IBM DB2 Alphablox

DB2 Alphablox 開発者用ガイド

バージョン 8.4

SD88-6490-02
(英文原典：SC18-9434-02)

IBM DB2 Alphablox

DB2 Alphablox 開発者用ガイド

バージョン 8.4

SD88-6490-02
(英文原典：SC18-9434-02)

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、 319 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 4 および新しい版で明記されていない限り、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。

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： SC18–9434–02
IBM DB2 Alphablox
DB2 Alphablox Developer's Guide
Version 8.4

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 2006.3

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 DB2 Alphablox アプリケーションおよび基礎となる Blox

DB2® Alphablox は、カスタム・ビジネス分析アプリケーションの作成を可能にします。カスタム・ビジネス分析アプリケーションとは、各種のデータ・ソースからのライブ・ビジネス・データおよび取り引きを、エンド・ユーザーが視覚化し、分析するのを助けるアプリケーションです。DB2 Alphablox アプリケーションは、照会およびレポート作成ツールの様式で単にデータを供給するのではなく、通常ビジネス・ロジックを組み入れて、使いやすいインターフェースを介してガイドされた分析を提供します。

DB2 Alphablox アプリケーションは、Blox として知られる DB2 Alphablox 構築ブロックを含む任意の Web アプリケーションです。アプリケーションは 1 つの JSP ページほど単純な場合もあり、さまざまなアプリケーション・サーバーおよびデータ・ソースと通信しあう Web ページの集合全体といった複雑なものにもなり得ます。

Blox は、JSP タグや Java™ コードを使用して JSP ページに追加できる再使用可能なソフトウェア・コンポーネントであり、データ・ソースに接続したり、データ形式変更や計算を実行したり、対話式のデータ分析機能を提供したりします。

このセクションでは、DB2 Alphablox アプリケーションに共通する、鍵となる特性に焦点を合わせます。グラフィカル表現とサンプル・シナリオを用いて、このセクションでは、DB2 Alphablox 内のフィーチャーとコンポーネントがどのようにそれらの特性を生み出しているかを例示します。

DB2 Alphablox アプリケーション・プログラムの流れと開発アプローチの詳細は、11 ページの『第 2 章 DB2 Alphablox アプリケーション・プログラムの流れ』を参照してください。

DB2 Alphablox アプリケーションの鍵となる特性

DB2 Alphablox アプリケーションには通常、次のような特性があります。それぞれの特性は、DB2 Alphablox のフィーチャーのさまざまな組み合わせを使ってインプリメントすることができます。

- 『リアルタイムのデータ・アクセスおよび分析』
- 2 ページの『対話式ユーザー・インターフェース』
- 4 ページの『個別設定』
- 4 ページの『共用とコラボレーション (共同作業)』
- 5 ページの『リアルタイムの計画』

リアルタイムのデータ・アクセスおよび分析

DB2 Alphablox アプリケーションは、リレーショナルとマルチディメンションの両方の、複数のデータ・ソースからのデータの分析を行うことができます。データベ

ースへのネイティブ・アクセス (Microsoft® Analysis Services には MDX、DB2 OLAP Server™ および Essbase には Report Script、リレーショナル・データベースには JDBC) を通して、DB2 Alphablox はデータベース・エンジン内の分析機能を公開します。それには、ランキング、派生した計算、順序付け、フィルター操作、百分位数、分散、標準偏差、相関、傾向、統計関数、その他の複雑な計算があります。

ライブ・データをユーザーに提示するには、いくつかの異なる方法があります。 ユーザーがグリッドまたはチャート形式のデータを必要とする場合は、まず DataBlox をアプリケーションに追加してから、その DataBlox インスタンスで使用するデータ・ソースを指定します。これで、データベース・エンジンに備わっているすべての分析機能にアクセスできるようになります。次に、その DataBlox からのデータを使用するために PresentBlox を追加します。これに GridBlox と ChartBlox を組み込みます。これで、ユーザーは Blox ユーザー・インターフェースを通して最新のデータと対話して、データ分析のニーズを満たすことができます。

例えば CFO (経理部長) の場合、ログインしたときに最初に表示される画面は経営者のダッシュボードであり、それには月間所得の記述と市場収益のランキングの要約が含まれているかもしれません。データはライブであり、どの顧客がどの商品を購入しているかを調べようとする場合、CFO はデータをドリルダウンすることを選択できます。

リレーショナル・データベースからレポートを作成する場合には、Relational Reporting を使用できます。 Relational Reporting の中核を成すのは ReportBlox コンポーネントであり、このコンポーネントは DHTML ベースのレポートにリレーショナル結果セットをレンダリングします。その他の Blox コンポーネントは、Relational Reporting でのデータ・アクセス、データ形式変更、計算、およびフォーマットをサポートします。これらの Blox は、それぞれの名前によって表される特定のタスクを実行します。

リレーショナル・レポートは、静的なレポートまたは対話式レポートにすることができます。対話モードのレポートをレンダリングした場合、ユーザーは Relational Reporting の Report Editor ユーザー・インターフェースを使用して、その場でデータをソートしたり、フィルターに掛けたり、再配列したりして、独自のリレーショナル・レポートを設計することができます。

対話式ユーザー・インターフェース

通常、DB2 Alphablox アプリケーションには DHTML レンダリングで利用できるグリッドとチャートが備わっており、サポートされている Web ブラウザーを使用してそれらにアクセスできます。

DHTML クライアントでレンダリングされたグリッドとチャートには、使いやすいユーザー・インターフェースがあるので、データ分析が複雑になることはありません。PresentBlox を追加すると、それには GridBlox、 ChartBlox、 ToolbarBlox、 PageBlox、および DataLayoutBlox をネストさせることができ、対話式のデータ分析、ブックマーク、データのエクスポート、およびビューのカスタマイズ能力をユーザーに提供します。開発者はインターフェース内のさまざまなコンポーネントをカスタマイズおよび個別設定することにより、設計上のニーズに応えることができます。

DataLayoutBlox はデータ・レイアウト・パネルとして現れ、ユーザーは対話式に軸間でディメンションを移動したり表示したりすることができます。 ToolbarBlox はツールバーとして現れ、共通に実行されるデータ分析タスクには、マウスのクリック一つでクイック・アクセスできます。 メニュー・バーは、ユーザーが選択可能なすべてのオプションとアクションを提供します。ユーザーはビューにブックマークを付けたり、グリッドやチャートを表示したり隠したり、データをソートしたり、データを PDF や Excel にエクスポートしたり、データをナビゲートすることができます。PageBlox はページ・フィルターとして現れ、ユーザーは GridBlox および ChartBlox に現れるデータをフィルターに掛けることができます。アプリケーション開発を単純化し、画面の領域を節約するため、これらすべての Blox は PresentBlox の中にネストされています。

ユーザー・インターフェース内のコンポーネントは、DB2 Alphablox Tag Libraries で提供される JSP タグを使ってカスタマイズできます。例えば、使用する色を指定したり、ツールバー内のボタンを追加または除去したり、メニュー・バーでメニューを追加または除去したり、データ・ナビゲーション・オプションを追加または除去することができます。さらに、セルの強調表示のための基準のセットを指定することができます。これはセル・アラートという機能で、たとえば最小値より低い値が指定されたときにセルを赤で表示します。

このユーザー・インターフェースの長所は、ユーザーがデータを対話式に処理するたびにページが最新表示されることはないという点です。ページ全体がダウンロードし終わるまで待つ必要がないので、長く待たされたために何をしようとしていたかを忘れてしまったというようなことはありません。またポータル環境の場合には、ポータル・ページ全体を最新表示するとなるとページ上のすべてのポートレットを再ロードすることが必要になり、膨大な時間を要する可能性があるため、この長所による利点は計り知れません。

DB2 Alphablox テーマ

DB2 Alphablox では、さまざまな関連スタイル・シートと GIF イメージから成るすぐに使用可能なテーマがいくつか DHTML クライアント向けに用意されており、これらはそのまま使用することができます。また、既存の DB2 Alphablox テーマをコピーし、そのテーマで使用されているスタイル・シートとイメージに変更を加えることによって、独自のテーマを作成することもできます。

メンバー・フィルター

時折ユーザーは、一度に 1 レベル上または下にドリル移動するよりも、さらに特定のデータを見たいと思うことがあります。同じディメンション階層内の異なる親に属する特定のメンバーのデータが対象となるかもしれません。例えば、あるユーザーは各地域内の代表的な州からのデータを比較することだけを希望します。

メンバー・フィルター・インターフェースを使用することにより、Market ディメンションをナビゲートして、East 地域から New York を選択し、West 地域から California を選択し、Central 地域から Illinois を選択し、South 地域から Texas を選択することができます。

メンバー・フィルターは、データ・ナビゲーション・オプションが提供されている場合に、GridBlox、DataLayoutBlox、および PageBlox 内で、さまざまな右クリック・メニューおよびポップアップ・メニューから使用可能です。「メンバー・フィ

ルター」ダイアログ・ウィンドウにリストされるディメンションは、メニューを立ち上げるためにユーザーがどこで右クリックするかによって決まります。

Relational Reporting ユーザー・インターフェース

ReportBlox とそれをサポートする Blox を使用して対話式レポートを作成すると、ユーザーは Report Editor ユーザー・インターフェースを介して、列をソート、隠蔽、再配列したり、ブレーク・グループを作成したり、各ブレーク・グループごとにサマリー・データを追加することができます。

Report Editor はコンテキストに依存した 3 つのポップアップ・メニューから成ります。これらすべては DHTML でサポートされます。レポートと対話式メニューは、特定の CSS スタイル・クラスによってレンダリングされます。 使用するスタイルを指定することにより、色とフォントをカスタマイズすることができます。

個別設定

ユーザーごとにデータや業務上の必要は異なるため、DB2 Alphablox アプリケーションを個別設定することが必要になる場合がよくあります。例えば、ログイン時に最初に表示される画面をユーザーごとに違ったものにすることができます。データ・ナビゲーションを制御して、西部 (West) 地域のユーザーに東部 (East) 地域のデータが表示されないようにすることもできます。また、チャート・タイプの好みの設定やグリッド内のデータを強調表示する際のしきい値を、ユーザーが自分で指定するようにもできます。

カスタム・プロパティー

DB2 Alphablox では、カスタム・プロパティーを使用して個別設定をサポートします。カスタム・ユーザー・プロパティーを定義して、それぞれのプロパティーに有効な値を指定することができます。次いでそれぞれのユーザーごとに、定義されたプロパティーおのおのについて、値を割り当てることができます。ユーザー・ログインと、ユーザーに関連付けられたプロパティー値に基づいて、何を表示するか、データをどのように表示するか、どんなデータ・ナビゲーション機能を使用可能あるいは使用不可にするかを動的に指定することができます。

共用とコラボレーション (共同作業)

共用とコラボレーション (共同作業) のサポートに使用される、DB2 Alphablox の共通フィーチャーとして、ブックマーク、データのコメント付け、そして PDF 変換が挙げられます。

ブックマークの設定

DB2 Alphablox の重要な機能としてブックマーク機能があります。ユーザー・インターフェースを通して、ユーザーはデータ・ビューにブックマークを付け、あとで同じビューを最新のデータで取り出すことができます。ブックマークは、プライベートなもの、または特定グループ内のユーザーが使用可能なもの、あるいはサーバーにアクセスできるすべてのユーザーの公用とすることができます。

BookmarksBlox は、ブックマークの管理と操作のための広範な API を提供します。例えば、データ一括表示内で変更を反映するために、すべてのブックマークをプログラマチックに更新することができます。

データのコメント付け

共同作業による分析をサポートするため、CommentsBlox を利用して、セル・レベル、ページ・レベル、あるいはアプリケーション・レベルの注釈をサポートすることができます。ユーザーは、セルを右クリックして「コメントの追加」を選択することにより、GridBlox 内のデータ・セルにコメントを追加することができます。コメントの付いたセルには隅にコメント標識があるので、ユーザーは素早くスポットを合わせて、コメントを表示するよう選択できます。

PDF にエクスポート

ユーザーはしばしば、作業を保管したり、データのビューを共用しようとします。DB2 Alphablox には「PDF にエクスポート」するオプションがあり、Blox 内のデータを PDF 形式に保管することを可能にしています。これにより、ブラウザーを使用した Web ページの印刷や保管に共通した問題の大半が解決されます。例えば、不適切な改ページ、チャートを表示するには不適当なページ幅、ブラウザー間の印刷の相違、およびレポートで使用されるすべての HTML およびイメージの E メール送信といった問題です。

「PDF にエクスポート」オプションでは、ページ・レイアウト、ページの向き、改ページ、ヘッダーおよびフッターのテキスト、およびチャートのサイズをユーザーが制御できます。必要に応じて、ダイアログをカスタマイズし、フォント・サイズ、色、ヘッダーとフッターの位置を制御できるようにすることも可能です。

リアルタイムの計画

分析アプリケーションの範囲は、履歴の分析から、将来の予測や、先行して行うリソース割り振りにまで及びます。予算の作成、販売予測、および共同作業によるデマンド・プランニングといったリアルタイム計画アプリケーションを、DB2 Alphablox データ書き戻し機能を使用して構築することができます。

例えば、データ・ソースからデータを抽出して GridBlox に入れたり、地域管理者が販売予測数を GridBlox 内に入力するのを許可したり、データのサブミットの際にデータをデータ・ソースに書き戻すことができます。カスタム・プロパティーとの併用により、アプリケーションは、ユーザーが属する地域をベースとした販売予測ワークシートを動的に作成することができます。

基礎となる Blox コンポーネント

DB2 Alphablox アプリケーションの基礎となるキー・コンポーネントは Blox コンポーネントです。 Blox は再使用可能なソフトウェア・コンポーネントで、データ・ソースへの接続、さまざまなデータ操作およびプレゼンテーション・タスクの実行を可能にし、動的で個別設定されたデータ分析アプリケーションを構築できるようにします。

1 つの Blox で、そのプロパティーと関連メソッドを通して、上記の機能性のいくつかを提供することがあります。そうしたプロパティーとメソッドにより、Blox の外観と動作を指定し、制御することができます。ドリルアップ/ダウン、ピボット、ページ・フィルターの変更、あるいはブックマークのロードといったユーザー・イベントの処理のためのイベント・フィルターもあります。

次の表では、それぞれの Blox の要旨を説明しています。

Blox	説明
DataBlox	• 7 ページの『DataBlox』: 鍵となる Blox で、すべてのデータ・プレゼンテーション Blox のために、データのアクセス、検索、および操作の機能性を提供します。 • 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。
ユーザー・インターフェース Blox	対話式ユーザー・インターフェースを持つグリッドまたはチャート形式でデータを提示できるようにする 6 つの Blox があります。それらによってユーザーは、データを分析し、ページ・フィルターを変更し、ブックマークを追加またはロードし、グリッドまたはチャートのレイアウトを指定することなどが可能です。これらの Blox は次のとおりです。 • 8 ページの『GridBlox』 • 8 ページの『ChartBlox』 • 8 ページの『DataLayoutBlox』 • 8 ページの『PageBlox』 • 8 ページの『ToolbarBlox』 • 9 ページの『PresentBlox』 これらの Blox はそれぞれユーザー・インターフェース・コンポーネントとアプリケーション・プログラミング・インターフェース (API) を持ち、許可されたプレゼンテーションとアクションの制御を可能にします。 その多くは結果セットを戻し、そこからさらに多くのメタ情報を得ることができます。
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。 • RepositoryBlox: DB2 Alphablox Repository 内のアプリケーション・プロパティーを保管および検索する手段を開発者に提供します。 • 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。 • AdminBlox: DB2 Alphablox ホーム・ページ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。
ビジネス・ロジック Blox	以下の Blox コンポーネントにより、ビジネス・ロジックをアプリケーションに追加します。 • 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。
FormBlox	一連の FormBlox を使用すれば、使い慣れた HTML フォーム・インターフェースが提供され、状態管理が処理されます。これらの FormBlox はデータを認識し、データ・ソース、ディメンション、メンバー、または開発者が提供する他のオプションをユーザーが選択して、個別設定された照会を作成できるようにします。
ContainerBlox	ContainerBlox は DHTML モードでのみ使用可能であり、これを使用してカスタム Blox コンポーネントを作成し、ユーザー・セッション中の持続性を管理することができます。
ReportBlox	対話式 HTML 形式でレンダリングされる ReportBlox は、リレーショナル・データ・ソースからレポートを構築するための中核となる Blox です。 ReportBlox とそれをサポートする Blox の詳細は、「*Relational Reporting 開発者用ガイド*」を参照してください。

以下のセクションでは、DataBlox と各ユーザー・インターフェース Blox が何を可能にし、それらが協働してどのように下記の機能を提供するかに焦点を当てます。

- プログラマチック制御を開発者に提供
- 視覚に訴えるデータ分析経験をユーザーに提供

DataBlox

DataBlox は、データ・アクセスに必要な機能性を特に提供する Blox です。これにはグラフィカル・ユーザー・インターフェースはありません。代りに、ユーザーがデータと対話するためのグラフィカル・ユーザー・インターフェースを提供するすべての Blox の中心となります。これは広範なアプリケーション・プログラミング・インターフェース (API) を備えています。例えば、必要なデータ・ソースへの接続が成功したか、あるいは現行のデータベース操作が完了したかどうかを検出することができます。ユーザーがだれであるかに応じて、ユーザーが特定のデータ・ナビゲーション・アクションを実行したり、結果セット内の特定のデータを見たりするのを阻止することができます。 次の表は、DataBlox の API の広範さを例示する、DataBlox に関連したプロパティーとメソッドの一部を示しています。

DataBlox プロパティー/メソッドのカテゴリー	**説明**	**例**
データ・ソース	データ・ソースに関連したプロパティー	`aliasTable, catalog, query, schema, connectOnStartup, userName, password, dataSourceName`
データ操作	算出メンバー、ソート、ドリルなどのデータ操作に関連したプロパティー	`calculatedMembers, columnSort, rowsort, hiddenMembers, keepOnly, parentFirst`
データの外観	重複、欠落、ゼロのデータやメンバー名の接頭部を表示するかどうか、またどのように表示するかといった、データ外観に関連する	`memberNameRemovePrefix, memberNameRemoveSuffix, suppressDuplicates, suppressMissing, suppressNoAccess, suppressZeros`
書き戻し	データ更新に関連する	`commitData()`
結果セット	データを含む結果セットに関連する	`clearResultSet(), getResultSet()`
メタデータ	現行 DataBlox の基礎となるデータ・ソースの MetaData オブジェクトに関連する	`dimensionRoot, getMetaData()`
MDB 結果セット	マルチディメンション・データ結果セット内の軸、ディメンション、タプル、およびセルに関連する	`((MDBResultSet) getResultSet())`
RDB 結果セット	リレーショナル・データ結果セット内の列および行に関連する	`((RDBResultSet) getResultSet())`
MDB メタデータ	結果セットのためのマルチディメンション・メタデータに関連する	`((MDBMetaData) getMetaData())`
RDB メタデータ	結果セットのためのリレーショナル・メタデータに関連する	`((RDBMetaData) getMetaData())`
イベント・フィルター	サーバー・サイドのイベント・フィルターに関連する	`addFilter(), removeColumnSort()`

GridBlox

GridBlox はリレーショナル・データまたはマルチディメンション・データを拡張されたグリッド形式で表示し、ユーザーはデータをドリル、ピボット、ソート、および探索することができます。その外観、数値フォーマット、その他を制御するための、プロパティーと関連メソッドの広範なセットを備えています。デフォルトでは、スタンドアロンの GridBlox は以下のものを持っています。

- メニュー・バー。GridBlox および基礎となる DataBlox で使用可能なすべてのオプションと機能性を提供します。
- ToolbarBlox。ボタンをクリックするだけで、ユーザーは共通の機能性にクイック・アクセスできます。

ChartBlox

ChartBlox はリレーショナル・データまたはマルチディメンション・データをさまざまなチャート形式で表示し、ユーザーはチャートの外観を変更したり、データを探索することができます。ChartBlox にデータ・アクセスおよびデータ操作の機能を提供するには、DataBlox が必要です。デフォルトでは、スタンドアロンの ChartBlox は以下のものを持っています。

- メニュー・バー。ChartBlox および基礎となる DataBlox で使用可能なすべてのオプションと機能性を提供します。
- ToolbarBlox。ボタンをクリックするだけで、ユーザーは共通の機能性にクイック・アクセスできます。

DataLayoutBlox

DataLayoutBlox は、選択可能なデータ・ディメンションおよびそのディメンションが現在置かれている軸を表示し、ユーザーは軸間でディメンションを移動することができます。DataLayoutBlox は PresentBlox の中にネストされます。スタンドアロンの GridBlox やスタンドアロンの ChartBlox の中にネストすることはできません。

ユーザーがディメンションをある軸から別の軸に移動すると、同じ PresentBlox にネストされている GridBlox と ChartBlox の両方のデータは自動的に変更を反映します。

PageBlox

PageBlox はページ軸上にあるディメンションを表示します。その際、チャートまたはグリッド内のデータをフィルター操作しており、ユーザーはデータ・フィルターを変更することができます。PageBlox は PresentBlox の中にネストしています。スタンドアロンの GridBlox やスタンドアロンの ChartBlox の中にネストすることはできません。ユーザーが PageBlox でディメンションから選択を行うと、同じネスティング PresentBlox の中にある GridBlox および ChartBlox 内のデータは、選択されたフィルターを反映します。

ToolbarBlox

ToolbarBlox は、ユーザーがさまざまな Blox の機能性を利用するためのボタンを表示します。ToolbarBlox は、PresentBlox の中か、スタンドアロンの GridBlox または ChartBlox の中にネストする必要があります。デフォルトでは、ToolbarBlox はそれ

らのユーザー・インターフェース Blox 内でオンになっており、2 つのツールバーとして現れます。これらのツールバーはカスタマイズ可能です。既存のツールバーでボタンを追加したり除去することができます。ツールバー自体を追加したり除去することもできます。

PresentBlox

PresentBlox は上記のすべての Blox を単一の Blox に結合することで、アプリケーション開発を単純化し、画面の領域を節約します。PresentBlox 内にネストされているすべての Blox は互いに対話します。それらは同じデータ・ソースと、PageBlox で行われる同じデータ・ナビゲーション・アクションを使用します。例えば、ネストされた GridBlox と ChartBlox の両方で、表示されるデータが影響を受けます。指定されるデータ・オプションは、該当するなら、ネストされたすべての Blox に反映されます。例えば、メンバー名に別名を使用するように指定した場合、別名は GridBlox、ChartBlox、および PageBlox で使用されます。

DB2 Alphablox FastForward

DB2 Alphablox FastForward は、カスタム分析ビューを素早く開発、デプロイ、共用するための、サンプル・アプリケーション・フレームワークです。すぐに使用可能な FastForward フレームワークは、セキュリティー、コラボレーション、カスタマイズ、および個別設定を含む共通アプリケーション・サービスを配信します。アプリケーション管理者 (通常は OLAP 管理者) は、新しいバージョンの FastForward アプリケーションを作成し、レポート・テンプレートを選択してレポート・パラメーターを構成することによりレポートをパブリッシュした後、コードをまったく見ることなく新規アプリケーションをデプロイすることができます。JSP 開発者はアプリケーション・フレームワークを変更または拡張して、アプリケーション管理者が構成およびデプロイできるように新規カスタム・レポート・テンプレートを追加できます。詳しくは、「*開発者用ガイド*」の『DB2 Alphablox FastForward での作業』のトピックを参照してください。

第 2 章 DB2 Alphablox アプリケーション・プログラムの流れ

このセクションでは、DB2 Alphablox アプリケーションのファイル構造、DB2 Alphablox およびアプリケーション・サーバーがアプリケーションを処理する方法、およびアプリケーション開発者が標準的な Web テクノロジーを使用して該当するエンド・ユーザーと対話したり、プログラムを制御したりするための方法を説明します。

アプリケーション・ファイル構造

DB2 Alphablox は Java 2 Enterprise Edition (J2EE) Web アプリケーション・サーバー環境で実行されるため、このセクションでは、DB2 Alphablox アプリケーションを作成する際の基礎となる Web アプリケーション・サーバーのファイル構造について説明します。

アプリケーション・コンテキスト

DB2 Alphablox ホーム・ページからアプリケーションを作成する場合、アプリケーション・コンテキスト、表示名、ホーム URL、デフォルトの保管状態、および書き込み特権のセキュリティー役割などの情報を指定するように要求されます。こうした一連の情報に基づき、DB2 Alphablox は DB2 Alphablox リポジトリーに加え、アプリケーションのディレクトリー構造にもアプリケーションの定義を作成します。指定したアプリケーション・コンテキストの名前が付けられたディレクトリーが作成され、このディレクトリーはアプリケーション「docroot」、アプリケーション・コンテキスト、またはアプリケーション・ディレクトリーと呼ばれます。

このアプリケーション・ディレクトリーの物理的な位置は、アプリケーション・サーバーによって異なります。DB2 Alphablox が WebSphere® を使用してインストールされる場合、アプリケーション・ディレクトリーは WebSphere の `installedApps` ディレクトリーにあります。詳しくは、「*管理者用ガイド*」を参照してください。

作成するアプリケーションに関するファイルはすべて、このアプリケーション・ディレクトリー構造の中に置く必要があります。一般に、これらは JSP、 HTML、 CSS、 JavaScript™、およびイメージ・ファイルの組み合わせです。さらに、Java クラス、またはサーブレット、Beans、他のユーティリティー・クラスの入った他の Java アーカイブ・ファイルは通常、J2EE の仕様で推奨されているとおり、`classes` および `lib` ディレクトリー内の `WEB-INF` ディレクトリーのサブディレクトリーに入れられます。

DB2 Alphablox リポジトリー

DB2 Alphablox リポジトリーは、アプリケーション、ユーザー、グループ、ブックマーク、および他の情報をトラックするために、DB2 Alphablox が使用するオブジェクトの保管場所です。DB2 Alphablox リポジトリーに関係した物理ファイルは、`<db2alphablox_dir>/repository` ディレクトリー (DB2 Alphablox ファイル・システム・リポジトリーを使用した場合) に置かれます。ここで、`<db2alphablox_dir>` は DB2 Alphablox のインストール・ディレクトリーです。

たとえば、DB2 Alphablox ホーム・ページから「MyApp1」という名前のアプリケーションを作成する場合、そのアプリケーションのための「MyApp1」という名前のフォルダーが、<db2alphablox_dir>/repository/applications/ ディレクトリーの下に作成されます。カスタム・アプリケーションのプロパティーを定義する際に、その情報を保管するためにアプリケーション・プロパティー記述子ファイル appprodesc.properties が更新されます。

同様に、ユーザーを追加する際には、そのユーザーの名前のフォルダーが <db2alphablox_dir>/repository/users/ ディレクトリーの下に作成されます。各ユーザーには、パスワード、E メール・アドレス、およびグループの関連など、DB2 Alphablox ホーム・ページを通して定義される、関連した情報を保管するためのユーザー・プロパティー・ファイルがあります。

