ

メニュー・バーをカスタマイズするために UI タグを使用すると、PresentBlox、GridBlox、または ChartBlox のデフォルトのメニュー・バーで、メニューやメニュー項目を追加、除去、使用不可にすることができます。このタグは、このようなユーザー・インターフェース Blox のタグ内にネストする必要があります。デフォルトでは、メニュー・バーは PresentBlox、スタンドアロンの GridBlox、またはスタンドアロンの ChartBlox 内で使用可能です (`menubarVisible="true"`)。

このセクションでは、`Menubar`、`Menu`、および `MenuItem` コンポーネントに関連した一般概念を説明し、こうしたコンポーネントのタグ・リファレンスを提供しています。

- 446 ページの『Menubar、Menu、および MenuItem』
- 446 ページの『Menu タグ・リスト』
- 447 ページの『<bloxui:menu> タグ属性』
- 448 ページの『ネストされた <bloxui:menuItem> タグ属性』
- 450 ページの『ネストされた <bloxui:clientLink> タグ属性』
- 450 ページの『組み込みメニューおよびメニュー項目名』
- 452 ページの『メニュー・タグの例』

Menubar、Menu、および MenuItem

メニュー・バーの各メニューは、除去、使用不可、位置変更可能な Menu コンポーネントです。各メニューにはメニュー項目があります。各メニュー項目も、除去、使用不可、位置変更を行うことができます。さらに、メニュー・バーにカスタムのメニューやメニュー項目を追加したり、メニュー項目に関連する操作をカスタマイズすることができます。

メニュー項目に関連したアクションを指定するには、 `<bloxui:clientLink>` タグを使用して URL をロードしたり、JavaScript 関数を呼び出すことができます。さらに、`<bloxui:actionFilter>` タグを介してサーバー・サイド・コードを呼び出すこともできます。詳しくは、463 ページの『<bloxui:actionFilter> タグ』を参照してください。

Menu タグ・リスト

```
<!--Nested within a PresentBlox, GridBlox, or ChartBlox -->

<bloxui:menu
   name=""
   bloxRef=""
   accesskey=""
   disabled=""
   positionBefore=""
```

```
   resourceName=""
   title=""
   tooltip=""
   visible=""
   >
   <bloxui:menuItem
      name=""
      accesskey=""
      checkable=""
      checked=""
      disabled=""
      imageURL=""
      positionBefore=""
      separator=""
      themeBasedImage=""
      title=""
      tooltip=""
      visible=""
      >
      <bloxui:clientLink
         features=""
         link=""
         target="" />
   </bloxui:menuItem>
</bloxui:menu>
```

<bloxui:menu> タグ属性

属性	必要	説明
name	はい	メニューの名前。指定された名前の Menu を検出すると、このタグをコンポーネントで実行します。そうでない場合、新規 Menu が作成されます。メニュー・バーでデフォルトのメニューをカスタマイズするには、以下の有効な値の 1 つを指定します。 • bookmarkMenu • chartMenu • dataMenu • editMenu • fileMenu • helpMenu • toolsMenu • viewMenu 定数を使用して値を指定することもできます。続く例では、メニュー・バーで「ツール」メニューを指定する 2 つの方法を示しています。 `name="<%= ModelConstants.TOOLS_MENU %>"` `name="toolsMenu"`
bloxRef	いいえ	タグが Blox タグ以外で使用される場合に、このタグに適用される既存の Blox を参照します。422 ページの bloxRef 属性の例を参照してください。

属性	必要	説明
accesskey	いいえ	アクセラレーター・キー。指定の文字を使用して、この Menu のタイトル中を検索します。それが見つかると、ユーザー・インターフェースの Menu のタイトル内の文字に下線が引かれ、それが Menu のキーボード・アクセラレーターであることを示します。 **注:** これはネストされたメニューでのみ機能し、メニュー・バーの最上位のメニューでは機能しません。
disabled	いいえ	true に設定すると、メニューが使用不可になります。このメニューは、メニュー・バーに表示されていますが、ぼかし表示です。
positionBefore	いいえ	メニューがこの前に表示される位置。このタグが指定されていないと、新しく追加されたメニューは、メニュー・バーの最後に追加されます。 DB2 ロゴの前にメニューを配置するには、この属性の値を次のように logo に設定します。 `positionBefore="logo"`
resourceName	いいえ	指名されたリソース・ファイルをコンポーネントにロードします。これにより、メニュー XML ファイルから新規メニューを作成するメニュー・タグを持つことができます。詳しくは、479 ページの『第 25 章 XML リソース・ファイル・リファレンス』を参照してください。
title	はい (カスタム・メニューの場合)	メニューの表示タイトル。メニュー・バーに追加されるカスタム・メニューにはタイトルがなければなりません。タイトルにスラッシュ (″/″) を含めることはできません。
tooltip	いいえ	マウスオーバーで表示されるツールチップ。
visible	いいえ	メニューの可視性。false に設定すると、このメニューはメニュー・バーに表示されません。デフォルトは true です。

ネストされた <bloxui:menuItem> タグ属性

属性	必要	説明
name	はい	メニュー項目の名前。カスタム・メニュー項目名を指定して、カスタム・メニュー項目を追加します。指定した名前の MenuItem が検出されると、タグがコンポーネントに作用します。そうでない場合、新規 MenuItem が作成されます。組み込みメニュー項目をカスタマイズするには、有効値について450 ページの『組み込みメニューおよびメニュー項目名』を参照してください。 定数を使用して値を指定することもできます。続く例では、データ・メニューで「すべて展開」メニュー項目を指定する 2 つの方法を示しています。 `name="<%= ModelConstants.DATA_EXPAND_ALL %>"` `name="dataExpandAll"`

属性	必要	説明
accesskey	いいえ	アクセラレーター・キー。指定の文字を使用して、この MenuItem のタイトル中を検索します。それが見つかると、ユーザー・インターフェースの MenuItem のタイトル内の文字に下線が引かれ、それが MenuItem のキーボード・アクセラレーターであることを示します。
checkable	いいえ	メニュー項目をチェック可能にする場合は、true です。メニュー項目が選択されると、そのメニュー項目の前にチェック・マークが表示されます。 **注:** 組み込みメニュー項目に checkable または checked 属性を設定する場合、カスタム・イベント・ハンドラーおよびコントローラーを追加する必要があります。追加しない場合、それらを設定しても効果がありません。
checked	いいえ	メニュー項目の前にチェック・マークが表示されるようにする場合は、true。
disabled	いいえ	true に設定すると、メニュー項目が使用不可になります。このメニュー項目はメニューに表示されていますが、ぼかし表示です。
imageURL	いいえ	使用するイメージの URL。themeBasedImage を true に設定すると、テーマのイメージが DB2 Alphablox リポジトリーに保管されているディレクトリーに、カスタム・イメージを入れておく必要があります。通常、このディレクトリーには以下の場所にあります。 <alphablox_dir>/repository/theme/<themeName>/i/ themeBasedImage を false 設定する場合、イメージの URL を指定します。URL は次のようになります。 • 絶対 URL。このストリングは “http://” で始めなければなりません。 • 相対 URL は以下のとおりです。 – ストリングをスラッシュ (/) で始めると、URL がサーバー・ルートに対して相対であることを示します。アプリケーション・コンテキストを URL に含める必要があるということに注意してください。 – ストリングをスラッシュ (/) なしで始めると、URL が現行の文書に対して相対であることを示します。
positionBefore	いいえ	指名されたメニュー項目が置かれる位置。この属性が指定されないと、新しく追加されたメニュー項目が、メニューの最後に追加されます。
separator	いいえ	true に設定すると、区切り線を追加します。

属性	必要	説明
themeBasedImage	いいえ	true に設定すると、テーマ・ベースのイメージを追加します。イメージは、テーマのイメージがリポジトリーで保管されるディレクトリーに入れておく必要があります。通常、このディレクトリーには以下の場所にあります。 <alphablox_dir>/repository/theme/<themeName>/i/ false に設定すると、テーマのイメージ・ディレクトリーに入っていないイメージを使用します。 imageURL 属性を使用し、イメージ・ファイルの URL を指定します。
title	はい (カスタム・メニュー項目の場合)	このメニュー項目の表示タイトル。追加されるカスタム・メニュー項目には、タイトルがなければなりません。タイトルにスラッシュ ("/") を含めることはできません。
tooltip	いいえ	マウスオーバーで表示されるツールチップ。
visible	いいえ	メニュー項目の可視性。false に設定すると、このメニュー項目はメニューに表示されません。デフォルトは true です。

ネストされた <bloxui:clientLink> タグ属性

複数の Blox UI タグに使用するネストされたタグです。詳しくは、468 ページの『<bloxui:clientLink> タグ』を参照してください。

組み込みメニューおよびメニュー項目名

Blox UI モデルが使用する共通コンポーネント名はすべて定数です。Javadoc の com.alphablox.blox.uimodel パッケージにある ModelConstants インターフェースでこの定数をすべて検出できます。定数名はすべて大文字になります。その値を Blox UI タグ属性で指定する場合、2 番目以降の語の先頭文字を大文字にして、それ以外のすべてを下線 ("_") なしの小文字にしなければなりません。便宜のために、組み込みメニュー名および組み込みメニュー項目名を以下の表で示しています。モデル定数の詳細なリストについては、470 ページの『モデル定数とその値』を参照してください。

メニュー	メニュー項目	メニュー定数
fileMenu		FILE_MENU
	fileOpen	FILE_OPEN
	fileSaveAs	FILE_SAVE_AS
	fileExportToExcel	FILE_EXPORT_TO_EXCEL
	fileExportToPDF	FILE_EXPORT_TO_PDF
editMenu		EDIT_MENU
	editUndo	EDIT_UNDO
	editRedo	EDIT_REDO
	editFind	EDIT_FIND

メニュー	メニュー項目	メニュー定数
	editHistory	EDIT_HISTORY
	editCopy	EDIT_COPY
	editDelete	EDIT_DELETE
	editSelectAll	EDIT_SELECTALL
viewMenu		VIEW_MENU
	viewChart	VIEW_CHART
	viewGrid	VIEW_GRID
	viewPageFilter	VIEW_PAGE_FILTER
	viewDataLayout	VIEW_DATA_LAYOUT
	viewToolbarMenu	VIEW_TOOLBAR_MENU
	viewToolbarCustomize	VIEW_TOOLBAR_CCUSTOMIZE
	viewPoppedOut	VIEW_POPPED_OUT
dataMenu		DATA_MENU
	dataSortAscending	DATA_SORT_ASCENDING
	dataSortDescending	DATA_SORT_DESCENDING
	dataDrillUp	DATA_DRILL_UP
	dataDrillDown	DATA_DRILL_DOWN
	dataExpandAll	DATA_EXPAND_ALL
	dataPivot	DATA_PIVOT
	dataShowOnly	DATA_SHOW_ONLY
	dataRemoveOnly	DATA_REMOVE_ONLY
	dataKeepOnly	DATA_KEEP_ONLY
	dataHide	DATA_HIDE
	dataUnhideAll	DATA_UNHIDE_ALL
	dataSwapAxes	DATA_SWAP_AXES
	dataOptions	DATA_OPTIONS
	dataNavigateButton	DATA_NAVIGATION_BUTTON
	dataAdvancedMenu	DATA_ADVANCED_MENU
	dataAdvancedDrillThrough	DATA_ADVANCED_DRILL_THROUGH
	dataAdvancedFormatMask	DATA_ADVANCED_FORMAT_MASK
	dataAdvancedMergedHeaders	DATA_ADVANCED_MERGED_HEADERS
	dataAdvancedSetHiddenColumns	DATA_ADVANCED_SET_HIDDEN_COLUMNS
	dataAdvancedSetHiddenMembers	DATA_ADVANCED_SET_HIDDEN_MEMBERS
	dataAdvancedSetHiddenMenu	DATA_ADVANCED_SET_HIDDEN_MENU
	dataAdvancedSetHiddenRows	DATA_ADVANCED_SET_HIDDEN_ROWS
	dataAdvancedShowBottomLevel	DATA_ADVANCED_SHOW_BOTTOM_LEVEL
	dataAdvancedShowSiblings	DATA_ADVANCED_SHOW_SIBLILING
	dataAdvancedTrafficLights	DATA_ADVANCED_TRAFFIC_LIGHTS
	dataCalculationEditor	DATA_CALCULATION_EDITOR
	dataCommentsMenu	DATA_COMMENTS_MENU
	dataCommentsAddComment	DATA_COMMENTS_ADD_COMMENT

メニュー	メニュー項目	メニュー定数
	dataCommentsDisplayComments	DATA_COMMENTS_DISPLAY_COMMENTS
	dataMemberFilter	DATA_MEMBER_FILTER
chartMenu		CHART_MENU
	chartTypesMenu	CHART_TYPES_MENU
	chartTypesLine	CHART_TYPES_LINE
	chartTypesBar	CHART_TYPES_BAR
	chartTypes3DBar	CHART_TYPES_3DBAR
	chartTypes3DPie	CHART_TYPES_3DPIE
	chartTypesMore	CHART_TYPES_MORE
	chartAxisPlacement	CHART_AXIS_PLACEMENT
	chartComboTypes	CHART_COMBO_TYPES
	chartDataValues	CHART_DATA_VALUES
	chartAllData	CHART_ALL_DATA
	chartSelectedData	CHART_SELECTED_DATA
	chartOptions	CHART_OPTIONS
toolsMenu		TOOLS_MENU
	toolsGridOptions	TOOLS_GRID_OPTIONS
	toolsPresentOptions	TOOLS_PRESENT_OPTIONS
	toolsManageMenu	TOOLS_MANAGE_MENU
	toolsManageTrafficLights	TOOLS_MANAGE_TRAFFIC_LIGHTS
helpMenu	helpHelp	HELP_HELP
	helpAbout	HELP_ABOUT

メニュー・タグの例

例 1: メニュー項目の除去

この例では、メニュー項目の可視性を false に設定して、メニュー・バーからメニュー項目を除去する方法を示します。この例では、「ツール」の「編集」メニュー項目と、「グリッド・オプション...」サブメニューが除去されます。

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxuitld" prefix="bloxui"%>
...
<html>
<head>
   <blox:header />
</head>
<body>
...
<blox:present menubarVisible="true" ...>
...
   <bloxui:menu name="editMenu" visible="false" />

   <bloxui:menu name="toolsMenu" >
      <bloxui:menuItem name="toolsGridOptions" visible="false" />
   </bloxui:menu>
...
```

```
</blox:present>
...
</body>
</html>
```

例 2: メニュー項目を使用不可にする

この例では、メニュー項目の disabled 属性を true に設定して、メニュー・バーからメニュー項目を使用できなくする方法を示します。この例では、「ツール」の「グリッド・オプション...」サブメニューが使用不可になります。

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

<blox:present menubarVisible="true"...>
...
   <bloxui:menu name="toolsMenu" >
      <bloxui:menuItem name="toolsGridOptions" disabled="true" />
   </bloxui:menu>
...
</blox:present>
</body>
</html>
```

例 3: メニュー項目の作成

この例では、3 つのオプションがある “Quick Links” と呼ばれるメニュー項目を作成します。2 番目のオプションにはサブメニューがあります。

- 最初のオプション “Today’s Stock Quotes” が選択されると、別のサーバーにあるページが、別のブラウザー・ウィンドウにロードされる。
- 区切り記号が、オプション 1 とオプション 2 の間に追加される。
- 2 番目のオプション “Reports...” の 2 つのサブメニューのいずれかを選択すると、同一のサーバーにあるページが別のブラウザー・ウィンドウにロードされる。
- 3 番目のオプション “Calendar” が選択されると、JavaScript 関数が呼び出される。

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

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

<blox:present menubarVisible="true"...>
...
   <bloxui:menu name="myMenu" title="Quick Links" visible="true">
      <bloxui:menuItem name="option1" title="Today's Stock Quotes">
         <bloxui:clientLink link="http://myserver/quotes.jsp"
            target="mywindow" />
      </bloxui:menuItem>

      <bloxui:menuItem separator="true" />
```

```
      <bloxui:menu name="option2" title="Reports...">
         <bloxui:menuItem name="submenu1" title="YTD Sales- East">
            <bloxui:clientLink link="east.jsp"
               target="mywindow" />
         </bloxui:menuItem>

         <bloxui:menuItem name="submenu2" title="Google">
            <bloxui:clientLink link="central.jsp"
               target="myotherwindow" />
         </bloxui:menuItem>
      </bloxui:menu>

      <bloxui:menuItem name="option3" title="Calendar">
         <bloxui:clientLink link="javascript:getCalendar();" />
      </bloxui:menuItem>

   </bloxui:menu>

</blox:present>

</body>
</html>
```

カスタム・ツールバーのタグ

ツールバーのカスタマイズに UI タグを使用すると、PresentBlox、GridBlox、または ChartBlox のデフォルトのツールバーで、メニューとメニュー項目の追加、除去、使用不可化が可能になります。このタグは、このようなユーザー・インターフェース Blox のタグ内にネストする必要があります。しかし、可能な場合はいつでも、ToolbarBlox のタグ属性を使用して、そのプロパティー値を設定するべきです。たとえば、removeButtonタグ属性を使用して、ボタンを除去することができます。DHTML クライアントを使用していて、ツールバーを Blox プロパティーで提供したものよりも高いレベルでカスタマイズする必要がある場合のみ、<bloxui:toolbar> および <bloxui:toolbarButton> タグを使用します。すべての Blox UI タグの場合と同様、これらのタグは DHTML クライアントで使用されるテーマ・ベースの Cascading Stylesheet クラスの設定値をオーバーライドするスタイルを使用します。

このセクションでは、Toolbar および ToolbarButton コンポーネントに関連した一般概念を説明し、こうしたコンポーネントのタグ・リファレンスを提供しています。

Toobar および ToolbarButton

PresentBlox には標準およびナビゲーションの 2 つのデフォルトのツールバーがあります。各ツールバーは、除去、使用不可化、位置変更が可能なツールバー・コンポーネントです。各ツールバーには、ツールバー・ボタンが含まれます。各ツールバー・ボタンも、除去、使用不可化、位置変更を行うことができます。さらに、カ

スタムのツールバーやツールバー・ボタンを追加したり、ツールバー・ボタンに関連する操作をカスタマイズすることができます。

カスタム・ツールバーを追加すると、メニュー・バーの「表示」-> 「ツールバー」メニュー・オプションは、自動的にリスト内にカスタム・ツールバーを含めます。

ツールバー・タグ・リスト

```
<bloxui:toolbar
   disabled=""
   bloxRef=""
   name=""
   positionBefore=""
   resourceName=""
   title=""
   tooltip=""
   visible="">
   <bloxui:toolbarButton
      checkable=""
      checked=""
      disable=""
      imageURL=""
      name=""
      positionBefore=""
      separator=""
      themeBasedImage=""
      title=""
      tooltip=""
      visible="" >
      <bloxui:clientLink
         features=""
         link=""
         target="" />
   </bloxui:toolbarButton>
</bloxui:toolbar>
```

<bloxui:toolbar> タグ属性

属性	必要	説明
name	はい	ツールバーの名前。カスタム・ツールバー名を指定して、カスタム・ツールバーを追加します。指定された名前の Toolbar を検出すると、このタグをコンポーネントで実行します。そうでない場合、新規 Toolbar が作成されます。プレゼンテーション Blox がツールバーをオンにして作成されている場合、すぐに使用可能な 2 つのツールバーをカスタマイズするには、以下のいずれかを指定します。 • 以下の定数を使用する。 – NAVIGATION_TOOLBAR – STANDARD_TOOLBAR • 以下の有効な値を使用する。 – navigationToolbar – standardToolbar 続く例では、「標準」ツールバーを指定する 2 つの方法を示しています。 `name="<%= ModelConstants.STANDARD_TOOLBAR %>"` `name="standardToolbar"`
bloxRef	いいえ	タグが Blox タグ以外で使用される場合に、このタグに適用される既存の Blox を参照します。422 ページの bloxRef 属性の例を参照してください。
disabled	いいえ	true に設定すると、ツールバーが使用不可になります。ツールバーは表示されますが、ボタンをクリックしても、何の影響もありません。
positionBefore	いいえ	ツールバーがこの前に表示される位置。このタグが指定されていないと、新しく追加されたツールバーは、ツールバーの最後 (ナビゲーション・ツールバーの後) に追加されます。 たとえば、次のようにナビゲーション・ツールバーの前にカスタム・ツールバーを置きます。 `positionBefore="navigationToolbar"`
resourceName	いいえ	指名されたリソース・ファイルをコンポーネントにロードします。これにより、ツールバー XML ファイルから新規ツールバーを作成するツールバー・タグを持つことができます。詳しくは、479 ページの『第 25 章 XML リソース・ファイル・リファレンス』を参照してください。

属性	必要	説明
title	はい (カスタム・ツールバーの場合)	ツールバーの表示タイトル。追加されるカスタム・ツールバーには、タイトルがなければなりません。このタイトルは、ToolbarBlox の textVisible プロパティーが true に設定されると (デフォルトは false)、ボタン・アイコン・イメージの下に表示されます。また、Blox の menubarVisible プロパティーが true に設定されると、同一のネストしているプレゼンテーション Blox (PresentBlox、GridBlox、または ChartBlox) にあるメニュー・バーの「表示」->「ツールバー」メニュー・オプションに自動的に追加されます。タイトルにスラッシュ (″/″) を含めることはできません。
tooltip	いいえ	マウスオーバーで表示されるツールチップ。
visible	いいえ	ツールバーの可視性。false に設定すると、ツールバー全体が表示されません。デフォルトは true です。

<bloxui:toolbarButton> タグ

属性	必要	説明
name	はい	ツールバー・ボタンの名前。カスタム・ツールバー・ボタン名を指定して、カスタム・ツールバー・ボタンを追加します。固有名を指定した ToolbarButton が検出されると、タグがコンポーネントに作用します。そうでない場合、新規 ToolbarButton が作成されます。組み込みツールバー・ボタンをカスタマイズするには、有効値について 459 ページの『組み込み Toolbar および ToolbarButton 名』を参照してください。 続く例では、「標準」ツールバーで「コピー」ボタンを指定する 2 つの方法を示しています。 `name="<%= ModelConstants.EDIT_COPY %>"` `name="editCopy"`
checkable	いいえ	ツールバー・ボタンをスティッキーにする場合は、true です。これは、そのボタンが、他のボタンをクリックするまで押されたままの状態になることです。 **注:** 組み込みツールバー・ボタンに checkable または checked 属性を設定する場合、カスタム・イベント・ハンドラーおよびコントローラーを追加する必要があります。追加しない場合、それらを設定しても効果がありません。
checked	いいえ	ツールバー・ボタンが押された状態で表示されるようにする場合は、true です。
disabled	いいえ	true に設定すると、ボタンが使用不可になります。ボタン・アイコンはツールバーに表示されますが、mouseOver または onClick イベントには応答しません。使用不可にしたいカスタム・ボタン接尾部 “_disabled” が付いた同一の名前のイメージを提供しなければなりません。詳しくは、imageURL 属性を参照してください。

属性	必要	説明
imageURL	いいえ	使用するイメージの URL。themeBasedImage を true に設定すると、テーマのイメージが DB2 Alphablox リポジトリーに保管されているディレクトリーに、カスタム・イメージを入れておく必要があります。通常、このディレクトリーには以下の場所にあります。 `<alphablox_dir>/repository/theme/<themeName>/i/` themeBasedImage を false 設定する場合、イメージの URL を指定します。URL は次のようになります。 • 絶対 URL。このストリングは “http://” で始めなければなりません。 • 相対 URL は以下のとおりです。 – ストリングをスラッシュ (/) で始めると、URL がサーバー・ルートに対して相対であることを示します。アプリケーション・コンテキストを URL に含める必要があるということに注意してください。 – ストリングをスラッシュ (/) なしで始めると、URL が現行の文書に対して相対であることを示します。 **ヒント:** アイコンごとに、ToolbarBlox の rolloverEnabled プロパティーが true (デフォルト) に設定される場合、2 つのイメージ (非アクティブ・モードおよびアクティブ・モード) を提供してください。ツールバー・ボタンが選択されると (アクティブ・モード)、DB2 Alphablox は自動的に同じ名前のイメージを検索しますが、“_active” 接尾部が付きます。このイメージ・ファイルがない場合、ブラウザーは欠落アイコンを表示します。使用不可のボタン (表示されても、mouseOver または onClick イベントに応答しないボタン) には、同じ名前のイメージに “_disabled” 接尾部を追加したものも提供してください。
positionBefore	いいえ	指名されたツールバー・ボタンが置かれる位置。このタグが指定されていないと、新しく追加されたツールバー・ボタンは、ツールバーの最後に追加されます。
separator	いいえ	true に設定すると、区切りバーを追加します。

属性	必要	説明
themeBasedImage	いいえ	true に設定すると、テーマ・ベースのイメージを追加します。イメージは、テーマのイメージがリポジトリーで保管されるディレクトリーに入れておく必要があります。通常、このディレクトリーには以下の場所にあります。 <alphablox_dir>/repository/theme/<themeName>/i/ false に設定すると、テーマのイメージ・ディレクトリーに入っていないイメージを使用します。 imageURL 属性を使用し、イメージ・ファイルの URL を指定します。
title	はい (カスタム・メニュー項目の場合)	ツールバー・ボタンの表示タイトル。追加されるカスタム・ツールバー・ボタンには、タイトルがなければなりません。タイトルにスラッシュ ("/") を含めることはできません。
tooltip	いいえ	マウスオーバーで表示されるツールチップ。
visible	いいえ	ツールバー・ボタンの可視性。false に設定すると、このツールバー・ボタンはメニューに表示されません。デフォルトは true です。

組み込み Toolbar および ToolbarButton 名

Blox UI モデルが使用する共通コンポーネント名はすべて定数です。Javadoc の com.alphablox.blox.uimodel パッケージにある ModelConstants インターフェースでこの定数をすべて検出できます。定数名はすべて大文字になります。その値を、Blox UI タグ属性で指定する場合、2 番目以降の語の先頭文字を大文字にして、それ以外のすべてを小文字にしなければなりません。便宜のために、組み込みツールバー名および組み込みツールバー・ボタン名を以下の表で示しています。この名前は、<bloxui:toolbar> および <bloxui:toolbarButton> タグだけで使用するべきであり、ToolbarBlox の removeButton プロパティーには適用されません。モデル定数の詳細なリストについては、470 ページの『モデル定数とその値』を参照してください。

ツールバー

standardToolbar のボタン	navigationToolbar のボタン
fileOpen	dataNavigateButton
fileSaveAs	dataSortAscending
editCopy	dataSortDescending
standardToolbarSeparator1 (editCopy ボタンの後の区切り記号)	dataMemberFilter
viewPoppedOut	navigationToolbarViewSeparator
editRedoButton	viewGrid
editUndoButton	viewChart
fileExportToPDF	viewPageFilter

standardToolbar のボタン	navigationToolbar のボタン
fileExportToExcel	viewDataLayout
standardToolbarSeparator3	
helpHelp	

「取り消し」、「再実行」、およびデータ・ナビゲーション・ボタン (「ドリルダウン」、「ドリルアップ」、「ピボット」、および「選択的表示」などのさまざまなオプションが含まれている) は、DropDownToolbarButton コンポーネントであり、ToolbarButton コンポーネントではありません。ただし、以下のように <bloxui:toolbarButton> タグを使用してそれらを除去することもできます。