RepositoryBlox API を使用することによって、アプリケーションの状態の取得、設定、保管、または削除を行ったり、ユーザー名およびユーザーが属するグループを取得したりすることができます。

JavaServer Pages での Blox の処理

J2EE 環境では、動的コンテンツを提供する場合に使用される重要なテクノロジーは JavaServer Pages (JSP) です。JSP テクノロジーでは、HTML、JavaScript、および Java コードを 1 つの物理ファイルに組み合わせることができます。

Blox は通常 Java bean であるため、Blox を追加するには、Java bean のときのように、JSP タグ (<jsp:useBean> タグ) を使用して、その bean を組み込みます。さらに、DB2 Alphablox のカスタム JSP タグを活用して、XML に似た構文を使って Blox を追加できます。

要求の処理

このセクションでは、DB2 Alphablox アプリケーションの HTTP 要求が、基礎となるアプリケーション・サーバーおよび DB2 Alphablox によって処理される方法を説明します。以下のセクションでは、そうしたプロセスを大まかに単純化して概観します。さらに詳細な全体像については、JavaServer Pages テクノロジーの資料を参照してください。

説明は、以下に示すファイルで、アプリケーション・コンテキスト MyApp1 を持つアプリケーションに基づいています。

- welcome.html: これはアプリケーションの入り口ページです。このページには、intro.jsp と firstGrid.jsp へのリンクがあります。
- intro.jsp: これは、一般的な Java および JavaScript コードを持つ JSP ファイルです。
- firstGrid.jsp: すでに 『JavaServer Pages での Blox の処理』セクションで示されたのと同様の、GridBlox が入っている JSP ファイル。

さらにこの説明では、別個の Web サーバーはなく、アプリケーション・サーバーが Web ページを提供することを想定しています。

ユーザー要求 1 (http://myAppServer/MyApp1/welcome.html)

1. ユーザー「dave」は、ブラウザーを使用して http://myAppServer/MyApp1/welcome.html にアクセスします。
2. アプリケーション・サーバーは MyApp1/ に行き、WEB-INF/ ディレクトリーにあるアプリケーション・デプロイメント記述子ファイル web.xml に定義されているセキュリティー情報を参照します。
3. 定義されているセキュリティーの制約に基づいて、アプリケーション・サーバーはユーザー名とパスワードの要求に応答します。
4. J2EE セッションが認証によって開始されます。アプリケーション・サーバーは、応答の中で cookie をブラウザーに返します。送信される cookie には、セッション ID が入っています。
5. アプリケーション・サーバーは welcome.html を解析して、それをブラウザーに戻します。

ユーザー要求 2 (http://myAppServer/MyApp1/intro.jsp)

1. Dave は intro.jsp へのリンクをクリックします。http://myAppServer/MyApp1/intro.jsp の HTTP 要求が送信されます。
2. アプリケーション・サーバーは、cookie とヘッダー情報にアクセスして、J2EE セッション ID を探し、セキュリティーを確認します。
3. アプリケーション・サーバーには、JSP ファイルをコンパイルして実行する JSP エンジンが用意されています。アプリケーション・サーバーは最初に、このファイルがコンパイルされているかどうか、または最後にコンパイルされてから変更されているかどうかを確認します。

   コンパイルが必要な場合、エンジンはファイルを処理し、Java クラス・ファイルにコンパイルします。ファイルで参照されているクラスとパッケージが JSP ファイルに存在しているかどうか、さらに構文が正しいかどうかを確認します。
4. アプリケーション・サーバーはコンパイルされたファイルを実行して、ブラウザーに応答を戻します。

ユーザー要求 3 (http://myAppServer/MyApp1/firstGrid.jsp)

1. Dave は welcome.html に戻り、firstGrid.jsp へのリンクをクリックします。http://myAppServer/MyApp1/firstGrid.jsp の HTTP 要求が送信されます。
2. アプリケーション・サーバーは、cookie とヘッダー情報にアクセスして、J2EE セッション ID を探し、セキュリティーを確認します。
3. アプリケーション・サーバーは最初に、このファイルがコンパイルされているかどうか、または最後にコンパイルされてから変更されているかどうかを確認します。
4. コンパイルが必要な場合、その JSP エンジンはファイルを処理し、Java クラス・ファイルにコンパイルします。アプリケーション・サーバーは、<%@ ...%> ディレクティブで参照されている、クラス、パッケージ、およびタグ・ライブラリー記述子ファイル (TLD) が存在しているかどうか、使用される Java メソッドとカスタム・タグが有効で、構文が正しいかどうかを確認します。その後、アプリケーション・サーバーは firstGrid.jsp を実行します。

5. それは以下に示すスクリプトレットを検出し、それを処理します。変数 banding は、true または false の値を取得します。

```
<% String banding =
 (Math.random() >= 0.5) ? "true" : "false"; %>
```

6. その後、見慣れないタグ <blox:grid...> を検出します。接頭部 blox は、taglib ディレクティブに指定した内容 <%@ taglib uri="bloxtld" prefix="blox" %> と一致しています。
7. アプリケーション・サーバーは、taglib ディレクティブに定義されているように、タグ・ライブラリーに進みます。タグは、beans を作成して初期化する、実際の Java コードによって置き換えられる「マクロ」です。アプリケーション・デプロイメント記述子ファイル /MyApp1/WEB-INF/web.xml は、以下のようにタグ・ライブラリー記述子ファイル (TLD) が置かれているアプリケーション・サーバーを示します。

```
<taglib>
   <taglib-uri>bloxtld</taglib-uri>
   <taglib-location>/WEB-INF/tlds/blox.tld</taglib-location>
</taglib>
```

8. DB2 Alphablox が呼び出され、準備が整いました。DB2 Alphablox は bean、ユーザー・セッション、アプリケーション・インスタンス、およびピアを初期化し、結果をアプリケーション・サーバーに戻します。 Blox がどのように処理され、サービスを提供するかについては、『DB2 Alphablox プログラムの流れ』で説明します。
9. アプリケーション・サーバーは、firstGrid.jsp の各行を処理し、終わりまで続けます。
10. アプリケーション・サーバーは結果をブラウザーに戻します。

アプリケーション・サーバーの役割

要約すると、アプリケーション・サーバーは以下のタスクを担当します。

- ユーザー認証およびセキュリティー
- HTML ファイルの処理とサービス提供
- サーブレットまたは JSP エンジンからの助けを得ながら、JSP ファイルを処理し、コンパイルして、生成した応答全体をブラウザーに戻します。

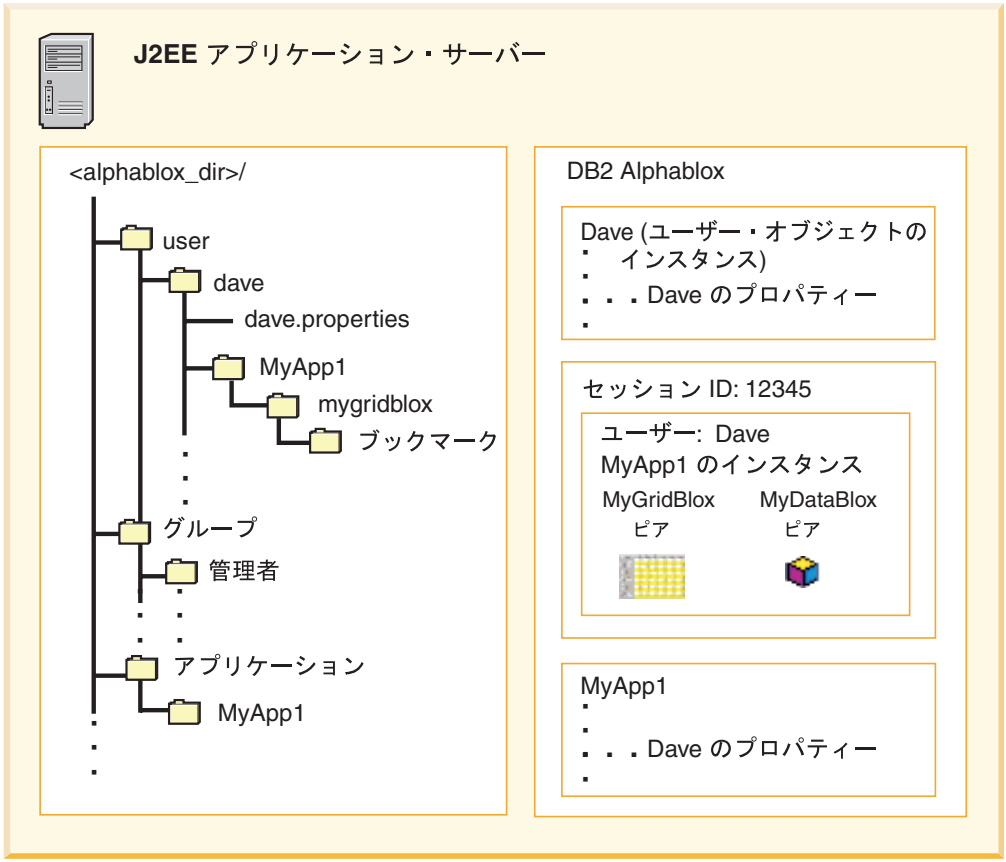
DB2 Alphablox プログラムの流れ

アプリケーション・サーバーは <blox:grid> などのカスタム・タグを検出すると、指定したタグ・ライブラリーを呼び出します。アプリケーションの web.xml ファイルの仕様に基づいて、アプリケーション・サーバーはタグを Java コードに置き換えるために使用する Java パッケージを判断します。

この時点で、DB2 Alphablox は以下のタスクを実行します。

1. DB2 Alphablox は要求オブジェクト (J2EE の API) からユーザーを取得して、Dave のユーザー・オブジェクトがすでに作成されているかどうかを確認します。これがユーザーからの最初の要求であれば、DB2 Alphablox は DB2 Alphablox ユーザーを作成します。
2. DB2 Alphablox は DB2 Alphablox リポジトリーからユーザー・プロファイルをロードし、ユーザー・インスタンスを作成します。

3. 次に DB2 Alphablox はセッション・インスタンスを作成し、応答ヘッダーに組み込まれる新しいセッション ID を割り当てます。ユーザー・オブジェクトのインスタンスがそのセッションに追加されます。
4. それから DB2 Alphablox はアプリケーション・インスタンスを作成します。
5. 次に DB2 Alphablox は要求オブジェクトからアプリケーション名を取り出し、そのアプリケーション名と一致しているアプリケーション・オブジェクトがすでに存在しているかどうかを確認します。一致しなければ、DB2 Alphablox はアプリケーション・オブジェクトを作成し、そのアプリケーションのインスタンスをセッション・インスタンスに追加します。
6. これで、ユーザー、セッション、およびアプリケーションのインスタンスが作成されました。 DB2 Alphablox はピアを作成します。
   a. `firstGrid.jsp` には、ID が `MyGridBlox` の GridBlox があります。DB2 Alphablox は、`MyGridBlox` のグリッド・ピアがすでに存在しているかどうか確認します。存在しない場合、DB2 Alphablox が作成します。
   b. グリッド・ピアは、関連のあるデータ・ピアを探します。まだ存在していない場合、DB2 Alphablox が作成します。
7. DB2 Alphablox は、レンダリングされた結果をアプリケーション・サーバーに戻します。アプリケーション・サーバーは、結果をブラウザーに戻す前に、DB2 Alphablox によって送られた出力を残りのファイルとマージします。



同じセッションで同じアプリケーションに関して同じユーザーから別の要求が来る場合、既存のピアが再使用されます。

DB2 Alphablox の役割

要約すると、DB2 Alphablox は以下のタスクを担当します。

- データのアクセスと操作
- 対話式の分析アプリケーションの構築とデプロイ
- データ・ビューの個別設定 (以下のセクションでさらに詳しく説明します)

ブックマーク、アプリケーション状態、および DB2 Alphablox Repository

ユーザー「dave」がデータ・ビューにブックマークを付ける際に、そのブックマークが名前指定されたグループに対してプライベート、パブリック、または可視のどの設定で保管されるかに応じて、プレゼンテーション Blox のインスタンスの名前 (通常は PresentBlox、GridBlox、または ChartBlox) が付けられるフォルダーの作成場所は、リポジトリー内のユーザーのフォルダー、アプリケーションのフォルダー、またはグループのフォルダーのいずれかになります。

リポジトリーに保管されている各ブックマークに関係した情報には、可視性 (プライベート、パブリック、または指定のグループ)、アプリケーション名、プレゼンテーション Blox の幅と高さ、そのビューのデータ・ソースとデータ照会、ブックマークの説明に加えて、そのビューに関係するカラー・スキーマとその他のデータ表示オプションが含まれます。

提供される API を通して、プログラマチックにブックマークの名前を指定の可視性で入手できます。さらに、ブックマークの保管、削除、名前変更、またはリストアを行うことができ、イベントを保管およびロードするブックマークを検出できます。さらに、ブックマークをプログラマチックに作成したり、ブックマーク付きで保管されているすべてのデータ照会をデータ・アウトラインでの変更を反映するように変更したりすることもできます。

さらにリポジトリーは、DB2 Alphablox 管理ページのアプリケーション定義ページで、アプリケーション状態を自動的に保管することを指定する場合に、アプリケーション状態を格納します。DB2 Alphablox は、アプリケーション内のすべての Blox (複数の PresentBlox または複数の独立した Blox がある場合) に関する情報を保管します。これには、照会の結果セット、グリッドおよびチャートの外観、およびユーザーによる他の変更が含まれます。

アプリケーション開発およびプログラミング・モデル

「*開発者用ガイド*」では、アプリケーション開発環境、JavaServer Pages、Blox Tag Library のセットアップ、およびタスク・ベースのインプリメンテーション・ステップと詳細情報について扱っています。詳細な情報について参照する前に、以下に示す概念について理解しておくと役立ちます。

Blox コンポーネント

Blox コンポーネントは、Java bean 上に構築されています。Java、スクリプトレット、または Blox カスタム JSP タグを使用して、サーバー上の Blox と Java オブジェクトにアクセスしたり、それらを制御したりするために、拡張 API が用意されています。サーバー・サイドの API により、Blox のプレゼンテーションと動作を

制御し、ビジネス・ロジックがユーザーに公開される (ブラウザーからのソース表示またはファイル保管オプションによる) のを防ぎ、開発チームのページ設計者がプログラムを複雑にするのを防ぐことができます。

JSP およびカスタム・タグ

JSP は、1 つのファイルで静的コンテンツと動的コンテンツを組み合わせるのを可能にする、J2EE のキー・テクノロジーです。デフォルトでは、アプリケーション・サーバーはその JSP エンジンだけを呼び出して、ファイルの拡張子が `.jsp` である場合に、要求を処理し、動的コンテンツを生成します。HTML ページでは、サーブレット・コンパイルおよび要求生成プロセスは行われません。アプリケーション・サーバーは、静的 HTML としてページを提示し、ブラウザーは JSP コードと Java スクリプトレットを無視します。そのため、DB2 Alphablox によって提供される機能を使用するために、Blox を JSP ページに追加する必要があります。

Blox は Java bean で作成されているため、Java bean と同じ属性を持つと想定できます。たとえば、プロパティーと、そのプロパティーの setter および getter メソッドを持っています。Blox を追加するには標準的な `<jsp:useBean>` タグを使用し、Blox プロパティーを取得または設定するには `<jsp:getProperty>` および `<jsp:setProperty>` タグを使用します。しかし、可能な場合はいつでも、DB2 Alphablox Tag Library を使用する必要があります。カスタム Blox タグを使用する場合、有効範囲は自動的に「セッション」に設定され、DB2 Alphablox はセッション管理を行い、未使用または期限の切れたリソースを自動的にクリーンアップします。

両方のアプローチについて、さらに Blox Tag Library の詳細については、51 ページの『Blox 表示タグ』を参照してください。

サーバー・サイド API とクライアント・サイド API の比較

Blox は Java Bean であるため、サーバー上の Blox とそのピアにアクセスして、情報を取得し、Blox の動作と外観を制御することができます。サーバーでの処理は、出力がクライアントに送信される前に行われます。これにより、サーバー上の他のリソースの使用およびコンポーネントの再使用が可能になり、さらに異なるブラウザー間またはブラウザーのバージョン間でたびたび起こる不整合や不一致を防ぐことができます。一般に、サーバー・サイドでは、JSP、Java スクリプトレット、またはカスタム・タグを使用して以下のことを行えます。

- Blox のインスタンスの作成
- Blox のプロパティーの動的な設定
- Blox のプロパティーの取得

データを表示する領域の選択やグリッドを表示するためのパラメーターの指定など、ユーザーが選択を行えるようにするには、サーバーに選択を伝える JavaScript 関数を呼び出さなければならない場合があります。DHTML クライアントには、サーバー上の JSP ページまたはサーバー・サイド bean を呼び出して、プロパティーを設定できるようにする直接的なクライアント・サイド API があります。そうしたクライアント・サイド API の詳細については、本書で後述します。

第 3 章 開発環境

DB2 Alphablox ソリューションはワールド・ワイド・ウェブのオープン・スタンダードに基づいており、DB2 Alphablox コンポーネントを使用した分析アプリケーションを構築するための、開発ツール用の多くのオプションが使用可能になります。

すでに快適なウェブ開発ツールをお持ちの場合は、おそらくそれらを引き続き使用して、DB2 Alphablox を用いた分析アプリケーションを開発することができます。このセクションでは、最大限の成功を期待しつつ、いくつかの重要な開発上の事項について論じます。

アプリケーション開発ツールの選択

DB2 Alphablox ソリューションは、HTML、CSS、JavaScript、その他を含むインターネットの標準的なオープン・スタンダード・テクノロジーをサポートするように、意図的に設計されています。専有の開発環境を要求することなく、DB2 Alphablox では、最もよく慣れた、あるいは快適なツールを選択することが可能です。経験を積んだ Java 開発者はすでに、IBM® Rational® Application Developer、Rational Web Developer、 Web Tools Platform プラグインを使用した Eclipse、あるいは他の IDE を快適に使用しています。Java に精通していなかったり、頻繁に Web ページを作成する場合には、お好きな HTML エディターに精通するようにしてください。 Visual SlickEdit や jEdit といった強力なテキスト・エディターを快適に使用される方もおられるでしょう。まだ理想的な開発環境を持っておられない方は、多くの選択肢を探索することができ、どれでも選択するものを使用して、DB2 Alphablox に基づく分析アプリケーションを開発できることが分かるでしょう。

Web ブラウザー

DB2 Alphablox アプリケーションは、Microsoft Internet Explorer を使用しないとサポートされません (特定の要件については、「*インストール・ガイド*」をご覧ください)。

開発者としては、開発環境でのコーディングと Web ブラウザーでのコードのテストとの間を行き来して、作業に影響しうるいくつかの共通の問題に遭遇することに気付かれるでしょう。第 1 に、開発中に認識しておくべき一般的なブラウザー考慮事項や問題がいくつかあります。第 2 に、開発の経験を強化するために、開発中に使用するテスト用ブラウザーを最適に構成したいと思われるでしょう。これらの問題については、以下のセクションで網羅します。

一般的な考慮事項

アプリケーション開発中に、いわゆる「開発モード」でブラウザーを構成するのが有利であることに気付かれるでしょう。以下のタスクでは、開発の効率を最適化するように Microsoft Internet Explorer を構成するためのステップを記載します。念頭に置いておくべき点として、最終的なアプリケーションとその動作を、ユーザーが使用するであろう Web ブラウザーと構成を使用して常にテストしてください。ほ

とんどの場合、開発モードとエンド・ユーザー・モードの構成における相違は最終的な結果に影響を与えませんが、エンド・ユーザーが遭遇すると考えられる広範囲のブラウザーと構成を用いてアプリケーションをテストするのが常にベスト・プラクティスです。

DHTML モードでの作業

アプリケーション開発中に DHTML クライアントで作業しているときに留意しておくべき、いくつかの重要な点があります。DHTML テクノロジーを使用したコーディングの方法を完全に論じることはこの部分の意図するところではありませんが、DB2 Alphablox アプリケーションで作業しているときに理解しておくべき、頻繁に遭遇する動作について若干説明しておきます。

Blox タグの変更

DHTML クライアントで作業しているときに最初に学習することの一つは、Blox タグに変更を加えても、現行のブラウズ・セッション中は有効にならないということです。つまり、Blox タグの変更後に変更内容を見るためには、ブラウザーをいったん閉じて、新しいブラウズ・セッションを開始する必要があります。これは、DHTML クライアントとサーバー・サイド・コードとの協働の結果として予期される動作です。

アプリケーション開発の際によくある別の誤りは、不注意にも複数の Blox コンポーネントを同じ `id` 属性を使って作成してしまうことです。これが生じる典型的な例は、Blox 定義タグを含むコードをコピー貼り付けして別の Blox を同じページまたは別のページに作成する際に、新しい Blox 用の `id` 属性を変更し忘れてしまう場合です。2 つの Blox が同じ `id` を持っていると、ブラウザー・メモリーにロードされる最初の Blox が 2 番目の Blox の外観を決定し、2 番目の Blox のプロパティー設定は無視されてしまいます。

Microsoft Internet Explorer による構成と開発

以下のステップは Microsoft Internet Explorer バージョン 5 以降に適用されます。

1. Microsoft Internet Explorer ブラウザーをオープンします。
2. 「ツール」メニューをクリックし、サブメニューの「インターネット オプション」を選択して、「インターネット オプション」ウィンドウをオープンします。
3. 「インターネット一時ファイル」セクションで「設定」ボタンをクリックします。
4. 「保存しているページの新しいバージョンの確認」のデフォルト設定は「自動的に確認する」です。この設定を、「ページを表示するごとに確認する」に変更します。この選択により、オープンする Web ページは、作業中のページの最新バージョンになるはずです。
5. 「OK」ボタンをクリックしてこのダイアログを閉じますが、「インターネット オプション」ダイアログ・ウィンドウは閉じないでください。
6. 今度は、「インターネット オプション」の「詳細設定」タブを選択します。長いスクロール可能なチェック・ボックスのリストが表示されます。以下のセクシ

ョンで、この長いオプション・ウィンドウの別個の部分を網羅します。 下記の設定はオプショナルですが、Web ページのトラブルシューティングを強化するために推奨されています。

JavaScriptエラー通知

7. [オプショナル] JavaScript エラーを認識するのを助けるため、「詳細設定」オプションの「ブラウズ」セクションで、「スクリプト エラーごとに通知を表示する」をチェックするようお勧めします。そうすれば、JavaScript エラーが発生したことを示すダイアログ・ボックスがポップアップされ、エラーを見逃すことがありません。これを使用可能にしないと、状況ウィンドウの左下隅に表示される JavaScript アラート・メッセージに常に気を配り、それに気付く必要が生じます。

ヒント: DB2 Alphablox アプリケーションのページを Microsoft Internet Explorer 内で見ているとき、表示しようとするページをブラウザーは期待通りに処理しないかもしれません。時々ブラウザーは、新たに更新されたページではなく、キャッシュ・ページを再表示することがあります。上記のように設定すれば、それを避けられますが、必ずしも頼りにはなりません。ブラウザーの再ロード (更新) ボタンをクリックしたときでさえ、表示されるページが引き続きキャッシュ・ページである場合もあります。このことが問題と思われる場合は、いくつかの別のオプションがあります。第 1 に、ページのハード・リフレッシュ (キャッシュ・コピーではなくサーバーからフレッシュ・コピーを得ること) を強制することができます。これは、Ctrl リフレッシュ技法を使用して行います。すなわち、Ctrl キーを押したまま、最新表示ボタンをクリックします。 第 2 のオプションは、ブラウザーをいったん閉じてから再オープンすることです。その結果、新しいブラウザー・セッションとなるので、表示されるページが最新のページになることが保証される最も確実な方法です。

Web ブラウザー - Mozilla に関する既知の問題

Microsoft Internet Explorer とは異なる、Mozilla に関する既知の問題を取り上げます。

ご自分の組織でサポートされている Web ブラウザーを使用して、アプリケーションをテストするのがベスト・プラクティスです。以下の表では、サポートされている Microsoft Internet Explorer ブラウザーの場合とは異なる、 Blox UI コンポーネントに関する Mozilla および Mozilla Firefox Web ブラウザーの既知の動作について特に取り上げています。

表 1. Mozilla に関する注目すべき問題

問題	Mozilla に関する注目すべき問題
スプリッター・コンテナーにおけるチャートのサイズ変更	チャートのサイズは同じままで、その後再描画されます。 Firefox はチャートのアスペクト比を保持して、コンテナー・サイズを変更します。
コピーの編集 (Blox Model API および UI フィーチャー)	サポートされていません。 Gecko エンジンでは、クリップボードにコピーする方法はありません。

表 1. Mozilla に関する注目すべき問題 (続き)

エディット・フィールド選択とキャレット位置	サポートされていません。 Gecko エンジンで使用される window.getSelection() メソッドは、エディット・フィールドにテキストを戻しません。これは既知の制限で、今後のリリースで修正される可能性があります。
ComboBox オートコンプリート強調表示	サポートされていません。作業を完了しても、完成したテキストを強調表示する方法はありません。
ツールチップでの改行	サポートされていません。ツールチップに改行を追加する方法はありません。改行文字には、固有文字が表示されます。 Firefox では DHTML ビュー・コードが、これらをスペースに置換します。
ポップアップ・メニュー、右クリック・メニュー、ツールバー・ドロップダウン	フレームに限定されています。ポップアップ・メニューおよび右クリック・メニューは Firefox のフレーム内になければならず、収まるだけのメニュー数に自動調整されます。
ドラッグ・アンド・ドロップの違い	Firefox では、ドラッグしてもカーソルが変わることはなく、ドロップできません。フィールドの選択は、ドラッグではできません。ユーザーは、ドラッグ可能なエディット・フィールドで、カーソルを使用してテキストを選択することはできません。
サイズ変更可能なダイアログ	枠がありません。 Firefox では、サイズ変更可能なダイアログで枠が除去されます。主要な問題は、Firefox で使用できないパーセンテージ・サイズのダイアログ・コンテンツで余白が使用されている (コンテンツが常に大きすぎる) ことに起因します。
静的テキスト・コンポーネントのサイズ変更	静的コンポーネントに設定されたサイズ変更は無視されます。この問題に対処するには、2 つのブラウザーで互換性を持つ親コンポーネントでサイズを設定できます。
キーボード・サポート (アクセラレーター)	サポートされていません。 `tabindex` は `div` エレメントでは設定できないため、Firefox ではサポートされていません。 Mozilla Firefox 1.8 では、サポートされる予定です。
アクセシビリティ	サポートされていません。
右から左 (RTL) グリッド表示	サポートされていません。
Grid.setFixedScrollbarPosition(boolean) および Grid.isFixedScrollbarPosition()	サポートされていません。

Application Studio

Application Studio には、学習および開発の目的で役立つ例とその他のツールが用意されています。Application Studio は、DB2 Alphablox ホーム・ページの「アセンブリー (Assembly)」タブからアクセスできます。ご自分のアプリケーション用にサンプル・コードを調査したり再利用する場合、それらのファイルは次の場所にある Application Studio ディレクトリーの下にあります。

```
<db2alphablox_dir>¥system¥ApplicationStudio¥Examples
```

本書の随所で参照されている Blox Sampler 例のセットは、 本書で論じられている技法の多くを例示しており、`Examples` ディレクトリーにあります。

第 4 章 設計上の考慮事項

アプリケーション開発に関して、設計および開発を進める前に、まず要件を明確に識別し、それからアプリケーションの成功度を評価する必要があります。 このトピックでは、一般的な要件の収集ガイドラインをいくつか記載しています。それらはユーザーのニーズや、事前に考慮に入れるべき他の問題を見極める助けとなるでしょう。

アプリケーション要件の定義

アプリケーション設計のゴールは、特定の対象ユーザーの特定の必要にかなうように、アプリケーションが適切な情報と機能性を確実に提供するようになることです。要件を収集する際には、調査すべき 3 つの分野があります。すなわち、データ、ユーザー・インターフェース、およびアプリケーション・ロジックです。

- 『データ要件』
- 26 ページの『ユーザー・インターフェース要件』
- 28 ページの『アプリケーション・ロジックの要件』

データ要件

DB2 Alphablox がサポートするアプリケーションの特定クラスは、OLAP (online analytical processing) です。OLTP (online transactional processing) アプリケーションがトランザクション・データ・ソース内でデータを生成およびアクセスするのに対し、OLAP アプリケーションはデータ・ソース内のデータにアクセスします。これらのデータ・ソースには普通、トランザクション詳細から統合されてマルチディメンション・アーキテクチャーに格納されるデータが入っています。

アプリケーション設計プロセスの一部として、必要なデータを識別することと、そのデータへのアクセスを取り巻くセキュリティー問題を識別することが挙げられます。以下の質問に答えることによって、アプリケーション・データ要件を定義する助けが得られます。

1. ユーザーはどんな情報を期待し、必要としていますか?

   この質問にできるだけ正確に答えることは、情報を位置決めし、それに対して効果的な照会を定義するための第一歩となります。 必要な情報にユーザーがすでにアクセスできるかどうか、アクセスの使用可能化に関係するセキュリティー問題があるかどうか、などの事項を見極めることは重要です。例えば、地域の販売管理者はすべての地域のデータを見られるようにしますが、地域の営業担当員は自分の地域のデータだけを見られるようにすればよいでしょう。

2. 情報はどこにありますか?

   DB2 Alphablox アプリケーションは、さまざまな種類のマルチディメンション・データベースとリレーショナル・データベースからのデータにアクセスすることができます。アプリケーション開発を始める前に、どのデータ・ソースにアクセスする必要があるかを考慮に入れ、DB2 Alphablox が要件をサポートすることを

検証してください。 (特定のデータ・ソースのサポートについての詳細は、「*インストール・ガイド*」を参照してください。) 多くの分析アプリケーションは、ディメンションとメンバーの階層に情報を編成するマルチディメンション・データ・ソースに依存しています。Blox は特にこのようなデータ階層を活用するように設計されています。そのユーザー・インターフェースにより、階層内をドリルアップおよびドリルダウンしたり、ディメンションおよびメンバーに基づいてデータをフィルター操作したり、 1 つ以上のディメンションを別の軸に移動するなどのことができます。

DB2 Alphablox はさらに、情報を行と列の形式に編成したリレーショナル・データ・ソースもサポートします。リレーショナル・データの一つの使用法は「詳細へのドリル」で、これによりユーザーは、統合された情報のマルチディメンション・データ・ソースから、リレーショナル・データ・ソース内の基礎となる詳細に移動することができます。

DB2 Alphablox の DB2 Alphablox Cube Server コンポーネントにより管理者は、リレーショナル・データ・ソースにある情報からマルチディメンション・データ・キューブを作成することができます。DB2 Alphablox Cube Server は、フル装備の OLAP データ・ソースのスケーラビリティーとオーバーヘッドとを必要としないアプリケーションに有用です。DB2 Alphablox Cube Server にはディメンション・メタデータが含まれているので、ユーザーは、ドリル、ピボット、フィルターなどの操作を実行することができます。リレーショナル・データをマルチディメンション・データにトランスフォームする方法については、「*DB2 Alphablox Cube Server 管理者用ガイド*」を参照してください。

DB2 Alphablox アプリケーションにデータが表示される際、基礎となるデータ・フォーマットはユーザーには見えないことに注意してください。ただし、データがリレーショナル形式の場合は、マルチディメンション形式を必要とするユーザー処置 (ドリルなど) は使用不可になります。

DB2 Alphablox を起点として、ReportBlox を使用してリレーショナル・データ・ソースから対話式レポートを開発することができます。それによりユーザーは、標準装備の Report Editor ユーザー・インターフェースを通して、ブレーク・グループを追加し、列を再配列し、データをソートし、列を名前変更し、セルおよびヘッダーのスタイルを編集することができます。ReportBlox とそのサポート Blox については、「*Relational Reporting 開発者用ガイド*」を参照してください。

ユーザー・インターフェース要件

ユーザー・インターフェースはアプリケーションの使いやすさの鍵です。アプリケーションはコンテンツ・プレゼンテーション、アプリケーション・ナビゲーション、およびユーザー支援を備えている必要があります。効果的なユーザー・インターフェースおよび Web ページ設計についての包括的な論議は本書の範囲を越えていますが、このセクションでは以下の分野のいくらかのガイドラインを提供します。

- 27 ページの『ユーザー・グループ』
- 27 ページの『コンテンツ・プレゼンテーション』
- 27 ページの『ユーザー指示』

- 28 ページの『ユーザー・ナビゲーション』
- 28 ページの『データ操作』
- 28 ページの『保管および復元作業』

ユーザー・グループ

ユーザー・グループの要件を定義する際には、以下の考慮事項に留意してください。

- 多くの場合、ユーザーにはそれぞれ異なる使いやすさの要件があります。そうした差異に対処するための一つの方法は、ユーザーを大きなグループの中の小さなグループに入れることです。例えば、Financial (財務) ユーザー・グループの中に Analyst (分析者) グループと Administrative (管理) グループを含めることができます。ユーザーが所属するグループに基づいて、アプリケーションのインターフェースを動的に変更することができます。
- さらにユーザーには、それぞれ異なるデータ・アクセス要件があります。多くの場合、データ・ソースのセキュリティー機構自体が、ユーザー・レベルおよびグループ・レベルのアクセス制限をサポートします。データへの容易なユーザー・アクセスを確保するために、DB2 Alphablox ユーザー・グループは、データ・ソース上でインプリメントされるものと並行して設けるべきです。

コンテンツ・プレゼンテーション

DB2 Alphablox では、コンテンツ・プレゼンテーションを開発者が制御でき、ユーザーでも多少制御できるようになっています。開発者には、情報の編成と提示に関してかなりの自由があります。 アプリケーション・ページの外観は、経営者のダッシュボードのようにもでき、イントラネット・ポータル、あるいは印刷可能レポートなどとすることもできます。

開発者はデータ・プレゼンテーションを選択することもできます。適切な Blox を選択し、それらの Blox 上でプロパティー値を設定することにより、データをグリッド、チャート、あるいはグリッド/チャートの組み合わせとして表示するように選択できます。 DB2 Alphablox はいろいろなチャート・タイプを提供しているので、最も効果的なデータ・プレゼンテーションで実験することができます。

適切な場合には、チャート・タイプを変更したり、グリッドおよびチャート・プレゼンテーションを切り替えるなどのことをユーザーに許可することができます。例えば、一部のユーザーはパーセンテージと傾向がすぐに分かる円グラフを好みますが、数値を記載して複雑な分析をサポートするグリッドを好む人もいます。適切で効果的なコンテンツ・プレゼンテーションの設計を容易にするために、アプリケーションの対象ユーザーをはっきりと理解してください。

ユーザー指示

DB2 Alphablox では、各 Blox の使用に際しての包括的な指示を含むオンライン・ヘルプを提供しています。DB2 Alphablox ヘルプ・ページにアクセスするためのデフォルトの手段は、ツールバーで疑問符 (?) をクリックするか、メニュー・バーで「ヘルプ」>「ヘルプ...」メニューをクリックすることです。しかし、ユーザーの学習曲線を緩やかにするためには、各アプリケーション・ページにアプリケーション・レベルのユーザー指示を提供することが助けになります。

この方法が適切でない場合や、ユーザー指示が極めて高度な場合は、DB2 Alphablox オンライン・ヘルプを編集して拡張することができます。詳しくは、 295 ページの『第 24 章 ユーザー・ヘルプの追加』を参照してください。

ユーザー・ナビゲーション

最も単純なアプリケーションは、単一の JSP ページと 1 つ以上の Blox から成ります。しかし、アプリケーションが、ユーザー対話を伴ういくつかの Blox を呼び出す場合には、2 つの状況が起こり得ます。複数の Blox が単一ページにある場合、ある Blox がブラウザー・ウィンドウに現れると、別の Blox はスクロールアウトされることがあります。ですから、ページ内にリンクを張って、領域間を素早く移動できるようにすることを考慮してください。

さらに一般的なのは、いくつかの JSP ページで 1 つのアプリケーションを構成している場合です。このアプリケーション設計では、ページ間を前後に移動するリンクが必要です。アプリケーションの「ホーム・ページ」は、アプリケーション内の他のすべてのページにリンクし、アプリケーションのフィーチャーと機能拡張についてユーザーに通知するための適切な場を提供します。

データ操作

十分な対話式分析と「仮定の」シナリオを備えたアプリケーションから、素早い管理スナップショットのための静的なプレゼンテーションに至るまで、広範なアプリケーションを製作することができます。実際、Blox の対話性とツールバーを使用可能化/使用不可化するだけで、複数のユーザー・ニーズに応える単一アプリケーションを開発することができます。アプリケーション設計の成功には、対象ユーザーがデータとどのように対話するかに関する知識が欠かせません。

保管および復元作業

複雑な分析を実行するユーザーはしばしば、ある時点で作業を保管したり、データの特定のビューを他のユーザーが利用できるようにしておきたいと思います。DB2 Alphablox は、ツールバー・ボタンをとおして、これらの要求をサポートします。

アプリケーション・ロジックの要件

別の主要な設計領域はアプリケーション・ロジックです。Blox がデータ・アクセス、プレゼンテーション、および操作を提供する一方、多くの DB2 Alphablox アプリケーションが、以下のような特定のユーザー・ニーズに応えるロジックを提供します。

- ユーザーがそこから選択を行う、定義済み照会のリストを提供する。
- 一連の関連した選択を通してユーザーが動的に照会を構成できるようにする。
- ユーザー・ログインに基づいて、初期照会、配信フォーマット、およびアプリケーションの外観を設定する。
- ユーザーが入力した値に基づいて、例外データを強調表示する。
- ユーザー処置に基づいて、コンテンツ・プレゼンテーションを切り替える。
- 「仮定の」シナリオを実行し、オプションで結果をデータ・ソースに書き戻す。

JSP、HTML フォーム、JavaScript 関数、Java、および DB2 Alphablox カスタム・プロパティーを組み合わせて使用し、アプリケーション・ロジックをインプリメントすることができます。

カスタム・プロパティー

DB2 Alphablox 管理ページを通して、アプリケーション、データ・ソース、およびユーザーから使用可能なカスタム・プロパティーを定義することができます。カスタム・プロパティーを定義したら、RepositoryBlox のサーバー・サイド Java コードを介してそれを使用することができます。詳しくは、255 ページの『カスタム・ユーザー・プロパティーの作成』を参照してください。

注: ポートレットは、ユーザー・プロファイル情報へのアクセスを、ポータルのインフラストラクチャーに依存しています。ポータルのユーザー・プロファイルと DB2 Alphablox ユーザー情報は別々に保持されていることに留意してください。

ポートレット開発の計画

ポータル・アプリケーションに Blox コンポーネントを使用する場合、計画段階で以下の設計上の要件について考慮する必要があります。Blox が有効になっているポートレットを他のポートレットと同じページに置くことが必要なため、ページ上のすべてのポートレットがみな適正に動作するようにするには、いくつかの注意すべき点があります。

- Blox コンポーネントは、Internet Explorer v5.5 以上で動作します。この要件がポータル・サーバーのサービスを受ける他のポートレットにも等しく適用されるようにしてください。「*インストール・ガイド*」で、固有のブラウザー要件を確認してください。
- Blox コンポーネントを含むアプリケーションを使用する場合は、キャッシングをオフにします。 Blox コンポーネントの強力な対話式ユーザー・インターフェースとライブ・データのサポートを利用するには、サーバーとの通信が必要です。キャッシングがオンになっていると、これらの機能がうまく働きません。そのため、同じページ上のポートレットはすべてキャッシュなしの環境で実行する必要があります。
- 上記の理由のため、Blox が有効になっているポートレットはオフライン・モードでは動作しません。
- Blox コンポーネントによって使用されるデータ・ソースは、DB2 Alphablox 管理ページから DB2 Alphablox に定義しておく必要があります。DB2 Alphablox 管理権限を持つユーザーを、ポータル管理から別個にセットアップする必要があります。データ・ソースの準備を行うには、DB2 Alphablox 管理者およびデータベース管理者との協力が必要です。

アクセス可能アプリケーションの設計

身体障害を持つユーザーのため、すべてのアクションに対してキーボードの同等機能を提供するのはとても重要なことです。視覚に制約を持つユーザーのため、テキスト・ブラウザーおよびスクリーン・リーダーの限界について考慮に入れる必要もあります。DB2 Alphablox の UI コンポーネントでは、標準装備のキーボード・ショートカットおよびアクセラレーターにより Internet Explorer のアクセシビリティをサポートしています。

分析アプリケーションを開発しカスタマイズする際、アクセシビリティをサポートするため留意する必要のあるいくらかの点を記します。

- カスタム・メニュー・オプションにアクセラレーター・キーを作成してください。メニュー・バーに表示されるカスタム・メニュー項目では、独自のアクセラレーター・アクセス・キーを指定してください。これを行うには、<bloxui:menu> および <bloxui:menuItem> タグに accesskey タグ属性を設定します。詳細については、「*開発者用リファレンス*」の『Blox UI タグ・リファレンス』を参照してください。
- チャート・コンポーネントの使用は、使用不可にするか制限してください。非グラフィカル・ブラウザーおよびスクリーン・リーダーは、視覚障害を持つユーザーにイメージを見せることはできません。グラフィカルな性質を持つチャート・コンポーネントは、キーボードではアクセスできません。視覚に制約を持つユーザーのためには、PresentBlox (chartAvailable="false") からチャート・コンポーネントを除去するか、グリッド・コンポーネントのみを使用することをお勧めします。
- ハイ・コントラスト・テーマの形で Blox をレンダリングします。ハイ・コントラスト・テーマを使用してレンダリングした Blox ユーザー・インターフェースでは、色の使用で灰色の陰影を付けるだけでなく、ブラウザーで設定される好みのフォント・サイズ表示が尊重されます。ハイ・コントラスト・テーマを使用するには、theme URL 属性を highcontrast に設定します。例:
  http://*server*/*application*/*file.jsp*?theme=highcontrast
- 視覚に障害を持つユーザーのため、メニュー・バーやツールバーを介するタブを使用せずに、データ内容を簡単に取得できるようにします。
  <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強化するために設計されています。ユーザーがメニュー・バーまたはツールバーをスキップして、データをすぐに取得できるようにするタグ属性が提供されています。さらに、フォーカスが失われているダイアログを開くためのキーボード・アクセスも提供されています。詳細については、「*開発者用リファレンス*」の『Blox UI タグ・リファレンス』を参照してください。

通常、アクセス可能アプリケーションの設計には、ユーザー・プロファイルによる個別設定が関係しています。そのための 1 つの方法は、DB2 Alphablox 内のカスタム・ユーザー・プロパティーを使用することです。たとえば、DB2 Alphablox 管理ページを介して accessible という名前のカスタム・ユーザー・プロパティーを指定できます。視覚障害を持つユーザーの場合、このプロパティーを「はい (Yes)」などの指定値に割り当てます。 RepositoryBlox getUserProperty() メソッドを使用してこの値にアクセスし、PresentBlox 内のチャート・コンポーネントをプログラマチックに非表示にできます。

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

<%
   String accessible = myRepository.getUserProperty("accessible");
   if (accessible.equals("Yes")) {
      myPresent.setChartAvailable(false);
   } else {
      myPresent.setChartAvailable(true);
   }
%>

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

一般的なアプリケーション・アクセシビリティに関する問題の詳細については、IBM アクセシビリティ・センターにアクセスしてください。このサイトでは、アクセス可能アプリケーションの習得および開発に役立つ開発者リソースおよびガイドラインが数多く提供されています。

複数ロケールの設計

Blox ユーザー・インターフェースとオンライン・ヘルプが表示される言語は、ブラウザーの言語設定によって決まります。要求を受け取ると、DB2 Alphablox は要求の `Accept-Language` ヘッダーに基づいて、クライアントのロケールを判別します。ロケールとは 2 文字の言語コードで、その後に下線 (″_″) で区切られて 2 文字の国別コードが続く場合もあります。DB2 Alphablox はブラウザーの言語セットのリストを調べ、最初の言語セットがサポートされているロケールと完全に一致するものではない場合は、サポートされているロケールが見つかるまでリスト内の調査を継続します。サポートされているロケールがない場合、DB2 Alphablox はリストの最初の言語を使用し、関連する言語で表示します。関連する言語がサポートされていないか、または何らかの理由でブラウザーに言語が設定されていない場合には、DB2 Alphablox はサーバーのロケールに基づいてロケールを判別します。これらすべてを試してうまくいかないときは、DB2 Alphablox はデフォルトで英語に設定されます。

たとえば、ブラウザーの言語設定で `fr_CA` が第 1 言語、`en` が第 2 言語となっている場合、`fr_CA` はサポートされているロケールと完全に一致するロケールではないため、リストの 2 番目のロケールが調べられます。`en` はサポートされているロケールなので、Blox ユーザー・インターフェースは英語で表示されます。リストに言語が `fr_CA` しかない場合には、DB2 Alphablox は関連する言語つまり `fr` を探します。そして、`fr` はサポートされているので、Blox ユーザー・インターフェースはフランス語で表示されます。

ブラウザーでの言語設定

言語設定を指定するには、次のようにします。

- Internet Explorer の場合、「**ツール**」 → 「**インターネット オプション...**」を選択し、「**言語...**」ボタンをクリックします。
- Firefox または Mozilla ブラウザーの場合は、「**ツール**」 → 「**オプション...**」を選択し、「**言語...**」ボタンをクリックします。

オープンしたウィンドウで、ユーザーは言語を追加したり、言語の優先順位を変更することができます。

Internet Explorer で、ユーザーがリストからすべての言語を誤って削除すると、ブラウザーの既知の XMLHTTP 問題のために、問題が生じることがあります。

Blox UI のロケールとフォーマット設定のロケールの関係

前述したように、DB2 Alphablox はブラウザーに設定されているロケールのリストを調べて、Blox ユーザー・インターフェースの言語を判別します。しかし、データのフォーマット設定は、Java フォーマットに基づいているので、DB2 Alphablox によってサポートされるロケールに制限されることはありません。数値のフォーマット設定に使用する設定を判別する際に、DB2 Alphablox は常にロケール・リストの最初の言語を使用します。結果として、Blox ユーザー・インターフェースが表示されるロケールがフォーマット設定のロケールと異なる場合もあり得ます。

チャート・ラベルのフォント・マッピング

チャートのユーザー・インターフェースが複数のロケールをサポートしている場合に、サーバーのロケールがクライアントのロケールと異なると、サーバーはクライアントのロケールで絵文字を表示できない場合があります。ユーザーがこの問題に直面する可能性がある場合、チャート軸ラベルと凡例用にマップされるシステム・フォントを指定することができます。<alphablox_dir>/repository/serversの下にあるchartfonts.xmlファイルを使用して、Java が認識する 5 つの論理フォント名 (Serif、SansSerif、 Dialog、DialogInput、および Monospaced) を物理フォント名 (Courier、 Arial、Times New Romans など) にマップすることができます。

各ロケールについて、言語と、各システム・フォントのフォント名を指定してください。以下は中国語のフォント・マッピングの例を示しています。

```
<locale language="ZH">
  <Serif>MingLiu</Serif>
  <SansSerif>SimHei</SansSerif>
  <Monospaced>SimHei</Monospaced>
  <Dialog>SimHei</Dialog>
  <DialogInput>SimHei</DialogInput>
</locale>
```

必要に応じて、ロケールごとに、国と異体も指定できます。ブラジル・ポルトガル語の場合の例を以下に示します。

```
<locale language="PT" country="BR">
...
</locale>
```

論理フォントごとに、style属性を使用してスタイルも指定できます。有効な値は、plain、bold、bolditalic、および italic です。データ・タイプ定義 (DTD) ファイルchartfonts.dtdは、<alphablox_dir>/xmlディレクトリーの下に置かれています。

フォント名は大/小文字が区別されます。 chartfont.xmlで指定されているフォントがクライアントで見つからない場合、チャートは基本 JVM フォント・マッピングを使用してレンダリングされます。

設計に関する情報と開発上のヒント

留意しておくべき全般的な Blox の動作、サーバーの動作、およびアプリケーション開発上のヒントを以下に示します。

- BloxContext には、要求したロケール設定をブラウザーから検索する getLocales() メソッドがあります。
- Blox にはセッション有効範囲があるので、セッションを確立してから言語設定を変更しても、新しいセッションを開始するまでその設定は有効になりません。新しいロケールが反映された Blox ユーザー・インターフェースを表示するには、新しいブラウザー・ウィンドウをオープンする必要があります。
- DB2 Alphablox には、Java ResourceBundle クラス用のラッパー・タグのセットが用意されています。このタグを使用することにより、カスタム・コードのロケール固有リソースにアクセスすることができます。このタグについては、リソース・バンドル関連タグで説明されています。実際の使用例については、FastForward アプリケーションを参照してください。このサンプル・アプリケーションでは、ラッパー・タグを使用して、WEB-INF/classes/fastforward/FastForwardBundle.properties ファイル内のリソース・バンドルにアクセスします。
- JSP コードにチャート・タイプを指定する場合は、必ず英語のストリングまたはチャート・タイプの数値を使用してください。
- ブックマークの名前と説明は、常に保存時の言語で表示されます。フランス語で保存されたブックマークの名前と説明は、ブックマークをロードするときのブラウザーの言語設定に関わりなく、常にフランス語で表示されます。
- DB2 Alphablox によって生成されるエラー・メッセージはクライアントのロケールに基づいて表示されますが、データベースからの例外およびエラー・メッセージはデータベース・サーバーと同じロケールで表示されます。DB2 Alphablox はデータベースからのその情報を引き渡すだけです。
- サーバー・ログは常にサーバーのロケールに基づきます。DB2 Alphablox がドイツ語のマシンにインストールされている場合には、サーバー・ログはドイツ語になります。
- DB2 Alphablox Cube Server はサーバーのロケールで作動し、すべてのエラー・メッセージとログ・メッセージはサーバーのロケールに基づきます。あるキューブのサーバーおよびリレーショナル・データ・ソースが異なるロケールで稼働している場合は、照会の結果にソートの面で不整合が生じることがあります。MDX照会の処理中に、ソートの処理が一部は DB2 Alphablox Cube Server で、一部がリレーショナル・データ・ソースで実行されることがあるために、そのようになります。

双方向言語用の設計

双方向 (BiDi とも呼ばれる) 言語は、数字は左から右に読むものの、言葉は右から左に読む言語です。デフォルトでは、Web ブラウザーは左から右に HTML ページのコードを解釈してビジュアル・コンポーネントを表示します。アラビア語やヘブライ語などの双方向言語の場合、ブラウザーまたはユーザーの Windows® システムの設定によっては、表示コンポーネントを右から左に自動的に表示できないこともあります。Internet Explorer ユーザーは、「**表示**」>「**エンコード**」メニュー・オプションから表示方向を設定できますが、すべてのブラウザー・バージョンでこのオプションが提供されているわけではありません。また Web ページ設計者は、`dir` 属性を `<body>` タグに追加して、値を `rtl` に設定すると、方向を指定できます。

Blox コンポーネントが表示される言語と方向は、ブラウザーのロケール設定によって決まります。ChartBlox の場合、左から右方向というのは X 軸ラベルがチャートの左側に表示されるということです。右から左方向の場合、X 軸ラベルは右に配置されます。GridBlox では、左から右方向というのは行ヘッダーがデータ・セルの左側に表示されるということです。右から左方向の場合、行ヘッダーは右に配置されます。

アプリケーションが双方向言語をサポートするために、追加のコードは必要ありません。設計目標によって、Blox コンポーネントを右から左に表示することが必要な場合には、以下の手法でそのようにすることができます。

HTML コードにおける dir 属性の設定

Blox ユーザー・インターフェースはダイナミック HTML にレンダリングされるので、ブラウザーの機能を活用して、HTML コードで指定された方向に解釈することができます。この `dir` 属性は、`<body>` タグ (`<body dir="rtl">`) または内部の `<div>` タグ (`<div dir="rtl">`) に設定できます。この方法によってブラウザーには Blox コンポーネントを右から左に表示するよう指示が出されますが、引き続きユーザーは提供される「**グリッド・オプション (Grid Options)**」および「**チャート・オプション (Chart Options)**」ダイアログを使用して表示方向を変更することができます。これら 2 つのダイアログにより、アプリケーションのユーザーは左から右の表示に戻すこともできます。

動的に方向を変更する必要がある場合、1 つの方法として、パラメーターのある JSP を渡します。そして、そのパラメーター用に渡された値に基づいて、出力の方向を設定します。たとえば、`test.jsp` ファイルを持っている場合には、以下のようなパラメーターで呼び出すことができます。