```
 <bloxui:toolbar name="navigationToolbar" visible="true">
   <bloxui:toolbarButton
      name="<%=ModelConstants.DATA_NAVIGATE_BUTTON%>" visible="false"/>
</bloxui:toolbar>
```

またはその代わりに、以下のように総称 <bloxui:component> タグを使用してこれらの DropDownToolbarButtons を除去することもできます。

```
<bloxui:toolbar name="navigationToolbar" visible="true">
   <bloxui:component name="<%=ModelConstants.DATA_NAVIGATE_BUTTON%>"
      visible="false"/>
</bloxui:toolbar>
```

注: maximumUndoSteps の設定。共通 Blox プロパティーは、「取り消し」および「再実行」ボタンの可用性を制御します。maximumUndoSteps を 0 に設定した場合、「取り消し」および「再実行」ボタンは除去されます。maximumUndoSteps が 0 ではない場合、これらのボタンは表示されます。 44 ページの『maximumUndoSteps』を参照してください。

ツールバー・タグの例

例 1: ツールバー・ボタンの除去

この例では、ツールバー・ボタンの可視性を false に設定して、ナビゲーション・ツールバーからツールバー・ボタンを除去する方法を示します。

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

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

<blox:present ....>
...
<bloxui:toolbar name="navigationToolbar" >
   <bloxui:toolbarButton name="viewGrid" visible="false" />
   <bloxui:toolbarButton name="viewChart" visible="false" />
   <bloxui:toolbarButton name="viewPageFilter" visible="false" />
   <bloxui:toolbarButton name="viewDataLayout" visible="false" />
   </bloxui:toolbar>
...
</blox:present>
</body>
</html>
```

例 2: カスタム・ツールバーの追加

この例では、“My Toolbar” (title="My Toolbar") という表示名の付いた “myToolbar” (name="myToolbar") と呼ばれる Toolbar を作成します。

この例では、以下の点を例示します。

- ツールバーの位置を指定する positionBefore 属性の使用。
- イメージ URL を指定する絶対および相対 URL の使用。
- ツールバー・ボタンがクリックされる場合、URL を指定する <bloxui:clientLink> ネストされたタグの使用 (詳細は、468 ページの 『<bloxui:clientLink> タグ』を参照)。

メニュー・バーは、この新しいツールバーを「表示」->「ツールバー...」メニュー・オプションで自動的に反映します。

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

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

<body>
<blox:present id="myPresentBlox" width="700" height="500" >
   <blox:data dataSourceName="TBC" useAliases="true"
      query="<SYM <ROW(Product) <ICHILD Product <COLUMN(Year,
         Scenario) Qtr1 Qtr2 <CHILD Scenario Sales !" />

   <bloxui:toolbar name="myToolbar" title="My Toolbar"
      visible="true" positionBefore="navigationToolbar">

      <bloxui:toolbarButton name="option1" title="mail"
         themeBasedImage="false"
         imageURL="http://myserver/myApp/email.gif"
         tooltip="Check email alerts">
         <bloxui:clientLink link="emailAlerts.jsp"
            target="mywindow"
            features="toolbar=no,status=no" />
      </bloxui:toolbarButton>

      <bloxui:toolbarButton name="option2" title="Stocks"
         themeBasedImage="false" imageURL="../money.gif"
         tooltip="Today's Stocks">
         <bloxui:clientLink link="http://www.my.com/app/file.jsp"
            target="mywindow" />
      </bloxui:toolbarButton>

      <bloxui:toolbarButton name="option3" title="KPI"
         themeBasedImage="false" imageURL="/myApp/lookup.gif"
         tooltip="Show KPI" >
         <bloxui:clientLink link="javascript:myLookupFunction()"
                            target="mywindow" />
      </bloxui:toolbarButton>

      <bloxui:toolbarButton separator="true"
         positionBefore="option1" />
   </bloxui:toolbar>

</blox:present>

</body>
</html>
```

アクセシビリティ・タグ

<bloxui:accessibility> タグは、身体障害を持つユーザーのユーザー・エクスペリエンスを拡張するために設計されたものです。

- これは、視力の限られたユーザーが、メニュー・バーまたはツールバーをタブ操作せずに Blox の正しいデータ内容に移動できるようにする screenReaderMode 属性を提供します。たとえば、PresentBlox に <bloxui:accessibility> タグが追加され、screenReaderMode 属性が true に設定されると、レンダリングされた PresentBlox の前に、ネストされた GridBlox、ChartBlox、および DataLayoutBlox にアクセスするための 3 つのリンクが追加されます。ユーザーがリンクにタブ移動すると、スクリーン・リーダーはリンクのテキストを読み上げ、サブコンポーネントに直接移動するには Enter を押すように指示します。
- フォーカスを失ったオープン・ダイアログへのキーボード・アクセスを提供するための windowMenuEnabled 属性があります。 windowMenuEnabled 属性が true に設定されると、各オープン・ダイアログごとのメニュー項目を含むメニュー・バーに、**Windows** メニューが追加されます。万一ページ上のオープン・ダイアログにフォーカスがなくなり、キーボード・ユーザーがダイアログにタブ移動できなくなる場合は、メニュー・バーおよび特定のメニュー項目にタブ移動してダイアログにフォーカスを移すことができます。

<bloxui:accessibility> タグ

このタグには、以下の属性があります。

```
<bloxui:accessibility
   screenReaderMode=""
   windowMenuEnabled=""
 />
```

ここで、それぞれ以下のとおりです。

属性	必要	デフォルト	説明
screenReaderMode	いいえ	false	レンダリングされた Blox コンポーネントの前に、各主要サブコンポーネントへのリンクを追加します。これが PresentBlox の場合、ネストされた GridBlox、ChartBlox、および DataLayoutBlox の 3 つのリンクが追加されます。この属性があることによって影響を受けるのは、PresentBlox、 GridBlox、 ChartBlox、および DataLayoutBlox のみです。

属性	必要	デフォルト	説明
windowMenuEnabled	いいえ	false	各オープン・ダイアログのメニュー項目を含むメニュー・バーに、**Windows** メニューを追加します。オープン・ダイアログのメニュー項目を選択すると、ユーザーはキーボードのみを使ってそのダイアログにフォーカスを移すことができます。さらにすべてのダイアログを閉じるため、「**すべてのダイアログを閉じる (Close All Dialogs)**」メニュー項目も追加します。この属性があることによって、PresentBlox、スタンドアロンの GridBlox、およびスタンドアロンの ChartBlox が影響を受けます。

例

以下の例は、ネストされた GridBlox、ChartBlox、および DataLayoutBlox に素早くアクセスするために、PresentBlox の前のメニュー・バーおよびリンクに **Windows** メニューを追加します。

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxuitld" prefix="bloxui"%>
...
<blox:present id="accessiblePresent" menubarVisible="true">
   <blox:toolbar visible="true" />
   <blox:data bloxRef="myData" />
   <bloxui:accessibility
      screenReaderMode="true"
      windowMenuEnabled="true"
   />
</blox:present>
```

ユーティリティー・タグ

Blox UI タグ・ライブラリーには一式のユーティリティー・タグが含まれます。このようなタグを使用すると、ClickEvent が UI コンポーネントでトリガーされた場合にとるべきアクションを指定したり、<bloxui:customLayout> および <bloxui:customAnalysis> タグによって参照されるクラスでプロパティーを設定したり、サーバー・サイドのコードを起動して、GridBlox レイアウトをカスタマイズすることができます。以下のタグが含まれています。

- 463 ページの『<bloxui:actionFilter> タグ』
- 465 ページの『<bloxui:gridFilter> タグ』
- 468 ページの『<bloxui:clientLink> タグ』
- 469 ページの『<bloxui:setProperty> タグ』

<bloxui:actionFilter> タグ

<bloxui:actionFilter> タグを使用すると、 Blox UI タグ・ライブラリーを使用して Blox UI コンポーネントがクリックされたときに、そこからサーバー・サイドのコードを起動できます。

```
<bloxui:actionFilter
   componentName=""
   filter="" />
```

ここで、それぞれ以下のとおりです。

属性	説明
componentName	このアクション・フィルターが接続されるコンポーネント名。この名前のコンポーネントがクリックされると、ClickEvent がトリガーされ、名前付きクラスの actionFilter メソッドで指定されたアクションが実行されます。
filter	フィルター・オブジェクトを指定します。たとえば、以下のようにします。 `<bloxui:actionFilter` `filter="<%= new MyActionFilterClass() %>"` `componentName="dataPivot" />` ここで、MyActionFilterClass は IActionFilter をインプリメントし、これは JSP ページ内で定義されます。

com.alphablox.blox.uimodel.tags パッケージの IActionFilter インターフェースは、<bloxui:actionFilter> タグを使用して Blox に追加されたアクション・フィルターすべてでインプリメントしなければなりません。このインターフェースには、以下のシグニチャーがある 1 つのメソッドがあります。

```
void actionFilter(DataViewBlox blox, Component component);
    //throws java.lang.Exception
```

ここで、それぞれ以下のとおりです。

引数	説明
blox	アクション・フィルターの Blox
component	ClickEvent を生成するコンポーネント

このメソッドは、このアクション・フィルターが接続されて ClickEvent を生成するコンポーネントで毎回呼び出されます。このメソッドをインプリメントして、関連コンポーネントがクリックされるときにとるべきアクションを追加することができます。

このメソッドをインプリメントするため、少なくとも以下のパッケージがインポートされていることを確認してください。

```
<%@ page import="com.alphablox.blox.uimodel.*,
                 com.alphablox.blox.uimodel.tags.IActionFilter,
                 com.alphablox.blox.DataViewBlox,
                 com.alphablox.blox.uimodel.core.Component" %>
```

他のパッケージをインポートすることもできます。Java Integrated Development Environment (IDE) を使用すると、どのパッケージをインポートするかを識別できます。

ユーティリティー・タグの例

以下の例では、<bloxui:actionFilter> タグの使用と、IActionFilter インターフェースのインプリメント方法、および、コンポーネントがクリックされた時にアクションを実施する actionFilter メソッドへの拡張方法を示しています。この場合、カスタム・メニュー項目がクリックされると、MessageBox には、メッセージが表示されます。

```
<%@ page import="com.alphablox.blox.uimodel.*,
                 com.alphablox.blox.uimodel.tags.IActionFilter,
                 com.alphablox.blox.DataViewBlox,
                 com.alphablox.blox.uimodel.core.Component,
                 com.alphablox.blox.uimodel.core.MessageBox"%>

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

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

<blox:present ....>

   <bloxui:menu name="toolsMenu" >
      <bloxui:menuItem name="myToolMenuItem" title="Get Message" />
   </bloxui:menu>

   <bloxui:actionFilter
      filter="<%= new MyActionFilterClass() %>"
      componentName="myToolMenuItem" />
...
</blox:present>
...
<%!
public static class MyActionFilterClass implements IActionFilter
{
   public void actionFilter( DataViewBlox blox, Component component )
throws Exception {
      MessageBox.message( component, "Get Message", "The myToolMenuItem has
been clicked!" );
   }
}
%>
```

<bloxui:gridFilter> タグ

<bloxui:gridFilter> タグを使用すると、サーバー・サイドのコードを起動し、Blox UI タグ・ライブラリーを使用して GridBlox レイアウトのカスタマイズができます。これには、以下のタグ属性があります。

```
<bloxui:gridFilter
filter="" />
```

ここで、それぞれ以下のとおりです。

属性	説明
filter	フィルター・オブジェクトを指定します。たとえば、以下のようにします。 `<bloxui:gridFilter` `filter="<%= new MyGridFilterClass() %>"` `componentName="dataPivot" />`

ここで、MyGridFilterClass は IGridFilter をインプリメントし、これは JSP ページ内で定義されます。

グリッド・フィルターを使用して、ロードされる前にグリッドのカスタマイズや再ビルドを行うことができます。com.alphablox.blox.uimodel.tags パッケージの IGridFilter インターフェースは、 <bloxui:gridFilter> タグを使用して Blox に追加されたアクション・フィルターすべてでインプリメントしなければなりません。このグリッドは、各データ・ナビゲーション・コマンドが処理された後に再ビルドされます。このインターフェースには、以下のシグニチャーがある 2 つのメソッドがあります。

```
void gridFilter(DataViewBlox blox, GridBrixModel grid);
                  // throws java.lang.Exception

void cellFilter(DataViewBlox blox, GridCell cell);
                  // throws java.lang.Exception
```

ここで、それぞれ以下のとおりです。

引数	説明
blox	グリッド・フィルターの Blox
grid	再ビルドされたグリッド
cell	再ビルドされたグリッド・セル

このフィルターは、再ビルド後にグリッドに適用される多くのフィルターの 1 つである可能性があるので、このフィルターによってグリッドのレイアウトに関して推測を行うべきではありません。

このメソッドをインプリメントするため、少なくとも以下のパッケージがインポートされていることを確認してください。

```
<%@ page import="com.alphablox.blox.uimodel.*,
                 com.alphablox.blox.uimodel.tags.IActionFilter,
                 com.alphablox.blox.uimodel.tags.IGridFilter,
                 com.alphablox.blox.DataViewBlox,
                 com.alphablox.blox.uimodel.GridBrixModel,
                 com.alphablox.blox.uimodel.GridBrixCellModel,
                 com.alphablox.blox.uimodel.core.grid.GridRow,
                 com.alphablox.blox.uimodel.core.grid.GridCell,
                 com.alphablox.blox.uimodel.core.Component" %>
```

他のパッケージをインポートすることもできます。Java Integrated Development Environment (IDE) を使用すると、どのパッケージをインポートするかを識別できます。

gridFilter タグの例

以下の例では、<bloxui:gridFilter> タグの使用と、IGridFilter インターフェースのインプリメント方法、および、gridFilter メソッドを拡張して、グリッドを再ビルドする方法を示しています。この例の内容は以下のとおりです。

- 以下のように移動します。
  - 行ヘッダーはグリッドの末尾に移動される。
  - 列ヘッダーはグリッドの下部に移動される。
- ボタン・コンポーネントが各列の末尾に追加されます。

- グリッドが再ビルドされると、グリッドの変更を通知する MessageBox がポップアップ表示されます。
- そして、グリッドが表示されます。

同様の例については、Blox Sampler を参照してください。

```
<%@ page import="com.alphablox.blox.uimodel.tags.IGridFilter,
                 com.alphablox.blox.DataViewBlox,
                 com.alphablox.blox.uimodel.GridBrixModel,
                 com.alphablox.blox.uimodel.core.grid.GridRow,
                 com.alphablox.blox.uimodel.core.grid.GridCell,
                 com.alphablox.blox.uimodel.core.Button,
                 com.alphablox.blox.uimodel.ModelConstants,
                 com.alphablox.blox.uimodel.core.grid.GridColumn,
                 com.alphablox.blox.uimodel.core.MessageBox,
                 com.alphablox.blox.uimodel.GridBrixCellModel"%>
<%@ taglib uri="bloxtld" prefix="blox" %>
<%@ taglib uri="bloxuitld" prefix="bloxui" %>

<blox:data id="dataBlox" dataSourceName="qcc-essbase"
   useAliases="true" visible="false"
   query="<ROW ('All Locations') 'Central' 'East' 'West' 'All Locations'
<ASYM <COLUMN ('Scenario', 'All Time Periods') 'Actual' 'Actual' 'Forecast'
'Forecast' '2000.Q3' '2000.Q4'  '2001.Q1'  '2001.Q2'!" />

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

<body>
<blox:grid id="testGridMoveFilter" width="80%" height="500"
   bandingEnabled="true" menubarVisible="true">
   <blox:toolbar visible="true" />
   <bloxui:gridFilter
      filter="<%= new MyGridFilterClass() %>" />
   <blox:data bloxRef="dataBlox" />
</blox:grid>

</body>
</html>

<%!
public static class MyGridFilterClass implements IGridFilter
{
   public void gridFilter( DataViewBlox blox, GridBrixModel grid ) throws
Exception {
      // Move row headers to the end of the grid
      if ( grid.getColumnCount() > 1 )
         while ( grid.getColumn( 0 ).isHeader() )
            grid.moveColumn( 0, grid.getColumnCount() );

      // Move column headers to the end of the grid
      if ( grid.getRowCount() > 1 )
         while ( grid.getRow( 0 ).isHeader() )
            grid.moveRow( 0, grid.getRowCount() );

      GridRow row = new GridRow( );

      // Add a button to the end of each column. For this example,
      // these buttons do not do anything.
      for ( int i=0; i < grid.getColumnCount(); i++ ) {
         GridCell cell = new GridCell( "myCell" + (i+1)  );
         cell.add( new Button( cell.getName(), cell.getName() ) );
         cell.setClickable( false );
         row.add( cell );
```

```
      }

      grid.addRow( row );

      row = new GridRow();
      row.setHeight( 4 );
      row.setThemeClass( ModelConstants.THEME_STYLE_ROW_DATA_GENERATION +
"3" );
      grid.insertRow( 4, row );

      GridColumn column = new GridColumn();
      column.setWidth( 4 );
      column.setThemeClass( ModelConstants.THEME_STYLE_ROW_DATA_GENERATION
+ "3" );
      grid.insertColumn( 4, column );
      MessageBox.message( grid, "Change", "The grid has changed" );
   }
   public void cellFilter( DataViewBlox blox, GridCell cell ) throws
Exception {
   }
}
%>
```

この例の目的は、<bloxui:gridFilter> タグによるグリッド・レイアウトのカスタマイズと再ビルドの方法を示すことだけです。これは高度な技法であり、グリッドのスクロールの仕方に影響する可能性があります。さらに、グリッドに追加されるボタンには、それがクリックされるときの関連付けられたアクションがありません。

<bloxui:clientLink> タグ

<bloxui:clientLink> タグを使用すると、URL を指定してコンポーネントがクリックされるとすぐに、既存のまたは別のブラウザー・ウィンドウにロードすることができます。これは、<bloxui:menuItem> および <bloxui:toolbarButton> といったコンポーネント・タグ内に追加します。これには、以下のタグ属性があります。

```
<bloxui:clientLink
   features=""
   link=""
   target="" />
```

属性	説明
features	コンマで区切られたブラウザー機能ストリング。target 属性が指定されている場合、この属性は新規のブラウザー・ウィンドウのための機能を設定します。たとえば、features="toolbar=no,status=no" などとなります。ブラウザー機能ストリングは、JavaScript の window.open() メソッドと同じ方法で指定する必要があります。
link	URL。これは、以下のいずれかになります。 • 絶対 URL。このストリングは “http://” で始めなければなりません。 • 相対 URL は以下のとおりです。 – ストリングをスラッシュ (/) で始めると、URL がサーバー・ルートに対して相対であることを

示します。アプリケーション・コンテキストを URL に含める必要があるということに注意してください。

– ストリングをスラッシュ (/) なしで始めると、URL が現行の文書に対して相対であることを示します。

- JavaScript 関数名。ストリングは「javascript:」接頭部で始めます。

target — link 属性で指定された URL をロードするブラウザー・ウィンドウの名前。ブラウザー機能は features 属性で指定されます。この属性が指定されていないと、URL は現在のブラウザー・ウィンドウにロードされます。

このタグを他の Blox UI タグと連動して使用する方法の例については、453 ページの 『例 3: メニュー項目の作成』および 461 ページの 『例 2: カスタム・ツールバーの追加』を参照してください。

<bloxui:setProperty> タグ

<bloxui:setProperty> タグを使用すると、レイアウトまたは分析クラスのプロパティーの値を設定できます。たとえば、<bloxui:customAnalysis> または <bloxui:customLayout> タグ内でクラスを使用するように指定する場合、この <bloxui:setProperty> タグを使用して、指名されたプロパティーの値を指定できます。これには、以下のタグ属性があります。

```
<bloxui:setProperty
   name=""
   value="" />
```

ここで、それぞれ以下のとおりです。

属性	説明
name	プロパティーの名前。
value	プロパティーの値。

setPropertyTag の例

以下に、デフォルトのメンバー数を設定して、TopN 分析クラスによってトリガーされるポップアップ・ダイアログで表示する方法の例を示します。

```
<bloxui:customAnalysis
   analysis="<%= new TopN()" >
   <bloxui:setProperty name="number" value="15" />
</bloxui:customAnalysis>
```

詳しい例については、427 ページの『<bloxui:customAnalysis> タグ』を参照してください。

モデル定数とその値

このセクションでは、共通のモデル定数とその値をリストしています。完全なリストについては、 Javadoc の `com.alphablox.blox.uimodel` パッケージにある `ModelConstants` インターフェースを参照してください。*定数名はすべて大文字になります。その値を Blox UI タグ属性で指定する場合、2 番目以降の語の先頭文字を大文字にして、それ以外のすべてを下線 ("_") なしの小文字にしなければなりません。*

チャート・エレメント

定数	値
CHART	chart
CHART_FILTER	chartFilter
CHART_FILTERS_CONTAINER	chartFiltersContainer
CHART_TOTALS_FILTER	chartTotalsFilter

メニュー

定数	値
MAIN_MENU	mainMenu
HELP_MENU	helpMenu
TOOLS_MENU	toolsMenu
DATA_MENU	dataMenu
DATA_ADVANCED_MENU	dataAdvancedMenu
FORMAT_MENU	formatMenu
BOOKMARK_MENU	bookmarkMenu
VIEW_MENU	viewMenu
CHART_MENU	chartMenu
FILE_MENU	fileMenu
EDIT_MENU	editMenu
VIEW_TOOLBAR_MENU	viewToolbarMenu
CHART_TYPES_MENU	chartTypesMenu
DATA_COMMENTS_MENU	dataCommentsMenu
FORMAT_LAYOUT_MENU	formatLayoutMenu
TOOLS_MANAGE_MENU	toolsManageMenu

メニュー・エレメント

定数	値
BOOKMARK_ADD	bookmarkAdd
BOOKMARK_LOAD	bookmarkLoad
BOOKMARK_ORGANIZE	bookmarkOrganize
CHART_COMBO_TYPES	chartComboTypes
CHART_AXIS_PLACEMENT	chartAxisPlacement
CHART_DATA_VALUES	chartDataValues
CHART_ALL_DATA	chartAllData
CHART_SELECTED_DATA_ONLY	chartSelectedDataOnly
CHART_OPTIONS	chartOptions
CHART_TYPES_PIE	chartTypePie
CHART_TYPES_LINE	chartTypeLine
CHART_TYPES_BAR	chartTypesBar
CHART_TYPES_MENU	chartTypesMenu
CHART_TYPES_3DPIE	chartType3DPie
CHART_TYPES_MORE	chartTypesMore
CHART_TRENDLINES	chartTrendlines
HELP_HELP	helpHelp
HELP_ABOUT	helpAbout
DATA_SORT	dataSort
DATA_SORT_ASCENDING	dataSortAscending
DATA_SORT_DESCENDING	dataSortDescending
DATA_DRILL_UP	dataDrillUp
DATA_DRILL_DOWN	dataDrillDown
DATA_EXPAND	dataExpand
DATA_COLLAPSE	dataCollapse
DATA_EXPAND_ALL	dataExpandAll
DATA_PIVOT	dataPivot
DATA_SHOW_ONLY	dataShowOnly
DATA_REMOVE_ONLY	dataRemoveOnly
DATA_KEEP_ONLY	dataKeepOnly
DATA_HIDE	dataHide
DATA_UNHIDE_ALL	dataUnhideAll
DATA_SWAP_AXES	dataSwapAxes
DATA_MEMBER_FILTER	dataMemberFilter
DATA_CALCULATION_EDITOR	dataCalculationEditor
DATA_OPTIONS	dataOptions
DATA_ADVANCED_DRILL_THROUGH	dataAdvancedDrillThrough
DATA_ADVANCED_FORMAT_MASK	dataAdvancedFormatMask
DATA_ADVANCED_MERGED_HEADERS	dataAdvancedMergedHeaders

定数	値
DATA_ADVANCED_SHOW_BOTTOM_LEVEL	dataAdvancedShowBottomLevel
DATA_ADVANCED_SHOW_SIBLINGS	dataAdvancedShowSiblings
DATA_ADVANCED_SET_HIDDEN_MENU	dataAdvancedSetHidden
DATA_ADVANCED_SET_HIDDEN_ROWS	dataAdvancedSetHiddenRows
DATA_ADVANCED_SET_HIDDEN_COLUMNS	dataAdvancedSetHiddenColumns
DATA_ADVANCED_SET_HIDDEN_MEMBERS	dataAdvancedSetHiddenMembers
DATA_ADVANCED_TRAFFIC_LIGHTS	dataAdvancedTrafficLights
DATA_COMMENTS_DISPLAY_COMMENTS	dataCommentsDisplayComments
DATA_COMMENTS_ADD_COMMENT	dataCommentsAddComment
EDIT_UNDO	editUndo
EDIT_REDO	editRedo
EDIT_COPY	editCopy
EDIT_DELETE	editDelete
EDIT_SELECT_ALL	editSelectAll
EDIT_FIND	editFind
EDIT_HISTORY	editHistory
FILE_OPEN	fileOpen
FILE_SAVE_AS	fileSaveAs
FILE_EXPORT_TO_PDF	fileExportToPDF
FILE_EXPORT_TO_EXCEL	fileExportToExcel
TOOLS_GRID_OPTIONS	toolsGridOptions
TOOLS_MANAGE_TRAFFIC_LIGHTS	toolsManageTrafficLights
TOOLS_PRESENT_OPTIONS	toolsPresentOptions
VIEW_GRID	viewGrid
VIEW_CHART	viewChart
VIEW_PAGE_FILTER	viewPageFilter
VIEW_DATA_LAYOUT	viewDataLayout
VIEW_POPPED_OUT	viewPoppedOut
VIEW_TOOLBAR_CUSTOMIZE	viewToolbarCustomize
LOGO	logo
DATA_NAVIGATE_BUTTON	dataNavigateButton
EDIT_UNDO_BUTTON	editUndoButton
EDIT_REDO_BUTTON	editRedoButton

ダイアログ・ボタン

定数	値
OK	ok
CANCEL	cancel
YES	yes
NO	no

定数	値
APPLY	apply
HELP	help

ツールバー

定数	値
STANDARD_TOOLBAR	standardToolbar
NAVIGATION_TOOLBAR	navigationToolbar

汎用エレメント

定数	値
HEADER_CONTAINER	headerContainer
BODY_CONTAINER	bodyContainer
PRESENT_SPLITTER	presentSplitter
TREENODE_LABEL	treeNodeLabel
GRID_CELL_VALUE	gridCellValue
DATA_LAYOUT_LIST	dataLayoutList
DATA_LAYOUT_TREE	dataLayoutTree
DATA_LAYOUT_ROW_CONTAINER	dataLayoutRowContainer
DATA_LAYOUT_COLUMN_CONTAINER	dataLayoutColumnContainer
DATA_LAYOUT_PAGE_CONTAINER	dataLayoutPageContainer
DATA_LAYOUT_OTHER_CONTAINER	dataLayoutOtherContainer

第 24 章 PDF レンダリング・タグ

Blox ユーザー・インターフェースのデフォルトのツールバーには「PDF にエクスポート」アイコンがあり、ユーザーはページ上の Blox の現在のビューを印刷またはアーカイブするために PDF フォーマットに変換することができます。開発者は、<blox:pdfReport> タグを使用して、ヘッダー、フッター、それらの高さ、ページ・マージン、およびページ・サイズを指定することができます。さらに、<blox:pdfDialogInput> タグを使ってポップアップ・ダイアログをカスタマイズして、こうしたさまざまな設定を指定するようユーザーに促すことができます。また、提供されたタグを使用してページ上の複数の Blox を 1 つの PDF にレンダリングすることもできます。

PDF にレンダリングするためのタグ

以下に PDF レンダリングに関連したタグと属性を示します。

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

PDF レポート作成およびタグの使用に関する詳細については、「*開発者用ガイド*」の『PDF に変換』のトピックを参照してください。

pdfReport タグ属性

pdfReport の属性	説明
header	テキストおよびレイアウトを含むヘッダー。 XHTML およびマクロを使用して定義されます。 477 ページの『PDF 変換のマクロ』を参照してください。
headerHeight	ヘッダーの高さ。有効な単位は、ピクセル (px)、ポイント (pt)、インチ (in)、ミリメートル (mm)、およびセンチメートル (cm) です。指定しない場合、ピクセル (px) が使用されます。
footer	テキストおよびレイアウトを含むフッター。 XHTML タグおよびマクロを使用して定義されます。 477 ページの『PDF 変換のマクロ』を参照してください。
footerHeight	フッターの高さ。有効な単位は、ピクセル (px)、ポイント (pt)、インチ (in)、ミリメートル (mm)、およびセンチメートル (cm) です。指定しない場合、ピクセル (px) が使用されます。

pdfReport の属性	説明
margin	マージン。有効な単位は、ピクセル (px)、ポイント (pt)、インチ (in)、ミリメートル (mm)、およびセンチメートル (cm) です。指定しない場合、ピクセル (px) が使用されます。
pageBreak	改ページ設定の規則。規則は、ディメンションとメンバーの指定リストの後に、@ before または @ after キーワードが続いたもので構成されます。たとえば、以下のようになります。 `Dim1: M1, M2; Dim2: M3, M4 @ after` 上の例では、ディメンション Dim1 のメンバー M1 または M2、またはディメンション Dim2 の M3 または M4 を変更する際は常に、データの**後ろ (after)** に改ページを追加するよう指定しています。 ディメンションとメンバーの指定の区切りには、セミコロンを使います。同じディメンションでのメンバーの区切りには、コンマを使います。指定の条件を満たす場合には、改ページが追加されます。キーワード @ before および @ after は、大文字小文字を区別しません。キーワードのスペルが間違っていると、例外がスローされます。
size	用紙サイズ。用紙サイズ (A3、A4、Letter、Legal) および用紙の向き (landscape、portrait) の定義に使用されます。有効な属性は以下のとおりです。 `[ A3 | A4 | Letter | Legal | Custom` `[ [Portrait | Landscape] | [`*width* `| [ `*height*`]]]` 例: `size="Letter Portrait"` `size="A4 Landscape"` `size="Custom 15in 100mm"` デフォルトのページ・サイズはロケールによって異なります。米国またはカナダにおけるデフォルトは Letter で、それ以外の国のページ・サイズのデフォルトは A4 です。デフォルトの用紙の向きは、Landscape です。
theme	テーマ名。DB2 Alphablox Repository で使用されるテーマ名と同じものです。
themeListEnabled	使用可能なテーマ・リスト。値は true (デフォルト) または false です。

注: XHTML タグおよび CSS の制限:

- はサポートされていません。
- 改行なしスペースの場合、XHTML 文字 ( ) の代わりに Unicode 文字 ( ) を使用します。
- CSS 省略属性は CSS の仕様に従う必要があります。

PDF 変換のマクロ

ヘッダーおよびフッターのタグ属性で使用可能なマクロは、以下のとおりです。

マクロ	説明
<date/>	今日の日付
<time/>	現在時刻
<totalpages/>	ページ数
<pagenumber/>	現行ページ
<pdfDialogInput*N*/>	PDF ダイアログの入力テキスト。 N は、1 から 5 までの整数です。デフォルトで、<pdfdialogINput1/> はヘッダーを定義し、<pdfDialogInput2/> はフッターを定義します。

pdfReport タグの例

```
<%
  String footer="<span style='color:red'>Total: <totalpages/> pages</span>";
%
<blox:pdfReport
    footer="<%=footer>"
    footerHeight="20px"
    size="Letter portrait"
    margin="0"
    theme="financial"
    themeListEnabled="false"/>
```

pdfDialogInput タグ属性

pdfDialogInput の属性	説明
index	テキストおよびレイアウトを含むヘッダー。 XHTML およびマクロを使用して定義されます。
defaultValue	ヘッダーの高さ。有効な単位は、ピクセル (px)、ポイント (pt)、インチ (in)、ミリメートル (mm)、およびセンチメートル (cm) です。指定しない場合、ピクセル (px) が使用されます。
displayName	テキストおよびレイアウトを含むフッター。 XHTML タグおよびマクロを使用して定義されます。

pdfDialogInput タグにより、ユーザーに値を指定させたい設定を指定することができます。たとえば、以下のようにすることができます。

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

ユーザーは、レポートのヘッダーとフッターを指定できるダイアログを受け取ります。

第 25 章 XML リソース・ファイル・リファレンス

この章では、Blox UI モデル・コンテナーの作成に使用する XML リソース・ファイルの記述方法に関する一般的な解説を提供します。これらの XML ファイルを使用すると、定義済みのエレメントや属性を使用して、ダイアログ、ツールバー、メニューおよびメニュー・バーといったモデル・コンテナーを作成できます。その後、ご自分のコントローラーを書いて、このようなコンポーネントを制御することができます。

リソース・ファイル概説

リソース・ファイルとは、複合モデル・コンテナーの言語ローカライズ可能な記述のことです。これらは、定義済みのエレメントと属性を含む、標準 XML ファイルです。Blox UI モデルはリソース・ファイルを読み取り、リスト済みの Java モデル・オブジェクトをすべて構成し、指定した属性を各オブジェクトに設定します。これらのリソース・ファイルは、DHTML クライアントでユーザー・インターフェースが構築される手段です。

このセクションで説明するフォーマットを使用して、ご自身の XML リソース・ファイルを書くことができます。XML リソース・ファイルの `com.alphablox.blox.uimodel.core` パッケージの下にリストされているモデル UI コンポーネントを追加し、コードでさらに拡張したり修正したりできるすべての Java モデル・オブジェクトを必要に応じて DB2 Alphablox で構成できます。通常、モデル・コンポーネントへの更新と、モデル・コンポーネントからのイベントの処理のために、リソース・ファイルから作成されたモデルのコントローラーをアタッチする必要があります。

作成したリソース・ファイルはさまざまな方法でロードできます。たとえば、それらのファイルをいくつかの共通なサーバー・ロケーションに置く方法や、アプリケーション・ディレクトリーからロードする方法などです。一般的には、ロードはクラス・パスの設定によって行われます。クラス・パスの設定方法に関する指示は、「*管理者用ガイド*」の、DB2 Alphablox を拡張する方法に関するトピックを参照してください。 XML リソース・ファイルを使用して作成したカスタム・ダイアログを立ち上げるコントローラーを記述する方法を示す完全な例に関しては、「*開発者用ガイド*」の『DHTML Client UI の拡張性』セクションにある『ダイアログ』のトピックを参照してください。

最上位エレメント

リソース・ファイルごとに 1 つの最上位コンテナーしか定義できません。最上位エレメントは、以下のいずれかになります。

- Dialog
- Menubar
- Menu
- Toolbar
- ComponentContainer

このようにして作成された UI は通常、ダイアログ・ボックス、メニュー・バー、メニュー (右クリック・メニューなど)、またはツールバーのいずれかです。その後、このコンポーネント・コンテナーは他のコンポーネントを含めます。例えば、ダイアログ・ボックスには、テキスト (Static)、編集テキスト・ボックス (Edit)、数個のチェック・ボックス (CheckBox)、ラジオ・ボタンのセット (RadioButton)、および「OK」ボタンと「取り消し」ボタン (Button) が入っています。最上位エレメントにすることに加えて、 ComponentContainer エレメントはレイアウトとスタイルの操作を向上させるために、しばしばエレメント・セットのグループ化に使用されます。

すべてのビジュアル UI コンポーネントは、Component 基本クラスから生じており、コンポーネントはフォーマット制御と基本的なコンポーネントのセットを集中管理する手段とを提供する階層に配列されます。XML リソース・ファイルでは、ComponentContainer エレメントは、たびたび複数のコンポーネントを「グループ化」するために他のコンポーネントで使用されます。これを使用すると、レイアウトの操作性を改善し、同じ ComponentContainer にあるすべてのエレメントの属性を設定できます。

Blox UI モデルについて詳しくは、13 ページの『Blox UI モデル』を参照してください。

サポートされる引数タイプ

XML リソース・ファイルの使用する場合に作成されるモデルでサポートされるのは以下の引数タイプだけです。

- string
- boolean
- integer
- レイアウト (horizontal、vertical、または grid)
- Style
- alignment

サポートされる引数タイプは、Component で対応する「setter」メソッドのある属性タイプに関連があります。このようなタイプを使用する setter だけを XML ファイルで使用できます。属性について詳しくは、484 ページの『属性』を参照してください。

リソース・ファイルのキャッシング

モデル・リソース・ファイルは、デフォルトでキャッシュされます。リソースのロードを最初に要求した後は、リソース・ファイルを変更すると、DB2 Alphablox の再起動が必要になります。キャッシュできないようにするには、cache="false" 属

性を最上位リソース・エレメントに配置し、この属性が既に起動している場合は、サーバーを再起動します。たとえば、次のようにします。

```
<Dialog name="myDialog" title="My Own Dialog"
   cache="false" modal="true"
   height="420" width="450" layout="vertical">
   <!--other components omitted -->
</Dialog>
```

ローカリゼーション

リソース・ファイルは、リソース・バンドルと同じローカリゼーション命名規則に従います。指定したロケールの言語コードが、ロードされる前にリソース・ファイル名に追加されます。例えば、ロケールをフランス語に設定する場合、リソース・ファイル名には _fr 接尾部を付加します。リソース・ファイルがない場合は、未変更のリソース・ファイル名が使用されます。

XML リソース・ファイルのエレメント

com.alphablox.blox.uimodel.core パッケージのすべてのモデル UI コンポーネントは、リソース・ファイルに追加可能なエレメントです。各エレメントには、指定可能な属性のセットがあります。これらの UI コンポーネントは、Component クラスから同じプロパティーのセットを継承しているので、このエレメントには指定可能な類似の属性があります。この共通の属性には、name、title、alignment、valignment、height、width、layout などが含まれています。これらの共通の属性のリストについては、 488 ページの『全 UI エレメントに共通の属性』を参照してください。

エレメントのリスト

UI コンポーネントのエレメント名には、com.alphablox.blox.uimodel.core パッケージのクラスと同じ名前があり、この名前は各語の先頭の文字が大文字です。以下は、エレメントのリストです。

コンポーネント	説明
Button	プッシュボタン・コンポーネント。
CheckBox	チェック・ボックス・コンポーネント。
ComponentContainer	UI モデル・オブジェクトのための汎用コンテナー。
ControlbarContainer	コントロール・バーのコンテナー、ControlbarContainer に入れられるコントロール・バー (メニューおよびツールバー) の基本クラス。
Dialog	Dialog コンポーネント。ダイアログとは、ユーザーから入力を収集したり、ユーザーに状況を表示したりするのに使用される浮動コンテナーです。ダイアログを作成してから、そのダイアログに Button、CheckBox、および RadioButtons のようなコンポーネントを追加して、ユーザーにオプション・リストを表示したり、決定を下してもらったりします。
DropDownList	ドロップダウン・リスト・コンポーネント。

DropDownList は、他の選択項目のリストから選択するというメカニズムを持つ 1 つの表示オプションで構成されています。一時に 1 つの選択項目しか選択することができません。 DropDownList は、スペースが限られていて、考えられる選択項目を常時表示することが必要でない場合に使用します。

DropDownToolbarButton

ドロップダウン・ツールバー・ボタン・コンポーネント。DropDownToolbarButton には、選択項目のドロップダウン・リストと、現在表示されているドロップダウン・リストを呼び出すアクション・ボタンの両方があります。選択が変更されたり、アクション・ボタンがクリックされると、このコントロールは ClickEvent を生成します。

Edit

編集フィールド (テキスト・フィールド) コンポーネント。Edit コンポーネントにより、ユーザーは 1 行以上のテキストの入力および変更を行うことができます。テキストは、標準のユーザー UI メカニズムを使用して、編集フィールドにコピー、移動、挿入することができます。

GroupBox

ダイアログと他のモデルに名前付きコンテナーを提供する GroupBox コンポーネント。GroupBox コンポーネントは、主にダイアログ・ボックスでコンポーネントをグループ化するために使用されます。たとえば、チャートにオプションを設定するため専用のコンポーネントがいくつかある場合、これらを 1 つの GroupBox 内にまとめて、「チャート・オプション」とタイトルを付けます。

このコンポーネントに名前が付けられないと、RadioButton コンポーネントは、GroupBox 内で別の振る舞い方をします。名前付きグループ内の名前のない RadioButtons はすべて自動的にグループ化されます。1 つのラジオ・ボタンを押すと、グループ内の他のラジオ・ボタンは選択解除されます。

Image

GIF、JPEG、または他の互換可能なイメージを表示する Image コンポーネント。 StaticImage とは異なり、クリックされると、Image コンポーネントは、ClickEvent を生成します。

ListBox

ListBox コンポーネント。

Menu

MenuItem と他の Menu で構成されている Menu コンポーネント。Menu 内部の Menu は、適当なサブメニュー動作をするサブメニューとして扱われます。 MenuItem は選択項目として表示され、選択されると ClickEvent を生成します。デフォルトでは、MenuItem の名前はコントローラー内のハンドラー・メソッドを構成するのに使用されます。たとえば、「drillDown」という名前を持つ MenuItem

	は、コントローラー内の「actionDrillDown」というメソッドにマップされます。新しいメニューにはすべて、デフォルトで垂直レイアウトが割り当てられます。
Menubar	ControlbarContainer と連動して使用され、最上位メニューを表示する Menubar コンポーネント。
MenuItem	MenuItem コンポーネント。これは、メニューで選択可能な項目です。
RadioButton	ラジオ・ボタン・コンポーネント。
Spacer	コンポーネント間に固定された高さと幅のスペーシングを追加する Spacer コンポーネント。
SpinnerButton	ユーザーからの整数入力を受け入れ、値を増減するボタンを提供する Spinner コンポーネント。初期値、増分、最小値、および最大値を設定することができます。
SplitterContainer	ユーザーが調整可能な 2 つのコンポーネント間にスプリッター・バーを指定して表示する SplitterContainer コンポーネント。HorizontalLayout と VerticalLayout のいずれかを使用して、スプリッターの向きを制御します。
Static	対話を必要としない、指示、ラベル、および値といった単純な静的テキストを表示する Static コンポーネント。
StaticImage	ユーザー入力に応答しない静的イメージを表示するコンポーネント。
TabbedContainer	子コンテナーすべてのためのタブを持つコンテナー・ウィンドウ。このコンテナーは、すべての子コンテナーに対応する一連のタブを表示するのに使用します。典型的な使用法は、タブ付きのダイアログ・ボックスをインプリメントすることです。このコンテナーに含めることができるのは、他のコンポーネント・コンテナーだけです。このコンテナーに付加されたスタイルはタブに適用されます。 子コンテナーのタイトルは、そのコンテナーのタブ・ラベルに使用されます。子コンテナーの選択状態が、選択されているタブを決定します。どの子コンテナーも選択されていなければ、最初のコンテナーが自動的に選択されます。複数の子コンテナーが選択されているとマークされている場合、そのうちの最初のものが選択されていると見なされます。 子コンテナーがタブ付きコンテナーに追加された順番が、タブ順序を決定します。上部および下部 (水平) レイアウトでは、最初のコンテナーは右側にな

	ります。左および右 (垂直)レイアウトでは、最初のコンテナーは上になります。
Toolbar	ControlbarContainer と関連してツールバーを表示するのに使用するツールバー・コンポーネント。
ToolbarButton	ToolbarButton コンポーネント。コンポーネント・モデルのどこでも使用できますが、主に ControlbarContainer 内で動作するように設計されています。このコンポーネントの name は、「.gif」拡張子を付加してイメージ名を構成するために使用されます。

上記にリストしたエレメントに加えて、次に説明する Item および ClientLink があります。

Item エレメント

XML リソース・ファイルで、ListBox、DropDownList、または DropDownToolbarButton エレメントが追加されると、Item エレメントを使用して、次のように個々の項目を指定します。

```
<DropDownList name="selectList"
      title="Undo"
      tooltip="Select an option" >
      <Item value="A" />
      <Item value="B" />
      <Item value="C" />
</DropDownList>
```

ClientLink エレメント

ClientLink エレメントは、コンポーネントがクリックされた場合にブラウザーによって処理される URL ベースのリンクを定義します。このエレメントを使用すると、リンク、新規ページがロードされるターゲット・ウィンドウ、および JavaScript の window.open() メソッドとほぼ同じようなブラウザー・ウィンドウ機能ストリング (例えば、"toolbar=no,scrollbars=yes") を指定することができます。

属性

属性名は最初の語が小文字で、後続の各語の最初の文字は大文字です。次のようになります。name、title、width、height、themeClass、imageURL、および themeBasedImage。すべての UI コンポーネントは、Component クラスから派生しているので、共通の属性を多く共有しています。これらの属性は、 Component オブジェクトの「setter」メソッドと対応しています (例えば、setName()、setTitle()、setWidth()、および setHeight())。これらの共通の属性のリストについては、488 ページの『全 UI エレメントに共通の属性』を参照してください。このメソッドの詳細については、Javadoc で com.alphablox.blox.uimodel.core パッケージを参照してください。

以下の単純なダイアログの例では、異なるエレメントを追加する方法、その属性を指定する方法、そして、レイアウトを操作して Blox UI モデル・コンポーネントを使用するダイアログ・ボックスを作成する方法を示します。

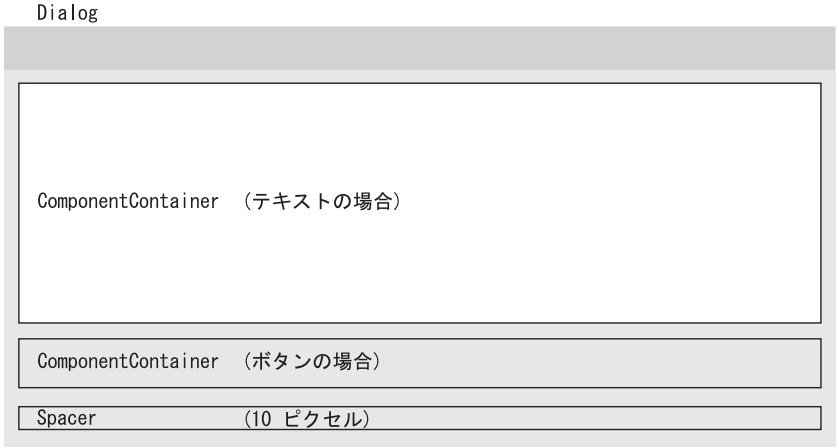
リソース XML ファイルの例

例 1:「About」ダイアログ・ボックス

XML リソース・ファイルの最初の例では、以下のように「About MyApp」ダイアログを作成します。

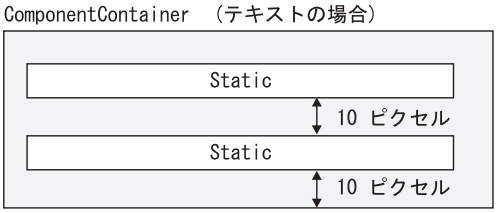
```
<?xml version="1.0" ?>
<Dialog name="aboutDialog" title="About MyApp"
   height="150" width="400" layout="vertical">

   <ComponentContainer layout="vertical" alignment="center">
      <Static name="credit"
       title="Brought to you by the Information Technology group"
       themeClass="csLblFnt csThmClr csAbtTxt" />

      <!-- Add a 10px space in between two Static components -->
      <Spacer />
      <Static name="company"
       title="Copyright 2003 Your company name here."
       themeClass="clsIbar" />
      <Spacer />
   </ComponentContainer>

   <!-- Add another ComponentContainer to have the button aligned in
        the center -->
   <ComponentContainer layout="horizontal" alignment="center">
      <!--If button text is less than 7-8 characters, add width="70" -->
      <Button name="ok" title="OK" />
   </ComponentContainer>
   <!-- Add 10px margin from bottom -->
   <Spacer />
</Dialog>
```