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

`test.jsp` ファイルはパラメーター値を取得し、`<body>` タグに方向を設定します。

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

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

<html>
```

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

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

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

UI コンポーネント内の方向の設定

Blox UI モデル内の可視コンポーネントすべての Component 基底クラスには、setDirection() メソッドがあります。デフォルトは DIRECTION_DEFAULT で、HTML コードまたはブラウザー内の方向設定を優先します。コンポーネントの方向を DIRECTION_RTL に設定すると、HTML コードで指定された言語、ブラウザー設定、または方向にかかわらず、サブコンポーネントも必ず右から左に書き込まれます。引き続きユーザーは、提供される「**グリッド・オプション (Grid Options)**」および「**チャート・オプション (Chart Options)**」ダイアログを使用して、表示方向を変更できます。

第 5 章 JavaServer Pages および Blox Tag Library の使用

JavaServer Pages テクノロジーを DB2 Alphablox アプリケーションで使用すると、開発者は Web ベースの分析アプリケーションを素早く作成して簡単に保守することができます。DHTML テクノロジー (HTML、JavaScript、および CSS を含む) を使用して開発することに加えて、JSP テクノロジーは Java を習得しなくても Java の機能を使用できる動的スクリプト要素を追加します。このトピックでは、JavaServer Pages テクノロジーが DB2 Alphablox アプリケーション内でどのように使用されるかについて説明します。

JavaServer Pages テクノロジー

JavaServer Pages (JSP) テクノロジーによって、HTML、CSS、JavaScript などのよく知られた DHTML テクノロジー、および開発者が Java とサーバー・サイド処理を使用することを可能にする動的スクリプト要素を使用して、開発者は Web ベースの分析アプリケーションを素早く作成して簡単に保守することができます。さらにこのテクノロジーは、各ブラウザーの特異性が持つぜい弱さの影響を受けにくいアプリケーションを開発者が作成するためにも役立ちます。要約すると、JSP テクノロジーの主な利点は以下のとおりです。

- コンテンツの生成と表示との分離

  JSP テクノロジーを使用すると、Web ページ開発者は HTML または XML タグを使用して Web アプリケーション・ページを設計およびフォーマットできます。 JSP タグおよびスクリプトレットにより、Web ページ開発者は、中核となるプログラム・ロジックがカスタム・タグ・ライブラリーおよび Java Bean の中に隠れている、よく知られたタグ構文およびスクリプト機能を使用してページを生成できます。上級の Java 開発者は Java を使用して、 Web ページ設計者およびアプリケーション開発者が使用できる再使用可能なコンポーネントを作成できます。

- 再使用可能なコンポーネントの強調

  ほとんどの JSP ページは、Java Bean やサーブレットなどの、クロスプラットフォームで再使用可能なコンポーネントの使用に依存しています。JSP を使用すると、 Web ページの設計者および開発者が Java Bean およびサーブレット・コンポーネントを使用してコンテンツを生成することが容易になります。たとえば、Blox はサーバー・ピアと対話する Java Bean ですが、開発者は簡単なタグを使用してこれらの Bean を定義できます。

- タグによる Web 開発の単純化

  JSP テクノロジーによって、使いやすい JSP 特定の XML タグ内に機能の多くをカプセル化することにより、動的なコンテンツ生成が可能になります。これら標準の JSP タグを使用して、JavaBeans™ コンポーネントとの対話、 Bean 属性の設定と取得、および他の方法ではコーディングがより困難で時間のかかるその他の機能の実行を行うことができます。JSP カスタム・タグ・ライブラリーを使

用して、DB2 Alphablox などが作成できるタグは、Web ページ設計者および開発者が活用できるもので、使いやすい上に、考慮する必要のない複雑な点は表に出ないようになっています。

「*開発者用ガイド*」は、 JavaServer Pages テクノロジーに関する基本的な知識を前提にしていますが、その知識がない場合でもいくつかの基本的な DB2 Alphablox アプリケーションを作成可能です。このトピックの残りの部分では、JSP を DB2 Alphablox と共に使用する方法について主に説明します。

JavaServer Pages テクノロジーの詳細を学習するには、 Alphablox が推奨する以下の資料および Web サイトを参照してください。

推奨される資料

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

熟練した開発者にならなくても JavaServer Pages およびアプリケーション開発について理解できる、優れた解説書。Web ページ設計者および開発者を対象にした最初の部分では、 JSP の概念について、および JSP が Web アプリケーション開発にどのように適合するかについて説明します。プログラミングに注目した後半の部分では、JSP コンポーネントおよびカスタム JSP タグの作成方法について説明しています。

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

JavaServer Pages に関するもう一つの優れた解説書。Web ページ設計者と Java 開発者の両方を対象にしています。JSP 1.2 および Servlet 2.3 仕様を使用する方法についての説明、および共通 Web アプリケーション・タスクの例が含まれています。

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

JavaServer Pages を紹介するこの資料では、プログラミングの経験があることを前提としていません。ただし、以前に HTML の経験があることは前提となります。Web ベース・アプリケーションの作成方法を紹介するとともに、関連する JSP および Java の概念についても説明します。第 3 章までで、読者は簡単な Java Bean を作成できるようになります。

Web サイト

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

Sun は JavaServer Pages テクノロジーを開発しました。このサイトでは、ニュース、仕様、ソフトウェア、およびチュートリアルを含む、最新情報が提供されます。Technical Resources セクションでは、構文のクイック・リファレンス用カードおよびガイドの PDF 版を入手できます。

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

JavaBeans 仕様、チュートリアル、および JavaBeans テクノロジーについての最新のニュースがあります。

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

ニュース、仕様、およびチュートリアルを含む最新のサーブレット・テクノロジーに関する Sun のサイト。

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

JSP の記事、解説書、および他の JSP リソースへのリンクのための優れたソース。*JSP Buzz* ニュースレターは、ニュースと記事を提供しており、 JSP 製品および機能についての最新情報を入手するための優れた手段となります。

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

このサイトには、JSP Web アプリケーション開発に関連したトピックについての多数の記事があります。さらに、JSP およびサーブレットに関する役に立つ FAQ もあります。

JavaServer Pages を DB2 Alphablox と共に使用する

JavaServer Pages テクノロジーおよび DB2 Alphablox を使用すると、洗練された分析アプリケーションを素早く作成して、容易に保守できるようになります。JSP はサーバー・ベースのテクノロジーですが、HTML、JavaScript、および Cascading Style Sheets を含む標準のクライアント・サイド・テクノロジーを組み込むことが可能です。これにより、DB2 Alphablox 開発者はこれらのテクノロジーを使用して、柔軟で拡張可能なアプリケーションを作成できます。

分析アプリケーションは通常、クライアント・サイドの技法とサーバー・サイドの技法の両方を採用しながら、両方のテクノロジーの最良部分を使用してアプリケーションを提供します。 Blox Client API およびサーバー・サイド Java API を含む Blox API の全体について詳しくは、「*開発者用ガイド*」を参照してください。

プレゼンテーション Blox を含む Blox は、最も多くの場合、 Blox Tag Library を使用して JSP ファイル内で定義されます。JavaServer Pages テクノロジーを使用して開発した Blox Tag Library には、 Blox およびそのプロパティーを指定するための使いやすいタグが含まれます。その他の Blox タグを使用して、アプリケーションおよびビジネス・ロジック・デバッグを含む共通の開発者タスクを取り扱うことができます。標準の JSP アクション (`jsp:useBean`、`jsp:setProperty`、および `jsp:getProperty` タグを含む) を使用して DB2 Alphablox を伴う JSP アプリケーションを開発することもできますが、Blox Tag Library はより少ない作業でほとんど同一の機能を提供します。このトピックの残りの部分では、Blox Tag Library 内の中核となる Blox タグを使用して Blox を定義する方法について主に説明します。Blox Form Tag Library、 Blox Logic Tag Library、および Blox UI Tag Library を含む他の Blox Form Tag Library については、この章で後に説明します。

DB2 Alphablox を使用するサーバー・サイド・プログラム

DB2 Alphablox サーバー・サイド・プログラム・モデル (SSPM) は、可能なときには常に、Web アプリケーション・サーバー上の処理アプリケーションおよびビジネス・ロジックを強調します。DB2 Alphablox は、サーバー・サイド機能の豊富なセット、および JavaServer Pages と Java プログラム言語のサポートを提供します。BEA WebLogic や IBM WebSphere などの強力なアプリケーション・サーバーに関連して、Alphablox は開発者用に強力な J2EE 準拠の環境を提供します。DB2 Alphablox には、開発者に Java、JavaServer Pages、JavaBeans コンポーネント、および Java Servlets テクノロジーの完全な機能を提供する、Blox Java API が備わっています。

本書は読者に (読者の Java 経験が限られているかまったくない場合でも)、Java サーバー・サイド・プログラム・モデルの機能を使用して分析アプリケーションを素早く作成する方法を主に示します。本書ではタスクを中心に説明します。これは、DB2 Alphablox の機能を使用してビジネス上の即座の必要を解決する方法を素早く学習するために役立ちます。

Blox Tag Library の使用

JSP カスタム・タグをサポートする JavaServer Pages テクノロジーを使用して開発された DB2 Alphablox Blox Tag Library には、Web ページ作成者および Java 開発者が使用できる使いやすいタグが含まれています。

中核となる Blox タグを使用して、ChartBlox、 DataBlox、 DataLayoutBlox、 GridBlox、 PageBlox、PresentBlox、および ToolbarBlox を含む、共通ユーザー・インターフェース Blox を定義します。さらに、Blox タグを使用して、リレーショナル・レポート・アプリケーションの作成に特に使用される Blox も定義します。その他の Blox Tag Library は、強力なフォーム要素の作成 (Blox Form Tag Library)、Blox UI の拡張 (Blox UI Tag Library)、複雑なビジネス・ロジックの処理 (Blox Logic Tag Library)、およびポータル・アプリケーション内の URL ベースのクライアント・サイド・リンクのサポート (Blox Portlet Tag Library) に役立ちます。リレーショナル・レポート要件に関して、ReportBlox および関連する他の Blox を含む Blox Reporting Tag Library が「*Relational Reporting 開発者用ガイド*」で説明されています。

Blox Tag Library を使用することの利点について説明する前に、以下のコード例を参照してください。この例から、Blox タグを使用することの利点を知ることができます。Blox タグの詳しい機能については、少し後に説明されますので、今は知る必要はありません。その代わりに、コード例のレイアウトおよび読みやすさに注目してください。

読者が標準の JSP 構文を使用して Java Bean を定義する方法をすでに知っている場合、以下のコード例を簡単に理解できるはずです。しかしこの構文が目新しいものである場合は、多くの疑問が生じて、多くを学習してからでなければ進歩は望めないのではないかと不安に感じる可能性があります。この例では PresentBlox を定義するために標準の JSP 構文を使用する、 Blox をコーディングするためのより高度な方法を強調していることに注意してください。

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

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

「*開発者用ガイド*」では、この構文については多くを説明していません。この構文が目新しいものである場合でも、Blox Tag Library には Blox コンポーネントおよびアプリケーションを作成するためのより簡単な方法が備わっているので安心してください。場合によっては標準の JSP 構文がソリューションをコーディングするための唯一の方法となりますが、ほとんどの場合には代わりに Blox タグを使用できます。ここで、直前のコード例を、同じ PresentBlox を定義するために Blox タグを使用する次のコード例と比較してください。

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

このように、このコードは判読および保守がより容易です。JSP カスタム・タグ・アプローチの使用に関心を持つべき主な理由を、以下に示します。

- 判読がより容易

  PresentBlox 例では、Blox Tag Library を使用してコードを作成するとより読みやすくなります。プロパティーはタグ属性を簡単な名前と値の対として使用してマップされます。これは読みやすくするためにフォーマット設定できます。

- コーディングがより容易

  入力が少ないために、Blox タグを使用して作成された Blox はより速くコーディングできます。初期化やデータ・ソースへの接続のために、追加の数行を加える必要はありません。そのような処理は、Blox タグによって自動的に行われます。

- 保守がより容易

  判読およびコーディングがより容易になるので、Blox タグの保守はより容易になります。Blox (Java Bean) の初期化およびデータ・ソースへの接続を含む詳細は、開発者に代わって自動的に処理されます。開発者は Blox およびそのプロパティーの定義に集中することができ、タグ・ライブラリーがそれらすべてを稼働するように処理します。さらに、Java コードをタグ・ライブラリーによってカプセル化することの 1 つの利点は、Java コードの将来の更新が Alphablox にもユーザーにも管理しやすくなることです。

それでは、これらの Blox タグを使用して分析アプリケーションで使用する Blox を定義する方法について、学習を開始します。

Blox Tag Library へのアクセス

デフォルトでは、Web ページは既知ではないタグを無視します。つまり、それについての情報をどこから入手できるかをページに知らせないで Blox タグをページ上に配置すると、そのタグは無視されます。

そのため、Blox タグを利用する前に、JSP ページの先頭に taglib ディレクティブと呼ばれる 1 行を追加する必要があります。使用する JSP `taglib` ディレクティブを以下に示します。

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

このコードの行は JSP コンパイラーに、`bloxtld` として指定された URI (Uniform Resource Identifier) にあるカスタム・タグ・ライブラリーを使用する予定であることを知らせます。この URI つまり `bloxtld` は、DB2 Alphablox が Blox カスタム・タグ・ライブラリー用のタグ・ライブラリー・ディスクリプター・ファイル (`blox.tld`) を検索できる場所を定義する略式ラベルです。`blox.tld` ファイルは、作成する各アプリケーション内の、通常は以下の場所に存在します。

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

`blox.tld` ファイルは、DB2 Alphablox アプリケーションでサポートされるタグおよびタグ属性を判別します。このファイルは、DB2 Alphablox ホーム・ページを使用して新規のアプリケーション内に自動的に作成されます。

注: タグ・ライブラリー・ディスクリプター (`.tld`) ファイルの詳細に関心がある場合、上記の『JavaServer Pages テクノロジー』セクションにリストされた推奨される JavaServer Pages テクノロジー・リソースの 1 つを参照してください。DB2 Alphablox アプリケーションが作成され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。

`taglib` ディレクティブは JSP ページで Blox タグを使用する前に出現するのであれば、そのページ上の任意の場所に配置できます。ただしベスト・プラクティスは、taglib ディレクティブを JSP ページの先頭の `<html>` タグの上に次のように配置することです。

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

ここでも、`taglib` ディレクティブは Web アプリケーション・サーバーに Blox タグを使用する予定であること、そのためにライブラリーが使用可能でなければならないことを知らせます。その後 JSP Engine は `taglib` ディレクティブで定義されているように JSP ページ内を解析して、`blox` プレフィックスで始まるタグをページ上で検索します。タグが検出されるとタグ・ライブラリー内の Java コードが実行されます (ユーザーはそれを見る必要がありません)。

Blox ヘッダー・タグの使用

taglib ディレクティブをページの先頭に追加した後、そのページに含める必要のある重要なタグは、Blox ヘッダー・タグ (<blox:header>) です。このタグは、ページでの Blox のレンダリングを管理して、重大な外部 JavaScript および Cascading Style Sheets (CSS) ファイルを使用可能にします。さらに、ファイル・キャッシングを管理する数行のコードを追加します。

Blox ヘッダー・タグは、JSP ページの <head> セクション内で taglib ディレクティブよりも後に配置する必要があります。

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

少なくとも、上の例に示されているように、略式の <blox:header> タグを追加する必要があります。ただし特定のアプリケーションによっては、このタグ内にネストされたタグを追加して他の重要な Blox アクションを実行しなければならないことがあります。この使用方法については、本書の後の部分で説明されます。タグの構文と使用法の詳細については、「*開発者用リファレンス*」の<blox:header> タグを参照してください。

注: <jsp:include> を使用して JSP ページに Blox のあるファイルを含めるときには、 <blox:header> タグをそのページの先頭に追加して Blox が初期化の段階でハングしないようにする必要があります。

注: 複数のアプリケーションからのフレームセットおよび複数のフレームを使用してアプリケーションを開発するとき、 <blox:session> タグを使用して、ユーザー・セッションの適切な管理を支援できます。このタグについての詳細は、「*開発者用リファレンス*」の<blox:session> タグを参照してください。

Blox の定義に関する次のセクションでは、Blox Tag Library の Blox タグを使用してアプリケーション・ページ上に Blox を定義する方法について説明します。

Blox の定義

以下の表では、ユーザー・インターフェース Blox およびその JSP カスタム・タグをリストします。

Blox 名
: Blox タグ

ChartBlox
: <blox:chart>

DataBlox
: <blox:data>

DataLayoutBlox
: <blox:dataLayout>

GridBlox
<blox:grid>

PageBlox
<blox:page>

PresentBlox
<blox:present>

注: これらの Blox とそのタグの完全な構文、属性、および使用法について詳しくは、「*開発者用ガイド*」を参照してください。

注: blox プレフィックス、コロン、およびタグの名前の間には、スペースはありません。コロンの後にスペースを入れると、JSP コンパイラー・エラーが生じます。

注: 「*開発者用ガイド*」の全体で、Blox についての説明の際に、タグの略式構文について参照されています。これにより、Blox 自体ではなくタグについての説明であることが明白になります。たとえば、Blox GridBlox タグと記述する代わりに、このガイドではしばしば <blox:grid> タグと記述します。

1 ページの『第 1 章 DB2 Alphablox アプリケーションおよび基礎となる Blox』で説明されたように、使用する Blox およびその特定の使用法に応じて、 Blox はスタンドアロンで使用することも、他の親 Blox 内に子としてネストすることも可能です。デフォルトでは、スタンドアロンの Blox にはデフォルト値に設定されたネストされた Blox が含まれています。たとえば、PresentBlox には、ネストされた ChartBlox、DataBlox、DataLayoutBlox、GridBlox、 PageBlox、および ToolbarBlox が含まれています。

以下の表では、スタンドアロンのプレゼンテーション Blox ごとに、ネストされた Blox コンポーネントがリストされています。

スタンドアロン Blox
ネストされた Blox コンポーネント

ChartBlox
DataBlox、ToolbarBlox

DataBlox
CommentsBlox [オプショナル]

DataLayoutBlox
DataBlox

GridBlox
DataBlox、ToolbarBlox

PageBlox
DataBlox

PresentBlox
ChartBlox、DataBlox、DataLayoutBlox、GridBlox、PageBlox、ToolbarBlox

ネストされた Blox にネストされたタグを含める必要はなく、最上位の親 Blox タグだけを含めることができます (上にリストされたネストされた Blox は、暗黙的に含められます)。

たとえば、<blox:present> タグと共に定義された最小の PresentBlox は、開始タグおよび終了タグと共にコーディングした場合に以下のようになります。

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

または、略式メソッドを使用すると、以下のようになります。

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

略式メソッドは、ほとんどの場合に推奨されます。開始タグおよび終了タグが必要なのは、それらのタグの間にコンテンツ (ボディと呼ばれる) を追加する必要がある場合だけです。

注: 上記の最小の例には、id 属性が含まれます。これは、Blox が正常にレンダリングされるために必要な最小の定義です。アプリケーション内のすべての親 Blox は一意的に識別される必要がありますが、ネストされた Blox には id 属性は必要ありません。

PresentBlox に暗黙的に含まれている、ネストされた Blox タグをすべて含める場合、同じ PresentBlox は以下のようになります。

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

ネストされた Blox は暗黙的に含まれているので、ネストされた Blox タグを明示的に追加しない場合でも、必須のタグ属性またはデフォルト設定から変更しなければならないプロパティーのタグ属性を含める必要がある場合にのみ、ネストされたタグを含めてください。

ネストされた Blox が定義されていない、またはタグ属性が定義されていないページに PresentBlox タグを配置する場合、PresentBlox はそのページ上でレンダリングされますが、Blox は実用的に機能しません。少なくともデータ・ソースおよび照会をネストされた <blox:data> タグに定義しなければ、データは検索されません。Blox が実用的に機能するためには、通常はタグ属性またはネストされたプロパティー・タグを追加する必要があります。

ここまでで、Blox を JSP ページに表示させる方法、およびネストされた Blox を含める方法について学びました。次のセクションでは、簡単なタグ・プロパティーを Blox に追加する方法について説明します。

タグ属性を使用する Blox プロパティーの設定

Blox が JSP ページでインスタンス化するときに使用される初期 Blox プロパティーは、Blox タグ属性またはネストされたプロパティー・タグで定義された設定値によって決まります。プロパティー・タグについては、次のセクションの『プロパティー・タグを使用する Blox プロパティーの設定』で説明します。

Blox には、共通プロパティーと固有プロパティーがあります。そして、他の Blox 内で生じる暗黙的な Blox と同様に、Blox 上のすべてのプロパティーにはデフォルト値があります。ほとんどのデフォルト値については変更の必要がありませんが、値を変更する必要がある場合には、これらのプロパティーのほとんどはタグ属性を使用して公開および変更できます。 Blox のカスタマイズに使用できる何百ものタグ属性について詳しくは、「*開発者用リファレンス*」を参照してください。ここでは、しばしば変更される 2 つの属性を例として検討します。

PresentBlox のデータ・ソースおよび初期照会を定義しなくても、これは正常にレンダリングされて、グリッドおよびチャートのセクションに「使用可能なデータはありません」メッセージが表示されます。表示するデータを検索するには、2 つの DataBlox プロパティーを、`dataSourceName` および `query` 属性を使用して定義する必要があります。前に説明したように、ネストされた Blox にアクセスしてそれを変更するには、ネストされた Blox タグを最上位の Blox 内に追加してから、必要なタグ属性またはネストされたプロパティー・タグを追加しなければなりません。以下の PresentBlox の例では、ネストされた DataBlox タグが 2 つのタグ属性、`dataSourceName` および `query` を使用して追加されます。

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

PresentBlox でデフォルトのチャート・タイプ設定を変更したい場合、 ChartBlox の `chartType` 属性を使用して代替のチャート・タイプを定義する、ネストされた ChartBlox タグを追加する必要があります。以下の例では、PresentBlox はデフォルトの棒グラフではなく線グラフを表示するようになります。

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

ユーザー要件に適合するように Blox をカスタマイズするため、タグ属性を使用して多数の Blox プロパティーを追加および変更することになります。タグ属性に加えて、いくつかの Blox ではプロパティーを定義するために独自のプロパティー・タグが必要となります。次の 2 つのセクションでは、これらの「ネストされた」プロパティー・タグを使用していくつかの特定の Blox スタイル・プロパティーを定義する方法について説明します。

スタイル・プロパティー・タグを使用する Blox プロパティーの設定

ほとんどの Blox プロパティーはタグ属性を使用して公開および定義されますが、すべてのスタイルおよび索引付きプロパティー (索引値を使用して同じ Blox 内に複数のインスタンスを含めることを可能にするプロパティー) を含む他のプロパティーは、相対的により複雑で、そのほとんどにさらに定義の必要なサブプロパティーが含まれます。これらのプロパティーのいくつかでは独自のタグを使用する必要があり、他のプロパティーは Blox 上のプロパティー・タグまたはタグ属性として使用できます。

以下の表では、プロパティー・タグを使用して公開および定義される、すべての索引なし Blox プロパティー (それぞれが索引値で識別される複数のインスタンスを持つことができないプロパティー) をリストします。

プロパティー	関連したサブプロパティーまたは属性	適用対象
titleStyle	foreground font	ChartBlox
footnoteStyle	foreground font	ChartBlox
labelStyle	foreground font	ChartBlox
axisTitleStyle	foreground font	ChartBlox

上にリストされたそれぞれのスタイル・プロパティーで、 Blox プロパティー・タグはプロパティーおよび関連したサブプロパティーを定義します。タグ上の属性を使用して、スタイル・プロパティーの個別のサブプロパティーを設定します。ネストされた Blox 定義タグと同様に、Blox プロパティー・タグはそのプロパティーが適用される Blox 内でネストされます。しかし、Blox 定義タグとは異なり、これらのタグはオブジェクトを定義するのではなく、Blox 上にプロパティーを定義するために使用されます。

以下の GridBlox タグ例には、id、bandingEnabled、および defaultCellFormat などのいくつかの GridBlox タグ属性が含まれます。 GridBlox タグの本体には、ネストされたプロパティー・タグ (<blox:titleStyle>) があります。これはグリッド内のすべてのセルの表示方法を定義するために使用されています。

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

プロパティー・タグを使用すると、開発者はこれらのサブプロパティーのすべてを単一値ストリングに書き込まなくても、複雑なプロパティーをコーディングすることができます。これは読みやすくてコーディングに便利であり、コーディング・エラーの可能性を削減するためにも役立ちます。さらに、長い値ストリングでは改行ができないために、上記の例のようにきれいなフォーマットにはなりません。単一値ストリングを使用すると、上の例は以下のようになります。

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

スタイルの定義にタグ属性を使用するかまたはネストされたプロパティー・タグを使用するかは、開発者の個人的な好みの問題ですが、アプローチに一貫性があればデバッグが容易になります。さらに、スタイル・タグ属性内にネストされた引用符を使用することは面倒です。

スタイル・プロパティー・タグの使用法について詳しくは、「*開発者用リファレンス*」で各スタイル・プロパティーに関する説明を参照してください。

プロパティー・タグを使用する索引付き Blox プロパティーの設定

索引付き Blox プロパティーも Blox プロパティー・タグを使用して定義され、これらのプロパティーの複数のインスタンスを 1 つの Blox 内で使用可能にするための索引値が含まれています。前のスタイル・プロパティー・タグとは異なり、これらのプロパティー・タグには、 Blox タグ内で同じタグに対する複数のインスタンスを取り扱うことを可能にする index 属性が含まれています。

索引付き、および索引なしプロパティー・タグの間には、次の 2 つの重要な相違があります。

- 親 Blox タグ内で、同じ索引付きプロパティー・タグに対する複数のインスタンスを持つことができます。
- 属性で索引値を明示的に定義していなければ、索引付き Blox プロパティー・タグをコード内に配置する順序は出力に影響します。

索引付きプロパティー・タグには、複数の例が存在するときに解釈の順序を定義する、共通の index 属性があります。 index 属性を使用すると、以下が可能になります。

- これらのプロパティー・タグで定義された索引付きプロパティーにスクリプト記述すること
- 解釈の順序を割り当てて、これらのプロパティー・タグをネストされた Blox 内で再配列する必要がないようにすること (ただし、それらを適正な順序で配列すれば、開発者が予想される動作を解釈するために役立ちます)。

index 属性が定義されていない場合、暗黙的な索引値が自動的に割り当てられます。最初の index 属性には、値の 1 が割り当てられます。複数の索引付きプロパティー・タグを使用する予定であり、これらのタグにスクリプト記述するのであれば、索引属性をタグに追加してコード内に表示される値を割り当てることを検討してください。これにより、正しいタグにスクリプト記述していることを確認できます。

以下の表では、すべての索引付きプロパティー、それらのサブプロパティー、およびそれらが属する Blox をリストします。

索引付きプロパティー	関連したサブプロパティーまたは属性	適用対象
cellAlert	index enabled condition value value2 description font foreground background apply format align valign link image image_align scope	GridBlox
cellFormat	index format scope	GridBlox
cellEditor	index scope	GridBlox
cellLink	index description image image_align link scope	GridBlox

索引付きプロパティー	関連したサブプロパティーまたは属性	適用対象
generationStyle	index foreground background font align valign	GridBlox

前に説明したとおり、索引付きプロパティー・タグにスクリプト記述すると、暗黙のまたは定義された索引属性が付与されます。以下の例では、 GridBlox には 2 つの GridBlox cellAlert タグがありますが、どちらにも index 属性は定義されていません。

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

2 番目の <blox:cellAlert> タグを変更するには、 Java メソッドは以下のようになります。

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

GridBlox タグに index 属性が明示的に示されていなくても、 2 番目の cellAlert プロパティーに対する index プロパティーは自動的に 2 に設定されます。そうではあっても、以下のように明示的に index 属性を設定してセル・アラートを定義することをお勧めします。

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

より多くのセル・アラートが定義されているときは特に、これを行うと、各セル・アラートの索引値を知ることが容易になります。

Blox コンポーネントの可視性の制御

visible 共通 Blox プロパティーにより、開発者は JSP ページ上の Blox のレンダリングまたは表示を制御できます。このプロパティーは、ChartBlox、DataBlox、DataLayoutBlox、GridBlox、PageBlox、PresentBlox、RepositoryBlox、およびネストされた ToolbarBlox の各 Blox に適用できます。デフォルトでは、これらの Blox の visible の値は true です。 visible プロパティーを false に設定してから、後に <blox:display> タグを使用していくつかの処理ロジックの終了後にそれを表示できます。

注: DHTML クライアントを使用するとき、Blox JavaScript オブジェクトがクライアント・ページに作成されて、そのページが Client API を使用して DB2 Alphablox と通信できるようになります。しかし、visible プロパティーが false に設定されていると、クライアント・サイドの JavaScript Blox オブジェクトは作成されません。

visible タグ属性について詳しくは、「*開発者用リファレンス*」を参照してください。

レンダリング前の処理ロジック

より高機能な分析アプリケーションでは、Java メソッドをスクリプトレットで使用して、 JSP ページに Blox を表示する前にいくつかのビジネス・ロジックを処理しなければならないことがあります。その場合、Blox の `visible` 属性を `false` に設定してから、 Blox 表示タグ (`<blox:display>`) を使用して Blox の可視性を制御できます。

ビューを JSP ページにレンダリングする前に、Blox の `visible` 属性を `false` に設定して、処理ロジックのコードを含めてから、処理の完了後に Blox をレンダリングできます。

以下の例では、PresentBlox の `visible` 属性が `false` に設定された後に、 Java を使用するいくつかの処理ロジックのあるスクリプトレットが示されて、最後に `<blox:display>` タグによって PresentBlox がページに表示されます。

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

<%
  your processing logic would go here
%>
<blox:display bloxRef="myPresentBlox"/>
```