- 図に示されているダイアログ・ボックスは、高さ 150 ピクセルおよび幅 400 ピクセルです。このダイアログに含まれているエレメントは、垂直に積み重ねられます (layout="vertical")。スペーサーの幅または高さが指定されていない場合、デフォルトは高さ 10 ピクセルおよび幅 10 ピクセルです。



- 最初の ComponentContainer には、クレジット・ステートメント (Static) および会社/著作権ステートメント (Static) が入っています。
- この 2 つの Static コンポーネントは、垂直に積み重ねられており (layout="vertical")、中央に位置合わせ (alignment="center") されています。

- CSS クラスには、Static コンポーネントに適用されるものもあります。テーマ・クラスについては、「*開発者用ガイド*」の『データの提示』の章で説明しています。



- 2 番目の ComponentContainer には、ボタンが入っています。ボタンを中央に位置合わせするには、固有の ComponentContainer 内に置く必要があります。そうしないと、このボタンは、2 つのStatic コンポーネントの左に位置合わせされます。

例 2:「Confirmation」ダイアログ・ボックス

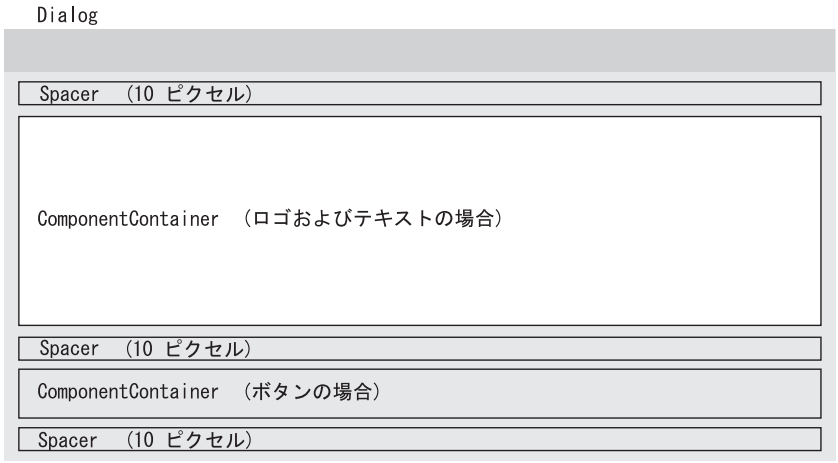
XML リソース・ファイルの 2 番目の例では、以下のように確認ダイアログを作成します。

上記のダイアログを生成する XML コードは、次のとおりです。

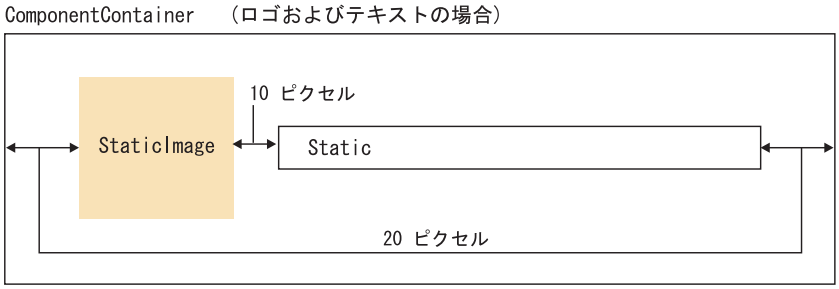
```
<?xml version="1.0" ?>
<Dialog name="myDialog" title="Confirmation" modal="true"
   height="140" width="400" layout="vertical">

   <!-- Add 10px margin from top -->
   <Spacer />
   <!-- Need a horizontal layout in the main area in order to add 20px
        margin on each side -->
   <ComponentContainer layout="horizontal">
      <!-- Add 20px margin on left side -->
      <Spacer width="20" />

      <!-- CONTENT AREA-->
      <StaticImage imageURL="/SalesApp/images/logo.gif" />
      <Spacer width="10" />
      <Static name="credit"
         title="Do you want to apply the change?"
         themeClass="csLblFnt csThmClr csAbtTxt" />

      <!-- Add 20px margin on right side -->
      <Spacer width="20" />
   </ComponentContainer>

   <!-- Add 10px margin between content area and buttons -->
   <Spacer />

   <ComponentContainer name="buttonContainer" layout="horizontal"
    alignment="right">
      <!--If button text is less than 7-8 characters, add width="70" -->
      <Button name="ok" title="Yes" width="70" />

      <!-- Add 5px margin between buttons -->
      <Spacer width="5" />
      <Button name="cancel" title="Cancel" width="70" />
      <!-- Add 20px margin on right side, matching main content area -->
      <!-- Only put this in if there is equivalent or greater margin on the
           left already -->
      <Spacer width="20" />
```

```
    </ComponentContainer>

<!-- Add 10px margin from bottom -->
    <Spacer />
</Dialog>
```

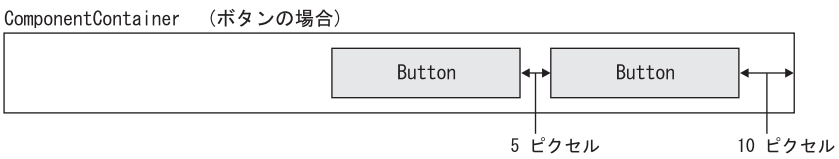
- ダイアログ・ボックスは、高さ 150 ピクセルおよび幅 400 ピクセルです。このダイアログに含まれているエレメントは、垂直に積み重ねられます (layout="vertical")。



- ComponentContainer が追加され、ロゴ (StaticImage) とテキスト (Static) を組み込みます。
  - ComponentContainer のレイアウトは、horizontal に設定されるので、このコンテナー内のコンポーネントは、左から右へ積み重ねられます。
  - CSS クラスには、このコンポーネントに適用されるものもあります。テーマ・クラスについては、「*開発者用ガイド*」の『データの提示』の章で説明しています。



- 別の ComponentContainer が追加され、2 つのボタン (Button コンポーネント) を組み込みます。この 2 つのボタンは、最後に枠の前に 10 ピクセルのスペースを追加して右に位置合わせ (alignement="right") します。



エレメント属性

このセクションでは、すべてのエレメントに共通の属性をリストします。

全 UI エレメントに共通の属性

この表では、すべてのエレメントに共通の属性をリストします。

属性	説明
name	コンポーネントの名前を指定します。これは、コンポーネントの識別に使用され、コンポーネントのアクション名を与えます。例えば、コンポーネント名が `myButton` の場合、コンポーネントの ClickEvent は、メソッド `actionMyButton` を起動します。 **重要:** コンポーネントに命名する場合、予約済みのコンポーネント名は避けてください。コンポーネント名は、com.alphablox.blox.unimodel パッケージの *ModelConstants* インターフェースの定数と同じです。
title	コンポーネントのタイトルを指定します。これは表示テキストです。コンポーネントごとのタイトル属性の使用法について詳しくは、490 ページの『タイトル属性の使用』を参照してください。
alignment	コンポーネントの水平位置合わせを指定します。有効な値は、`right`、`left`、および `center` です。
batchEvents	ユーザーがコンポーネントの状態を変更するアクションをとると即時に、対応するイベントをサーバーに送信するかどうかを指定します。この属性を `false` に設定している場合、ユーザーがコンポーネントの状態を変更するアクションをとるとすぐに、クライアントはイベントをサーバーに送信します。この属性を `true` に設定していると、このコンポーネントに生成されたイベントはクライアントで、(他のコンポーネントからイベントを送信するなど) 他のアクションがクライアントをサーバーに接続するまで、バッチ処理されます。

この設定は、ダイアログでコンポーネントにより送信されたイベントにのみ有効です。

setBusyAfterEvent

コンポーネントがイベントを生成するたびに、UI をビジー状態に設定するかどうかを指定します。`true` に設定すると、このコンポーネントの UI は、イベントが生成されるとビジーに設定されます。有効な値は `true` および `false` です。これは、長い、あるいは重要な操作中に、ユーザー入力を停止する場合に便利です。この設定は、イベント生成後、即時に UI の振る舞いを制御します。

clickable

コンポーネントがマウス・クリック・イベントを生成するかどうか指定します。有効な値は `true` および `false` です。

disabled

コンポーネントを使用できなくするかどうか指定します。有効な値は `true` および `false` です。

height

コンポーネントの高さをピクセル単位で指定します。

layout

コンテナーに付加するレイアウトを指定します。有効な値は、`vertical`、`horizontal`、および `grid(`*numOfColumns*`)` です。この属性は、ComponentContainer とそこから派生したコンポーネントだけに適用されます。

例えば、4 列のグリッドのあるレイアウトを作成するには、次のようにします。

layout=″grid(4)″

setRightClickMenu

このコンポーネントの右クリック・メニューを設定します。このメニューは、ユーザーがコンポーネントを右クリックすると表示されます。

style

コンポーネントに付加するスタイルを指定します。このスタイルは、CSS のようなスタイルのストリングで表されます。また、これは、Style オブジェクトの名前にもなるでしょう。

tabStop

タブ停止位置としてコンポーネントを設定します。有効な値は、`true` および `false` です。デフォルトは `true` です。しかし、タブ停止位置は、ラジオ・ボタンとは連動しないことに注意してください。これは、ブラウザーの振る舞いです。

themeClass

このコンポーネントに使用されるテーマ・クラス (1 つ以上) の名前を指定します。

tooltip

コンポーネントに付加するツール・ヒント (ユーザーがコンポーネントをマウスオーバーするときにポップアップするテキスト) を指定する。

valignment	コンポーネントの垂直位置合わせを指定します。有効な値は、top、bottom、および center です。
visible	コンポーネントの可視性を指定します。有効な値は true および false です。
width	コンポーネントの幅をピクセル単位で指定します。

タイトル属性の使用

以下の表では、エレメントごとの title 属性の使用について説明します。

エレメント	タイトル属性の使用
Button	ボタン・ラベル。
CheckBox	チェック・ボックスの後に表示されるテキスト。
ComponentContainer	最上位エレメントのタイトル。それ以外のところでは、無視されます。
Edit	無視されます。
GroupBox	GroupBox のタイトル。例えば、title="Report Options" を指定すると、以下のように表示されます。

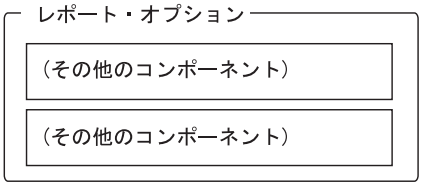


Image StaticImage	無視されます。
ListBox DropDownList	無視されます。
Menu MenuItem	メニュー・ラベル。
Menubar Toolbar	メニュー内でツールバーを参照するために使用されます。それ以外のところでは、無視されます。
Spacer	無視されます。
Static	表示テキスト。

CheckBox および RadioButton の追加の属性

488 ページの『全 UI エレメントに共通の属性』 で説明された共通の属性に加えて、CheckBox および RadioButton には以下の属性があります。

属性	説明
checked	チェック・ボックスまたはラジオ・ボタンがチェックされている (選択されている) かどうか指定します。有効な値は true および false です。

ControlbarItem、MenuItem、および ToolbarButton の追加の属性

488 ページの『全 UI エレメントに共通の属性』 で説明されている共通の属性に加えて、ControlbarItem、MenuItem、および ToolbarButton エレメントには以下の属性があります。

属性	説明
checked	MenuItem がチェックされている (選択されている) かどうか指定する。有効な値は `true` および `false` です。
checkBox	MenuItem が選択されたときに、その隣にチェック・マークを付けるチェック・ボックスのように機能するかどうか指定します。有効な値は `true` および `false` です。
imageURL	イメージ URL を指定します。 • 絶対 URL の場合、ストリングは「`http://`」で開始する必要があります。 • 相対 URL の場合、スラッシュ (/) で始まるストリングは、URL がサーバー・ルートと相対であることを示します。アプリケーション・コンテキストを URL に含める必要があるということに注意してください。スラッシュで始まっていないストリングは、URL が現在のテーマ・ディレクトリーに相対であることを示します。通常、このディレクトリーは以下の場所にあります。 `<alphablox>/repository/theme/<themeName>/i/`
separator	このアイテムがツールバーかメニュー区切り記号かを指定します。有効な値は `true` および `false` です。
themeBasedImage	このイメージが現在のテーマのディレクトリーにあるかどうかを指定します。有効な値は `true` および `false` です。この属性を true に設定すると、サーバーは、コンポーネントと同一の名前に `.gif` 拡張子を追加したイメージ・ファイルを検索します。

ダイアログの追加の属性

488 ページの『全 UI エレメントに共通の属性』 で説明された共通の属性に加えて、ダイアログには以下の属性があります。

属性	説明
defaultButton	ユーザーが Enter キーを押したときにクリックされるデフォルトのボタンを指定します。すべてのダイアログは作成された時点では、デフォルトのボタンが「OK」ボタンに設定されています。ユーザーが別のボタンをクリックすると、そのボタンがデフォルトになります。

modal	このダイアログがモデル・ダイアログであるということを指定します。モデル・ダイアログは、固有の分離した移動可能ウィンドウにあり、それが消されるまで、残りの UI は入力を受け入れません。
resizable	ユーザーがこのダイアログ・ウィンドウをサイズ変更できるように指定します。有効な値は `true` および `false` です。デフォルトは `false` です。

Image および StaticImage の追加の属性

488 ページの『全 UI エレメントに共通の属性』 で説明された共通の属性に加えて、Image と StaticImage エレメントには以下の属性があります。

属性	説明
themeBasedImage	このイメージが現在のテーマのディレクトリーにあるかどうかを指定します。有効な値は `true` および `false` です。この属性を `true` に設定すると、サーバーは、テーマのイメージ・ディレクトリーでコンポーネントと同一の名前に `.gif` 拡張子を追加したイメージ・ファイルを検索します。通常、このディレクトリーは以下の場所にあります。 `<alphablox>/repository/theme/<themeName>/i`。
imageURL	イメージ URL を次のように指定します。 • 絶対 URL の場合、ストリングは「`http://`」で開始する必要があります。 • 相対 URL の場合、 – ストリングをスラッシュ (/) で始めると、URL がサーバー・ルートに対して相対であることを示します。アプリケーション・コンテキストを URL に含める必要があるということに注意してください。 – ストリングにスラッシュを付けないで開始すると、`themeBasedImage` が `true` に設定される場合、URL が現在のテーマ・ディレクトリーが相対であることを示します。`themeBasedImage` が `false` に設定される場合、ストリングにスラッシュを付けないで開始すると、URL が現在のアプリケーション・ディレクトリーと相対であることを示します。

Static の追加の属性

488 ページの『全 UI エレメントに共通の属性』 で説明された共通の属性に加えて、Static には以下の属性があります。

属性	説明
wrapText	Static コンポーネントの `title` 属性で指定されるテ

キストを折り返すかどうか指定します。有効な値は true および false です。デフォルトは false です。

最上位コンポーネント・コンテナーの特殊な属性

5 つの最上位コンポーネント・コンテナー (Dialog、Menu、Menubar、Toolbar、および ComponentContainer) には、1 つの特殊な属性があり、次のようにリソースのキャッシングを指定します。

属性	説明
cache	リソースがロード後にキャッシュされるかどうかを指定します。デフォルトは、true で、リソース・ファイルに対して変更を行うといつでもサーバーを再起動しなければなりません。

Item の属性

Item エレメントには 1 つしか属性がありません。

属性	説明
value	リストに追加する項目を指定する。

ClientLink の属性

ClientLink エレメントには 3 つの属性があります。この属性を使用すると、window.open()JavaScript メソッドで受け渡し可能な引数と同一の引数を指定することができます。

属性	説明
link	コンポーネントのクライアント・リンク (URL) を指定します。この属性の指定は必須です。
target	ロードする URL のターゲット・ウィンドウを指定します。これはオプションです。この属性が指定されていないと、新規 URL が同一ウィンドウにロードされます。
features	target 属性が指定されると、ウィンドウ特性をコンマで区切られたストリングで指定します。たとえば、features="scrollbars=yes,width=300,height=300" などとなります。

最上位エレメントの例

このセクションでは、各最上位エレメントの XML リソース・ファイルの例を示します。XML リソース・ファイルを使用して作成したコンポーネントを制御するコントローラーを記述する方法を示す完全な例に関しては、「UI の拡張性」セクションにある Blox Sampler (DHTML バージョン) のダイヤル・チャート例を参照してください。

ComponentContainer エレメント

以下の XML は、Chart Types と Configuration ダイアログが使用する実際のリソース・ファイルです。このダイアログには 3 つのタブがあり、そのうちの 1 つが Chart Types です。ユーザーがこのタブをクリックすると、以下のリソース・ファイルが呼び出されます。

```
<?xml version="1.0" ?>
<ComponentContainer name="chartTypesTab"
   title="Chart Types"
   layout="vertical"
   alignment="left">
   <Spacer />
   <ComponentContainer layout="horizontal">
      <Spacer width="20" />
      <ComponentContainer layout="vertical">
         <ComponentContainer layout="horizontal" alignment="left">
            <Static title="Chart Type" alignment="left" />
            <Spacer width="10" />
            <DropDownList name="chartTypesSelector" />
         </ComponentContainer>
         <Spacer height="5" />
         <Image name="chartTypeImage" />
      </ComponentContainer>
      <Spacer width="20" />
   </ComponentContainer>
</ComponentContainer>
```

以下に示すのは、上記のリソース・ファイルをロードする Chart Types および Configuration ダイアログ・リソース・ファイルにある XML の一部です。

```
<?xml version="1.0" ?>
<Dialog name="chartTypesDialog" title="Chart Types and Configuration"
   modal="false" height="420" width="450" layout="vertical">
   <TabbedContainer name="ChartTypesTabContainer"
      themeClass="csChrtCntnr" height="360" width="440"
      layout="horizontal" alignment="left">
   </TabbedContainer>
   <!--other components omitted -->
</Dialog>
```

Menu エレメント

以下の XML は、2 つのメニュー項目がある myFormatMenu と呼ばれるメニューを作成します。その後、このメニューは右クリック・メニューのようなコンポーネントに付加することができます。

```
<?xml version="1.0" ?>
<Menu name="myFormatMenu" title="Format" valignment="top">
   <MenuItem name="layout1" title="Special Layout 1"
      valignment="top" />
   <MenuItem name="layout2" title="Special Layout 2"
      valignment="top" />
</Menu>
```

Menubar エレメント

以下の XML は、myMenubar と呼ばれるカスタム・メニュー・バーの下に「表示」メニューを作成します。「表示」メニューには以下の 4 つのオプションがあります。「グリッド」、「チャート」、「ページ・フィルター」および「データ・レイアウト」。以下の「表示」メニューは、クリックされても、ClickEvent を送信しない StaticImage です。

```
<?xml version="1.0" ?>
<Menubar name="myMenubar" layout="horizontal"
   themeClass="csCmpBg csThmClr csCmpBrdr csMnbr" >
   <Menu name="view" layout="vertical" title="View"/>
      <MenuItem name="viewGrid" title="Grid" checkBox="true"
          themeBasedImage="true" setBusyAfterEvent="true" />
      <MenuItem name="viewChart" title="Chart" checkBox="true"
          themeBasedImage="true" setBusyAfterEvent="true" />
      <MenuItem name="viewPageFilter" title="Page Filter"
          checkBox="true" themeBasedImage="true"
          setBusyAfterEvent="true" />
      <MenuItem name="viewDataLayout" title="Data Layout"
          checkBox="true" themeBasedImage="true"
          setBusyAfterEvent="true" />
   </Menu>
   <StaticImage name="logo" imageURL="smallLogo.gif"
      tooltip="Copyright (C) 2004 My Company"
      valignment="top" themeClass="clsStaticImage"
      themeBasedImage="true" />
</Menubar>
```

Dialog エレメント

レイアウトの検討に関する詳細は、485 ページの『リソース XML ファイルの例』を参照してください。

Toolbar エレメント

この XML リソース・ファイル例では、Excel ボタン、PDF ボタン、および Help ボタンのあるツールバーを作成します。PDF ボタンと Help ボタンの間には区切り記号があります。

```
<Toolbar name="myToolbar" title="Exporting" layout="horizontal">

   <ToolbarButton name="fileExportToExcel" title="Excel"
    tooltip="Export to Excel"
    Bookmark" themeBasedImage="true" />

   <ToolbarButton name="fileExportToPDF" title="PDF"
    tooltip="Export to PDF"
    themeBasedImage="true" />

   <ToolbarButton separator="true" />

   <ToolbarButton name="helpHelp" title="Help" tooltip="Help"
    themeBasedImage="true" />

</Toolbar>
```

この場合、fileExportToExcel、fileExportToPDf、および helpHelp は、組み込まれた ToolbarButtons (ツールバー定数と値については 454 ページの『カスタム・ツールバーのタグ』を参照) なので、これらのコンポーネントを制御するために固有のコントローラーを記述する必要はありません。独自の ToolbarButton を作成する場合は、こうしたコンポーネントに対する更新やこうしたコンポーネントからのイベントを処理するコントローラーが必要です。ボタンに使用するイメージはアクティブ、非アクティブ、使用不可のモードで使用可能な異なるイメージ・ファイルのあるテーマ・ベースで、<alphablox_dir>/repository/theme/i ディレクトリーにあります。また、imageURL 属性を使用して、イメージに URL を指定することもできます。themeBasedImage および imageURL について詳しくは、457 ページの『<bloxui:toolbarButton> タグ』を参照してください。

第 26 章 クライアント・サイド API

この章にはクライアント・サイド API の参照資料が記載されています。このリファレンスの使用方法については、1 ページの『第 1 章 このリファレンスの使用法』を参照してください。

クライアント・サイド API 概説

DHTML クライアント用の Client API の JavaScript オブジェクト、メソッド、およびイベントのセットは、DB2 Alphablox 中の他のクライアントよりも比較的単純なものです。 DHTML クライアントの主な意図は、サーバー・サイドのアプリケーション・ロジックおよび API へのアクセスを容易なものにすることです。Client API の 4 つの主なコンポーネントのサマリーを次に説明します。クライアント・サイド API の詳細な説明と例については、「*開発者用ガイド*」および付随の Blox サンプラーの例を参照してください。

Blox メソッド

各 Blox には、フレーム内に関連した JavaScript Blox オブジェクトがあります。PresentBlox を ″salesPresent″ の `id` で作成した場合、同じ名前の JavaScript オブジェクトをそのページで使用できます。いくつかの JavaScript メソッド-`call()`、`getBloxAPI()`、`getName()`、`isBusy()`、`setBusy()`、`setDataBusy()`など - により、サーバーでの JSP ページの呼び出し、Blox プロパティーのサーバーでの更新、Blox のビジー状態の設定などが可能です。

これらのメソッドはユーザー・インターフェース Blox (PresentBlox、 GridBlox、ChartBlox、 PageBlox、および ToolbarBlox) に共通であり、499 ページの『クライアント・サイド Blox メソッド』で説明されています。

BloxAPI

各フレームにはそれぞれ、サーバーとクライアント間のすべての出入力トラフィックをコントロールする 1 つの BloxAPI オブジェクトがあります。このオブジェクトは、グローバル bloxAPI オブジェクトを通して、JavaScript コードで使用可能です。これが提供するメソッドにより、サーバーにポーリングしてイベントを送信し、JSP ページを呼び出すか、またはサーバー・サイド Bean でメソッドを呼び出すか、あるいはイベント・リスナーを追加することができます。bloxAPI オブジェクトに使用可能なメソッドは、499 ページの『BloxAPI メソッド』で説明されています。詳しい例は、561 ページの『例 2: bloxAPI.call() メソッドを使用したサーバー上のチャート・プロパティーの設定』を参照してください。

イベント

ブックマークのロードや軸の交換などのユーザー処置をインターセプトするには、関連したサーバー・サイドのイベント・フィルターを使用する必要があります (Javadoc の com.alphablox.blox.filter パッケージを参照)。代わりに、イベントがサーバーに送信される前に DHTML クライアント上のクリック・イベントをインターセプトすることもできます。例えば、クリック・イベントをサーバーに送信する必要がある場合には、クライアント API (bloxAPI オブジェクト) の sendEvent メソッドを以下のように使用できます。

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

ときに、サーバーが介入する前に、クライアント・サイドでインターセプトすることが望ましい、または必要でさえある場合があります。例えば、ドリルスルー・イベントをキャンセルする必要がある場合、サーバー・サイド・イベント・フィルターの cancelEvent() メソッドでは、データ操作のみがキャンセルされ、ポップアップ・ウィンドウはキャンセルされません。クライアント・サイドでイベントをインターセプトするなら、サーバーに到達する前に、操作すべてをキャンセルすることができます。