この例で、PresentBlox の `visible` 属性を `false` に設定しなかった場合、JSP コンテナーはページに 2 つの Blox をレンダリングすることになります。 1 つは処理ロジックの実行前、1 つは実行後です。そして、最初の PresentBlox には、実行したロジックの影響は反映されません。

本来ページ上に表示されない RepositoryBlox や DataBlox などの Blox では、`visible` プロパティーを `false` に設定しても影響はありません。これらの Blox が可視にはならないので、これらの Blox ではどちらも `visible` プロパティーは無視されます。

複数のページへの Blox のレンダリング

`<blox:display>` タグは、Blox を 1 つのページ、またはフレームセット内のフレームに作成して、同じ Blox を別のページにレンダリングする必要があるときに便利です。これが役立つ 2 つの一般的な場合があります。その 1 つは、ページ上に Blox があり、印刷または Microsoft Excel へのエクスポートのためにカスタム・ページを持ちたいときです。たとえば、Blox ビューが Microsoft Excel にエクスポー

トされると、「Excel にエクスポート」ボタンのある 1 つのページ上に Blox を定義できます。ユーザーがそのボタンをクリックすると、Excel にページがロードされます。その際に、グリッドとチャートだけが表示されて、元のページにある不必要なテキストまたはボタンは表示されません。

さらに、セッション中に Blox がインスタンス化された後、 <blox:display> タグを使用して現行のビューを他のページで表示できます。

<blox:display> タグについて詳しくは、「*開発者用リファレンス*」、および本書の 183 ページの『<blox:display> タグを使用したカスタム印刷ページの作成』を参照してください。

Blox ユーティリティー・タグ

いくつかの Blox タグは、ページに表示される Blox を定義するために使用されるのではなく、追加機能へのアクセスを提供します。ここでは Blox ユーティリティー・タグについて簡潔に説明していますが、詳しくは一般 Blox リファレンス情報を参照してください。

Blox ヘッダー・タグ

Blox ヘッダー・タグ (<blox:header>) については、前に 43 ページの『Blox ヘッダー・タグの使用』で説明されています。

Blox コンテキスト・タグ

<blox:bloxContext> タグは、実際の要求および応答タイプ (HTTP またはポートレット・ベース) に合わせて適切な BloxRequest、BloxResponse、および BloxContext オブジェクトを作成する点で <blox:header> タグと似ています。しかし、テーマやレンダリング用の JavaScript コードは出力しません。このタグを使用する例としては、使用する自分の JSP ページには Blox が含まれていないものの、Blox を含む他の JSP ページを含むポートレット上にある Blox コンテキスト情報にアクセスすることが必要な場合が挙げられます。<blox:bloxContext> タグと <blox:header> タグはともに同じ変数を宣言しようとするため、これらを同じ JSP ページに共存させることはできません。

Blox デバッグ・タグ

Blox debug タグ (<blox:debug>) も特殊タグであり、 JSP ページに追加すると、完全なデバッグ情報をシステム・コンソールに送信できます。

Blox debug タグの使用方法について詳しくは、『トラブルシューティング』セクションを参照してください。システム・コンソールの使用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。

Blox 表示タグ

前述の <blox:display> タグは、いくつかの処理ロジックを実行した後に Blox をレンダリングする場合、または Blox を最初に作成されたページとは別のページにレンダリングする場合に便利です。<blox:display> タグの使用方法について詳しくは、177 ページの『第 15 章 データの提示』 および 「*開発者用リファレンス*」の <blox:display>タグを参照してください。

リソース・バンドル・タグ

リソース・バンドルを使用することにより、ローカライズや他言語への翻訳が可能なアプリケーションを作成できます。DB2 Alphablox には Java ResourceBundle クラス用のラッパー・タグが用意されており、このタグを使用することで、アプリケーションのリソース・バンドルからロケール固有のリソースにアクセスできます。ラッパー・タグは次のとおりです。

- `<blox:resourceBundle>`
- `<blox:message>`
- `<blox:messageArg>`

これらのタグとその使用法の詳細については、「*開発者用リファレンス*」のリソース・バンドル関連タグを参照してください。

標準 JSP 構文の使用

Blox は、Blox カスタム・タグの代わりに標準 JSP 構文を使用して定義することもできます。前に説明されたように、ほとんどの場合には Blox タグが Blox を定義するための最良の方法となります。しかし、標準 JSP 構文を使用する以外に方法がない場合もあります。

開発者が以前に標準 JSP 構文だけを使用してコーディングしていて、タグ・ライブラリーを使用したことがない場合、Blox タグを使用するよりもその慣れている構文を使用したいと思う可能性があります。しかし標準 JSP 構文の使用を決定する前に、このトピックで前に説明された理由のために、Blox カスタム・タグを使用してみてください。

注: 同じページ上で標準 JSP 構文と Blox タグの両方を使用する場合、ページの先頭に以下の 2 行を入れる必要があります。

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

注: Blox タグは、`session` に設定された `scope` を使用して、Blox (Java Bean) を定義します。標準 JSP 構文を使用している場合は、ほとんど常に Bean (この場合は Blox) を、セッションを有効範囲にして使用してください。`useBean` 構文のデフォルトでは、有効範囲が `page` に設定されます。以下の例を参考にしてください。

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

次のステップ

このセッションでは、中核となる Blox タグを使用して、プレゼンテーション Blox およびそのプロパティーを定義する方法を学習しました。さらに、`<blox:display>`、`<blox:debug>`、および `<blox:header>` などの Blox ユーティリティー・タグについても学習しました。これらのタグの構文および使用法について詳しく学ぶには、「*開発者用リファレンス*」および本書を参照してください。

「*開発者用ガイド*」の残りの部分では、共通に経験するタスク、およびいくつかの可能なソリューションを紹介します。「*開発者用ガイド*」、「*開発者用リファレンス*」、およびいくつかの優れた JavaServer Pages の解説書は、ユーザーのために洗練された分析アプリケーションを開発するために大変に役立ちます。

第 6 章 Blox Form Tag Library

Blox Form Tag Library には、機能拡張が組み込まれた HTML フォーム要素を生成するための、FormBlox および他のタグが含まれています。そのタグの中には、データ・ソース、ディメンション、およびディメンション・メンバーの選択リストを自動的に生成するものがあります。また、ラジオ・ボタンおよびチェック・ボックスの管理、またはナビゲーションその他のためのツリー・コントロールを作成するために使用できるタグもあります。そして、これらのタグを使用することにより、セッション中の状態の持続が維持されます。したがって、ユーザーのブラウズ・セッション中の選択状態が持続されるようにするために、追加の Java または JavaScript コードを記述する必要はありません。ユーザーがページを離れてから同じセッション中にそのページに戻ったとしても、チェック・ボックスのチェック状態は維持され、最後に選択されたラジオ・ボタンは選択されたままの状態であり、ツリー・メニューも元の状態を維持します。

Blox Form Tag Library の使用

FormBlox および関連するタグは、`bloxform.tld` ファイルで定義されます。新規の DB2 Alphablox アプリケーションを作成するとき、このファイルは以下のディレクトリーに自動的に含まれます。

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

注: TLD ファイルが見つからない場合、または間違えて削除した場合には、この TLD ファイルの現行バージョンのコピーが以下のディレクトリーにあります。

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

FormBlox および関連するタグは、`bloxform.tld` ファイルで集合的に定義されます。新規の DB2 Alphablox アプリケーションを作成するとき、このファイルは以下のディレクトリーに自動的に含まれます。

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

Blox フォーム・タグを使用するには、以下の taglib ディレクティブを JSP ページの先頭に含めてタグ・ライブラリーが認識されるようにする必要があります。

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

FormBlox コンポーネントの概説

以下に、FormBlox コンポーネントについて簡潔に説明し、いくつかの簡単な使用例についても説明します。FormBlox コンポーネントは、開発者としてアクセス可能なものと同じ低レベルの Blox UI コンポーネントを使用して構築されていますが、これら便利な Blox コンポーネントを構築する作業はすでに実行済みです。

FormBlox API について、および他のフォームに関連したタグについての詳細な情報 (構文、使用法、例を含む) は、「*開発者用リファレンス*」の『Blox Form Tags リファレンス』セクション、および「*Blox API Javadoc*」資料を参照してください。

FormBlox コンポーネントのカテゴリー

Blox Form Tag Library の FormBlox コンポーネントは、フォーム・コントロール、メタデータ選択リスト、タイム・スキーマ選択リスト、およびツリー・コントロールの 4 つのカテゴリーに分類できます。これらのコンポーネント・グループについての概説を以下に簡潔に示します。

基本的なフォーム・コントロール

FormBlox コンポーネントのこのグループは、チェック・ボックス、テキスト・フィールドまたはテキスト域、ラジオ・ボタン、および選択リストを含む、基本的な HTML フォーム・コントロールを作成します。ほとんどの場合、これらの Blox コンポーネントは標準の HTML 要素と類似の機能を提供し、さらにセッションを通じて状態を維持するという利点があります。チェック・ボックスにチェックを入れたり選択リストで項目を選択するなどのユーザー・イベントが生じると、ページがリフレッシュされることなく、変更された値が適切なオブジェクトに送られます。

FormBlox コンポーネント
説明

CheckBoxFormBlox
HTML の `<input type="checkbox">` タグを使用して、項目を選択または選択解除するための (個別のまたはグループ化された) チェック・ボックスを作成します。

EditFormBlox
HTML の `<input type="text">` または `<textarea>` タグを使用して、テキスト・フィールドまたはテキスト域を作成します。

RadioButtonFormBlox
`<input type="radio">` タグを使用して、ラジオ・ボタン・フォームのコントロールを作成します。

SelectFormBlox
HTML の `<select>` および `<option>` タグを使用して、ドロップダウンおよびスクロール選択リストを作成します。

メタデータ選択リスト

このグループの FormBlox コンポーネントは、データ・ソース・メタデータから選択リスト・オプションを生成する特殊な HTML 選択リストを作成します。これらには、データ・ソース、マルチディメンション・データベース、キューブ、ディメンション、およびメンバーの選択リストが含まれます。これらのリストは動的に生成されるので、リスト・オプションを追加または除去する操作について覚えている必要はありません。

他の基本的な HTML 選択リストとは異なり、これらのメタデータ選択リストは、ユーザー・セッションを通じて状態を保持します。チェック・ボックスにチェックを入れたり選択リストで項目を選択するなどのユーザー・イベントが生じると、ページがリフレッシュされることなく、変更された値が適切なオブジェクトに送られます。

FormBlox コンポーネント
説明

DataSourceSelectFormBlox
DB2 Alphablox データ・ソースの、動的に生成された HTML 選択リストを作成します。

CubeSelectFormBlox
指定された DB2 Alphablox データ・ソースに、動的にデータが取り込まれた、使用可能なキューブを含む HTML 選択リストを作成します。

DimensionSelectFormBlox
指定されたマルチディメンション・キューブに、動的にデータが取り込まれた、使用可能なディメンションの HTML 選択リストを作成します。

MemberSelectFormBlox
指定されたディメンションに、動的にデータが取り込まれた、使用可能なメンバーを含む HTML 選択リストを作成します。

タイム・スキーマ選択リスト

タイム・スキーマに関連した Blox Form タグによって、開発者はユーザーが共通ビジネス時間枠および時間単位を選択するための選択リストを動的に生成できます。これらの選択リストを使用して、ユーザーに表示される分析ビューを操作できます。2 つの主なタイム・スキーマ・タグである TimePeriodSelectFormBlox および TimeUnitSelectFormBlox の要約を、以下に示します。

Blox コンポーネント
説明

TimePeriodSelectFormBlox
現在の週、現在の月、現在までの月、現在までの四半期、現在までの年、最近の 1 四半期、最近の 2 四半期、4 四半期、最近 2 カ月、最近 3 カ月、最近 6 カ月、最近 1 年、および最近 2 年を含む、共通ビジネス時間枠を示す選択リストを作成します。カスタムの時間枠を含めることもできます。

TimeUnitSelectFormBlox
時間単位のオプションを示す HTML 選択リストを作成します。これには、日、週、月、四半期、および年を含めることができます。

ツリー・コントロール

このカテゴリーには 1 つの項目、TreeFormBlox が含まれます。TreeFormBlox は、複数のフォルダーおよび項目から構成されるツリー・コントロールを作成します。これらのフォルダーおよび項目を選択すると、それらにアクションを関連付けることができます。TreeFormBlox を使用して、階層選択リストおよびナビゲーション・メニューを作成できます。さらに、 HTML フォーム POST メソッドも使用できます。使用可能にした場合、項目およびフォルダーをツリー・コントロール内でドラッグ・アンド・ドロップすることができます。

Blox コンポーネント
説明

TreeFormBlox
階層選択リストおよびナビゲーション・メニューを作成するための、DHTML ツリー・コントロールを作成します。

Blox および JavaBeans コンポーネントでのプロパティーの取得および設定

Blox Form Tag Library には、ネストされた 2 つのタグである <blox:getChangedProperty> および <blox:setChangedProperty> が含まれます。<blox:getChangedProperty> は、2 つの FormBlox の間で値を引き渡す、つまりリンクさせるのに便利です。<blox:setChangedProperty> は、2 つの FormBlox の間、FormBlox と他の Blox コンポーネント (DataBlox など) との間、または FormBlox と他のカスタム JavaBeans コンポーネントとの間で使用できます。<blox:setChangedProperty> には、変更が生じた後にサーバー・サイド Java メソッドを呼び出すことのできる callAfterChange 属性も含まれます。ブール debugEnabled 属性は、予期しない動作をデバッグする際に役立ちます。

ネストされたこれら 2 つの FormBlox タグの要約を、以下に示します。

Blox コンポーネント
説明

<bloxform:getChangedProperty>
別の FormBlox をターゲットとするように FormBlox 内でネストされています。ターゲット FormBlox からの指定されたプロパティー値は、このタグと共に FormBlox に渡されます。

<bloxform:setChangedProperty>
別の FormBlox、Blox、またはカスタム Bean を含む他の JavaBeans コンポーネントをターゲットとするように、FormBlox 内でネストされています。所有する FormBlox からの指定されたプロパティー値は、このタグと共にターゲット Bean に渡されます。

<bloxform:getChangedProperty> および <bloxform:setChangedProperty> タグについての詳細情報 (構文、使用方法、および例を含む) は、「*開発者用リファレンス*」の『Blox Form Tags リファレンス』セクション、および「*Blox API Javadoc*」資料を参照してください。

FormBlox イベント・モデル

必要であれば、FormBlox コンポーネント用に独自のイベント・ハンドラーを記述できます。簡単な FormBlox イベント・モデルは、変更が生じたとき常に、コントロールからの前の値および後の値を提供します。FormBlox コンポーネントは、開発用の包括的なソリューションとなることを意図したものではありませんが、ほとんどの場合に生じる基本的なイベントを処理するための簡単なイベント・モデルを提供します。より複雑な要求を処理できるより洗練されたコンポーネントを作成する必要がある場合は、高度なイベント・モデルもサポートする Blox UI Model コンポーネントを使用して、独自のコンポーネントを作成できます。

FormBlox タグの使用例

Blox Sampler およびその他の場所にある多くの例では、FormBlox コンポーネントを使用します。そしてこのガイドでも、FormBlox を使用する例を示しています。FormBlox コンポーネントは、リストの動的生成や状態管理のコーディングを含む煩雑なタスクの多くを扱う、開発者のための強力なリソースです。さまざまな FormBlox コンポーネントを利用するいくつかの例が、以下で強調されています。

DataSourceSelectFormBlox を使用する随時分析

Blox Sampler の「FormBlox および Logic Blox の使用 (Using FormBlox and Logic Blox)」セクションの下に、 DataSourceSelectFormBlox を使用してユーザーがデータ・ソースを選択するための簡単なビューを作成してから、完全に対話式の PresentBlox を使用して指定されたキューブを使用する分析を行う例があります。この例で使用されるコードのステップバイステップの記述は、このガイドの『データへの接続』のトピックの 133 ページの『DataSourceSelectFormBlox を使用する異なるデータ・ソースの設定』に示されています。

照会ビルダー

DB2 Alphablox 管理ページの「アセンブリー (Assembly)」タブの「ワークベンチ (Workbench)」セクションの下にある照会ビルダーは、開発者が DB2 OLAP Server、Hyperion Essbase、および Microsoft Analysis Services で使用するマルチディメンションのクエリー・ステートメントを生成するのに役立つインターフェースです。この背後では、照会ビルダーのソース・コードを表示すると分かりますが、 DataSourceSelectFormBlox および CubeSelectFormBlox を含む FormBlox コンポーネントのインスタンスが使用されています。

FormBlox を使用するレポート・オプションの指定

FormBlox コンポーネントの使用方法を示す別の良い例は、 Blox Sampler の「データとの対話 (Interacting with Data)」セクションの下にある「HTML フォーム要素を使用する例 (Using HTML Form Elements example)」にあります。この例では、RadioButtonFormBlox および CheckboxSelectFormBlox を使用してレポート・オプションを選択します。

TreeFormBlox を使用するナビゲーション・メニュー

Blox Sampler アプリケーションをオープンすると、左側のフレームでオープンするナビゲーション・メニューが TreeFormBlox を使用します。この FormBlox を使用して作成される大きなアプリケーション・メニューの例については、Blox Sampler アプリケーション内の `navigation.jsp` ファイルを参照してください。

FastForward アプリケーション内のレポート・テンプレート

Alphablox FastForward アプリケーションは、レポート・テンプレートを作成するため、およびナビゲーション・メニューのために、FormBlox コンポーネントを広範囲に使用します。Alphablox FastForward アプリケーションで使用されるコードを検討するには、アプリケーションの新規コピーを作成してください。FormBlox コンポーネントの使用方法を示すコード例も、297 ページの『第 25 章 DB2 Alphablox FastForward での作業』で説明されています。

第 7 章 Blox Logic Tag Library

Blox Logic Tag Library には、ユーザーが Essbase レポート・スクリプト (DB2 OLAP Server および Essbase で使用) または MDX ステートメント (Microsoft Analysis Services で使用) およびメンバー・セキュリティーの作成方法を知らなくても、時間枠の選択の処理およびマルチディメンション・データベースのクエリーを操作するために使用できる、より使いやすいタグが含まれています。

Blox Logic Tag Library の使用

Blox Logic Tag Library で使用可能なタグは、`bloxlogic.tld` ファイルで定義されます。新規の DB2 Alphablox アプリケーションを作成するとき、このファイルはアプリケーションの以下のディレクトリーに自動的に含まれます。

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

注: TLD ファイルが見つからない場合、または間違えて削除した場合には、この TLD ファイルの現行バージョンのコピーが以下のディレクトリーにあります。

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

ページ上の Blox Logic Tag Library にアクセスするには、以下の JSP taglib ディレクティブを含める必要があります。

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

Blox Logic Tag Library コンポーネント

Blox Logic Tag Library 内の主な Blox Logic コンポーネントには、以下に要約されている MDBQueryBlox、MemberSecurityBlox、および TimeSchemaBlox が含まれます。

Blox Logic Tag Library コンポーネントについての詳細情報 (構文、使用方法、および例) は、「*開発者用ガイド* 」の『ビジネス・ロジック Blox』と『TimeSchema DTD リファレンス』セクション、および「*Blox API Javadoc* 」資料を参照してください。

Logic Blox
: **説明**

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。

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 を取得するには、`addMember()` 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

ユーザーによるタイム・スキーマの定義に基づいて、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、時間ディメンションの名前 (複数も可)、生成レベル(「年」、「四半期」、「月」、および「週」の) 、キューブ内の時間枠の開始日、通常の暦時間または週時間のどちらを適用するか、および年の長さが例外的なもの (48 週の年など) かどうかを指定することにより、「時間 (Time)」ディメンションの構造を定義できます。

MDBQueryBlox コンポーネントを使用する製品の選択

MDBQueryBlox を使用することにより、DB2 OLAP Server、Hyperion Essbase、または Microsoft Analysis Services に固有のロジックがなくても、使いやすいタグによってマルチディメンション・クエリーを取り扱うことができます。

Blox Sampler には、「Logic Blox の使用 (Using Logic Blox)」の下に、MDBQueryBlox を PresentBlox と共に使用して、ユーザーが (MemberSelectFormBlox を使用して作成された) 選択リストから製品を選択可能にするための例があります。ここでは、類似のページを作成するための主なステップを簡単に示します。

1. ページで使用される Blox タグ・ライブラリーの JSP `taglib` ディレクティブを追加します。

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

2. `visible` 属性を false に設定し、 `useAliases` を `true` に設定することにより別名のメンバー名を使用可能にして、使用される DataBlox を定義します。

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

3. 列、行、およびページの軸で使用されるタプルのリストを指定します。

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

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

```
<!-- Column Measures tuples -->

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

<!-- Page tuples -->

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

4. ユーザーが製品ディメンションから製品を選択できるように、MemberSelectFormBlox を追加します。

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

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

5. MDBQueryBlox を追加します。

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

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

6. 以前に指定した DataBlox を参照する PresentBlox を追加します。

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

7. Blox をレンダリングしてビューをレイアウトするために、ページの残りの部分を追加します。

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

<body>

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

Blox Sampler で「Blox Logic タグの使用 (Using Blox Logic Tags)」の下にある、MDBQueryBlox および MemberSelectFormBlox を使用する完全なコードおよび作動例を参照してください。

MemberSecurityBlox を使用するキューブ・メンバーのリスト

MemberSecurityBlox タグによって、アクセス許可権限に基づくディメンション内のメンバーをリストできます。これは DataBlox の suppressNoAccess プロパティーを使用して、メンバーをフィルターに掛けます。また、複数のルート・メンバーを使用することができます。さらに、このタグを使用して、フィルター処理のために複数の dimension:members の対を指定できます。

MemberSecurityBlox の使用例を以下に示します。

1. アクセスする必要のある Java クラスを指定した JSP ページ・ディレクティブを、ファイルの先頭に追加します。

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

2. 使用する Blox タグ・ライブラリーの JSP taglib ディレクティブを追加します。

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

3. ページのヘッド・セクションに、<blox:header> タグを忘れずに追加してください。

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

4. DataBlox をページに追加します。

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

5. MemberSecurityBlox タグを追加します。

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

6. SelectFormBlox を追加します。

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

7. 選択リストを表示するページに <blox:display> タグを追加します。

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

TimeSchemaBlox コンポーネント

TimeSchemaBlox は、ユーザーによるタイム・スキーマの定義に基づいて、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、時間ディメンションの名前、生成レベル(「年」、「四半期」、「月」、および「週」の) 、キューブ内の時間枠の開始日、通常の暦時間または週時間のどちらを適用するか、および年の長さが例外的なもの (48 週の年など) かどうかを指定することにより、「時間 (Time)」ディメンションの構造を定義できます。

<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。

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 XML の定義に使用されるデータ・タイプ定義 (DTD) は、TimeSchema XML DTD に記述されます。

TimeSchemaBlox および TimeSchema XML DTD の構文および使用方法について詳しくは、「*開発者用リファレンス*」の『ビジネス・ロジック Blox』および『TimeSchema DTD リファレンス』セクションを参照してください。

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 の listFromMetadataTuples プロパティーはそれに応じて変更されます。

```
<blox:data id="dataBlox"
  dataSourceName="QCC-MSAS"/>
  <bloxlogic:timeSchema id="timeSchema"
    name="MSAS"
    dataBloxRef="dataBlox" />
<bloxlogic:tupleList id="histTuples">
    <bloxlogic:dimension
      list="<%=timeSchema.getDimensions()%>">
    </bloxlogic:dimension>
    </bloxlogic:tupleList>
<bloxform:timePeriodSelect id="historySelector"
   timeSchemaBloxRef="timeSchema"
   selectedSeriesString="SEQUENCE(QUARTER,-1,1)(QUARTER)"
   visible="false">
   <bloxform:setChangedProperty
     formProperty="tuples"
     targetRef="histTuples"
     targetProperty="listFromMetadataTuples"
     callAfterChange="changed"/>
</bloxform:timePeriodSelect>
```

第 8 章 Blox Portlet Tag Library

Blox Portlet Tag Library のタグを使用することにより、Blox コンポーネントまたは UI コンポーネント内にポートレット・リンクやアクション・リンクを定義することができます。また、これらのリンクによって、ポートレット間メッセージングにポートレット API が使用できるようになります。このセクションでは Blox Portlet Tag Library について説明し、Blox Portlet Tag Library を使用して ClientLink ベースのリンクまたはアクション・リンクをポートレット内で最上位の Blox または UI コンポーネントにアタッチする方法について示します。

注: 最上位の Blox とは、ネスト関係にある Blox どうしで最も外側にある Blox のことです。たとえば、PresentBlox が GridBlox と ChartBlox をネストしている場合には、PresentBlox が最上位 Blox です。

Blox Portlet タグの概説

あるポートレット内のリンクをクリックするだけで別のポートレット内での更新をトリガーできたらよいと思うことがよくあります。Blox UI Model の `ClientLink` オブジェクトを使用することにより、特定の Blox UI コンポーネントへのリンクをアタッチすることができます。この URL ベースのリンクは、コンポーネントをクリックした時にブラウザーによって処理されます。`ClientLink` オブジェクトによって、ビュー・レイヤーはユーザーによるクリックを処理する際に、その動作をサーバーに戻すのではなく独自のロジックを使用するようになります。`ClientLink` には、ロードする URL だけでなく、新規ページをロードする宛先のウィンドウやブラウザー・ウィンドウ機能ストリング (たとえば、`features="scrollbars=yes,width=300,height=300"`) も指定できます。ただし、ポータル環境の場合は、最初にリンクのみが処理されます。リンクがページの再ロードをトリガーすると、ポータル・サーバーが各要求を処理する方法が原因となって、ポートレット・リンクは不整合になります。それ以降、リンクをクリックしても、実際のアクションはサブミットされません。Blox Portlet Tag Library を使用することによって、PortletLinkDefinition または ActionLinkDefinition を追加することができ、これらにより以下の機能が提供されます。

- PortletLinkDefinition を追加した場合は、このタグを使用して PortletLink オブジェクトを作成できます。ついで、PortletLink オブジェクトを使用して、指定したパラメーター値の URI を呼び出す実際のリンクを定義します。URL はページが更新されるたびに、不整合にならないようポートレットによってエンコードし直されます。
- ActionLinkDefinition を追加した場合は、このタグを使用して PortletLink を作成できます。または、このタグを使用して、アクション名を `BloxResponse.getActionURL()` に渡し、このリンクのポートレット URI を取得できます。

PortletLinkDefinition または ActionLinkDefinition を Blox と結合すると、定義は Blox に割り当てられ、一方で PortletLink は Blox UI Model 内で使用する ClientLink の生成に使用されます。Blox Portlet タグは、すべてのデータ・プレゼン

テーション Blox、FormBlox、ReportBlox、またはクリックされたイベントの概念を持つ Blox UI Model コンポーネントすべてに組み込んで使用することができます。PortletLinkDefinition と ActionLinkDefinition はともに PortletLink の作成に使用できますが、ActionLinkDefinition は PortletURI を作成するこのリンク定義のアクション名の設定にも使用できます。ただし、この定義を使用して PortletLink を生成した後は、アクション名を `BloxResponse.getActionURL()` に渡すことはできません。

基本的なコード構造、およびポートレット JSP に Blox を追加する方法に関する開発上のヒントについては、はじめに「入門」セクションにあるポートレットのチュートリアルを参照してください。

Blox Portlet Tag Library の使用

Blox Portlet Tag Library で使用可能なタグは、`bloxportlet.tld` ファイルで定義されます。ポートレット・プロジェクトの場合、構造やデプロイメント記述子ファイルを適正に設定するために、ポートレット開発ツールを使用してください。必要な Alphablox Tag Library およびサーブレット・マッピングがすべて適正に参照および指定されるようにプロジェクトを作成する方法については、「入門」セクションの Rational Application Developer を使用したポートレットのビルドに関するチュートリアルを参照してください。ポートレットに Blox コンポーネントを追加する方法については、「入門」セクションのポートレットに関するチュートリアルの部分を参照してください。

注: TLD ファイルが見つからない場合、または間違えて削除した場合には、この TLD ファイルの現行バージョンのコピーが以下のディレクトリーにあります。

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

Blox Portlet タグを Blox または Blox UI コンポーネントに追加するには、`<bloxportlet:actionLinkDefinition>` または `<bloxportlet:portletLinkDefinition>` タグを Blox タグの中にネストします。これにより、リンク定義がコンポーネントにアタッチされます。パラメーターとその値は、ネスト内の `<bloxportlet:parameter>` タグを使用して指定します。

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

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

<portletAPI:init/>

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

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

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

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

%>
</blox:container>
```

この後、指定されたアクションから PortletLink に進み、リンク情報またはパラメーター値をスクリプトレット内に設定できます。

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

PortletLink は、PortletLinkDefinition を呼び出すマークアップを生成するために使用します。 PortletLink メソッドには以下のものが含まれます。

- `getClientLink()`: この PortletLink が Blox UI Model 内で使用されることを表す ClientLink を戻します。
- `getLinkHRef()`: HTML アンカー・タグ内で使用可能であった `HREF` を表すストリングを戻します。
- `setParameterValue()`: このリンク内でのパラメーターの値を設定します。

ClientLink について詳しくは、110 ページの『DHTML Client API のフレームワーク』を参照してください。

Blox Portlet Tag Library の例

このセクションでは、Blox Portlet タグの Button (Blox UI コンポーネント) および ReportBlox 内での使用法を示す例を取り上げます。それぞれの例では、アクション・リンクまたはポートレット・リンクを追加する基本的な方法が使用されています。

1. `<bloxportlet:actionLinkDefinition>` タグをリンク定義を付加する Blox または UI コンポーネントに追加し、`action` 属性を使用するアクション名を指定します。
2. ネストされた `<bloxportlet:parameter>` タグを使用して、パラメーター名およびその値を指定します。

その後、名前付きアクションの PortletLink を取得し、スクリプトレットにリンク情報またはパラメーター値を設定できます。API の詳細については、Javadoc 資料内の com.alphablox.blox.portlet パッケージを参照してください。GridBlox および TreeFormBlox へのポートレット・リンクの追加方法についてさらに詳しくは、「*開発者用リファレンス*」のBlox Portlet Tag Library Reference というトピックを参照してください。

ボタンへのリンクの追加

以下の例では、Button に 3 つのパラメーターを持つ「showData」という名前のアクションを定義します。PortletLink の ClientLink をボタンと関連付けるので、ボタンをクリックすると、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。この情報を別のポートレットなどで使用するには、追加コードが必要です。この例で示されているのは、リンクの設定方法に限定されています。

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

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

<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 コンポーネントへのリンクの追加

以下の例では、ReportBlox に「selectProductCode」という名前のアクションを定義します。 このリンクは、「Product」列に付加されます。レポート内の商品名がクリックされると、パラメーター「code」の値が商品コードに設定されます。この情報を別のポートレットなどで使用するには、追加コードが必要です。この例で示されているのは、リンクの設定方法に限定されています。

```
<%@ 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>
```

以下は、Portlet API を介してこの情報を利用するために作成された actionPerformed() メソッドを用いてこのアクションを処理する方法を示しています。

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

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

第 9 章 Blox UI Tag Library

DHTML Client UI モデルには、一般的に使用される UI 操作に簡単にアクセスするためのタグのライブラリーが備わっています。これらのタグはライブラリー `bloxui.tld` に含まれています。また、各タグとそのプロパティーの完全なリストは、「*開発者用リファレンス*」の『Blox UI Tags リファレンス』セクションにあります。

DB2 Alphablox には、Blox Tag Library と呼ばれる、Blox を操作するためのタグ・ライブラリーが備わっています。Blox UI Tag Library は、Blox Tag Library を補完するものです。開発者は Blox Tag Library を使用して、データ・プロパティーの設定、一般的な UI 操作 (チャート/グリッドの方向、メニュー・バーの表示、分割ペインの調整、など) の実行、およびセル・アラートや算出されるメンバーなどの一般的な DB2 Alphablox 機能へのアクセスを行います。DHTML Client を使用していて、ユーザー・インターフェースに対する、 Blox ライブラリーでは提供されないより高レベルの制御が必要な場合、Blox UI Tag Library がその必要とされる機能を提供できることがあります。

Blox UI Tag Library のカテゴリー

Blox UI タグは、次の 4 つのカテゴリーに分類されます。

カテゴリー
　説明

コンポーネント・カスタマイズ
　UI をコンポーネント・レベルでカスタマイズするためのもの。例: メニューおよびツールバーのカスタマイズ。

カスタム・レイアウト
　空白行や空白列を追加したり、グリッドをバタフライ・レイアウトで生成するなど、グリッドのレイアウトに対する制御を提供する。

分析　分析機能をアプリケーションに組み込むために使用される。

ユーティリティー
　アクションの処理を容易にするための便利なタグ。開発者はユーティリティー・タグを使用して、ユーザー選択をインターセプトしたり、グリッドの変更時にアクションを実行することができます。

Blox UI タグの例

このセクションでは、これらの使いやすいタグが持つ能力についての理解の助けとして、多数の Blox UI タグの中からいくつかの例を示します。

Blox UI コンポーネント・カスタマイズ

以下の例は、コンポーネント・カスタマイズのタグを示しています。これらのタグを使用して、ユーザー・インターフェース内のメニューを追加および除去、または使用可能および使用不可にすることができます。

たとえば、以下の Blox および Blox UI タグを使用することにより、ツール・メニューを使用不可にしてブックマーク・メニューを除去できます。

```
<blox:grid id="testGridBlox"
    width="600"
    height="700"
    bandingEnabled="true"
    rowIndentation="None"
    commentsEnabled="false">
    <blox:data bloxRef="dataBlox" />
    <bloxui:menu name="toolsMenu" disabled="true" />
    <bloxui:menu name="bookmarkMenu" visible="false" />
</blox:grid>
```

カスタム・レイアウトのタグ

次の例は、以下の図に示すように行ヘッダーをグリッドの中央に移動するカスタム・レイアウトのタグを使用して、グリッドの表示をバタフライ・レイアウトに変更する方法を示しています。

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

このタグのプロパティーを使用して、開発者は行ヘッダーの位置、および区切り列をヘッダーとデータとの間に入れるかどうかを指定できます。

分析タグ

開発者は分析タグを使用して、分析をアプリケーションに直接組み込むこともできます。たとえば、アセンブラーは「bottom N」の計算をアプリケーションに組み込みたい場合があります。

このビューは、以下の PresentBlox タグおよび Blox UI の bottomN タグを使用して実現できます。

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

```
    prompt="true"
    showRank="true"
    number="7"/>
</blox:present>
```

Blox UI 分析タグには、開発者が計算をアプリケーションに組み込むことを可能にする汎用タグも含まれています。

ユーティリティー・タグ

最後に、Blox UI Tag Library は、ユーザー入力の処理を大幅に簡易にするタグを提供します。以下のコード例は、ユーティリティー・タグを使用することにより、ピボット・メニュー項目のユーザー・クリックをインターセプトしてダイアログを表示します。

```
<blox:grid id="testActionFilter"
  width="80%"
  height="500"
  bandingEnabled="true">
  <blox:toolbar visible="true" />
  <blox:data bloxRef="dataBlox" />
  <bloxui:actionFilter
    className="<%= MyActionFilter.class.getName() %>"
    componentName="dataPivot" />
</blox:grid>

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

この例で、actionFilter クラスは JSP ファイルで定義されてから、グリッドおよびピボット・メニュー項目に関連付けられます。イベント・フィルターについては、「*開発者用リファレンス*」にある『Blox UI タグ・リファレンス』の『ユーティリティー・タグ』セクションで詳しく説明されています。

その他の例

これらの、およびその他の Blox UI Tag Library タグの実行例、およびソース・コードについては、 Blox Sampler を確認してください。

第 10 章 DHTML Client UI の拡張性

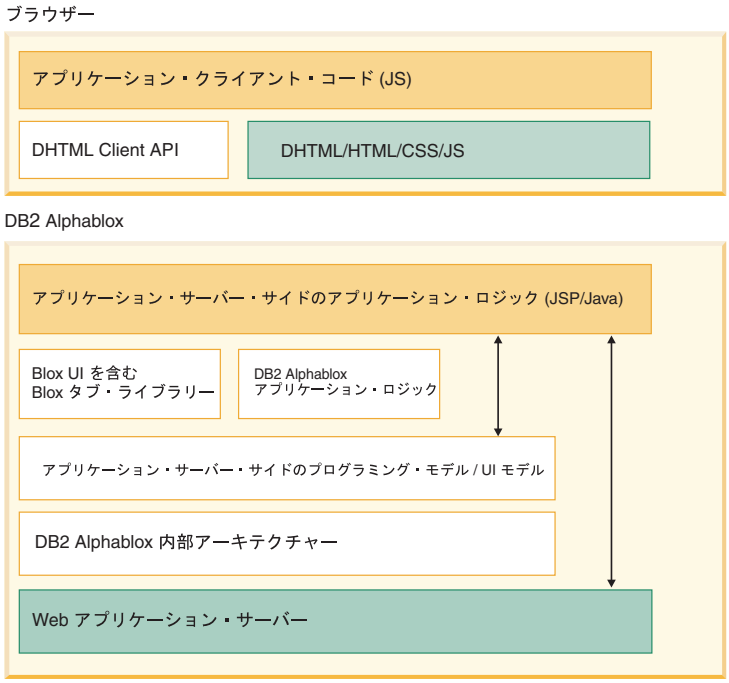
この上級のトピックは、サーバー・サイド Java API での作業に習熟した開発者を対象としています。ここで詳しく解説する Blox UI Model は、簡単な Blox UI タグの使用を越えてユーザー・インターフェースをカスタマイズするための多機能で強力な API を提供します。次のトピックの 109 ページの『第 11 章 DHTML Client API』では、アプリケーション開発者がクライアント・サイドとサーバー・サイドのプログラム・モデル間のギャップを埋める DHTML Client API について説明します。

Blox UI Model

DHTML User Interface Model は、アプリケーション開発者がエンド・ユーザーのユーザー・インターフェースを検討および制御できるようにする、プログラマチック API です。この API により、表示内容やユーザー入力の処理など、ユーザー・インターフェースのすべての局面を直接制御できます。UI Model API によっても、ユーザー・インターフェースをいくらか制御できます。これは、開発者がデータ API を使用してメタデータおよび結果セット・データを制御する場合と似ています。データ API およびユーザー・インターフェース API を共に使用すると、アプリケーションのすべての局面に対して、開発者による全体的な制御およびカスタマイズが可能になります。

以下の図に示すように、DB2 Alphablox フレームワークには次のものが含まれます。

- DB2 Alphablox 内部アーキテクチャー
- UI Model を含む、DB2 Alphablox サーバー・サイド・プログラム・モデル
- DB2 Alphablox アプリケーション・ロジック
- Blox UI Tag Library を含む Blox Tag Libraries
- DHTML Client API



Blox UI Model API は DB2 サーバー・サイド・プログラム・モデルの重要な部分で、PresentBlox、GridBlox、ChartBlox、DataLayoutBlox、および PageBlox を含むすべてのプレゼンテーション Blox によってサポートされます。サーバー・サイド `getBloxModel()` メソッドにより、アプリケーション開発者は特定の Blox インスタンスのモデルにアクセスできます。

Blox UI Model と述べるとき、実際にはコンポーネント、コントローラー、およびイベントの、 3 つの異なるユーザー・インターフェース概念を指します。これらのコンポーネントについて、以下に要約します。

概念　　説明

コンポーネント
ユーザー・インターフェースを構成する個々のコントロールおよびコンテナー。ボタン、リスト・ボックス、編集フィールド、グリッド、チャートなど。コンポーネントは、コンテナーの階層内に存在して、ユーザー・インターフェースを構造化します。結果として生じるモデルは、ユーザーに表示されるユーザー・インターフェースの論理表現です。

コントローラー
汎用のコンポーネント動作をアプリケーションで定義された動作に変換して、コンポーネントからイベントを生成するために使用します。たとえば、ユーザーがチェック・ボックスを選択するとチャートが表示されるとします。この場合、コントローラーはチェック・ボックスにチェックが入れられたことを、チャートを表示するためのシグナルとして解釈します。すべてのアプリケーション・ロジックは、モデル・コンポーネント (またはコンテナー) に付加された 1 つ以上のコントローラー内に存在する必要があります。

イベント
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、モデルの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。各コンポーネントおよびコントローラーには、それが認識および理解できる事前定義されたイベントのセットがあります。イベントが認識されると、通常はローカルに保管されたコンポーネントの状態が変更されます。アプリケーション・コードはイベントを使用して、ユーザー処置に基づくアプリケーション・ロジックをトリガーします。たとえば、ブラウザーで CheckBox の状態が変更された場合、`ClickEvent` が生成されてサーバーに送られます。アプリケーションのカスタム・コントローラーは `ClickEvent` を受け取ると、チャートの表示などの関連するアプリケーション動作を実行できます。

UI Model に関しては、以下の要点を覚えてください。

- このモデルは、クライアント上のユーザー・インターフェース・オブジェクトの状態をサーバー・サイドで表現したものです。これにより、サーバー・サイド Java コードは、コンポーネントのセットアップ、コンポーネント間の相互作用の処理、およびユーザー入力の処理を、実際のクライアント・サイド・コードを記述しなくても行うことができます。モデル自体には、実際には入力ボタンはありません。むしろそれは、ブラウザー上でユーザーに表示されているボタンの状態をサーバー・サイドのコードが制御できるようにします。たとえば、これによりサーバー・サイドのコードはユーザーがどのグリッド・セルを選択したかを判別できます。
- UI Model への直接のプログラミングは、オプショナルです。アプリケーション開発者がモデルと対話する必要があるのは、新しい機能を追加したい場合、または既存の機能を変更したい場合だけです。

Blox UI Model の目的

各プレゼンテーション Blox には、Blox の外観および動作を変更するための、一連の機能およびプロパティーがすでに備わっています。これらには、以下が含まれます。

1. **Blox タグ属性** (`bandingEnabled`、`chartFirst` など)
2. **Blox プロパティー**。メソッド呼び出しによって JSP スクリプトレット内で取り扱われます。
3. **Blox UI 修飾タグ**。バタフライ・レイアウト、圧縮されたヘッダーなど。

アプリケーション開発者が動作を組み込みたい場合、必要なことは、属性を Blox タグに追加するか、またはサーバー・サイド Java メソッドを呼び出すことだけです。

ときには、アプリケーション開発者が組み込まれたプロパティーには存在しない新規の機能を使用したい場合や、組み込まれた機能のインプリメンテーションを微調整したい場合があります。UI Model が導入される前は、変更をリクエストして、将来の製品リリースでそれが実現するまで待つしかありませんでした。

UI Model を使用すると、アプリケーション開発者は組み込まれた機能の動作を変更すること、および新規機能を UI に追加することができます。このようなカスタマイズの例としては、以下のものがあります。

- カスタムの計算操作を提供する新規のツールバーを追加する

- グリッドの外観を変更する
- チェック・ボックスまたは他のコントロールをグリッド・セルに追加して、ユーザーによる選択を処理するロジックを提供する
- ユーザーがクリックするユーザー・インターフェース・コンポーネントに応じて、右クリック・メニューを変更する

UI Model は、インプリメントが容易な組み込まれた製品機能に置き換わるものではありませんが、アプリケーション開発者はこれを使用して、必要なときに事実上無制限のカスタマイズを行うことが可能になります。

Blox UI コンポーネントの概説

モデル内のすべてのコンポーネントは、ユーザー・インターフェース上の特定のコントロールに対応します。これらのコントロールには、ボタン、グリッド、ツリー、チェック・ボックス、リスト・ボックス、チャート、メニュー、ツールバー、などがあります。UI Model 内のコンポーネントの状態を変更すると、ユーザー・インターフェース内のコントロールの状態が影響を受けます。同様に、ユーザーがユーザー・インターフェース内のコントロールと対話すると、UI Model はコントロールの状態を反映するように更新されます。

このモデルはすべてのコンポーネントの状態を維持するので、ユーザーがページをリフレッシュしても、サーバー・ベースのモデルはコンポーネントの状態を維持して、ページをリフレッシュする際にその状態を使用します。サーバー・ベースのアプリケーション・コードはいつでも、ユーザー・インターフェースで使用される任意のコンポーネントの状態を検査できます。状態に関する情報を別に保管または管理する必要はありません。たとえば、Checkbox コンポーネントが UI Model に追加された場合、`Checkbox.isChecked()` メソッドはチェックされた状態に関する最新の情報を提供します。

UI Model 内では、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。この階層は、1 つ以上の ComponentContainers を使用して可能になります。これにはコンポーネントおよび他の ComponentContainers を含めることができます。

結果として生じる階層は、単一のダイアログに関して以下のようになります。

Dialog ComponentContainer ComponentContainer CheckBox RadioButton RadioButton ComponentContainer Button Button

UI Model コンポーネントは、Blox の存続期間中のいつでも、変更、修正、追加、または削除できます。これにより、ユーザーがインターフェースと対話してオプションや機能を選択するときに、ユーザー・インターフェースが変更するようにすることができます。初期ページがブラウザーに送信された後にコンポーネントを変更するときは、開発者は以下のように `changed()` メソッドを呼び出す必要があります。

- コンポーネントが直接または間接に変更される (つまりそのスタイルが変更される) 場合、`changed()` はコンポーネント自体に対して呼び出す必要があります。
- コンポーネントが追加または削除される場合、`changed()` は親コンテナーに対して呼び出す必要があります。

初期ページがユーザーに送信される前に呼び出された場合、changed() メソッドはコンポーネントに対して何の影響も (積極的な影響も消極的な影響も) 与えません。

コンポーネント

モデル内のすべてのコンポーネントは、ユーザー・インターフェース上の特定のコントロールに対応します (非表示のものを除く)。これらのコントロールには、ボタン、グリッド、ツリー、チェック・ボックス、リスト・ボックス、チャート、メニュー、ツールバー、などがあります。UI Model では、Component はすべてのビジュアル・コンポーネントおよびコンテナーの基本クラスです。このように、すべてのビジュアル・コンポーネントは Component から派生します。Component クラスは、ビジュアル・コンポーネントが UI Model フレームワークに加わるために必要なプロパティーおよび動作の基本セットを提供します。同様に、スタイル、レイアウト、チャート軸定義などの、すべての非ビジュアル・モデル・オブジェクトは、Component 基本クラスからは生じません。

コンポーネント名

すべてのコンポーネントには、名前を割り当てることができます。名前は Component の存続期間内に変更可能ですが、ほとんどの場合は、一度設定されてからコンポーネントの存続期間内に変更されることはありません。この名前には、次の 2 つの目的があります。

1. この名前を使用して、コンポーネント、およびモデル内でのその役割を識別します。たとえば、各メニュー項目にはその特定の機能を識別する名前があります。コンポーネントに名前がない場合は、コンポーネントをモデル内のあちらこちらに移動する場合には特に、それらのコンポーネントの目的を識別することが大変に困難になります。名前付きコンポーネントであれば、モデル内で移動しても通常に操作できます。
2. コンポーネントの名前は、その「アクション・コード」となります。アクション・イベント (ClickEvent など) を生成するコンポーネントは、使用可能であればそのコンポーネントの名前を使用して、アクションをメソッドにマップします。これについては、後にコントローラーに関するセクションで詳しく説明します。

非固有のコンポーネント名の取り扱い

名前は固有である必要はなく、デフォルトの Blox モデルでは、いくつかの名前は複数の異なるコンポーネントで共有されています。これは複数のコンポーネントが同じ動作にマップしているときに便利であり、コンポーネントを異なるモデルの間で自由に移動させることも可能になります。

名前が固有であることは保証されていないので、コンポーネントを名前で検索するコードでは、複数の結果を処理するための準備が必要です。名前ベースの検索で複数の結果が生じる可能性を小さくするためには、可能な限りコンポーネント階層の深い位置から検索を開始します。たとえば、sort という名前のツールバー・ボタンを探す場合、モデルの最上位からではなく各ツールバー内から検索を開始してください。

以下の例は、同じ名前のコンポーネントをすべて検索して、それらに対して特定のアクションを実行する (この例では、それらを非表示にする) 方法を示しています。

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

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

最後に、名前が固有であってはいけないという明確な「規則」はありません。カスタム・モデルのコードでは、追加するコンポーネントの名前を固有にする命名規則に従うことにより、コンポーネント検索で複数の結果が生じることを心配しないようにすることが簡単にできます。

組み込まれた名前

Blox Model によって追加されたコンポーネントには、名前の標準セットがあります。関係のない機能に対して同じ名前を使用することは避けてください。すべての主な Blox コンポーネントの標準名は、「*Blox API Javadoc*」資料にある `com.Alphablox.blox.uimodel.ModelConstants` クラス・ファイル、および「*開発者用リファレンス*」の『Blox UI Tag リファレンス』セクションで定義されています。コード内で、ハードコーディング・ストリングを使用するのではなく、定義済みの定数を常に使用してください。

コンポーネント・タイトル

タイトルは、コンポーネントについてユーザーに説明するために使用します。たとえば、 CheckBox に追加されるタイトルは、ユーザーがチェック・ボックスにチェックを入れる理由を示すことになります。コンポーネントによってタイトルを使用する方法は多少異なりますが、その目的は同じです。各コンポーネント、およびタイトルが使用される方法のリストを以下に示します。

コンポーネント・タイプ
「タイトル」が使用される方法

Static 表示される値

ComponentContainer
最上位コンテナーのタイトル。その他の場合は無視される。

Checkbox
CheckBox の後に表示される

RadioButton
RadioButton の後に表示される

Edit 無視される

GroupBox
GroupBox のタイトル

ListBox、DropDownList
無視される

Image、StaticImage
無視される

Toolbar、Menu bar
メニュー内でツールバーを参照するために使用される。その他の場合は無視される。

Menu、MenuItem
メニュー・ラベル

Button ボタン・ラベル

Spacer 無視される

コンテナー

すべてのコンポーネントは、表示されるためにコンテナーに含まれている必要があります。コンテナーを階層に配置して、コンポーネントのセットをカプセル化すること、およびレイアウトを制御することができます。ComponentContainers は、コンテナー内のコンポーネントの検索やコンテナーの階層内すべてでのコンポーネントの検索などの、サービスを提供します。

ComponentContainers は Component の下層であり、コンテナーを他のコンテナーに追加すること、および基本 Component 機能を共有することを可能にします。たとえば、コンテナーには名前、UID、枠、および背景色があります。

コンテナー内のコンポーネントは、コンポーネントがコンテナーに追加された順序で表示されます。コンテナーのレイアウトは、この順序を解釈する方法を定義します。

レイアウト

ComponentContainer レイアウトは、コンテナー内に Component を表示するための方向の指定に限定されます。`VerticalLayout` をコンテナーに付加すると、複数のコンポーネントが縦に積み重ねられます。`HorizontalLayout` をコンテナーに付加すると、複数のコンポーネントが左から右へと表示されます。

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

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

複合コンポーネント

Model には、いくつかのコア・ユーザー・インターフェース Component が備わっています。しかし多くの場合、協調して機能する複数のコア・コンポーネントから構成される高水準 Component を作成する方が優れています。その後、これらの「複合コンポーネント」は他のコンポーネントと同様に取り扱うこと、および必要に応じて UI に追加することが可能になります。

複合 Component の作成は、`ComponentContainer` クラスを拡張して必要なユーザー・インターフェース Component を追加するだけの簡単な作業です。

以下に例を示します。