これらのクライアント・サイド・イベントについての参照情報は、512 ページの『クライアント・サイド・イベントのリファレンス』で説明します。

カスタム・イベント

DHTML クライアントからサーバー上の Blox UI モデルに基づいたコンポーネントへ送信できるカスタム・イベントを作成することもできます。これには、com.alphablox.blox.uimodel.core.event パッケージ中の ModelEvent クラスを拡張するクラスの作成が関係します。517 ページの『カスタム・イベントの作成』を参照してください。

Blox ヘッダー・タグを使用したクライアント Bean 登録

サーバーに Bean を登録して、サーバーがその Bean インターフェースを JavaScript オブジェクトの形でクライアントへ伝えるようにすることができます。これにより、JavaScript からどの Bean のメソッドでも他の JavaScript オブジェクトと同様に呼び出すことができます。例えば、サーバー上に ″myBean″ という Bean があり、それにメソッド:

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

がある場合、以下のように Bean を Blox ヘッダー・タグに登録できます。

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

それから、この登録済みのメソッドを以下のように呼び出すことができます。

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

多重定義されているメソッドについては、メソッドに下線を付加してから、データ・タイプを付加します。 たとえば、以下の例では、Bean およびその setSelectableSlicerDimensions メソッドを登録します。

```
<blox:header>
   <blox:clientBean name="myDataBlox">
      <blox:method name="setSelectableSlicerDimensions"/>
   </blox:clientBean>
</blox:header>
```

setSelectableSlicerDimensions メソッドでは、 Dimension オブジェクトの配列またはディメンションのストリングを使用できます。 このメソッドをストリングで呼び出すには、以下のようにします。

```
<input type="button" value="Set Slicer Dimensions"
   onclick="myDataBloxAPI.setSelectableSlicerDimensions_String('Market,Measures');"
```

注: この方法は、基本データ・タイプでのみ機能します。 サポートされるデータ・タイプのリストについては、 508 ページの『callBean()』を参照してください。

クライアント Bean の使用に関する詳しい説明は、「*開発者用ガイド*」を参照してください。

DHTML Client API 相互参照

このセクションでは、以下の相互参照表を提供します。

クライアント・サイド Blox メソッド

以下の表では、ページにある任意のユーザー・インターフェース Blox からサーバー・サイド・コードを呼び出すのに使用する、クライアント・サイド JavaScript メソッドをリストします。

メソッド
call()
getBloxAPI()
flushProperties()
getName()
isBusy()
setBusy()
setDataBusy()
updateProperties()

BloxAPI メソッド

以下の表では、bloxAPI オブジェクトを介して公開されるクライアント・サイド JavaScript をリストします。

メソッド
506 ページの『addBusyHandler()』
506 ページの『addErrorHandler()』
507 ページの『addEventListener()』
507 ページの『addResponseListener()』
508 ページの『call()』
508 ページの『callBean()』
510 ページの『getEnablePolling()』
510 ページの『getPollingInterval()』
511 ページの『poll()』
511 ページの『sendEvent()』
511 ページの『setEnablePolling()』
512 ページの『setPollingInterval()』

Blox JavaScript オブジェクト・メソッド

以下の表では、Blox JavaScript オブジェクトを介して公開されるクライアント・サイド JavaScript をリストします。これらのメソッドの説明は35 ページの『第 4 章 共通 Blox タグ・リファレンス』にあります。

JavaScriptメソッド
501 ページの『call()』
503 ページの『flushProperties()』
503 ページの『getBloxAPI()』
503 ページの『getName()』
504 ページの『isBusy()』
504 ページの『setBusy()』
505 ページの『setDataBusy()』
505 ページの『updateProperties()』

クライアント・サイドのイベントおよびイベント・メソッド

以下の表では、DHTML クライアントで使用可能なすべてのクライアント・サイド・イベントをリストします。

イベント
513 ページの ClickEvent
513 ページの CaretPositionChangedEvent
513 ページの ContentsChangedEvent
513 ページの ClosedEvent
513 ページの DoubleClickEvent
513 ページの DragDropEvent
513 ページの ExpandCollapseEvent
513 ページの HScrollEvent
513 ページの ResizeEvent
513 ページの RightClickEvent
513 ページの SelectedEvent
513 ページの SelectionChangedEvent
513 ページの UnselectedEvent
513 ページの VscrollEvent

以下の表では、すべてのクライアント・サイド・イベントに共通のメソッドをリストします。

メソッド
514 ページの『getBloxName()』
515 ページの『getDestinationName()』
515 ページの『getDestinationUID()』
515 ページの『getEventClass()』
515 ページの『isReplaceDuplicate()』
515 ページの『isUrgent()』
515 ページの『setAttribute()』
516 ページの『setReplaceDuplicate()』
516 ページの『setUrgent()』

複数の Blox に共通の JavaScript メソッド

このセクションでは、特定のプロパティーと関連していない、複数の Blox に共通するメソッドについて説明します。あるメソッドが特定の Blox に対して有効であるかどうかを確認するには、その Blox のメソッドのセクションを参照してください。

call()

サーバーで実行する URL を呼び出し、HTTP 要求の結果をストリングとして戻します。このメソッドはサーバー・サイド・コードをクライアントから実行するのに役立ちます。`call()` メソッドはページをリフレッシュすることなくサーバー・サイド・コードを実行します。

データ・ソース

すべて

構文

JavaScriptメソッド

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

ここで、それぞれ以下のとおりです。

引数	説明
callURL	サーバーで実行するファイル (通常は JSP ファイル) の URL を含むストリング。

使用法

call() メソッドを使用して、クライアント・サイド・メソッドから、サーバー・サイド・コードを実行します。このメソッドを使用してサーバーにプロパティーを設定したり、他のサーバー・サイドのロジックを実行したりすることができます。このコードはページをリフレッシュすることなく実行されます (アプリケーションが html モードでレンダリングされている場合には、call() メソッドはページをリフレッシュします)。

call() メソッドによって Blox 上の保留トランザクションが自動的にフラッシュされ、ユーザーが設定したプロパティーはすべてサーバーへ伝搬します。

callURL ストリングは JSP ファイルを参照し、このファイルは実際にクライアントに何も送信しないで、単にさまざまなサーバー・アクションを実行します。URL は絶対または相対のいずれかです。

- 絶対 URL の場合、ストリングは「http://」で開始する必要があります。
- 相対 URL の場合:
  - ストリングをスラッシュ (/) で始めると、URL がサーバー・ルートに対して相対であることを示します。アプリケーション・コンテキストを URL に含める必要があるということに注意してください。
  - ストリングをスラッシュ (/) なしで始めると、URL が現行の文書に対して相対であることを示します。

絶対 URL では、レンダリング・モードが Java である場合、アプレットを配信したのと同じサーバーを呼び出す必要があります。これは、Java アプレットのセキュリティー・ポリシーによります。

注: call() メソッドからの応答テキストのエンコード方式は、指定のない限り、UTF-8 です。別のエンコード方式を必要とする場合には、それを JSP ページ・ディレクティブで指定します。例えば、以下のようにします。

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

例

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

関連項目

499 ページの『BloxAPI メソッド』、505 ページの『setDataBusy()』、561 ページの『例 2: bloxAPI.call() メソッドを使用したサーバー上のチャート・プロパティーの設定』

flushProperties()

クライアントに設定されたすべてのプロパティー (例えば、ユーザー・インターフェースでのユーザー処置を通して) がサーバーへ伝搬される (「フラッシュされる」) ようにします。

データ・ソース

すべて

構文

JavaScriptメソッド

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

使用法

このメソッドは、DB2 Alphablox へ、すべての Blox のすべての保留プロパティー変更をフラッシュするため、これを 1 つの Blox から呼び出すだけですべての Blox 上のプロパティーをすべてフラッシュできます。

関連項目

501 ページの『call()』、505 ページの『updateProperties()』

getBloxAPI()

グローバル・フレームワーク・オブジェクトを戻します。

データ・ソース

すべて

構文

JavaScriptメソッド

```
BloxAPI getBloxAPI();
```

使用法

この BloxAPI オブジェクトは、各フレームで使用可能な `bloxAPI` 変数を使用して取得することもできます。

関連項目

497 ページの『第 26 章 クライアント・サイド API』

getName()

Blox の名前を戻します。

データ・ソース

すべて

構文

JavaScriptメソッド

```
String getName();
```

isBusy()

Blox の現行のビジー状態を戻します。

データ・ソース

すべて

構文

JavaScriptメソッド

```
boolean isBusy();
```

使用法

Blox がビジーである場合は `true` を、そうでない場合は `false` を戻します。

関連項目

504 ページの『setBusy()』、497 ページの『第 26 章 クライアント・サイド API』

setBusy()

クライアントでの Blox のビジー状態を一時的に制御します。

データ・ソース

すべて

構文

JavaScriptメソッド

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

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
`busy`	なし	`true` である場合、Blox は、ユーザー・インターフェースの対話性を使用不可にし、メニュー・バーのロゴをアニメーション表示することによって、ビジー状態を示します。

関連項目

504 ページの『isBusy()』、497 ページの『第 26 章 クライアント・サイド API』

setDataBusy()

クライアントでのビジー状態を設定します。このメソッドは、データ操作を実行するサーバー・サイド・コードを実行するときに役立ちます。

可用性

レンダリング・モード すべて

データ・ソース すべて

構文

JavaScriptメソッド

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

ここで、それぞれ以下のとおりです。

引数	デフォルト	説明
state	false	ブール引数。値 true は、クライアント・サイド Blox の状態をビジーに設定することにより、クライアント・サイド・コードの実行を使用不可にします。値 false は、状態がビジーではなく、クライアント・サイド・コードが自由に実行されることを示します。

使用法

setDataBusy() メソッドを、任意のサーバー・サイド・コード (例えば、call() メソッドを通して実行されるコード) によって更新されるクライアント・サイド DataBlox に設定します。

関連項目

501 ページの『call()』

updateProperties()

DB2 Alphablox にリフレッシュ・メッセージを送信し、現行の Blox プロパティーを更新します。これにより、Blox はその作動状態 (クライアント・サイド) プロパティーをサーバー・サイド Blox ピアへ伝送するため、サーバー・サイド・ピアとクライアント・サイド Blox 管の整合性が保たれます。

データ・ソース

すべて

構文

JavaScriptメソッド

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

関連項目

501 ページの『call()』、503 ページの『flushProperties()』

BloxAPI リファレンス

以下のメソッドは、フレーム内の BloxAPI オブジェクトで使用可能なグローバル・メソッドです。

addBusyHandler()

ページ上のすべての Blox にビジー・ハンドラーを追加します。

構文

JavaScriptメソッド

```
addBusyHandler(busyHandler);
```

ここで、それぞれ以下のとおりです。

引数	説明
busyHandler	JavaScript 関数の名前。

使用法

ビジー・ハンドラーは、Blox のビジー状態が変更される度に呼び出されます。供給されているビジー・ハンドラーは Blox オブジェクトと共に呼び出されます。このため、独自のカスタム・ハンドラーを提供することが可能です。ビジー・ハンドラーは、以下の形の JavaScript 関数です。

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

注: デフォルトのアクションは、ビジーのとき Blox をグレー表示します。`true` を戻してさらに状態変更が処理されないようにします。

addErrorHandler()

フレームワークにエラー・ハンドラーを追加します。

構文

JavaScriptメソッド

```
addErrorHandler(eventListener);
```

ここで、それぞれ以下のとおりです。

引数	説明
eventListener	JavaScript 関数の名前。

使用法

エラー・ハンドラーは、フレームワークが通信エラーまたは Simple Object Access Protocol (SOAP) エラー応答をサーバーから受信するときに呼び出されます。エラー・ハンドラーは、以下の形の JavaScript 関数です。

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

ここで、`SoapResponse` には以下のメソッドおよび属性があります。

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

- `String SoapResponse.faultCode;`
- `String SoapResponse.faultSubcode;`

エラー・ハンドラーは、エラーを処理した場合、`true` を戻し、さらにそのエラーが処理されるのを停止します。エラー・ハンドラーが `false` を戻した場合、エラーはリストにある残りのエラー・ハンドラーに送信されます。どちらの場合も、エラーの後、フレームワークは Blox UI コンテンツやビジー状態を更新せずに戻ります。

addEventListener()

フレームワークにイベント・ハンドラーを追加します。

構文

JavaScriptメソッド

```
addEventListener(eventListener);
```

ここで、それぞれ以下のとおりです。

引数	説明
eventListener	JavaScript 関数の名前。

使用法

イベント・リスナーは、DHTML クライアント・コードまたは sendEvent() メソッド使用の開発者のいずれかによって送信される各イベントごとに呼び出されます。イベント・リスナーは、以下の形の JavaScript 関数です。

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

ここで、`ClientEvent` は512 ページの『クライアント・サイド・イベントのリファレンス』のリスト中の任意のクライアント・イベントです。リスナーは、`true` を戻してそのイベントがさらに処理されるのを停止します。イベント・ハンドラーが `false` を戻した場合、クライアント API は他の残りのリスナーに、その後サーバーにイベントを送信し、イベントの処理を継続します。

addResponseListener()

ページ上のすべての Blox に応答リスナーを追加します。

構文

JavaScriptメソッド

```
addResponseListener(responseListener);
```

ここで、それぞれ以下のとおりです。

引数	説明
responseListener	JavaScript 関数の名前。

使用法

このメソッドでは、DHTML クライアント RPC が正常応答を戻す度に通知される、JavaScript 関数を追加できます。これは複数のフレームを通してアクションを調整するため、または他のイベント後処理のために使用できます。JavaScript 関数は、要求と応答の両方を取得します。

例

以下の例は、要求と応答がどのようにキャプチャーされるかを示しています。

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

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

call()

提供された URL に http 要求をし、要求の結果をストリングとして戻します。

構文

JavaScriptメソッド

```
call(url);
```

ここで、それぞれ以下のとおりです。

引数	説明
url	サーバーで実行するファイル (通常は JSP ファイル) の URL を含むストリング。

使用法

このメソッドは、HTTP 要求の直後、メソッドが結果を戻す前に、サーバーへの変更のポーリングをも行います。

callBean()

指定のサーバー・サイド Bean で指定のメソッドを呼び出します。

構文

JavaScriptメソッド

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

ここで、それぞれ以下のとおりです。

引数	説明
beanName	Bean の名前。Bean は以前にサーバー上に HttpSession で登録済みである必要があります。
methodName	Bean メソッドの名前。
argumentArray	Bean メソッドに渡されるすべての引数の配列。
argumentTypeArray	オプション。引数のデータ・タイプの配列。これはクライアント・サイド Bean 上の適切なメソッドに引数を突き合わせるのに役立ちます。サポートされるデータ・タイプは、以下の表を参照してください。

使用法

戻り値は適切な JavaScript データ・タイプに変換されます。 Bean メソッドが Java 例外をスローした場合、戻り値は JavaScript 例外オブジェクトになります。JavaScript例外オブジェクトに関する詳細は、「*開発者用ガイド*」を参照してください。

サポートされるメソッド引数タイプは大/小文字が区別され、以下の基本データ・タイプに限られます。

Java データ・タイプ	有効な argumentTypes 値
String	string または unspecified
整数	integer または int
ブール	boolean
Long	long
Double	double
Float	float
バイト	byte
配列	JavaScript array

引数が入力されず、サーバー・サイド Bean にリストされた引数と一致するメソッド・シグニチャーがない場合には、サーバーは基本タイプ以外に Java オブジェクトを取り入れるメソッドを探します。この場合サーバーは、ストリング・ベースのコンストラクターを使用して必要なオブジェクトの作成を試行します。

例

以下の例では、サーバー上の myBean という Bean で myMethod メソッドが呼び出されます。引数は文字列と整数の 2 つです。

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

以下のように、サーバー Bean myBean に単一のメソッドがあり、

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

引数のデータ・タイプを指定しないで以下のステートメントで上記の Bean メソッドをクライアントから呼び出す場合、

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

サーバーは以下のようにメソッドを呼び出します。

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

MyObject オブジェクトの作成にはストリング・ベースのコンストラクターが使用されることに注意してください。MyObject オブジェクトのストリング・ベースのコンストラクターがクライアントから提供される値を理解できる場合、メソッドが呼び出され、結果がクライアントに戻されます。

重要: クライアント・サイド・コードを、Java オブジェクトではなく、基本データ・タイプを取り入れ戻すメソッドのみの呼び出しに制限することをお勧めします。

getEnablePolling()

ポーリングが使用可能にされた設定を戻します。

構文

JavaScriptメソッド

```
getEnablePolling();
```

使用法

true の場合、自動ポーリング機構が使用可能にされ、DHTML クライアントの基礎であるフレームワークがサーバーのポーリングを行います。これはたいへん遅いポーリングで、たいへんまれですが、フレーム中の Blox への非同期変更を検出するものです。

関連項目

511 ページの『setEnablePolling()』

getPollingInterval()

非ビジー・ポーリングのポーリング間隔をミリ秒の単位で戻します。

構文

JavaScriptメソッド

```
getPollingInterval();
```

使用法

これは非同期更新のためにサーバーをチェックするための通常のポーリング間隔です。ポーリング機構は、サーバーがクライアントにビジーであると通知した場合、異なる間隔を使用します。

関連項目

512 ページの『setPollingInterval()』

poll()

フレーム中の blox への更新を即時にサーバーへポーリングします。

構文

JavaScriptメソッド

```
poll();
```

使用法

サーバーはそれぞれの Blox ごとに、変更の保留、またビジー状態で応答します。フレームワークが自動でポーリングするため、通常はサーバーに特別にポーリングする必要はありません。これは、サーバーの状態を変更する場合にのみ必要です。この場合には、それらの変更をタイミングよく選出するためにポーリングが必要である場合があります。これは通常、複数のフレームを使用するなどの方法で DHTML クライアントを迂回することを意味します。

sendEvent()

即時に、指定されたイベントをすべての登録済みイベント・ハンドラーへ送信し、最終的にサーバーへ送信します。

構文

JavaScriptメソッド

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

ここで、それぞれ以下のとおりです。

引数	説明
ClientEvent	JavaScript イベント・オブジェクト: ClickEvent、ContentsChangedEvent、ClosedEvent、DoubleClickEvent、DragDropEvent、RightClickEvent、ScrollEvent、SelectedEvent、SelectionChangedEvent、UnselectedEvent の 1 つ。
event	イベントの名前。

使用法

クライアントがサーバーにイベントを正常に送信した場合、この関数は `true` を戻します。その他のすべての場合は、関数は `false` を戻します。

setEnablePolling()

自動的サーバー・ポーリングを制御します。

構文

JavaScriptメソッド

```
setEnablePolling(enable);
```

ここで、それぞれ以下のとおりです。

引数	説明
enable	自動ポーリング機構を使用可能にする場合には `true` を指定し、使用不可にする場合には `false` を指定します。

使用法

`false` に設定した場合、クライアントの基礎となるフレームワークはサーバーへのポーリングを自動的には行いません。自動ポーリング機構はサーバーのたいへん遅いポーリングで、たいへんまれですが、フレーム中の Blox への非同期変更を検出するものです。デフォルトでは、ポーリングは使用可能です。

関連項目

510 ページの『getEnablePolling()』

setPollingInterval()

非ビジー・ポーリングのポーリング間隔をミリ秒の単位で設定します。

構文

JavaScriptメソッド

```
setPollingInterval(intervalMS);
```

ここで、それぞれ以下のとおりです。

引数	説明
intervalMS	ミリ秒単位の非ビジー・ポーリング間隔。

使用法

これは非同期更新のためにサーバーをチェックするための通常のポーリング間隔です。ポーリング機構は、サーバーがクライアントにビジーであると通知した場合、異なる間隔を使用します。

関連項目

510 ページの『getPollingInterval()』

クライアント・サイド・イベントのリファレンス

Blox UI モデルは、イベントのセットを JavaScript オブジェクトとして公開します。このため、JavaScript を使用してイベント・オブジェクトの作成、イベントのサーバーへの送信、またはこれらのイベントのインターセプトを行うことができます。各イベントには、イベントのディスパッチに使用されるサーバーにあるクラスの名前と一致するクラス名があります。クラス名は、各イベントの `EVENT_CLASS` 静的属性で常に使用可能です。たとえば、ClickEvent では、クラス名は

ClickEvent.EVENT_CLASS となります。 SelectionChangedEvent では、クラス名は SelectionChangedEvent.EVENT_CLASS となります。すべてのイベントには、宛先 Blox 名またはコンポーネント UID の識別、イベントを即時にディスパッチするかどうかの指定、その他を可能にする、共通のメソッドのセットがあります。

このセクションでは、以下に関する情報を提供します。

- 513 ページの『クライアント・サイド・イベントおよび構文』
- 514 ページの『共通のイベント・メソッド』
- 516 ページの『イベントの生成』
- 517 ページの『カスタム・イベントの作成』

クライアント・サイド・イベントおよび構文

以下の表では、DHTML クライアントで使用可能なクライアント・サイド・イベントをリストします。引数タイプは、引数の説明のために提供されています。JavaScript では、すべての引数が変数です。

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

引数について、以下で説明します。

引数

bloxName イベントの送信先である Blox の名前。

uid	イベントが呼び出されるコンポーネントに関連した固有の ID。
name	オプション。イベントを生成するコンポーネントの、実際のテキスト・ベースの名前であるコンポーネント名。name が必要なく、これが最後の引数でない場合には、NULL に設定します。
caretPosition	テキスト・カーソルの位置。
checked	オプション。クライアント上のチェック・ボックスの現行状態 (CheckBox コンポーネント用)。
droppedComponent	ドロップされるコンポーネントの uid。
expanded	拡張される場合は、true。
integerSelections	セレクションの uid のリストを含む整数の配列か、複数選択リストのインデックスを含む整数の配列のいずれか。
newContents	変更されたコンテンツ。これは通常、編集コンポーネント (テキスト編集ボックス) に入力されるストリングのようなストリングです。
newHeight	新規の高さ (ピクセル単位)。
newHorizontalPosition	0 ベースの水平スクロール・インデックス。スクロール単位は、イベントを使用しているコンポーネントによって定義されます。たとえば、グリッド・コンポーネントでは、これは列番号になります。
newVerticalPosition	0 ベースの垂直スクロール・インデックス。スクロール単位は、イベントを使用しているコンポーネントによって定義されます。たとえば、グリッド・コンポーネントでは、これは行番号になります。
newWidth	新規の幅 (ピクセル単位)。
operation	実行される操作。現在、有効な操作は move だけです。
positionAfterComponentUID	オプション。ターゲット・コンポーネントをその後ろにドロップするコンポーネントの UID。
textSelection	マウスでマークされた、現在選択されているテキスト。

共通のイベント・メソッド

各イベントは、以下のメソッドを公開します。

getBloxName()

宛先 Blox 名を戻します。

構文:

```
String getBloxName();
```

getDestinationName()

宛先コンポーネントの名前を戻します。

構文:

```
String getDestinationName();
```

使用法: このメソッドは一部のコンポーネントに対してヌルを戻すので、可能なときには常に `getDestinationUID()` を使用してください。このメソッドは、`getDestinationUID()` を使用できない場合 (たとえば、宛先コンポーネントがカスタム・コンポーネントであるかどうかを識別する必要がある場合など) にのみ使用してください。

getDestinationUID()

宛先コンポーネントの UID を戻します。

構文:

```
int getDestinationUID();
```

getEventClass()

イベントに関連したサーバー・サイド Java クラス名を戻します。

構文:

```
String getEventClass();
```

isReplaceDuplicate()

このイベントがクライアント・サイドのイベント・キュー内にある重複するイベントを置き換える場合は、`true` を戻します。

構文:

```
boolean isReplaceDuplicate();
```

isUrgent()

これが即時にサーバーに送る必要のある緊急イベントの場合は、 `true` を戻します。

構文:

```
boolean isUrgent();
```

setAttribute()

イベント内の属性名の値を設定します。

構文:

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

使用法: 属性値は、イベントと共にサーバーに渡されます。属性のタイプは、オプションのタイプ引数を使用して指定できます。サポートされるデータ・タイプについては、 509 ページの サポート・データ・タイプを参照してください。

setReplaceDuplicate()

クライアント・サイドのイベント・キューに存在する、同じイベント・クラス、宛先 UID、および宛先 Blox 名のイベントを置き換えるかどうかを指定します。

構文:

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

使用法: true のとき、クライアント・サイドのイベント・キューに存在する、同じイベント・クラス、宛先 UID、および宛先 Blox 名のイベント (つまり、非緊急イベント) は、置き換えられます。プロセスは既存の重複イベントをキューから除去して、キューの最後に置換するイベントを追加します。

setUrgent()

緊急イベントをサーバーに即時に送るように指定します。

構文:

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

使用法: 緊急イベントは、待機なしで即時にサーバーに送られます。非緊急イベントは、ポーリングの際や緊急イベントの送信を含む他のサーバー通信の際など、都合の良いときに送られます。

イベントの生成

イベントを生成するには、以下のようにします。

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

その後イベントを送信するには、以下のようにします。

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

イベントをインターセプトするには、以下のようにします。

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

eventHandler JavaScript 関数は以下のようになります。

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

ハンドラーが false を戻すと、イベントが処理されてサーバーに送信されることが可能になります。 true を戻すと、イベントのその後の処理がすべて停止します。

以下の JavaScript 例は、カスタム UI コンポーネント (この例では、コンポーネントの名前は "Show" です) で生じるイベントが即時にディスパッチされるようにする方法を示しています。

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

```
    event.setUrgent( false );

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

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

bloxAPI.addEventListener( eventListener );
```

詳細と例は、「*開発者用ガイド*」の『DHTML Client API』の章を参照してください。

カスタム・イベントの作成

提供されたイベントのように機能するカスタム・イベントを独自に作成できます。カスタム・イベントをクライアントから送る場合には、それらのイベントは既存のクライアント・サイド・インフラストラクチャーを使用する必要があります。クライアント・サイドのカスタム・イベントを生成する手順は、以下のとおりです。

1. イベントの名前で、JavaScript 関数を定義します。
2. コンストラクターに追加のイベント・パラメーターを追加します。
3. `EVENT_CLASS` 属性を設定して、関数にイベントのクラス名を設定します。
4. 関数内で `_modelEventConstructor()` を呼び出します。
5. `setAttribute()` メソッドを使用して、追加のパラメーターをイベントに設定します。

```
function MyEvent( bloxName, uid, name, myvalue )
{
   // Begin constructor
   _modelEventConstructor( this, MyEvent.EVENT_CLASS, bloxName, uid, name );
   this.setAttribute( "MyEvent.myValue", myvalue, "int" );
   // End constructor
}
MyEvent.EVENT_CLASS = "my.package.MyEvent";
```

イベント my.package.MyEvent がサーバー上で定義されていると想定します。

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

付録 A. JSP カスタム・タグのコピー・アンド・ペースト

この付録には、blox ごとのカスタム・タグ・ライブラリーのバージョンが収められています。これらのバージョンを使用して JSP ファイルにコピー・アンド・ペーストしたり、必要のない属性を除去したり、必要なふさわしい値を入力することができます。

AdminBlox JSP カスタム・タグ

以下に、AdminBlox カスタム・タグ・ライブラリー全体を示します。JSP ファイルにコピー・アンド・ペーストしたり、必要のない属性を除去したり、必要なふさわしい値を入力することができます。 JSP ファイルでは、属性の値を入力しなければなりません。そうしないと、ページはコンパイルされません。

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

BookmarksBlox JSP カスタム・タグ

以下に、BookmarksBlox カスタム・タグ・ライブラリー全体を示します。JSP ファイルにコピー・アンド・ペーストしたり、必要のない属性を除去したり、必要なふさわしい値を入力することができます。 JSP ファイルでは、すべての属性の値を入力しなければなりません。そうしないと、ページはコンパイルされません。

各属性とその値の構文の説明については、75 ページの『BookmarksBlox タグ属性』を参照してください。

```
<blox:bookmarks
       id=""
       bloxName=""
/>
```

ChartBlox JSP カスタム・タグ

以下に、ChartBlox カスタム・タグ・ライブラリー全体を示します。JSP ファイルにコピー・アンド・ペーストしたり、必要のない属性を除去したり、必要なふさわしい値を入力することができます。 JSP ファイルでは、すべての属性の値を入力しなければなりません。そうしないと、ページはコンパイルされません。

各属性とその値の構文の説明については、89 ページの『ChartBlox タグ属性』を参照してください。

```
<blox:chart
       id=""
       absoluteWarning=""
       aggregateIdenticalInstances=""
       applyPropertiesAfterBookmark=""
       areaSeries=""
       autoAxesPlacement=""
       axisTitleStyle=""
       backgroundFill=""
       barSeries=""
       bloxEnabled=""
       bloxName=""
       bookmarkFilter=""
       chartAbsolute=""
       chartCurrentDimensions=""
       chartFill=""
       chartType=""
       columnLevel=""
       columnSelections=""
       comboLineDepth=""
       dataTextDisplay=""
       dataValueLocation=""
       depthRadius=""
       dwellLabelsEnabled=""
       filter=""
       footnote=""
       footnoteStyle=""
       formatProperties=""
       gridLineColor=""
       gridLinesVisible=""
       groupSmallValues=""
       height=""
       helpTargetFrame=""
       histogramOptions=""
       labelStyle=""
       legend=""
       legendPosition=""
```

```
        lineSeries=""
        lineWidth=""
        localeCode=""
        logScaleBubbles=""
        markerShape=""
        markerSizeDefault=""
        maxChartItems=""
        maximumUndoSteps=""
        menubarVisible=""
        mustIncludeZero=""
        noDataMessage=""
        o1AxisTitle=""
        pieFeelerTextDisplay=""
        quadrantLineCountX=""
        quadrantLineCountY=""
        quadrantLineDisplay=""
        removeAction=""
        render=""
        rightClickMenuEnabled=""
        riserWidth=""
        rowHeaderColumn=""
        rowLevel=""
        rowSelections=""
        rowsOnXAxis=""
        seriesColorList=""
        showSeriesBorder=""
        smallValuePercentage=""
        title=""
        titleStyle=""
        toolbarVisible=""
        totalsFilter=""
        useSeriesShapes=""
        visible=""
        width=""
        x1AxisTitle=""
        x1FormatMask=""
        x1LogScale=""
        x1ScaleMax=""
        x1ScaleMaxAuto=""
        x1ScaleMin=""
        x1ScaleMinAuto=""
        XAxis=""
        XAxisTextRotation=""
        y1Axis=""
        y1AxisTitle=""
        y1FormatMask=""
        y1LogScale=""
        y1ScaleMax=""
        y1ScaleMaxAuto=""
        y1ScaleMin=""
        y1ScaleMinAuto=""
        y2Axis=""
        y2AxisTitle=""
        y2FormatMask=""
        y2LogScale=""
        y2ScaleMax=""
        y2ScaleMaxAuto=""
        y2ScaleMin=""
        y2ScaleMinAuto=""
>
</blox:chart>
```

<blox:chart> 内にネストされたタグ

```
<blox:chart ...>
   <blox:footnoteStyle
      font=""
      foreground="" />

   <blox:labelStyle
      font=""
      foreground="" />

   <blox:seriesFill
      index=""
      value="" />

   <blox:titleStyle
      font=""
      foreground=""/>

   <blox:dial
      borderColor=""
      borderType=""
      color=""
      formatMask=""
      radius=""
      showLabels=""
      startAngle=""
      stopAngle=""
      ticPosition="">
      <blox:needle
         color=""
         endType=""
         endWidth=""
         needleWidth=""
         scope=""
         tooltip=""
         value="" />
      <blox:scale
         maximum=""
         minimum=""
         scope=""
         stepSize="" />
      <blox:sector
         color=""
         innerRadius=""
         outerRadius=""
         scope=""
         startValue=""
         stopValue=""
         tooltip="" />
   </blox:dial>

</blox:chart>
```

CommentsBlox JSP カスタム・タグ

以下に、CommentsBlox カスタム・タグ・ライブラリー全体を示します。JSP ファイルにコピー・アンド・ペーストしたり、必要のない属性を除去したり、必要なふさわしい値を入力することができます。 JSP ファイルでは、すべての属性の値を入力しなければなりません。そうしないと、ページはコンパイルされません。

各属性とその値の構文の説明については、168 ページの『CommentsBlox タグ属性』を参照してください。

```
<blox:comments
       id=""
       bloxName=""
       bloxRef=""
       commentsOnBaseMember=""
       collectionName=""
       dataSourceName=""
       userName=""
       password="" >
       <blox:sortComments
          field=""
          order="" />
</blox:comments>
```

ContainerBlox JSP カスタム・タグ

以下に、ContainerBlox カスタム・タグ・ライブラリー全体を示します。JSP ファイルにコピー・アンド・ペーストしたり、必要のない属性を除去したり、必要なふさわしい値を入力することができます。 JSP ファイルでは、すべての属性の値を入力しなければなりません。そうしないと、ページはコンパイルされません。

```
<blox:container
       id=""
       bloxName=""
       enablePoppedOut=""
       height=""
       poppedOut=""
       poppedOutHeight=""
       poppedOutTitle=""
       PoppedOutWidth=""
       render=""
       visible=""
    width=""
>
</blox:container>
```

DataBlox JSP カスタム・タグ

以下に、DataBlox カスタム・タグ・ライブラリー全体を示します。JSP ファイルにコピー・アンド・ペーストしたり、必要のない属性を除去したり、必要なふさわしい値を入力することができます。 JSP ファイルでは、すべての属性の値を入力しなければなりません。そうしないと、ページはコンパイルされません。

各属性とその値の構文の説明については、183 ページの『DataBlox タグ属性』を参照してください。

```
<blox:data
       id=""
       bloxRef=""
       aliasTable=""
       applyPropertiesAfterBookmark=""
       autoConnect=""
       autoDisconnect=""
       bloxName=""
       bookmarkFilter=""
       calculatedMembers=""
       catalog=""
       columnSort=""
       connectOnStartup=""
       credential=""
       dataSourceName=""
       dimensionRoot=""
```

```
        drillDownOption=""
        drillKeepSelectedMember=""
        drillRemoveUnselectedMembers=""
        enableKeepRemove=""
        enableShowHide=""
        hiddenMembers=""
        hiddenTuples=""
        leafDrillDownAvailable=""
        memberNameRemovePrefix=""
        memberNameRemoveSuffix=""
        mergedDimensions=""
        mergedHeaders=""
        onErrorClearResultSet=""
        parentFirst=""
        password=""
        performInAllGroups=""
        provider=""
        query=""
        retainSlicerMemberSet=""
        rowSort=""
        schema=""
        selectableSlicerDimensions=""
        showSuppressDataDialog=""
        suppressDuplicates=""
        suppressMissingColumns=""
        suppressMissingRows=""
        suppressNoAccess=""
        suppressZeros=""
        textualQueryEnabled=""
        useAASUserAuthorization=""
        useAliases=""
        useOlapDrillOptimization=""
        userName=""
</blox:data>
```

DataLayoutBlox JSP カスタム・タグ

以下に、DataLayoutBlox カスタム・タグ・ライブラリー全体を示します。JSP ファイルにコピー・アンド・ペーストしたり、必要のない属性を除去したり、必要なふさわしい値を入力することができます。 JSP ファイルでは、すべての属性の値を入力しなければなりません。そうしないと、ページはコンパイルされません。

各属性とその値の構文の説明については、241 ページの『DataLayoutBlox タグ属性』を参照してください。

```
<blox:dataLayout
       id=""
       applyPropertiesAfterBookmark=""
       bloxEnabled=""
       bloxName=""
       bookmarkFilter=""
       height=""
       helpTargetFrame=""
       hiddenDimensionsOnOtherAxis=""
       interfaceType=""
       maximumUndoSteps=""
       noDataMessage=""
       render=""
       visible=""
       width=""  >
</blox:dataLayout>
```

GridBlox JSP カスタム・タグ

以下に、GridBlox カスタム・タグ・ライブラリー全体を示します。JSP ファイルにコピー・アンド・ペーストしたり、必要のない属性を除去したり、必要なふさわしい値を入力することができます。 JSP ファイルでは、すべての属性の値を入力しなければなりません。そうしないと、ページはコンパイルされません。

各属性とその値の構文の説明については、253 ページの『GridBlox タグ属性』を参照してください。

```
<blox:grid
       id=""
       applyPropertiesAfterBookmark=""
       autosizeEnabled=""
       bandingEnabled=""
       bloxEnabled=""
       bloxName=""
       bookmarkFilter=""
       columnWidths=""
       commentsEnabled=""
       defaultCellFormat=""
       drillThroughEnabled=""
       drillThroughWindow=""
       editableCellStyle=""
       editedCellStyle=""
       enablePoppedOut=""
       expandCollapseMode=""
       gridLinesVisible=""
       headingsEnabled=""
       height=""
       helpTargetFrame=""
       informationWindowName=""
       maximumUndoSteps=""
       menubarVisible=""
       missingValueString=""
       noAccessValueString=""
       noDataMessage=""
       paginate=""
       poppedOut=""
       poppedOutHeight=""
       poppedOutTitle=""
       PoppedOutWidth=""
       relationalRowNumbersOn=""
       removeAction=""
       render=""
       rightClickMenuEnabled=""
       rowHeadersWrapped=""
       rowHeadingWidths=""
       rowHeadingsVisible=""
       rowHeight=""
       rowIndentation=""
       showColumnDataGeneration=""
       showColumnHeaderGeneration=""
       showRowDataGeneration=""
       showRowHeaderGeneration=""
       toolbarVisible=""
       visible=""
       width=""
       writebackEnabled="" >
</blox:grid>
```

<blox:grid> 内にネストされたタグ

```
<blox:grid ...>
   <blox:cellAlert
      index=""
      apply=""
      background=""
      condition=""
      description=""
      enabled=""
      font=""
      foreground=""
      format=""
      group=""
      link=""
      image_align=""
      image=""
      scope=""
      value=""
      value2="" />]

   <blox:cellEditor
      index=""
      scope="" />

   <blox:cellFormat
      index=""
      background=""
      font=""
      foreground=""
      format=""
      group=""
      scope="" />

   <blox:cellLink
      index=""
      description=""
      link=""
      scope=""
      image_align=""
      image="" />

   <blox:drillThroughWindow
      height=""
      locationbarVisible=""
      menubarVisible=""
      name=""
      resizable=""
      scrollbarVisible=""
      statusbarVisible=""
      toolbarVisible=""
      url=""
      width="" />

   <blox:editableCellStyle
      background=""
      font=""
      foreground="" />

   <blox:editedCellStyle
      background=""
      font=""
      foreground="" />

   <blox:formatMask
      index=""
      mask="" />
```

```
<blox:formatName
   index=""
   name="" />
```

MemberFilterBlox JSP カスタム・タグ

以下に、MemberFilterBlox カスタム・タグ・ライブラリー全体を示します。JSP ファイルにコピー・アンド・ペーストしたり、必要のない属性を除去したり、必要なふさわしい値を入力することができます。 JSP ファイルでは、すべての属性の値を入力しなければなりません。そうしないと、ページはコンパイルされません。

各属性とその値の構文の説明については、299 ページの『MemberFilterBlox タグ属性』を参照してください。

```
<blox:memberFilter
       id=""
       applyButtonEnabled=""
       bloxEnabled=""
       bloxName=""
       dimensionSelectionEnabled=""
       height=""
       selectableDimensons=""
       selectedDimension=""
       visible=""
       width="" >
</blox:memberFilter>
```

PageBlox JSP カスタム・タグ

以下に、PageBlox カスタム・タグ・ライブラリー全体を示します。JSP ファイルにコピー・アンド・ペーストしたり、必要のない属性を除去したり、必要なふさわしい値を入力することができます。 JSP ファイルでは、すべての属性の値を入力しなければなりません。そうしないと、ページはコンパイルされません。

各属性とその値の構文の説明については、305 ページの『PageBlox タグ属性』を参照してください。

```
<blox:page
       id=""
       applyPropertiesAfterBookmark=""
       bloxEnabled=""
       bloxName=""
       bookmarkFilter=""
       fixedChoiceLists=""
       height=""
       helpTargetFrame=""
       maximumUndoSteps=""
       menubarVisible=""
       moreChoicesEnabled=""
       moreChoicesEnabledDefault=""
       noDataMessage=""
       render=""
       visible=""
       width="" >
</blox:page>
```

PresentBlox JSP カスタム・タグ

以下に、PresentBlox カスタム・タグ・ライブラリー全体を示します。JSP ファイルにコピー・アンド・ペーストしたり、必要のない属性を除去したり、必要なふさわしい値を入力することができます。 JSP ファイルでは、すべての属性の値を入力しなければなりません。そうしないと、ページはコンパイルされません。

各属性とその値の構文の説明については、314 ページの『PresentBlox タグ属性』を参照してください。

```
<blox:present
       id=""
       applyPropertiesAfterBookmark=""
       bloxEnabled=""
       bloxName=""
       chartAvailable=""
       chartFirst=""
       dataLayoutAvailable=""
       dividerLocation=""
       enablePoppedOut=""
       gridAvailable=""
       height=""
       helpTargetFrame=""
       maximumUndoSteps=""
       menubarVisible=""
       noDataMessage=""
       pageAvailable=""
       poppedOut=""
       poppedOutHeight=""
       poppedOutTitle=""
       PoppedOutWidth=""
       removeAction=""
       render=""
       splitPane=""
       splitPaneOrientation=""
       toolbarVisible=""
       visible=""
       width=""  >
</blox:present>
```

RepositoryBlox JSP カスタム・タグ

以下に、RepositoryBlox カスタム・タグ・ライブラリー全体を示します。JSP ファイルにコピー・アンド・ペーストしたり、必要のない属性を除去したり、必要なふさわしい値を入力することができます。 JSP ファイルでは、すべての属性の値を入力しなければなりません。そうしないと、ページはコンパイルされません。

各属性とその値の構文の説明については、324 ページの『RepositoryBlox タグ属性』を参照してください。

```
<blox:repository
       id=""
       bloxName=""
       render="">
</blox:repository>
```

ResultSetBlox JSP カスタム・タグ

以下に、ResultSetBlox カスタム・タグ・ライブラリー全体を示します。JSP ファイルにコピー・アンド・ペーストしたり、必要のない属性を除去したり、必要なふさわしい値を入力することができます。 JSP ファイルでは、すべての属性の値を入力しなければなりません。そうしないと、ページはコンパイルされません。

各属性とその値の構文の説明については、329 ページの『ResultSetBlox タグ属性』を参照してください。

```
<blox:resultSet
       id=""
       bloxName=""
       dataBlox=""
       resultSetHandler=""
/>
```

StoredProceduresBlox JSP カスタム・タグ

以下に、StoredProceduresBlox カスタム・タグ・ライブラリー全体を示します。JSP ファイルにコピー・アンド・ペーストしたり、必要のない属性を除去したり、必要なふさわしい値を入力することができます。 JSP ファイルでは、すべての属性の値を入力しなければなりません。そうしないと、ページはコンパイルされません。

各属性とその値の構文の説明については、337 ページの『StoredProceduresBlox タグ属性』を参照してください。

```
<blox:storedProcedures
       id=""
       bloxName=""
/>
```

ToolbarBlox JSP カスタム・タグ

以下に、ToolbarBlox カスタム・タグ・ライブラリー全体を示します。JSP ファイルにコピー・アンド・ペーストしたり、必要のない属性を除去したり、必要なふさわしい値を入力することができます。 JSP ファイルでは、すべての属性の値を入力しなければなりません。そうしないと、ページはコンパイルされません。

各属性とその値の構文の説明については、341 ページの『ToolbarBlox タグ属性』を参照してください。

```
<blox:toolbar
       id=""
       applyPropertiesAfterBookmark=""
       bloxEnabled=""
       bloxName=""
       bookmarkFilter=""
       helpTargetFrame=""
       removeAction=""
       removeButton=""
       rolloverEnabled=""
       textVisible=""
       toolTipsVisible=""
       visible=""
/>
```

blox.tld 内のその他のタグ

以下に、残りのカスタム・タグ・ライブラリーを示します。JSP ファイルにコピー・アンド・ペーストしたり、必要のない属性を除去したり、必要なふさわしい値を入力することができます。 JSP ファイルでは、すべての属性の値を入力しなければなりません。そうしないと、ページはコンパイルされません。

Display タグ

```
<blox:display
    blox=""
    bloxRef=""
    render=""
    width=""
    height="" />
```

Header タグ

```
<blox:header
    contextPath=""
    pageURL=""
    theme="" >
    <blox:clientBean name="" bean="">
        <blox:method name="" />
    </blox:clientBean>
</blox:header>
```

Blox Context タグ

```
<blox:bloxContext />
```

pdfReport および pdfDialogInput タグ

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

Debug タグ

```
<blox:debug />
```

このタグのいくつかの使用例については、「*開発者用ガイド*」の『Blox デバッグ・タグ』を参照してください。

Logo タグ

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

Session タグ

```
<blox:session
    key="" />
```

Blox フォームに関連したカスタム・タグ

このセクションでは、以下のタグ属性をリストします。

CheckBoxFormBlox タグ

```
<bloxform:checkBox
   id=""
   bloxName=""
      checked=""
   checkedValue=""
   formElementName=""
   themeClass=""
      title=""
   uncheckedValue=""
   visible=""
/>
```

CubeSelectFormBlox

```
<bloxform:cubeSelect
   id="cubes"
   id=""
   bloxName=""
   dataBlox=""
   dataBloxRef=""
   formElementName=""
   minimumWidth=""
   multipleSelect=""
   selectedCube=""
   selectedCubeName=""
   size=""
   themeClass=""
   visible=""
/>
```

DataSourceSelectFormBlox

```
<bloxform:dataSourceSelect
   id=""
   bloxName=""
   adapter=""
   adminBloxRef=""
   formElementName=""
   minimumWidth=""
   nullDataSourceLabel=""
   selectedDataSourceName=""
   themeClass=""
   type=""
   visible=""
/>
```

DimensionSelectFormBlox

```
<bloxform:dimensionSelect
   id=""
   bloxName=""
   cube=""
   cubeName=""
   dataBlox=""
   dataBloxRef=""
   formElementName=""
   minimumWidth=""
   multipleSelect=""
   selectedDimension=""
   selectedDimensionName=""
   size=""
   themeClass=""
   visible=""
/>
```

EditFormBlox

```
<bloxform:edit
   id=""
   bloxName=""
   charactersPerLine=""
   focus=""
   formElementName=""
   lines=""
   maskInput=""
   maxCharacters=""
   themeClass=""
   visible=""
/>
```

MemberSelectFormBlox

```
<bloxform:memberSelect
   id=""
   bloxName=""
   dataBlox=""
   dataBloxRef=""
   dimension=""
      dimensionName=""
   filterGeneration=""
   filterOperator=""
   formElementName=""
   minimumWidth=""
   multipleSelect=""
   rootMemberName=""
```

```
   rootMemberNames=""
   rootMembers=""
   selectedMember=""
   selectedMemberName=""
   size=""
   themeClass=""
   visible=""
/>
```

RadioButtonFormBlox

<bloxform:radioButton> タグは複数の <bloxform:button> タグを持つことができます。

```
<bloxform:radioButton
   id=""
   bloxName=""
   align=""
   borderEnabled=""
   formElementName=""
   themeClass=""
   visible="">

   <bloxform:button
            label=""
            object=""
      selected=""
      value=""
   />

</bloxform:radioButton>
```

SelectFormBlox

<bloxform:select> タグは複数の <bloxform:option> タグを持つことができます。

```
<bloxform:select
   id=""
   bloxName=""
   formElementName=""
   minimumWidth=""
   multipleSelect=""
   size=""
   themeClass=""
   visible=""
>
   <bloxform:option
            label=""
            object=""
      selected=""
      value=""
   />

</bloxform:select>
```

TimePeriodSelectFormBlox

```
<bloxform:timePeriodSelect
   id=""
   bloxName=""
   defaultSeriesVisible=""
   formElementName=""
   minimumWidth=""
   selectedSeries=""
   selectedSeriesString=""
```

```
   themeClass=""
   timeSchemaBloxRef=""
   visible=""
>
   <bloxform:timeSeries
      expression=""
      name=""
   />

</bloxform:timePeriodSelect>
```

TimeUnitSelectFormBlox

```
<bloxform:timeUnitSelect
   id=""
   bloxName=""
   formElementName=""
   minimumWidth=""
   multipleSelect=""
   selectedTimeUnit=""
   size=""
   themeClass=""
   timeSchemaBloxRef=""
   visible=""
/>
```

TreeFormBlox

<bloxform:tree> タグには、フォルダーおよびアイテム用の 2 つのネストされたタグがあります。<bloxform:tree> タグ内には、複数の <bloxform:folder> タグと <bloxform:item> タグを含むことができます。

```
<bloxform:tree
   id=""
   bloxName=""
   draggingEnabled=""
   itemPositoningEnabled=""
   rootVisible=""
   textWrapped=""
   themeClass=""
   visible=""
>
   <bloxform:folder> <%--root folder--%>

      <bloxform:folder
            label=""
            draggable=""
         expanded=""
            href=""
      imageURL=""
            label=""
      name=""
            object=""
   target=""
      themeBasedImage=""
      tooltip=""
      >
         <bloxform:item
            draggable=""
            href=""
      imageURL=""
            label=""
      name=""
            object=""
   target=""
```

```
      themeBasedImage=""
            tooltip="" />

   </bloxform:folder>

   </bloxform:folder>
</bloxform:tree>
```

<bloxform:getChangedProperty> タグ

```
<bloxform:getChangedProperty
   debugEnabled=""
   formBlox=""
   formBloxRef=""
   formProperty=""
   property=""
/>
```