```
Class MyComponent extends ComponentContainer {
    public MyComponent() {
    add(new Static("label:"));
    add(new ListBox());
    }
// Deal with events and add custom behaviors
}
```

上記の MyComponent クラスは、コア Component クラスの場合と同様に簡単に、任意の ComponentContainer に追加できます。

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

コア Component の 1 つと同様に簡単に使用できる、動作の組み込まれた再使用可能なカスタム Component を作成するための、複合 Component を作成します。

ContainerBlox の使用

UI Model コンポーネントをページ上に表示するには、そのコンポーネントは Blox フレームの中に配置されている必要があります。ContainerBlox は基本的には、DB2 Alphablox アプリケーション・ロジックが事前定義されていない空の Blox フレームです。これは開発者が UI Model のユーザー・インターフェース・コンポーネントを使用してカスタム・ユーザー・インターフェースを作成するための領域を、ページ上に提供します。 ContainerBlox には事前定義された動作がないので、開発者はメニュー、ツールバー、グリッドなどの、必要なすべてのコンポーネントを手作業で追加しなければなりません。

アプリケーション開発者は、プレゼンテーション Blox によって提供されないカスタム・ユーザー・インターフェースをアプリケーションが必要とするときに、ContainerBlox を使用します。たとえば、ユーザーのナビゲーションを支援するために、ContainerBlox を使用して UI Model Tree コンポーネントをページ上に配置します。この例では、ツリー操作は 100% アプリケーションによって定義されるので、既存の Blox 動作を継承しないことをお勧めします。

ContainerBlox は、以下のように UI Model のある他の Blox と同様に使用できます。

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

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

またはその代わりに、ContainerBlox をサブクラスにして、 Model に基づく内蔵タイプのカスタム Blox コンポーネントを作成することもできます。

```
class MyComponent extends ContainerBlox
{
   public MyComponent( )
   {
          BloxModel model = myComponent.getBloxModel();
// Add user interface components and handlers
   }
}
```

その後、上記のクラスを以下のように <jsp:useBean> タグで使用できます。

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

コントローラー

コントローラーは、1 つ以上のコンポーネントの動作を定義するアプリケーション・ロジックを提供します。UI Model の基本コントローラー・クラスはさらに、イベントの処理を簡単にするいくつかのサービス、および標準のコントローラー動作を簡単にオーバーライドするフレームワークを提供します。どの Component または Component から派生するクラスも付加されたコントローラーを持てますが、すべての Component でコントローラーを必要としてはいません。ほとんどの場合、必要ないことが普通です。

Component がイベントを受け取ると、イベントはコンポーネントに付加された Controller にディスパッチされます。コントローラーが使用不可である場合、または付加されたコントローラーが (イベント・ハンドラーから `false` を戻して) イベントを転送するように指示する場合は、イベントはコンポーネントの親に送られます。この処理は、イベントが取り扱われるときまで、または最上位コンテナーの場合のようにコンポーネントに 親が存在しないのでイベントが単に無視されるときまで、繰り返されます。

コントローラーは通常、多数のコンポーネントの状態を単一のアクションに変換するアプリケーション・ロジックを提供するので、ほとんどの場合は個別のコンポーネントよりもコンテナーに付加されます。この優れた例は、多数のコンポーネントがダイアログに付加されたコントローラーによって制御される Dialog です。

ほとんどの場合、コンテナーには付加されたコントローラーがありますが、コンポーネントが専用のコントローラーを持つことを禁じるものはありません。以下の状況では、コンポーネントは通常、独自のコントローラーを持ちます。

- Component が特定のタスクを行うユーザー・インターフェースに追加されたものである。アプリケーション開発者が追加したメニュー項目およびツールバー・ボタンのほとんどは、このカテゴリーに属します。追加されたコンポーネントに対して、単にコントローラーを追加することは実際的です。
- Component はコントローラーのあるコンテナーの一部だが、コントローラーはその状態に影響を与える特殊な処理を行う。たとえば、編集コントロールには、特にユーザー入力を検証するコントローラーを持つことがあります。

Controller 基本クラス

すべてのコントローラー・クラスは、`Controller` クラスから下降したものでなければなりません。この基本クラスは、以下を含むいくつかのサービスを提供します。

- 受け取るすべてのイベントをメソッド呼び出しに変換する。コントローラーが受け取る各イベントによって、`public boolean handle`*EventType*`(`*EventType*`event) throws Exception` というフォームのメソッドが呼び出されます。`EventType` は、`SelectionChangedEvent` などの実際のイベント・クラスに置き換えてください。メソッドが存在しない場合、コントローラーはイベントを無視します。
- 1 次ユーザー処置イベントである `ClickEvents` を、ユーザーがクリックした Component の名前に基づくメソッド呼び出しに変換する。たとえば、myButton という名前の Button Component にある `ClickEvent` によって、メソッド `public void actionMyButton(ModelEvent event) throws Exception` が呼び出されます。このメソッドが存在しない場合、コントローラーはイベントを無視して、該当する次の機能があればそこに転送されます。

- コントローラーに関連したコンポーネントによって作成されたダイアログが閉じたとき、closedDialogName() メソッドを呼び出す。
- カスタム・コードが、コントローラーの組み込まれたイベント動作をオーバーライドするイベント・ハンドラーを追加できるようにする、インフラストラクチャーを提供する。

「暗黙の」コントローラー

いくつかのコア Component には、オーバーライドできない暗黙のコントローラーがあります。これらの暗黙のコントローラーは、内部状態の変更を処理して、付加されたコントローラーが Component の状態を調べる前に Component が正しい状態を反映するようにします。たとえば、チェック・ボックスにチェックが入れられるとき、 CheckBox コンポーネントは ClickEvent を受け取ると即時にチェックされた状態を反映して、その後、他のコントローラーが ClickEvent を受け取るようにします。

コントローラーがイベントを受け取るとき、そのコントローラーは Component に状態についての情報を安全に問い尋ねることができ、サーバー上の Component のモデルがクライアント上のコントロールの状態を正確に反映していることを保証します。

Blox UI Model のイベント

イベントを使用して、ブラウザーと UI Model との間で、コンポーネント状態の変更およびアクションについて通信します。また、UI Model 内部で使用して、コントローラーにモデルおよびプロパティーの変更について通知します。

UI Model イベントについての要点を、以下に示します。

- ほとんどのイベントは、細かいユーザー・インターフェース・アクションを伝えます。たとえば、ボタン・クリック、メニュー・クリック、スクロールなどです。
- イベントは、アプリケーション開発者のコードによってインターセプトできます。たとえば、コードはドリルダウン・メニュー項目でのユーザー・クリックをインターセプトできます。
- UI Model 内のイベントは、ユーザー・インターフェース・アクションに関与する点で JavaScript イベントに似ています。
- イベントは、そのイベントを生成したコンポーネントにディスパッチされてから、その親にディスパッチされます。
- さらに、イベントを使用して、モデル内のコントローラーとコンポーネントとの間で情報を伝送できます。
- これらの内部イベントによって、コードはモデル内のキー・アクションおよびユーザー・インターフェース・アクションをインターセプトできます。たとえば、グリッドがセルを作成するとき、それはコードによってセルをカスタマイズ可能にするイベントを生成します。

以下のように、イベントは、関連するすべてのコンポーネントおよびコントローラーがイベントの処理に関与できる特定の順序でディスパッチされます。

1. モデル・オブジェクト - イベントを生成する特定のモデル・コンポーネント (編集フィールドの内容変更など)。
2. モデル・オブジェクト・コントローラー - コンポーネントにコントローラーがある場合、そのコントローラーはイベントを受け取ります。
3. モデル・オブジェクトの親 - 上記のステップ 1 および 2 が、最上位コンテナーに到達するまで繰り返されます。
4. モデル・コントローラー - モデルの最上位コントローラーは、イベントの最後の停止点となります。コントローラーがイベントを処理しない場合、それは廃棄されます。

すべての場合に、イベント・ハンドラーは以下の 1 つを行うことができます。

- イベントを無視して、何も行わない。これにより、イベントのディスパッチは継続します。
- イベントを吸収して、追加イベントの生成などのアクションをオプションで実行する。以後のイベントのディスパッチングは行われません。
- イベントを変更して、ディスパッチが続くようにする。
- イベントに反応するが、ディスパッチは続くようにする。
- イベントを生成したコンポーネントに `getComponent()` を使用してアクセスする。
- イベントに固有の他のプロパティーにアクセスする。

以下の例は、Button Component からイベントにインターセプトする方法を示しています。それぞれの例で、コンポーネントは `MyButton` という名前のボタンであり、`blox` という名前の Blox に追加されています。ユーザーがボタンを押すと、すべてのボタンは `ClickEvent` を生成します。

コンポーネントへの専用コントローラーの追加

この例では、ボタン・クリックを処理するコントローラーをボタン自体に追加します。これにより、単一の Component がモデルに追加されるときには動作の追加が容易になりますが、複数の Component 間での調整が困難になります。コンポーネントにコントローラーがまだ含まれていないとき、または既存のコントローラーを置き換えたいときに、この方法でイベントを取り扱うことができます。

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

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

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

既存のコントローラーへのリスナーの追加

この例では、ボタン・ハンドラーを Blox モデル・コントローラーに追加します。ここでは、リスナーをモデル階層のより高い位置にあるコントローラーに追加します。イベントは、そのイベントを生じたコンポーネント (つまり Button コンポーネント) に送られてから、コンポーネント階層にパーコレートされます。

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

上記の例すべてで、イベント・ハンドラーは Blox タグの内側に追加されることに注意してください。ページがリフレッシュされるたびにハンドラーが追加されないようにしたいので、これは大切なポイントです。Blox タグ内のすべてのコードは、ページがリフレッシュされるたびではなく、Blox が最初に作成されるときだけに実行されます。 `<jsp:useBean>` がセッションを有効範囲として使用されるときにも、類似の規則があります。

モデル・ディスパッチャー

コンポーネントが Blox モデルに付加された後、そのコンポーネントを使用してモデル・ディスパッチャーを取得できます。ディスパッチャーは、モデル内の最上位コンテナーに備わっているサービス・ポイントで、いくつかのモデル関連のサービスを提供します。以下のサービスが、ディスパッチャーによって使用可能です (`BloxModel` および `IModelDispatcher` インターフェースを参照してください)。

- ダイアログの表示 - `showDialog()` を使用して、クライアントにダイアログを表示します。
- ダイアログのクローズ - `closeDialog()` を使用して、クライアント上のダイアログをクローズします。
- `getTopLevelContainer()` によって、最上位コンテナーへの参照を取得します。
- イベントのディスパッチング - `dispatchEvent()` を使用して、モデル内にイベントをディスパッチします。
- ブラウザー・ウィンドウの表示 - `showBrowserWindow()` によって、ブラウザーが指定の URL を使用する新規のウィンドウを表示するようにします。
- `sendClientCommand()` によって、JavaScript コマンドをクライアントに送ります。
- 右クリック・メニューの表示 - `setAttachedRightClickMenu()` を使用して、ブラウザーが指定の場所に右クリック・メニューを即時に表示するようにします。
- ビジー状態の制御 - `setBusy()` を使用して、コードがブラウザー UI を解放されるまでビジー状態にできるようにします。

ディスパッチャーの最も一般的な使用方法は、以下のようにダイアログを表示することです。

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

Model に付加されていないコンポーネントには、モデル・ディスパッチャーがありません。

ダイアログ

ダイアログを使用して、ユーザーからの入力を集め、オプションの設定またはユーザーの意思確認を行います。UI Model により、アプリケーション開発者はダイアログを素早く作成してユーザーに表示することが容易になります。Dialog は、以下の 2 つの特殊機能を追加することにより、基本 ComponentContainer モデル・オブジェクトを拡張するコンテナーです。

1. ダイアログは、ブラウザー上で独自の別個な、サイズ変更可能で移動可能なウィンドウ内に存在する。
2. ダイアログはオプションで、それが解消するまで他のユーザー・インターフェースが入力を受け入れないようにすることができる。

それ以外については、Dialog は他の ComponentContainers と同様に機能します。ほとんどすべての Dialog では、Controller がユーザーの選択を解釈してアクションを実行することも必要です。

ユーザーがダイアログに注目するように、Dialog.setModal(boolean) メソッドを使用してダイアログのモーダル・プロパティーを設定します。Modal Dialog は、それが解消するまでユーザー・インターフェースの他の部分との対話を禁止します。Modeless Dialog は、ユーザー・インターフェースの対話を禁止しません。これは、apply 機能のあるダイアログで最もよく使用されます。ユーザー・インターフェースを制御する最後に表示された Modal Dialog と共に、複数のダイアログを同時に表示できます。

ユーザーがブラウザー・ページをリフレッシュすると、ダイアログは再表示します。

簡単なダイアログの作成

Blox UI Model を使用して、ユーザーからの入力を収集するダイアログを作成できます。ダイアログを作成するには、以下の基本ステップに従ってください。

1. UI Model リソース・ファイルからダイアログを作成する。
2. そのダイアログとのユーザー対話すべてを取り扱うために、DialogController を拡張するコントローラーを作成する。ほとんどの場合、コントローラーに必要なのは「OK」、「キャンセル」、および「適用」だけです。
3. コントローラーを Dialog オブジェクトに付加する。
4. モデル・ディスパッチャーにダイアログの表示を指示する。

以下の例では、JSP ページは PresentBlox のメニュー・バーで「データ」の下に、「My Menu Choice」というラベルの付いたカスタム・メニュー項目を追加します。ユーザーが「My Menu Choice」をクリックすると、XML リソース・ファイルで定

義されたダイアログが表示されます。この簡単な例では、ダイアログはユーザーに「OK」または「キャンセル」で応答する質問を尋ねます。以下に、PresentBlox およびカスタム・メニュー項目を表示する JSP ページ、およびダイアログ・ウィンドウの定義に使用する XML リソース・ファイルの、2 つのファイルのコードを示します。 JSP ページは、XML リソース・ファイルがアプリケーションのルート・ディレクトリーに存在すると想定します。

JSP ページ (customDialog.jsp)

以下の JSP ファイルは、PresentBlox をページ上に表示します。カスタムの「My Menu Choice」メニュー・オプションが、「データ」メニューの一番下に追加されます。実行される内容を理解するには、JSP ファイル内のコメントを参照してください。

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

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

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

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

  public static class MyActionFilter implements IActionFilter {
    String dialogPath;

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

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

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

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

```
        throw e;
      }
    }
  }

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

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

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

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

XML リソース・ファイル (MyDialog.xml)

上の例では、リソース・ファイルを使用してダイアログを作成しました。代わりの方法として、ダイアログを形成する個々のコンポーネントを作成および追加して、ダイアログを作成することもできます。リソース・ファイルは、ダイアログ内の Component を言語ローカライズ可能な XML で表現したものであり、ダイアログ、メニュー・バー、およびツールバーを作成するための簡単な方法となります。

この例のダイアログ・リソース MyDialog は、以下のようになります。

```
<?xml version="1.0" ?>
<Dialog name="MyDialog"
   title="My Dialog"
   cache="false"
   modal="true"
```

```
   height="150"
   width="500"
   layout="vertical">
<Static title="Do you really want to do this?" />
<ComponentContainer layout="horizontal" alignment="center">
   <Button name="ok" title="OK" />
   <Button name="cancel" title="Cancel" />
</ComponentContainer>
<Spacer />
</Dialog>
```

任意のコア Model Component を、コンテナーの子としてリソース・ファイルに追加できます。リソース・ファイルについて詳しくは、「*開発者用リファレンス*」の『XML リソース・ファイル・リファレンス』セクションを参照してください。

MessageBox

MessageBox は、開発者が素早くそして簡単に、テキスト・ベースのメッセージをユーザーに表示できるようにする、簡単な API のあるモーダル Dialog です。MessageBox は、ユーザーからの簡単な入力を、 OK、Yes/No、Yes/No/Cancel、および OK/Cancel の応答の形式で収集できます。MessageBox は性質が常にモーダルなので、ユーザーはユーザー・インターフェースの他の部分と対話を続ける前に、これに応答しなければなりません。

アプリケーション・ロジックがユーザーに何かの状況について知らせる必要がある場合、MessageBox をユーザーに送り、ユーザーの応答は無視することができます。

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

アプリケーション・ロジックがユーザーの応答を知る必要がある場合、 MessageBox クラスによって提供されるコールバック・メカニズムを使用して、ユーザーの選択を知ることができます。この場合、MessageBox は IMessageCallback インターフェース上のメソッドを呼び出して、ユーザーの応答をアプリケーション・コードに伝えます。どのクラスでも、このインターフェースをインプリメントして、ユーザーが MessageBox に応答したときに通知を受け取ることができます。

MessageBox を表示して、応答ハンドラーを提供するクラスの例を示します。

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

}

// To invoke the above MessageBox

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

DHTML クライアント・アプリケーションのロジックおよび流れ

UI Model は、1 つの層に DB2 Alphablox、他の層にブラウザーと、分割されて複数の層に存在するユーザー・インターフェースを制御します。これは層となっている HTML の性質と似ていますが、重大な相違もあります。

UI Model は、ページをリフレッシュすることなく DHTML ユーザーを更新します。ユーザーが UI と対話するときに、ページまたはフレーム・セットの全体をリフレッシュすることなく、ページに対して変更が行われます。

UI Model は、サーバー上に UI の状態を維持するユーザー・インターフェースの表現を持ち、 UI に対するサーバー・ベースのプログラマチック・インターフェースを提供します。つまり、2 つの層は互いに同期している必要があり、サーバー上の変更はクライアントに、そしてクライアント上の変更はサーバーに反映される必要があります。

ユーザー・インターフェースが分割されて複数の層にあり、 HTTP プロトコルを基にしていることは、サーバー・サイド・コードが記述される方法、およびそれがユーザーのアクションを処理する方法に影響を与えます。ユーザーの応答は異なるスレッドで生じるため、そして各スレッドは限定されたサーバー・リソースであるために、サーバー・サイド・コードはユーザーの応答を待てません。このほとんどは、他のユーザー・インターフェースがメッセージ・ループ、コールバック、またはハンドラーを使用して行う方法と比較して、大きな相違はありません。UI Model では、コントローラーを使用してユーザー処置を処理します。

コードは通常、以下のように構造化されています。

スレッド 1 (ユーザーがユーザー・インターフェース上のボタンを押したと想定)

1. `ClickEvent` をインターセプトする
2. `Dialog` オブジェクトを作成する
3. ダイアログに必要なコンポーネントを入れる
4. ダイアログからのイベントを処理するコントローラーを、ダイアログに付加する
5. ディスパッチャーを使用してダイアログを表示する (この時点ではダイアログは実際に表示されるのではなく、可能なときにすぐに表示できるようにキューに入れられる)
6. `ClickEvent` ハンドラーから戻る

いくらかの時間が経過した後 ...

スレッド 2 (スレッド 1 で作成されたダイアログで、ユーザーが「OK」をクリックしたと想定)

1. ダイアログ・コントローラー内の「OK」ボタンの `ClickEvent` をインターセプトする
2. ダイアログ内のコンポーネントの状態に基づく操作を実行する
3. ダイアログをクローズする (この場合も、ダイアログは可能なときにすぐにクローズするようにキューに入れられる)
4. `ClickEvent` ハンドラーから戻る

この例では、Dialog を表示して入力を収集する場合を示しましたが、すべての Model コンポーネントで一般的な構造は同じです。ユーザー処置の処理とモデルの作成とは、常に異なるスレッドで行います。

DHTML クライアントはテーマ・ベース

UI Model はテーマによって動作します。DHTML クライアントによって使用されるすべての CSS クラス名は公開されているので、独自のカスタム・テーマを作成したいアプリケーション開発者はそれらをオーバーライドすることができます。UI Model のチャート・オブジェクトは、イメージであるためにテーマの CSS 設定を使用しないことに注意してください (95 ページの『チャート作成』を参照)。

CSS ファイルは編成されているので、色、フォント、フォント・サイズ、および背景イメージなどの共通ビジュアル属性を簡単に見付けて変更できます。これらの属性すべては、各テーマのディレクトリー内で、*themeName*_dhtml.css (coleman_dhtml.css など) という名前のファイルに存在します。CSS クラス名とその機能のリストについては、CSS テーマについての資料 (184 ページの『CSS テーマ』) を参照してください。

テーマのレイアウトは、レンダリングされる直前に各 Blox に適用されます。UI Model のレイアウトを劇的に変更するカスタム・コードがある場合、その Blox に対するテーマ・レイアウト・アプリケーションをオフにしてください。テーマ・レイアウト・アプリケーションをオフにしない場合、おそらくテーマのデフォルト・レイアウトが優先されて変更は元に戻ります。

setApplyThemeLayout(boolean) メソッドは、テーマのレイアウトの Blox Model への適用を制御します。これを「false」に設定すると、サーバーはレイアウトの適用を停止します。

スタイル

すべての UI Model コンポーネントは、開発者がスタイルを適用して、制御しているテーマによって提供されたデフォルト・スタイルをオーバーライドすることを許可します。コンポーネントに適用可能なスタイルには、前景および背景色、フォント、枠、および下線、太字、イタリックなどの他のテキスト属性があります。

UI Model スタイルは、制御しているテーマの CSS クラスで定義されたスタイルと協調して機能します。スタイルがコンポーネントに適用されるとき、スタイルに対して明確に設定された属性だけがコンポーネントに適用されます。設定されていない属性は、テーマが提供する値の継承を続けます。これにより、開発者はコンポーネントの前景および背景を設定することができ、フォントや枠などの他の属性について心配する必要がなくなります。

チャートはテーマの CSS スタイルの影響を受けないイメージ・ファイルに基づくので、チャート・コンポーネントは、少し異なるスタイル・メカニズムを使用します。チャートに特定のスタイルについて詳しくは、チャートについてのセクションを参照してください。

スタイルを UI Model コンポーネントに適用するには、以下のように、 Style オブジェクトを作成してそれをコンポーネントに設定します。

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

またはその代わりに、CSS に似た省略表現を使用して Style オブジェクトを作成することもできます。省略表現は CSS の限定されたサブセットであり、Blox モデルの Style オブジェクトによって明確にサポートされている CSS スタイルだけをサポートすることに注意してください。

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

Style オブジェクトに加えて、開発者は各 UI Model コンポーネントで使用するための特定の CSS クラスも指定できます。Blox モデルの Style オブジェクトによってサポートされていない CSS スタイルを適用したい場合は、これを行ってください。クラス指定は CSS ファイルに配置するか、または <Style> エレメントを使用して HTML ページに指定できます。

```
// Set the CSS class to be used by tha component

myComponent.setThemeClass("myCssClass");
```

この方法は、いくつかのコンポーネントに指定されたデフォルトの DHTML テーマ・クラスをオーバーライドする効力もあるため、注意して行ってください。さらに、いくつかのモデル・コンポーネントはテーマ・クラス名をコンポーネントの状態に基づいて変更します。たとえば、Disabled または Enabled を CSS クラス名に追加することなどです。

複数のテーマ・クラスの設定

コンポーネントに使用される CSS クラスを拡張したい場合、スペースで区切った複数のテーマ・クラス名を追加できます。たとえば、CSS クラスをすでにクラスが定義されているコンポーネントに追加するには、以下のように行います。

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

チャート作成

特に興味深い UI Model の機能に、チャート作成があります。アセンブラーがチャートを作成したり、既存のチャートを変更できるようにする、Chart コンポーネントがあります。このセクションの目的は、チャート UI Model の一般構造および共通タスクのいくつかの例を示して、アセンブラーがチャート作成の拡張性を備えて開始できるようにすることです。

チャート作成関連の鍵となる用語

DataSeries

データ系列は、それぞれが通常は単一のメンバーを表すデータ値のリストです。たとえば、Sales 対 Region (East、West、North、および South) のグリッドがある場合、これは 4 つのデータ値 (East、West、North、および South に各 1 つ) を持つデータ系列で、そのデータ系列の名前は ″Sales″ となります。その後、このデータをチャート上にプロットすると、4 つの点のある 1 つの線となります。異なるチャートは、異なるタイプのデータ系列

オブジェクトを受け入れます。棒グラフ、線グラフ、面グラフ、および円グラフで使用される 1 つのタイプのデータ系列は SingleValueDataSeries で、グラフ上の各ポイントは単一の値だけを表します。BarDataSeries と LineDataSeries の両方は、SingleValueDataSeries を拡張するものです。さらに、複数値のデータ系列 (ScatterDataSeries や BubbleDataSeries など) もあります。たとえば、ScatterDataSeries では各データ・ポイントに X および Y 値があります。いくつかのチャート・タイプは、1 つのデータ系列だけをサポートします。現在、PieChart は単一の円だけをサポートして、円ごとに 1 つのデータ系列だけとなります。ほとんどのチャート・タイプは、複数のデータ系列をサポートします。3 本の線のある線グラフを想像してください。それぞれの線は、 1 つのデータ系列 (たとえば、Sales、COGS、および Inventory など) を表します。それぞれの線には、4 つのポイント (East、West、North、South) があります。それぞれのデータ系列は、1 つ以上の軸に対してプロットされます。軸は、OrdinalAxis または NumericAxis のどちらかとすることができます。独自の BarChart をセットアップする方法については、棒グラフのコード例を参照してください。

OrdinalAxis

OrdinalAxis は、基本的にストリング・ベースのラベルを持つ軸です。これは、データのグループに対するラベルを含む軸です。たとえば、データ系列に値の 3、5、4、6 をプロットして、これらの値が East、West、South、および North の Sales 数によるものである場合、 OrdinalAxis に含まれるラベルは″East″、″West″、″South″、および″North″となります。これが Ordinal と呼ばれる理由は、データ系列の順序がラベルの順序と一致するためです。各ラベルは、基本的にはデータ・ポイントのバケットです。

NumericAxis

NumericAxis は、実際のデータ値 (データ系列からの) がプロットされる、チャート上の軸です。たとえば、値が 3、5、4、6 のデータ系列がある場合、NumericAxis の範囲は 0 から 10 (制御可能) となり、目盛りが 1 ごと (制御可能) に示されて、チャートの左側 (制御可能) に表示されます。

Chart コンポーネント

単一の ″Chart″ コンポーネントで、すべてのチャートを表すものはありません。チャート UI Model には、チャートのすべての基本クラス (`PieChart`、`BarChart`、`LineChart`、`ScatterChart`、`DialChart` など) に対して、異なるコンポーネントがあります。チャートの基本クラスごとに、少しずつ異なる API があります。たとえば、棒グラフでは、棒の枠スタイルや棒の幅など、いくつかのビジュアル・プロパティーを設定できます。これらのビジュアル・プロパティーと類似のものは、`LineChart` には存在しません (線には厚さはありますが、枠や幅はありません)。そのため、`BarChart` には `setBarBorder(borderStyle)` メソッドがありますが、`LineChart` にはありません。

すべてのチャート・クラスは、`ChartObject` から派生します。さまざまなチャート・クラスが同様に取り扱われる、他の論理グループもあります。たとえば、グリッド線をチャート領域に入れるための、すべての `RectangularChart` オブジェクトに対して機能するコードが存在することは可能です。

Chart UI Model を使用する際の大切なポイントの 1 つは、Chart オブジェクトを適切なタイプに割り振ることと関係があります (上の図を参照してください)。BarChart API に到達するには、以下のようにまず Chart オブジェクトが実際に BarChart であることを確認してから、それを適切に割り振ります。