<bloxform:setChangedProperty> タグ

```
<bloxform:setChangedProperty
   callAfterChange=""
   debugEnabled=""
   formProperty=""
   target=""
   targetRef=""
   targetProperty=""
/>
```

Blox Logic のカスタム・タグ

Blox Logic タグ・ライブラリーには以下の Blox のタグがあります。

- 535 ページの『MDBQueryBlox』
- 536 ページの『MemberSecurityBlox』
- 536 ページの『TimeSchemaBlox』

MDBQueryBlox

MDBQueryBlox のタグはネスト構造です。ネスト構造は、アプリケーションの必要に応じ、変わる場合があります。詳細については、394 ページの『MDBQueryBlox のタグ』を参照してください。以下に、一般構造を示します。

```
<bloxlogic:mdbQuery
   id=""
   dataBloxRef=""
   cubeName="" >
   <bloxlogic:axis
      mutable=""
      queryFragment=""
      type="" >
      <bloxlogic:tupleList>
         <bloxlogic:dimension>
            [specify the dimension name here]
         </bloxlogic:dimension>
         <bloxlogic:tuple>
            <bloxlogic:member>
               [specify the member here]
            </bloxlogic:member>
            <bloxlogic:member>
             [specify another member here]
            </bloxlogic:member>
```

```
            ...
         </bloxlogic:tuple>
      </bloxlogic:tupleList>
   </bloxlogic:axis>
</bloxlogic:mdbQuery>
```

<bloxlogic:tupleList> タグは以下のようにネストしないで、独立型にすることもできます。

```
<bloxlogic:tupleList
   id=""
   tuplesRef="" />
```

<bloxlogic:dimension> タグには次の 1 つの属性があります。

```
<bloxlogic:dimension
   list="" />
```

<bloxlogic:tuple> タグには次の 1 つの属性があります。

```
<bloxlogic:tuple
   list="" />
```

<bloxlogic:crossJoin> タグは、<bloxlogic:axis> タグ内にネストされたタグです。これには属性がありません。<bloxlogic:member> タグにも属性はありません。

MemberSecurityBlox

```
<bloxlogic:memberSecurity
   id=""
   cubeName=""
   dataBlox=""
   dataBloxRef=""
   dimensionName="" >

      <bloxlogic:memberSecurityFilter
      dimensionName=""
      memberName="" />
   <bloxlogic:memberSecurityFilter
      dimensionName=""
      memberName="" />

</bloxlogic:memberSecurity>
```

TimeSchemaBlox

```
<bloxlogic:timeSchema
   id=""
   dataBloxRef=""
      name=""
   today=""
/>
```

Blox Portlet のカスタム・タグ

Blox Portlet タグ・ライブラリーには、Blox UI モデルの ClientLink に基づいた、ポートレット・ページへの HTML マークアップの追加を支援する以下のタグがあります。

- 537 ページの『<bloxportlet:actionLinkDefinition> タグ』
- 537 ページの『<bloxportlet:actionLink> タグ』
- 537 ページの『<bloxportlet:PortletLinkDefinition> タグ』

- 537 ページの『<bloxportlet:portletLink> タグ』
- 537 ページの『ネストされた <bloxportlet:parameter> タグ』

<bloxportlet:actionLinkDefinition> タグ

```
<bloxportlet:actionLinkDefinition
   action="" />
```

<bloxportlet:actionLink> タグ

```
<bloxportlet:actionLink
   action="" />
```

<bloxportlet:PortletLinkDefinition> タグ

```
<bloxportlet:PortletLinkDefinition
   action="" />
```

<bloxportlet:portletLink> タグ

```
<bloxportlet:portletLink
   action="" />
```

ネストされた <bloxportlet:parameter> タグ

このタグは、<bloxportlet:actionLinkDefinition> タグまたは <bloxportlet:PortletLinkDefinition> タグの内部にネストされる必要があります。

```
<bloxportlet:parameter
   name=""
   value="" />
```

Blox UI のカスタム・タグ

Blox UI 修飾子のカスタム・タグを使用するには、次のように bloxui.tld をインポートします。

```
<%@ taglib uri="bloxuitld" prefix="bloxui"%>
```

修飾子のカスタム・タグには以下が含まれます。

- コンポーネント・カスタマイズ・タグ
  - 539 ページの『<bloxui:calculationEditor> タグ』
  - 539 ページの『<bloxui:component> タグ』
  - 541 ページの『カスタム・メニュー・タグ』
  - 542 ページの『カスタム・ツールバー・レイアウト・タグ』
- カスタム分析タグ
  - 538 ページの『<bloxui:bottomN> タグ』
  - 539 ページの『<bloxui:customAnalysis> タグ』
  - 540 ページの『<bloxui:eightyTwenty> タグ』
  - 541 ページの『<bloxui:percentOfTotal> タグ』
  - 541 ページの『<bloxui:topN> タグ』

- カスタム・レイアウトのタグ
  - 538 ページの『<bloxui:butterflyLayout> タグ』
  - 539 ページの『<bloxui:compressLayout> タグ』
  - 540 ページの『<bloxui:customLayout> タグ』
  - 540 ページの『<bloxui:gridHighlight> タグ』
  - 540 ページの『<bloxui:gridSpacer> タグ』
  - 541 ページの『<bloxui:title> タグ』
- ユーティリティー・タグ
  - 538 ページの『<bloxui:actionFilter> タグ』
  - 539 ページの『<bloxui:clientLink> タグ』
  - 540 ページの『<bloxui:gridFilter> タグ』
  - 541 ページの『<bloxui:setProperty> タグ』
- : 538 ページの『<bloxui:accessibility> タグ』

<bloxui:actionFilter> タグ

```
<bloxui:actionFilter
   componentName=""
   filter="" />
```

<bloxui:accessibility> タグ

```
<bloxui:accessibility
   screenReaderMode=""
   windowMenuEnabled="" />
```

<bloxui:bottomN> タグ

```
<!--Nested within a PresentBlox or a GridBlox-->

<bloxui:bottomN
   description=""
   hideOthers=""
   membersToAnalyze=""
   name=""
   number=""
   preserveGrouping=""
   prompt=""
   showOtherSummary=""
   showRank=""
/>
```

<bloxui:butterflyLayout> タグ

```
<!--Nested within a PresentBlox, GridBlox, or ChartBlox -->

<bloxui:butterflyLayout
   addSeparatorColumns=""
   applyLayout=""
   description=""
   name=""
   position=""
   scope=""
   separatorWidth=""
   showOnLayoutMenu="" />
```

<bloxui:calculationEditor> タグ

このタグには属性がありません。

<bloxui:clientLink> タグ

```
<!--Nested within a component customization tag-->

<bloxui:clientLink
   features=""
   link=""
   target=""/>
```

<bloxui:component> タグ

```
<!--Nested within a PresentBlox, GridBlox, or ChartBlox -->

<bloxui:component
   alignment=""
   bloxRef=""
   clickable=""
   disabled=""
   height=""
   name=""
   positionBefore=""
   style=""
   themeClass=""
   title=""
   tooltip=""
   valignment=""
   visible=""
   width="" >

<bloxui:clientLink
   link=""
         target="" />

</bloxui:component>
```

<bloxui:compressLayout> タグ

```
<!--Nested within a PresentBlox or a GridBlox-->

<bloxui:compressLayout
   applyLayout=""
   compressColumns=""
   compressRows=""
   description=""
   memberSeparator=""
   name=""
   showOnLayoutMenu="" />
```

<bloxui:customAnalysis> タグ

```
<!--Nested within a PresentBlox, GridBlox, or ChartBlox -->

<bloxui:customAnalysis
   analysis=""
   name="" />
```

<bloxui:customLayout> タグ

```
<!--Nested within a PresentBlox, GridBlox, or ChartBlox -->

<bloxui:customLayout
   applyLayout=""
   layout=""
   name=""
   showOnLayoutMenu=""/>
```

<bloxui:eightyTwenty> タグ

```
<!--Nested within a PresentBlox or a GridBlox-->

<bloxui:eightyTwenty
   description=""
   hideOthers=""
   membersToAnalyze=""
   name=""
   number=""
   preserveGrouping=""
   prompt=""
/>
```

<bloxui:gridFilter> タグ

```
<!--Nested within a PresentBlox or a GridBlox-->

<bloxui:gridFilter
   filter="" />
```

<bloxui:gridHighlight> タグ

```
<!--Nested within a PresentBlox or a GridBlox-->

<bloxui:gridHighlight
   applyLayout=""
   description=""
   includeData=""
   includeHeaders=""
   name=""
   scope=""
   selection=""
   showOnLayoutMenu=""
   style="" />
```

<bloxui:gridSpacer> タグ

```
<!--Nested within a PresentBlox or a GridBlox-->

<bloxui:gridSpacer
   applyLayout=""
   axis=""
   description=""
   height=""
   locked=""
   name=""
   position=""
   scope=""
   showOnLayoutMenu=""
   style=""
   width="" />
```

<bloxui:percentOfTotal> タグ

```
<!--Nested within a PresentBlox or a GridBlox-->

<bloxui:percentOfTotal
   description=""
   hideOthers=""
   membersToAnalyze=""
   name=""
   number=""
   preserveGrouping=""
   prompt=""
/>
```

<bloxui:topN> タグ

```
<!--Nested within a PresentBlox or a GridBlox-->

<bloxui:topN
   description=""
   hideOthers=""
   membersToAnalyze=""
   name=""
   number=""
   preserveGrouping=""
   prompt=""
   showOtherSummary=""
   showRank=""
/>
```

<bloxui:setProperty> タグ

```
<!--Nested within a PresentBlox, GridBlox, or ChartBlox -->

<bloxui:setProperty
   name=""
   value="" />
```

<bloxui:title> タグ

```
<!--Nested within a PresentBlox, GridBlox, or ChartBlox -->

<bloxui=title
   title=""
   style=""
   alignment=""/>
```

カスタム・メニュー・タグ

カスタム・メニュー・レイアウトのタグには、以下のように <bloxui:menu> および <bloxui:menuItem> が含まれています。

<bloxui:menu> および <bloxui:menuItem> タグ

```
<!--Nested within a PresentBlox, GridBlox, or ChartBlox -->

<bloxui:menu
   name=""
   bloxRef=""
   accesskey=""
   disabled=""
   positionBefore=""
   resourceName=""
   title=""
```

```
   tooltip=""
   visible=""
   >
   <bloxui:menuItem
      name=""
      accesskey=""
      checkable=""
      checked=""
      disabled=""
      imageURL=""
      positionBefore=""
      separator=""
      themeBasedImage=""
      title=""
      tooltip=""
   visible=""
      >
<bloxui:clientLink
   link=""
         target="" />
   </bloxui:menuItem>
</bloxui:menu>
```

カスタム・ツールバー・レイアウト・タグ

カスタム・ツールバー・レイアウトのタグには、以下のように <bloxui:toolbar> および <bloxui:toolbarButton> が含まれています。

<bloxui:toolbar> およびこれにネストされた <bloxui:toolbarButton> タグ

```
<!--Nested within a PresentBlox, GridBlox, or ChartBlox -->

<bloxui:toolbar
   disabled=""
   bloxRef=""
   name=""
   positionBefore=""
   resourceName=""
   title=""
   tooltip=""
   visible="">

   <bloxui:toolbarButton
      checkable=""
      checked=""
      disable=""
      imageURL=""
      name=""
      positionBefore=""
      separator=""
      themeBasedImage=""
      title=""
      tooltip=""
      visible="" >
      <bloxui:clientLink
         link=""
         target="" />
   </bloxui:toolbarButton>

</bloxui:toolbar>
```

Relational Reporting Blox カスタム・タグ

Blox のサポートする Relational Reporting のカスタム・タグについては、「*Relational Reporting 開発者用ガイド*」を参照してください。

付録 B. Alphablox XML Cube の使用

Alphablox XML Cube は、アプリケーション・データ・ソースまたは DB2 Alphablox キューブから戻される照会結果セットを表す XML タグおよび属性を定義します。結果セットを Alphablox XML Cube 文書にトランスフォームする場合、この文書は、基礎となるデータ・ソースのレイアウトにかかわりなく、既存のエレメントを持つ公開された予測可能なデータ構造を示します。

Alphablox XML Cube は、W3C XML DOM 規格で行っているように、データのツリー・ビューを提示し、そのノードはデータ・エレメントに対応しています。W3C XML DOM には文書エレメントを操作する機能が含まれています。Alphablox は、DOM を拡張して、分析キューブ・データの操作に特にふさわしい便利なメソッドを提供します。これらの拡張機能について詳しくは、Javadoc の com.alphablox.blox.xml パッケージを参照してください。

アプリケーション・プログラマーは、XML Cube 文書にアクセスし、そのデータを処理してカスタム・ロジックまたはデータ・レイアウトをインプリメントすることができます。この章では、Alphablox アプリケーションで多く使用される、よく知られたデータの表現に使用する Alphablox XML Cube を説明しています。

データ表現

分析キューブ結果セットの DB2 Alphablox 表現に精通しているアプリケーション・プログラマーは、 Alphablox XML Cube の編成をすぐに理解するでしょう。このセクションでは、以下の単純な例を用いて、DB2 Alphablox 表現のかぎとなる概念を説明します。

Product	EAST	West	South	Market
Audio	12,460	15,507	0	27,967
Visual	33,138	26,605	24,565	84,308
Product	45,598	42,112	24,565	112,275

結果セットには、記述エレメント (ディメンションおよびメンバーの名前) と関連データの値が含まれています。この例で示されている通常の DB2 Alphablox 表現は記述エレメントを行または列軸に編成し、データ値をデータ・セルに編成します。この例については以下の点に注意してください。

- Market ディメンションは列軸にあり、East、West、および South の 3 つのメンバーを含んでいます。

- Product ディメンションは行軸にあり、Audio および Visual の 2 つのメンバーを含んでいます。
- 複数のディメンションを同じ軸に存在させることができます。このような場合、あるディメンションを別のディメンション内にグループ化する、暗黙のグループ化があります。
- タプル は、軸上のディメンションごとに 1 つずつあるメンバーのセットを表しています。この例では、各軸に 1 つしかディメンションがないので、各タプルは単一のメンバーを表しています。
- データ値は、タプルの交差部分のセルに現れます。例えば、値 12,460 は、Audio タプルと East タプルの交差部分に現れます。

注: GridBlox または PresentBlox では、未使用のディメンションは、「その他」の軸にあります。Alphablox XML Cube では、各未使用のディメンションは、別々のスライサー軸にあります。

次のセクションでは、XML フォーマットで照会結果セットをレンダリングする方法について説明します。

サンプル Alphablox XML 文書

XML 文書としてレンダリングされた結果セットの例を以下に示します。読みやすさを考慮して、改行が追加されている場合があります。

```
<?xml version="1.0"?>

<!DOCTYPE cube SYSTEM '/alphablox/AnalysisServer/xml/dtd/cube.dtd'>

<cube>
   <bloxInfo>
     <bloxID>15</bloxID>
     <bloxName>MyDataBlox</bloxName>
     <appName>MyXMLDoc</appName>
   </bloxInfo>
   <data>
   <slicer>
     <slicerDimension name="Period">Period</slicerDimension>
     <slicerMember name="Period" gen="1"
                   leaf="false">Period</slicerMember>
     </slicer>
     <slicer>
       <slicerDimension name="Accounts">Accounts</slicerDimension>
       <slicerMember name="Accounts" gen="1"
                     leaf="false">Accounts</slicerMember>
     </slicer>
     <slicer>
       <slicerDimension name="Scenario">Scenario</slicerDimension>
       <slicerMember name="Scenario" gen="1"
                     leaf="false">Scenario</slicerMember>
     </slicer>
  <axis name="columns" index="0">
       <dimensions>
        <dimension name="Market" index="0">Market</dimension>
       </dimensions>
       <tuple index="0">
        <member name="East" index="0" gen="2" spanInHierarchy="1"
            spanIndexInHierarchy="0" leaf="false">East</member>
       </tuple>
```

```
       <tuple index="1">
        <member name="West" index="0" gen="2" spanInHierarchy="1"
            spanIndexInHierarchy="0" leaf="false">West</member>
</tuple>

<tuple index="2">
        <member name="South" index="0" gen="2" spanInHierarchy="1"
            spanIndexInHierarchy="0" leaf="false">South</member>
</tuple>
<tuple index="3">
        <member name="Market" index="0" gen="1" spanInHierarchy="1"
            spanIndexInHierarchy="0" leaf="false">Market</member>
</tuple>
  </axis>
  <axis name="rows" index="1">
<dimensions>
        <dimension name="Product" index="0">Product</dimension>
</dimensions>
<tuple index="0">
        <member name="Audio" index="0" gen="2" spanInHierarchy="1"
            spanIndexInHierarchy="0" leaf="false">Audio</member>
</tuple>
<tuple index="1">
        <member name="Visual" index="0" gen="2" spanInHierarchy="1"
            spanIndexInHierarchy="0" leaf="false">Visual</member>
</tuple>
<tuple index="2">
        <member name="Product" index="0" gen="1" spanInHierarchy="1"
            spanIndexInHierarchy="0" leaf="false">Product</member>
</tuple>
  </axis>
  <cells>
    <row>
      <column>
        <cell>13438.0</cell>
      </column>
      <column>
        <cell>22488.0</cell>
      </column>
      <column>
        <cell>0.0</cell>
      </column>
      <column>
        <cell>35926.0</cell>
      </column>
    </row>
    <row>
      <column>
        <cell>33138.0</cell>
      </column>
      <column>
        <cell>40351.0</cell>
      </column>
      <column>
        cell>24565.0</cell>
      </column>

      <column>
        <cell>98054.0</cell>
      </column>
    </row>
    <row>
      <column>
        <cell>46576.0</cell>
      </column>
      <column>
        <cell>62839.0</cell>
      </column>
```

```
      <column>
        <cell>24565.0</cell>
      </column>
      <column>
        <cell>133980.0</cell>
      </column>
    </row>
  </cells>
 </data>
</cube>
```

Alphablox XML タグ

DB2 Alphablox は、このセクションで説明されている XML タグを使用して、アプリケーション・データ・ソースから戻される照会結果セットのエレメントを表します。これらのタグは、軸の数に制限のない分析キューブをサポートします。

すべての XML タグの場合と同様、以下の規則が Alphablox タグに適用されます。

- XML 文書は、以下のように XML 宣言で開始しなければなりません。

  ```
  <?xml version="1.0"?>
  ```

- XML 文書には、ルート・エレメントを 1 つのみ含めることができます。文書の内容を定義する他のすべてのタグは、ルート・エレメントに含まれます。以下のタグが、Alphablox XML Cube のルート・エレメントを定義します。

  ```
  <cube>...</cube>
  ```

- タグはネストできますが、オーバーラップできません。たとえば、以下は有効です。

  ```
  <bloxInfo><bloxName>myBlox</bloxName></bloxInfo>
  ```

  しかし、以下は無効です。

  ```
  <bloxInfo><bloxName>myBlox</bloxInfo></bloxName>
  ```

タグは XML 文書での出現順にリストされています。

XML タグ

<?xml version="1.0"?>	XML 文書を識別し、XML 文書が使用する W3C 仕様を指名します。 Alphablox XML Cube は、1.0 仕様を使用します。
<!DOCTYPE cube...>	文書タイプを (そのルート・エレメントに命名して) 識別し、関連 DTD ファイルを指定します。
<cube> </cube>	Alphablox XML Cube の文書ルート・エレメントを識別します。XML 文書には、キューブ・エレメントを 1 つのみ含めることできます。その他のエレメントはすべて、それに入っています。
<bloxInfo> </bloxInfo>	この Blox についての情報を提供します。
<bloxID> </bloxID>	この Blox インスタンス化 (この値は DB2 Alphablox が自動提供) を識別します。
<bloxName> </bloxName>	このインスタンス化 (この値は bloxName プロパティーが元になっている) に使用される Blox を識別します。

<appName> </appName> アプリケーション (この値は Blox applicationName プロパティーが元になっている) を識別します。

<data> </data> XML 文書の全データ域 (軸定義およびデータ・セルを含む) を識別します。

<slicer> </slicer> スライサー軸を定義する XML 文書の領域を識別します。スライサー軸は、キューブの「スライス」を提供します。OLAP データ・ソースを使用すると、データ域に現れない各ディメンションは、そのメンバーのいずれかと一緒に個別のスライサー軸上に置かれます。

<slicerDimension> </slicerDimension>
スライサー軸上のディメンションを指名します。

<slicerMember> </slicerMember>
スライサー・ディメンション内のメンバーを指名します。

<axis> </axis> 軸タイプ (通常は列または行) を識別し、その内容を定義します。この 5 つの名前付き軸は (昇順で)、列、行、ページ、章、およびセクションです。軸は Axis[index] とも表記されます。ここで index は、0 から N の範囲内にあります。たとえば、行軸を参照する 1 つの方法は、Axis[1] です。最初の名前なし軸を参照する方法は、Axis[5] です。

<dimensions> </dimensions> 軸上のディメンションに名前を付ける <axis> </axis> タグ内にある領域を識別します。

<dimension> </dimension) 特定のディメンションを識別します。

<tuple> </tuple> 1 つの軸にあるディメンション・メンバーすべてのセットが入っているエレメントを識別します。

<member> </member> 1 つのタプルに属しているメンバー 1 つを識別します。

<cells> </cells> (ディメンションおよびメンバー・ヘッダーとは対照的に) データ値の入っている XML 文書の領域を識別します。

<axisCells> </axisCells> 名前の付いていない軸 (Axis[5] から Axis[N] として参照される) を識別し、そのデータ・セルを含めます。

<section> </section> セクション軸 (Axis[4] として参照される) を識別し、そのデータ・セルを含めます。

<chapter> </chapter> 章軸 (Axis[3] として参照される) を識別し、そのデータ・セルを含めます。

<page> </page> ページ軸 (Axis[2] として参照される) を識別し、そのデータ・セルを含めます。

<row> </row> 行軸 (Axis[1] として参照される) を識別し、そのデータ・セルを含めます。

<column> </column>	列軸 (Axis[0] として参照される) を識別し、そのデータ・セルを含めます。
<cell> </cell>	タプルの交差部分のデータ値を識別します。

Alphablox XML タグ属性

Alphablox XML タグは、以下の属性を使用します。索引は 0 ベースであることを覚えていてください。

属性	説明
gen	このエレメントのデータ階層内での世代レベル (特に、ディメンション内のメンバーのレベル) を識別します。例えば、Product ディメンションで、Product は「1」の gen 値があり、 Audio および Visual は「2」の gen 値があり、VCR および TV は「3」の gen 値があります。
index	一連の類似エレメントでこのエレメントの位置を識別します。例えば、次の行は、これが最初のタプルであることを示しています。 <tuple index=″0″>...</tuple>
leaf	これがリーフ・ノード (true) かそうでないか (false) を指定します。
name	このエレメントに固有の名前を与えます。例えば、次の行では、ディメンション・エレメントの名前 (およびデータ値) を示しています。 <dimension name=″memberName″ index=″0″>AliasName</dimension>
span	1 つのメンバーがまたがっているタプルの数を識別します。例えば、Qtr1 という名前のメンバーは、スパンが「3」(January、February、および March) です。 MemberElement の場合、span ではなく spanInHierarchy を使用します。
spanInHierarchy	同一ルートの親によって定義される階層内で 1 つのメンバーがまたがっているタプルの数を識別します。例えば、March という名前のメンバーは、spanInHierarchy 値が 3 です。 MemberElement の場合、span ではなく spanInHierarchy を使用します。
spanIndex	一連のスパンされたメンバーでのこのメンバーの位置をゼロ・ベースで示します。例えば、January は spanIndex が “0”、February は “1”、そして March は “2” です。

MemberElement の場合、spanIndex ではなく spanIndexInHierarchy を使用します。

spanIndexInHierarchy

一連のスパンされたメンバーでのこのメンバーの位置 (階層内でルートの親からの相対位置) をゼロ・ベースで示します。例えば、April が spanIndex 3 で、列 Qtr2 内に出現する場合、spanIndexInHierarchy 値は 0 になります。

MemberElement の場合、spanIndex ではなく spanIndexInHierarchy を使用します。

XML データ・アイランド

XML データ・アイランドは、HTML 文書内に埋め込まれた有効な XML コードの集合です。データ・アイランドを使用すると、プログラマーは、XML 文書を (スクリプトまたは <OBJECT> タグを通して) ロードしなくても、XML 文書に対してスクリプトを記述することができます。現在、XML データ・アイランドは、Microsoft Internet Explorer 5.5 以降でのみ、サポートされています。

定義構文

ページでインライン・データ・アイランドを定義する構文を以下に示します。<XML> および </XML> タグの使用法に注意してください。

```
<XML ID="DataIslandID">
  <XMLDATA>
    <DATA>TEXT</DATA>
  </XMLDATA>
</XML>
```

例えば、以下の行では、3 つのデータ値のあるデータ・アイランドを定義します。

```
<XML ID="MyDataIsland">
  <dataSources>
    <dataSource name="DB2">IBM DB2 OLAP Server 8.1</dataSource>
    <dataSource name="MSOLAP">Microsoft OLAP Services 7.0</dataSource>
    <dataSource name="Essbase">Hyperion Essbase 6.5</dataSource>
  </dataSources>
</XML>
```

データ・アイランドの内容は、外部ファイルに含めることもできます。以下の構文を使用して、データ・アイランドとして外部 XML ファイルを含めます。

```
<XML SRC="http://<server>/MyXmlFile.xml"></XML>
```

XMLDocument プロパティー

XMLDocument プロパティーは、インラインまたは外部 XML データ・アイランドのルート・ノードを戻します。プログラマーは、標準 XML DOM を使用して、このルートからデータ・アイランドをナビゲートすることができます。例えば、以下の関数は MyDataIsland からデータすべてを戻します。

```
function returnXMLData(){
   return document.all("MyDataIsland").XMLDocument.nodeValue;
}
```

以下の構文も有効です。551 ページの『定義構文』の例を使用すると、この行は「IBM DB2 OLAP Server 8.1」の値を戻します。

```
MyDataIsland.XMLDocument.documentElement.childNodes.item(0).text
```

XML データ・アイランドとしての DataBlox

以下の行では、データ・アイランドとしての標準の DataBlox を定義します。

```
<XML ID="MyDataBlox">
   <blox:data id="MyDataBlox"
      dataSourceName = "qcc">
      query = "!"
      render = "XML">
   </blox:data>
</XML>
```

render 属性を XML に設定すると、DataBlox 結果セットは、XML フォーマットにレンダリングされます。ページが処理されると、Blox を定義する行は、レンダリングされた XML 行に置換されます。

このページの別の場所で、JavaScript 関数は次のような構文を通してデータ・アイランドの内容にアクセスできるようになります。

```
MyDataBlox.getCube.getUniqueName
```

付録 C. DB2 Alphablox XML Cube DTD

データベース・スキーマと同様に、*文書型定義* (Document Type Definition (DTD)) は、文書内に現れるデータ構造とそれらが現れる順序を定義します。

文書の構造が分かっていれば、プログラマーは文書の全探索、文書からの特定の値の抽出、およびその値に対する操作を行うコードを書くことができます。

DTD 構文の注記

DB2 Alphablox XML Cube DTD には、以下のマークアップが出現します。

要素型宣言

要素に名前を付け、その子を指定します。

構文

```
<!ELEMENT name (childElement1, childElement2,...childElementN)>
```

使用法

子要素の名前の後の正規表現文字 (+、*、または ?) は、親がいくつの子要素を持てるかを指定します。これらの文字のいずれもない場合、親要素が持つことのできる子要素は 1 つだけであることを意味します。

+	親要素は、この子要素を 1 つ以上持つことができます。
*****	親要素は、この子要素をゼロ以上持つことができます。
?	親要素は、この子要素をゼロまたは 1 つ持つことができます。

たとえば、次のような行は、

```
<!ELEMENT data (slicer*, axis+, cells)>
```

data 要素が、ゼロ以上の slicer 要素、1 つ以上の axis 要素、1 つだけの cells 要素を持つことができると指定することになります。

属性リスト宣言

要素に名前を付け、その属性を指定します。それぞれの属性ごとに、名前、データ型、および必須かオプションかを指定します。

構文

```
<!ATTLIST element-name
        childElementName1 dataType #REQUIRED
        childElementName2 dataType #REQUIRED
        childElementNameN dataType #REQUIRED
>
```

使用法

たとえば、以下の行は

```
<!ATTLIST dimension
       name CDATA #REQUIRED
       index CDATA #REQUIRED
>
```

dimension メンバーには 2 つの必須属性 (name と index) があり、両方とも生の文字データを持つことを指定します。

データ型

要素名と関連付けられて、その要素で許されるデータの型を指定します。

使用法

DB2 Alphablox XML Cube DTD は以下の 2 つのデータ型を使用します。

#PCDATA (解析対象文字データ (Parsed Character Data)): エンティティー参照を含むことが可能な生の (マークアップなし) テキスト。たとえば、& というストリングは解析されてアンパーサンド記号にならなければなりません。

CDATA (文字データ (Character Data)): エンティティー参照を含まない生の (マークアップなし) テキスト。たとえば、小なり記号 (<)、引用符 (“)、またはアンパーサンド (&) は生のテキストとして扱われ、解析されません。

ヒント: DTD 構文の完全な説明は、このガイドの範囲外です。

DTD 要素

リストに出現する DTD 要素それぞれの説明は、548 ページの『Alphablox XML タグ』を参照してください。

DTD リスト

このセクションの残りの部分は、DB2 Alphablox XML Cube DTD のリストです。

```
<!ELEMENT cube (bloxInfo, data)>

<!ELEMENT bloxInfo (bloxID, bloxName, appName)>

<!ELEMENT bloxID (#PCDATA)>
<!ELEMENT bloxName (#PCDATA)>
<!ELEMENT appName (#PCDATA)>

<!ELEMENT data (slicer*, axis*, cells)>

<!ELEMENT slicer  (slicerDimension, slicerMember)>

<!ELEMENT slicerDimension (#PCDATA)>
```

```
<!ATTLIST slicerDimension
        name CDATA #REQUIRED
>

<!ELEMENT slicerMember (#PCDATA)>

<!ATTLIST slicerMember
      name CDATA #REQUIRED
      gen CDATA #REQUIRED
      leaf CDATA #REQUIRED
>

<!ELEMENT axis (dimensions,tuple*)>

<!ATTLIST axis
       name CDATA #REQUIRED
       index CDATA #REQUIRED
>

<!ELEMENT dimensions (dimension*)>

<!ELEMENT tuple (member*)>

<!ATTLIST tuple
        index CDATA #REQUIRED
>

<!ELEMENT dimension (#PCDATA)>

<!ATTLIST dimension
       name CDATA #REQUIRED
       index CDATA #REQUIRED
>

<!ELEMENT member (#PCDATA)>

<!ATTLIST member
       name CDATA #REQUIRED
       gen CDATA #REQUIRED
       span CDATA #REQUIRED
       spanIndex CDATA #REQUIRED
       spanInHierarchy CDATA #REQUIRED
       spanIndexInHierarchy CDATA #REQUIRED
       index CDATA #REQUIRED
       leaf CDATA #REQUIRED
>

<!-- for zero axis, we have cell value only -->
<!ELEMENT cells (axisCells* | section* | chapter* | page*| row*| column*|
                 cell)>
<!ELEMENT axisCells (axisCells+ | section+)><!ATTLIST axisCells
       name CDATA #REQUIRED
>

<!ELEMENT section (chapter+)>
<!ELEMENT chapter (page+)>
<!ELEMENT page (row+)>
<!ELEMENT row (column+)>
<!ELEMENT column (cell)>
<!ELEMENT cell (#PCDATA)>
```

付録 D. コード例の相互参照

この章には、追加のコード例と本書内の例への相互参照があります。

- 558 ページの『例 1: リレーショナル結果セットのウォークスルー』
- 561 ページの『例 2: bloxAPI.call() メソッドを使用したサーバー上のチャート・プロパティーの設定』
- 562 ページの『例 3: サーバー・サイドの ChartPageListener を使用した、チャート・フィルター変更時の望むデータ・フォーマットのチャートに対する設定』
- 557 ページの BookmarksBlox のための例
- 557 ページの Blox Form Tag Library と FormBlox のための例
- 558 ページの Business Logic Blox と Blox Logic Tag Library のための例
- 558 ページの Blox UI Tag Library のための例
- 558 ページの ChartBlox のための例
- 558 ページの CommentsBlox のための例
- 558 ページの データ計算 のための例
- 558 ページの MemberFilterBlox のための例

カテゴリー	例
BookmarksBlox	• 67 ページの『例 1: すべてのブックマークのカウント数を取得する』 • 68 ページの『例 2: ブックマークのプロパティー・セットの取得』 • 70 ページの『例 3: 指定した基準と一致するブックマークのリストの取得』 • 70 ページの『例 4: BookmarksBlox API を使用したブックマークの作成』 • 72 ページの『例 5: サーバー・サイドの bookmarkLoad イベント・フィルターの使用』 • 73 ページの『例 6: ブックマークの照会をロード時に取得する』
Blox Form Tag Library と FormBlox	• 355 ページの『CheckBoxFormBlox の例』 • 358 ページの『CubeSelectFormBlox の例』 • 361 ページの『DataSourceSelectFormBlox の例』 • 363 ページの『DimensionSelectFormBlox の例』 • 365 ページの『EditFormBlox の例』 • 369 ページの『MemberSelectFormBlox の例』 • 371 ページの『RadioButtonFormBlox の例』 • 374 ページの『SelectFormBlox の例』 • 379 ページの『TimePeriodSelectFormBlox の例』 • 386 ページの『TreeFormBlox の例』

カテゴリー	例
Business Logic Blox と Blox Logic Tag Library	• 396 ページの『CrossJoin の例』 • 398 ページの『MDBQueryBlox の例』 • 400 ページの『MemberSecurityBlox の例』 • 402 ページの『TimeSchemaBlox の例』 • 403 ページの『IBM DB2 OLAP Server または Hyperion Essbase データ・ソースのサンプル TimeSchema』 • 404 ページの『Microsoft Analysis Services データ・ソースのサンプル TimeSchema』
Blox UI Tag Library	• 421 ページの『コンポーネント・タグの例』 • 426 ページの『bottomN タグの例』 • 452 ページの『メニュー・タグの例』 • 460 ページの『ツールバー・タグの例』 • 460 ページの『ツールバー・タグの例』
ChartBlox	• 561 ページの『例 2: bloxAPI.call() メソッドを使用したサーバー上のチャート・プロパティーの設定』 • SelectFormBlox を使用したチャート・タイプの動的設定: 374 ページの『SelectFormBlox の例』
CommentsBlox	• 164 ページの『例 1: セルのコメントの使用可能化』 • 165 ページの『例 2: ソートするフィールドおよびソート順序の指定』 • 166 ページの『例 3: MDBResultSet を使用したセル・コメントへのアクセス』 • 167 ページの『例 4: CommentAddedEvent リスナーの追加』
データ計算	• 558 ページの『例 1: リレーショナル結果セットのウォークスルー』 • 198 ページの『例 1: Profit という名前の算出メンバーを Measures ディメンションの末尾に追加する』 • 198 ページの『例 2: 算出メンバーの位置を指定する』 • 199 ページの『例 3: 世代番号および有効範囲を追加する』 • 199 ページの『例 4: 欠落値または NULL 値を 0 で置き換える』 • 200 ページの『例 5: 異なるディメンションからのメンバーを含む計算』 • 200 ページの『例 6: ランキングを追加する』 • 201 ページの『例 7: 各グループ内に個別のランキングを追加する』 • 201 ページの『例 8: 各グループ内に現在高を追加する』
MemberFilterBlox	• 297 ページの『例 1: 選択可能なすべてのディメンションでメンバーをフィルターに掛ける』 • 298 ページの『例 2: 指定されたディメンションのみでメンバーをフィルターに掛ける』 • 298 ページの『例 3: 1 つのディメンションのみでメンバーをフィルターに掛ける』

例 1: リレーショナル結果セットのウォークスルー

```
<%--  RDBResultSet.jsp
----  Example page to illustrate RDB ResultSet Methods
----
----  Walk a server-side RDB ResultSet and output the column
```

```
---   metadata information and the first and last rows of data.
 --%>

<%-- Import the Alphablox taglib --%>
<%@ taglib uri="bloxtld" prefix="blox" %>
<%-- Import the packages for accessing the server-side RDBResultSet--%>
<%@ page import="com.alphablox.blox.data.rdb.*" %>

<%-- creates sqlTypes variable & imports java.sql.Types & java.util.Hashtable--
%>
<%@ include file="SQLTypes.jsp"%>

<blox:data id="relationalDB"
  dataSourceName = "qcc-mssql"
  query = "SELECT * FROM qcc WHERE Sales > 9000 ORDER BY Week_Ending,
Product_Family_Code"
>
</blox:data>
<%
  RDBResultSet rs = (RDBResultSet) relationalDB.getResultSet();

  // Get the schema details
  ResultColumn[] cols       = rs.getColumns(); // column Metadata
  int[]          types      = rs.getTypes();
                              // jdbc/sql data types in the rs
  int            colCount   = cols.length;
                              // num cols in result set

  // each row of data is returned as an array of objects; use types
  // to determine their data types
  Object[]       firstRow   = null;
  Object[]       lastRow      = null;
  int            rowsRead   = 0;

  // iterate through the rows incrementing the row counter and
  // saving the first and last rows of data
  while( rs.hasMoreRows() )
  {
    rowsRead++;
    if( rowsRead == 1 )
    {
      firstRow  = rs.getNextRow( false );
    }
    lastRow = rs.getNextRow( false );
  }
%>

<html>
<head>
    <title>Relational JSP</title>
    <blox:header/><%-- Blox header tag for standard js and style inclusions --%>
</head>

<body>
<br>
The column count is: <b><%= colCount %></b><br />
The row count is: <b><%= rowsRead %></b><br />
The columns are: <br />

<table border="1" cellspacing="0" cellpadding="0">
  <tr>
    <th>Col Name</th>
    <th>Type</th>
    <th>Type Name</th>
    <th>First Row</th>
    <th>Last Row</th>
  </tr>
```

```
  <%
    // Display the column names, their types, typeNames, and
    // the first and last row of data
  for( int i = 0; i < colCount; i++ )
  {
    out.write("¥t<tr>");
    out.write("¥t¥t<td>" + cols[i].getName() + "</td>" ); // Col Name
    out.write("¥t¥t<td>" + String.valueOf(types[i]) + "</td>"); // Type
    // the names of the sql types is in the sqlTypes hashtable created in
SQLTypes.jsp
    out.write("¥t¥t<td>" + String.valueOf( sqlTypes.get( new Integer( types[i]
) ) ) + "</td>" ); // Type Name
    out.write("¥t¥t<td>" + String.valueOf(firstRow[i]) + "</td>"); //First Row
    out.write("¥t¥t<td>" + String.valueOf(lastRow[i]) + "</td>"); //Last Row
    out.write("¥t</tr>");
  }
%>
</table>
</body>
</html>
```

以下のコードは、上記の JSP ファイルで参照されている SQLTypes.jsp ファイルです。

```
<!-- SQLTypes.jsp
---- Helper page to create a hashtable with all of the SQL data types
---- 2002.03.28  -  YRL & REK
----
 -->

<!-- Imports for standard Java classes used -->
<%@ page import="java.sql.Types.*" %>
<%@ page import="java.util.Hashtable" %>

<%
    Hashtable sqlTypes = new Hashtable();

    sqlTypes.put( new Integer( java.sql.Types.ARRAY ), "ARRAY" );
    sqlTypes.put( new Integer( java.sql.Types.BIGINT ), "BIGINT" );
    sqlTypes.put( new Integer( java.sql.Types.BINARY ),"BINARY" );
    sqlTypes.put( new Integer( java.sql.Types.BIT ), "BIT" );
    sqlTypes.put( new Integer( java.sql.Types.BLOB ), "BLOB" );
    sqlTypes.put( new Integer( java.sql.Types.CHAR ), "CHAR" );
    sqlTypes.put( new Integer( java.sql.Types.CLOB ), "CLOB" );
    sqlTypes.put( new Integer( java.sql.Types.DATE ), "DATE" );
    sqlTypes.put( new Integer( java.sql.Types.DECIMAL ), "DECIMAL" );
    sqlTypes.put( new Integer( java.sql.Types.DISTINCT ), "DISTINCT" );
    sqlTypes.put( new Integer( java.sql.Types.DOUBLE ), "DOUBLE" );
    sqlTypes.put( new Integer( java.sql.Types.FLOAT ), "FLOAT" );
    sqlTypes.put( new Integer( java.sql.Types.INTEGER ), "INTEGER" );
    sqlTypes.put( new Integer( java.sql.Types.JAVA_OBJECT ) ,
                            "JAVA_OBJECT"   );
    sqlTypes.put( new Integer( java.sql.Types.LONGVARBINARY ),
                            "LONGVARBINARY" );
    sqlTypes.put( new Integer( java.sql.Types.LONGVARCHAR ),
                            "LONGVARCHAR"   );
    sqlTypes.put( new Integer( java.sql.Types.NULL ), "NULL" );
    sqlTypes.put( new Integer( java.sql.Types.NUMERIC ), "NUMERIC" );
    sqlTypes.put( new Integer( java.sql.Types.OTHER ), "OTHER" );
    sqlTypes.put( new Integer( java.sql.Types.REAL ), "REAL" );
    sqlTypes.put( new Integer( java.sql.Types.REF ), "REF" );
    sqlTypes.put( new Integer( java.sql.Types.SMALLINT ), "SMALLINT" );
    sqlTypes.put( new Integer( java.sql.Types.STRUCT ), "STRUCT" );
    sqlTypes.put( new Integer( java.sql.Types.TIME ), "TIME" );
    sqlTypes.put( new Integer( java.sql.Types.TIMESTAMP ),
                            "TIMESTAMP");
```

```
    sqlTypes.put( new Integer( java.sql.Types.TINYINT ), "TINYINT" );
    sqlTypes.put( new Integer( java.sql.Types.VARBINARY ),"VARBINARY");
    sqlTypes.put( new Integer( java.sql.Types.VARCHAR), "VARCHAR");
%>
```

例 2: bloxAPI.call() メソッドを使用したサーバー上のチャート・プロパティーの設定

```
<%--  chartSelect.jsp
----  Example page to illustrate how to use the call method to
----  execute some server-side code.
--%>

<!-- Import the taglib -->
<%@ taglib uri = "bloxtld" prefix = "blox"%>
<html>
<head>
   <title>Change Repository Values</title>
   <blox:header />
</head>

<!-- The JavaScript function that passes the chart type selected and
---- calls another JSP page (setSelection.jsp) on the server to set
---- the chart type.
--%>

<script language="JavaScript">
   function setChartChoice(ChrtType) {
      bloxAPI.call("setSelection.jsp?chart="+ChrtType);
   }
</script>

<body>
<blox:present id = "myPresent"
   height = "400"
   width = "600"
   >

   <blox:chart
      chartType = "Vertical Bar, Side-by-Side">
   </blox:chart>
   <blox:data
      dataSourceName = "QCC-Essbase"
      useAliases = "true"
      query = "<ROW ('All Products')  <ICHILD 'All Products' <SYM
         <COLUMN ('All Time Periods') <Ichild '2001' !"
   >
   </blox:data>
</blox:present>
<br>
Select a chart type:
<form name = form1>
   <input type="radio" name="chartSelect" value="Bar"
      onclick="setChartChoice(value);"> Bar
   <input type="radio" name="chartSelect" value="Line"
      onclick="setChartChoice(value);"> Line
   <input type="radio" name="chartSelect" value="Pie"
      onclick="setChartChoice(value);"> Pie
</form>
</body>
</html>
```

呼び出される JSP ファイルには、以下のコードがあります。

```
<%--  setSelection.jsp
----  Called by chartSelect.jsp to set the chart type to the one
----  selected.
--%>

<!-- Import the taglib -->
<%@ taglib uri="bloxtld" prefix="blox"%>

<%-- Reference the instance of PresentBlox created in chartSelect.jsp
--%>

<blox:present id="myPresent" />
<%
String chartChoice = request.getParameter("chart");
if (chartChoice != null && chartChoice.trim().length() != 0) {
   myPresent.getChartBlox().setChartType(chartChoice);
   };
%>
```

例 3: サーバー・サイドの ChartPageListener を使用した、チャート・フィルター変更時の望むデータ・フォーマットのチャートに対する設定

この例では、ユーザーがチャート内のフィルターを変更したときに、GridBlox で設定されている正しいフォーマット設定を保持するために、サーバー・サイドのイベント・リスナーを使用して、Y1Axis のためのフォーマットを設定 (setY1FormatMask()) する方法を示します。この例の内容は以下のとおりです。

- シナリオ・ディメンションは、ページ軸上にあります。 Actual と Variance % の両方にセル・フォーマットが指定されています。
- addEventListener() メソッドを使用して、ユーザーがチャートのフィルターを変更したときに呼び出す ChartPageListener オブジェクト (CPListener()) のインスタンスを指定します。
- CPListener が ChartPageListener インターフェースをインプリメントします。
- メンバーを選んでもらいます。この例の場合、メンバー Variance % は名前に "%" があるので、戻りストリングが "%" で終わっているかどうかをテストします。終わっていれば、それにしたがって Y1FormatMask を設定します。

```
<%@ page import="com.alphablox.blox.ChartBlox,
                 com.alphablox.blox.event.*,
                 com.alphablox.blox.uimodel.core.MessageBox,
                 com.alphablox.blox.uimodel.BloxModel,
                 com.alphablox.blox.ServerBloxException"%>
<%@ taglib uri="bloxtld" prefix="blox"  %>
<html>
<head>
   <blox:header />
</head>
<body>
<blox:present id="present" height="400" width="600" >
   <blox:grid defaultCellFormat="#,##0.00;[red](#,##0.00)" >
      <blox:cellFormat index="1" format="#,##0.00;[red](#,##0.00)"
         scope="{Scenario:Actual}" ></blox:cellFormat>
      <blox:cellFormat index="2" format="#,##0.00%;[red](#,##0.00%)"
         scope="{Scenario:Variance %}" ></blox:cellFormat>
   </blox:grid>
   <blox:chart chartType="bar" autoAxesPlacement="false"
      filter="Scenario" XAxis="All Time Periods" legend="All Locations" >
   </blox:chart>
   <blox:data dataSourceName="QCC-Essbase"
      query="{OUTALTNAMES} <ROW (¥"All Locations¥") <ICHILD ¥"All Locations¥"
```

```
<COLUMN (¥"All Time Periods¥", ¥"Scenario¥") <SYM ¥"Jan 01¥" ¥"Feb 01¥"
¥"Mar 01¥" ¥"Apr 01¥" ¥"May 01¥" ¥"Jun 01¥" ¥"Actual¥" ¥"Variance %¥" !">
   </blox:data>
   <%-- adds a ChartPageFilter to the ChartBlox --%>
   <% present.getChartBlox().addEventListener(new
CPListener(present.getBloxModel()));%>

</blox:present>
</body>
</html>

<%!
  public class CPListener implements ChartPageListener
  {
    BloxModel model;
    public CPListener (BloxModel model) {
       this.model = model;
    }
    public void changePage(ChartPageEvent cpe) {
      ChartBlox blox = cpe.getChartBlox();
      try {
        if (cpe.getSelection().endsWith("%")) {
          String msg = new String("Setting format mask to be a percentage");
          blox.setY1FormatMask("#%");
           MessageBox msgBox = new MessageBox(msg, "Chart Page Filter",
MessageBox.MESSAGE_OK, null);
          model.getDispatcher().showDialog(msgBox);
        }
        else {
          String msg = new String("Setting format mask to be currency");
          blox.setY1FormatMask("$#K");
          MessageBox msgBox = new MessageBox(msg, "Chart Page Filter",
MessageBox.MESSAGE_OK, null);
          model.getDispatcher().showDialog(msgBox);        }
      } catch (ServerBloxException e) {
        e.printStackTrace();
      }
    }
  }
%>
```

特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものです。

本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒106-0032
東京都港区六本木 3-2-31
IBM World Trade Asia Corporation
Licensing

以下の保証は、国または地域の法律に沿わない場合は、適用されません。 IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Corporation, J46A/G4, 555 Bailey Avenue, San Jose, CA 95141-1003 U.S.A.

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、 IBM より提供されます。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で決定されたものです。そのため、他の操作環境で得られた結果は、異なる可能性があります。一部の測定が、開発レベルのシステムで行われた可能性がありますが、その測定値が、一般に利用可能なシステムのものと同じである保証はありません。さらに、一部の測定値が、推定値である可能性があります。実際の結果は、異なる可能性があります。お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。 IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

IBM の将来の方向または意向に関する記述については、予告なしに変更または撤回される場合があり、単に目標を示しているものです。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。お客様は、IBM のアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、 IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。

商標

以下は、IBM Corporation の商標です。

IBM	DB2	DB2 OLAP Server
DB2 Universal Database™	WebSphere®	

Alphablox および Blox は、Alphablox Corporation の米国およびその他の国における登録商標です。

Microsoft、 Windows® および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。

Java およびすべての Java 関連の商標およびロゴは、Sun Microsystems, Inc. の米国およびその他の国における商標または登録商標です。

Linux は、Linus Torvalds の米国およびその他の国における商標です。

他の会社名、製品名およびサービス名等はそれぞれ各社の商標です。

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

[カ行]

[サ行]

[タ行]

［ハ行］

［マ行］

［ラ行］

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

プログラム番号: 5724-L14

Printed in Japan

SD88-6491-01

日本アイ・ビー・エム株式会社
〒106-8711　東京都港区六本木3-2-12