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

  // Checks to see if the chart is actually a BarChart
  if (chart != null && chart instanceof BarChart) {
  BarChart barChart = (BarChart) chart;

  // Now we can use the specific BarChart APIs
  barChart.setBarWidth(20);
}
```

チャート・オブジェクトは最初から作成することもできますが (下記の BarChart の例を参照してください)、アセンブラーは ChartBlox から作成された既存の Chart オブジェクトを変更することが多くあります。これを行う最良の方法を例示するために、下記のコンテキスト変更 (右クリック) メニューの例を参照してください。この例では、カスタム・コンテキスト (右クリック) メニューを Chart オブジェクトに配置します。

チャート設定の制御

アセンブラーが構成できるいくつかの共通項目があります。特定の API に関する詳細は、DB2 Alphablox 管理ページのヘルプ・メニューから入手可能な「*Blox API Javadoc*」資料を参照してください。

NumericAxis

属性　説明

axisTitle
軸のタイトル。可能な場合に表示されます (円グラフでは表示されません)。

formatMask
目盛りと共に示される数値の表示方法についてのフォーマット・マスクを設定します。

scale　この軸の、最小値、最大値、およびステップ・サイズ値を設定します。

OrdinalAxis

属性　説明

axisTitle
軸のタイトル。可能な場合に表示されます (円グラフでは表示されません)。

labels　各目盛りの下に表示されるテキスト・ラベルを設定します。

DataSeries

属性　説明

seriesName
データ系列のタイトル。凡例に表示されます。

dataValues
データ値を設定します。

凡例

属性　説明

legendTitle
凡例のタイトルは、凡例項目のすぐ上に表示されます。

position
Right および Bottom と共に、TopLeft、TopRight、BottomLeft、BottomRight、および Center があります。 Center は、Legend をチャートの内部に配置します。

legendLayout
凡例項目の垂直または水平方向の表示。水平は、すべての凡例項目が 1 行に表示されます。垂直は、各行に 1 つが含まれます。

ChartTitle、Footnote、AxisTitle

これらはすべて、ChartStatic オブジェクトです。

属性　説明

displayText
表示されるテキスト。

tooltip
dwellLabelsEnabled がオンに設定されているときに表示されるツールチップ。

textStyle
前景色、フォント名、フォント・サイズ、およびフォント角度を制御できるようになります。

regionStyle
背景色/イメージ、枠の幅、枠タイプ、および枠の色を制御できるようになります。

チャートのイベント処理

チャート・コンポーネント自体には、他の Component と同じイベント処理のメカニズムがあります。しかし、エンド・ユーザーはチャートのさまざまな部分 (データ・ポイント、ラベル、凡例項目など) をクリックすることができ、これらのイベントは完成した Component ではないので、別個に取り扱われます (これらは Component クラスを拡張しません)。この場合、Axis、Legend、AbstractDataSeries、および ChartStatic に対するスーパークラスとなる ChartComponent があります (タイトル、脚注)。一度に 1 つの ChartComponent を Chart 上で選択できます。以下の例を使用して、アセンブラーはこの選択されたイベントを Chart の Controller (または任意の親 Controller) でインターセプトできます。この例で、特殊な属性 (Chart.EVENT_ATTR_TARGET) を含む SelectedEvents だけが処理されることに注目してください。このターゲットは、選択された ChartComponent です。

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

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

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

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

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

  SingleValueDataSeries dataSeries = (SingleValueDataSeries)
     chartComponent;

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

チャートのカスタム・コンテキスト (右クリック) メニュー

この例では、カスタムの右クリック・メニューを既存の Chart オブジェクトに付加します。エンド・ユーザーがチャート・データ・ポイント、軸ラベル、凡例、その他を右クリックすると、カスタム・コンテキスト (右クリック) メニューが表示されます。大切な点の 1 つは、エンド・ユーザーがデータ操作 (ドリルダウンなど) またはチャート・タイプの変更を行うたびに、Chart オブジェクトが完全に再作成されることです。これが生じるたびに、カスタムの右クリック・メニューは再付加されることが必要になります。 ComponentRebuiltNotify イベントは、いつでも Chart オブジェクトが再作成されるときに送信されます。 handleComponentRebuiltNotify イベント・ハンドラーは、これらの場合にコンテキスト・メニューを再付加します。 Chart オブジェクトに対するすべての変更を handleComponentRebuiltNotify() イベント・ハンドラー内で行うことは大切です。そうしない場合、チャート・タイプが変更するとき (またはデータ操作が行われるとき)、すぐにカスタマイズ結果が失われます。

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

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

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

  /*** Make the menu ***/

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

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

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

Blox UI Model の例

単一のツールバー

この例では、2 つのデフォルトの DHTML クライアント・ツールバーを結合して単一のツールバーにします。これにより、水平方向の幅は増加しますが、垂直方向のスペースを節約できます。完全に新規のツールバーを作成する代わりに、コードはナビゲーション・ツールバー・ボタンのすべてを標準ツールバーの最後に付加してから、空のナビゲーション・ツールバーを除去します。 UI Model コンポーネントは、一度に 1 つのコンテナー内にのみ存在できるので、コンポーネントをコンテナーに追加すると、それが古いコンテナーから除去されます。

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

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

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

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

コンテキスト (右クリック) メニューの使用不可化

この例では、Blox のグリッド上での右クリック・イベントをインターセプトして、右クリック・メニューを表示する代わりに、ユーザーにメッセージを表示します。グリッドにはすでにコントローラーが付加されているので、この例ではすべての RightClickEvents をインターセプトするイベント・ハンドラーをそのコントローラーに追加します。イベント・ハンドラーから true が戻されると、それ以後のイベントの処理を停止します (false が戻されると、他のハンドラーがイベントを処理できるようになります)。

右クリック・メニューを使用不可にする Blox プロパティーは存在しますが、この例は右クリック動作をカスタマイズするための基礎となるので重要です。たとえば、ハンドラーはセルのタイプ (行ヘッダー、列ヘッダー、データ) に応じて、またはユーザーが選択したセルの内容に応じて、右クリック・メニューを使用可能または使用不可にすることができます。

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

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

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

  // Return true to stop the processing this event
    return true;
```

```
    }
  });
%>
</blox:present>
```

カスタマイズされたコンテキスト (右クリック) メニュー

静的ヘッダーまたはセルの右クリック・メニューが提供されないと (つまり、getHeaderCellRightClickMenu() または getCellsRightClickMenu() がヌルを戻す場合)、グリッド Brix のコンテキスト (右クリック) メニューは、Blox コンポーネントのメニュー・バーにある「データ」メニューのコピーとなります。これは、PresentBlox または GridBlox のいずれかに新規作成されたグリッド Brix のデフォルトの動作です。全置換からデフォルト・メニューへの項目の追加まで、右クリック・メニューのカスタマイズには、いくつかの方法があります。タスクに対して最も良い方法を決める助けとして、以下の表を参考にしてください。

表 2. メニューのカスタマイズに取ることのできるソリューション

要件	ソリューション
デフォルトの右クリック・メニューを、現行のセル選択によって影響を受けない静的メニューで置き換えます。	setHeaderCellRightClickMenu() および setCellsRightClickMenu() メソッドを使用します。
デフォルトの右クリック・メニューを、現行のセル選択に基づく動的メニューで置き換えます。	RightClickEvent を GridBrixController レベルでインターセプトし、セル選択に基づいてメニューを提供します。右クリック・イベントのデフォルトの処理を回避するために、イベント・ハンドラーから必ず ″true″ を戻してください。右クリック・メニューの起動について詳しくは、IModelDispatcher インターフェースを参照してください。
項目が現行のセル選択に依存しないデフォルトの右クリック・メニューに、メニュー項目を追加します。	メニュー・バーの「データ」メニューにメニュー項目を追加します。これは 1 度実行するだけでよく、追加されたメニュー項目は右クリック・メニュー上で自動的に複製されます。
項目が現行のセル選択に依存するデフォルトの右クリック・メニューに、メニュー項目を追加します。	RightClickEvent を BloxModelController レベルでインターセプトし、IModelDispatcher インターフェースを使って現行の右クリック・メニューを取得し、カスタム・メニュー項目を追加します。

この例では、グリッド・ヘッダー・セルにカスタムの右クリック・メニューを設定し、グリッド・データ・セルに別のカスタムの右クリック・メニューを設定することにより、このデフォルトの動作をオーバーライドします。このメニュー項目は、選択されたセルのタイプおよびセルの値を示す MessageBox をユーザーに表示します。

この例で追加される右クリック・メニューは静的ですが (つまり、選択されたセルに基づいて変更されません)、メニュー項目を選択した結果としてのアクションは、表示されるメッセージが選択されたセルに応じて調整されるので、動的なもので

す。これを行うには、各メニュー項目に付加されたコントローラーは現行のグリッド・セル選択を調べて、その選択を使用してメッセージを調整します。

簡単にするために、この例では選択リストの最初のセルだけを調べます。複数のセルが選択された場合には、これは実際にクリックされたセルではないことがあります。

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

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

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

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

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

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

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

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

カスタムのグリッド・レイアウト

この例は、グリッド・セルの作成時に個別のグリッド・セルの内容を変更できる、カスタムのグリッド・レイアウトを作成する方法を示しています。この例では、レイアウトは Microsoft Analysis Services セル属性をツールのヒントとして各ヘッダー・セルに追加して、セルを変更します (この例は、 Microsoft Analysis Services データ・ソースに対してのみ機能します)。

カスタム・レイアウト・タグは、グリッドにフックするために必要な詳細のほとんどを処理するので、アプリケーション開発者はグリッド全体または個々のセルをカスタマイズできるようになります。これらの詳細には、グリッド再ビルド通知を処理すること、およびセル変更をグリッドの作成時に事前にではなく、セルが必要とされるときに取り扱うことが含まれます。

デフォルトでは、グリッド内のセルは (1) ユーザーがセルを含むページを要求するとき、または (2) サーバー・サイド・コードがセルをグリッドから要求するときまで、実際には作成されません。一般に、グリッドがすべてのセルを作成するようにしない方が優れています。

この例の最初の部分では、カスタム・レイアウト・クラスを Grid にフックする、<bloxui:customLayout> タグを例示します。タグはレイアウト・クラスのインスタンスを作成して、それをグリッドに取り付けます。さらに、レイアウト・メニュー上のレイアウトの外観を管理します。これは、ユーザーがレイアウトのオン/オフを切り替えるようにするオプション・フィーチャーです。また、タグはユーザーが保管したすべてのブックマークにあるユーザーの設定を保管します。

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

タグ内で className 属性によって指定されるクラスは、サーバーのクラス・パスに存在する必要があります。そこに存在しない場合、タグはレイアウト・クラスのインスタンスを作成できません。クラス自体は、com.alphablox.blox.uimodel.layout.AbstractLayout を拡張する必要があります。提供されるすべてのメソッドおよびサービスについての完全なリストは、「AbstractLayout」Javadoc 資料を参照してください。この例では、レイアウト・クラスはセル作成だけに注目して、グリッド作成は無視します。

例の 2 番目の部分は、AbastractLayout を拡張してすべての作業を実行する、実際のクラスです。 getFormatName() メソッドは、レイアウトの名前を戻します。これは、レイアウトがメニュー・システムに追加された場合にメニュー項目として使用されます。

実際の作業は、すべて layoutCell() メソッド内で行われます。グリッド・セルが作成されるたびに、layoutCell() メソッドが新規に作成されたセルを参照して呼び出されます。このメソッドは、セルがヘッダー・セルであるかどうかを調べて、そうであればメンバー属性情報を含むツールのヒントをセルに追加します。

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

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

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

    // (i.e. not added by another layout)
    if (!gridCell.isColumnHeader() && !gridCell.isRowHeader())
        return;
    Property[] properties = getHeaderCellProperties(gridCell,
```

```
       dataViewBlox);

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

    GridBrixCellModel cell = (GridBrixCellModel)gridCell;

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

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

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

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

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

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

    return properties;
    }
}
```

基礎となる結果セットへのグリッド・セルのマッピング

カスタム・グリッド・レイアウトのこの例は、ヘッダー・セルの固有名に関する情報およびデータ・セルの値を含む、ツールのヒントを各セルに追加します。この例は、次の 2 つの重要な概念を例示します。

1. グリッド・レイアウト・クラスは、JSP ページに直接入れることができます。これは、レイアウトが単一の Blox とだけ使用される予定であれば適切であることがあります。クラス・コードを JSP ファイルに入れると、すべてのレイアウトで開発デバッグ・サイクルが速くなります。しかし、この方法でレイアウトを開発したときは、デバッグの後でそれを別のクラス・ファイルに入れてください。
2. レイアウトは MDBResultSet を GridBrixModel で使用可能な UI Model 変換メソッドに対して使用して、UI Model グリッド・セルを MDBResultSet オブジェクトにマップします。

この例の最初の部分では、実際の JSP ページで定義されたクラスを参照する方法を示します。ほとんどの Web サーバーは、JSP ファイルがコンパイルされるたびに

クラス名を廃棄しますが、このコードはレイアウトのクラス名をクラス自体から直接取得します。アプリケーション開発者は、レイアウト・クラスが JSP ファイルに配置されているとき、クラス・パスについて考える必要はありません。

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

この例の 2 番目の部分では、レイアウトをインプリメントする Java クラスについて例示します。それぞれの UI Model グリッド・セルが作成されるとき、カスタム・レイアウトは findGridBrixCell() を呼び出して、作成されたセルに対応する MDBResultSet オブジェクトを取得します。このメソッドから戻される値は、 UI Model セルと一致する結果セット・オブジェクトに依存します (このオブジェクトは、Cell、TupleMember、または null になります)。 GridBrixModel は、結果セットからのセルを UI Model 内のセルにマップする方法も提供します。

カスタム・レイアウトをインプリメントするときには、 UI Model グリッド内でのセルの順序が実際の MDBResultSet でのセルの順序と一致しないことがあることに注意してください。これは、行ヘッダーを移動するバタフライ・レイアウトなどのレイアウトに特に当てはまります。

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

Javadoc の資料

完全な UI Model Javadoc 資料については DB2 Alphablox Information Center を参照してください。サーバー・サイド Javadoc 資料には、すべての UI Model クラス、およびそれらのメソッドがリストされています。「*Blox API Javadoc*」資料は、DB2 Alphablox 管理ページでヘルプ・メニュー・オプションをクリックすると参照できます。

第 11 章 DHTML Client API

このトピックでは、DHTML Client API について説明します。この API は、JavaScript メソッドを使用してサーバー・サイド・アプリケーション・ロジックおよび API への簡単なアクセスを可能にするとともに、オフロード・ナビゲーション、いくらかの UI 操作、およびサーバーからの入力検証により、アセンブラーがクライアント・サイド・スクリプトを強化してアプリケーションに値を追加できるようにします。

DHTML Client API の概説

DHTML Client 用に構築されたアプリケーションの中核となるロジックは、 UI モデル、スクリプトレット、Bean、および他のサポートするクラスなどの、サーバー・サイド・コンポーネントによって形成されます。その結果、DHTML Client API の中心的な役割は、大きな RPC ベースの API をクライアントに公開するよりも、サーバー・サイド・アプリケーション・ロジックおよび API への簡単なアクセスを可能にすることになります。さらに、オフロード・ナビゲーション、いくらかの UI 操作、およびサーバーからの入力検証により、アセンブラーがクライアント・サイド・スクリプトを強化してアプリケーションに値を追加できるようにします。

DHTML Client API は、JavaScript コード用にイベント処理、エラー処理、通信サービス、および RPC メカニズムなどのサービスを提供するクライアント・フレームワークです。

DHTML Client API の使用

DHTML Client API は、周囲のページにある HTML または JavaScript が、そのページにある 1 つ以上の Blox と対話したいときに使用されます。クライアント API の主な使用法には、以下のものがあります。

- **サーバー・サイド・アプリケーション・ロジックの呼び出し**: DHTML Client API は、サーバー・サイド Bean 上のメソッドを直接呼び出すためのメソッドを提供します。さらに、サーバー・サイド Bean からの戻り値がクライアントに戻されて、適切な JavaScript オブジェクトに変換されます。
- **イベント処理**: API を通して、アセンブラーはイベントをサーバーに送り、 UI とのユーザー対話をシミュレートすることができます。JavaScript を使用して、クリック・イベントなどのイベント・オブジェクトを作成することができます。また、イベントをサーバーに送るためのメソッドが備わっています。これにより、Blox フレームワークの外部にある HTML ボタンおよびその他のコントロールはユーザー対話をシミュレートできるようになります。さらに、イベントを使用してモデル内のコンポーネントの状態を変更することもできます。
- **イベントのインターセプト**: JavaScript メソッドは、ページ上のすべての Blox によって生成されたすべてのイベント用のリスナーとして登録できます。イベント・リスナーは、イベントを無視するように選択することも、イベントが正常に

処理されるように選択することもできます。JavaScript を記述して、UI の動作を変更したり、クライアント上でいくつかのユーザー選択を処理することができます。

- **変更のためのサーバーのポーリング**: クライアント API フレームワークは、サーバーと共に、すべての UI 更新およびクライアントとサーバーとの間の情報の転送を自動的に処理します。ただし、アセンブラーには変更のために明示的にポーリングする機能があります。これが最も多く役立つのは、アプリケーションがクライアント・フレームワークの外部で変更を行うときです。この一般的な例としては、他のフレームを介しての、または XMLHTTP オブジェクトなどの HTTP 通信機能を使用しての、サーバーとの通信があります。
- **サーバーまたはコミュニケーション・レイヤーから戻されるエラーの処理**: クライアント・フレームワークは JavaScript メソッドを呼び出して、サーバー・エラーおよび通信エラーを処理します。クライアント・コードは、これらのエラーを処理するための独自のエラー・ハンドラーを登録できます。

このセクションの後の方で、DHTML Client API の一般的な使用例を示します。

DHTML Client API のフレームワーク

クライアント・フレームワークは、BloxAPI オブジェクトおよび Blox オブジェクトの 2 つの主なオブジェクトから成ります。これらは UI を強化して、サーバーとの通信を扱います。いくつかの関連するユーティリティー・オブジェクトも使用可能です。

BloxAPI オブジェクト

BloxAPI JavaScript オブジェクトには、ページ上のすべての Blox によって使用されるいくつかの汎用サービスが含まれています。これはクライアントとサーバーとの間の通信サービス、および JavaScript コード用の便利な RPC メカニズムを提供します。フレームごとに 1 つだけ BloxAPI オブジェクトがあり、これがそのフレーム内のサーバーとすべての Blox との間の着信および発信トラフィックを制御しています。

BloxAPI オブジェクトは、以下を処理します。

- 変更のためのサーバーのポーリング
- イベントおよびエラー管理用の API の提供
- フレーム内のさまざまな Blox への変更内容のディスパッチング
- 通信エラーおよびサーバー・エラーの処理
- RPC アクセスのための API の提供
- イベント送信のための API の提供

Blox オブジェクト

フレーム内の各 Blox は、JavaScript Blox オブジェクトと関連しています。 Blox オブジェクトは、以下を担当します。

- サーバーからの変更通知に対する応答
- ビジー状態およびビジー標識に対する応答および処理

- 次の DB2 Alphablox 4.x 互換性メソッドの提供: isBusy()、updateProperties()、flushProperties()、call()、および setDataBusy()
- この Blox に関連したダイアログの取り扱いおよび管理
- この Blox に関連した右クリック・メニューの取り扱い

ユーティリティー・オブジェクト

BloxAPI および Blox オブジェクトに加えて、フレームワークはいくつかのユーティリティー・オブジェクトもサポートします。最も重要なユーティリティー・オブジェクトには、以下のものがあります。

- *xxx*Event – クライアントが発行可能な、モデル・イベントのタイプごとに固有のイベント。例: ClickEvent
- Grid - 可視の選択済みセルのリストなど、いくつかのグリッド・プロパティーに対する読み取り専用アクセスを提供します。
- Exception - サーバー例外をクライアントに伝達するためのオブジェクト。

イベントの送信

UI モデルは、ClickEvent など、クライアントによって発行されるいくつかのイベントを公開します。それぞれのイベントごとに、DHTML Client API は JavaScript オブジェクトを定義します。その結果、JavaScript を使用してイベント・オブジェクトを作成し、イベントをサーバーに送信できるようになります。これにより、Blox フレームワークの外部にある HTML ボタンおよびその他のコントロールはユーザー対話をシミュレートできるようになります。さらに、イベントを使用してカスタム・モデル・コンポーネントの状態を変更することもできます。たとえば、以下の HTML コードは ClickEvent を UID が UID であるモデル・コンポーネントに送ります。

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

クライアントに公開されるイベントのリストは、「*開発者用リファレンス*」を参照してください。

JavaScript からの Blox UI Model イベントの開始

UI Model イベントを生成し、JavaScript からサーバーに送り返すことが可能です。これを行うと、ページ上の通常の HTML コントロールによって、Blox の使用するインターフェースに対するユーザーのクリックをシミュレートすることが可能になります。たとえば、Blox のメニューをオフにしても、HTML ボタンがページ上に表示されてユーザーに対して UI 機能の一部が提供されるようにすることが可能です。

以下の例では、クリックされるとデータ・オプション・メニューが呼び出される HTML ボタンが作成されます。

```
<blox:present id="samplePresent"
  width="700"
  height="500">
</blox:present>
<%
```

```
/* In order to send an event from the client, we need the component's UID */
  BloxModel model = samplePresent.getBloxModel();
  Component component = model.searchForComponent(
    ModelConstants.DATA_OPTIONS );
    int uid = component.getUID();

%>

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

input エレメントの onclick イベント・ハンドラーは、 BloxAPI を使用してサーバーに ClickEvent を送信します。 ClickEvent は、ターゲット・コンポーネントの Blox 名および UID を取得する JavaScript オブジェクトです。 UID は動的に割り当てられるので、ページが要求されるときに、コードはそれをモデル内で検索する必要があります。

イベントのインターセプト

クライアント・サイドのイベントをインターセプトするためには、2 つの機能を使用できます。

1. JavaScript コードは、すべてのクライアント・サイド・イベントのためのリスナーを登録できます。つまり、ユーザーによるすべての動作がインターセプトされて検討され、無視されるか処理されるかのどちらかになります。この方法によって、UI とのほとんどすべてのユーザー対話を管理できます。たとえば、ユーザーがメニュー項目を選択するたびに、 Blox、メニュー項目の UID、およびメニュー項目の名前を含む ClickEvent が生成されます。
2. UI Model には、クリックされる動作の概念を持つ (つまり、その ClickEvent を生成する) ほとんどのモデル・コンポーネントに付加可能な ClientLink オブジェクトが備わっています。 ClientLink オブジェクトによって、ビュー・レイヤーはユーザーによるクリックを処理する際に、その動作をサーバーに戻すのではなく独自のロジックを使用するようになります。DHTML クライアントでは、ClientLink が添付された Component はクライアント上で JavaScript 呼び出しの形式で、または新規のブラウザー・ウィンドウを開く形式で処理されます。

クライアント・サイドのイベントのインターセプト

この例は、クライアント上の UI Model イベントをインターセプトする方法を示しています。開発者は、UI Model イベントがサーバーによる関与を必要としないクライアント・サイド動作を実行するときに、これを行います。

JavaScript eventHandler() 関数が、UI によって生成されたすべてのイベントに対して呼び出されます。ハンドラーはすべてのイベントを認識するので、特定の UI イベントをインターセプトするために、コードはイベントのタイプおよび宛先 UID (または名前) を調べる必要があります。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。

クライアント・サイド JavaScript コードの例を以下に示します。

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

ユーザー・インターフェースからの JavaScript の直接の呼び出し

クライアント上で UI によって生成されたすべてのイベントをインターセプトせずに、コンポーネントに指示を与えて、クライアント・サイド JavaScript を直接呼び出すことができます。この場合、コンポーネントは ClickEvent イベントをサーバーに送りません。

クリック可能なコンポーネントに JavaScript メソッドを割り当てるためのコードは、サーバー上にあります。以下の例は、オプション・メニュー項目を「データ」メニュー上で検索して、そのメニュー項目が JavaScript メソッドを呼び出すようにします (またはその代わりに、ブラウザー URL をロードするようにメニュー項目を設定することもできます)。

```
<blox:present id="samplePresent" width="700" height="500">

<%
  // Find the component
  BloxModel model = samplePresent .getBloxModel();
  Component component = model.searchForComponent(
    ModelConstants.DATA_OPTIONS );

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

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

「データ・オプション」メニュー項目がクリックされると、クライアントはイベントをサーバーに送るのではなく、 JavaScript myJavaScriptFunction() 関数を呼び出します。 JavaScript 関数はページ上に定義済みであることが前提です。

例外処理

callBean を使用してサーバー・サイド・コードを呼び出すとき、サーバー・サイド・メソッドに例外をスローする可能性がある場合は、コードに Java 例外を処理するための準備が整っている必要があります。以下のように、クライアント Bean myBean 上に exceptionThrower() メソッドが指定された場合、結果を処理する前に、JavaScript コードは戻り値を調べて例外がスローされたかどうかを判別する必要があります。

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

DHTML Client API を使用するサーバー・サイド・ロジックの呼び出し

サーバー・サイド・ロジックを DHTML クライアントから呼び出すためには、基本的に 3 つのメソッドがあります。どのメソッドを選択するかは、どれだけの処理をサーバーによって自動化したいかに依存します。それらのメソッドを、自動化の小さいものから自動化の大きいものへの順序で、以下にリストします。

BloxAPI.call() および Blox.call() メソッド

これは DB2 Alphablox 4 で使用可能であったものと同じ call() メソッドです。これにより、サーバー上で URL を呼び出して、引数を URL パラメーターとして渡すことが可能になります。このメソッドは、rmi.jsp を呼び出して、自動的な引数の引き渡しおよび Bean メソッドの呼び出しを行うことができます。 JavaScript に戻される値は、解析されて、必要なデータ・タイプに変換されなければなりません。

たとえば、以下のコードでは bloxAPI.call() メソッドが使用されて、データ・レイアウト・パネルの可視性を切り替える Bean (MyBean) 上のメソッドを呼び出します。

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

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

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

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

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

BloxAPI.callBean() メソッド

このメソッドは、上で説明した call メソッドと rmi.jsp との組み合わせに似ている、サーバー上の Java Bean を呼び出します。これは以下の点で、その組み合わせと異なっています。

- これはサーバーと直接協働することにより、Bean メソッドを検索して呼び出します。他の JSP ファイルは関与しません (つまり、rmi.jsp は不要です)。
- 発信するメソッド引数のデータ・タイプを指定できます。
- 戻り値は、実際の JavaScript オブジェクトに変換されます。
- 引数および戻り値として、最も簡単なデータ・タイプおよび配列がサポートされます。

上の例で callBean を使用するには、call() 呼び出しを以下のものと単に置き換えてください。

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

clientBean (<blox:clientBean>) タグ

Blox clientBean タグ (<blox:clientBean>) は <blox:header> タグの内側にネストさせることが可能です。その結果、サーバーは指定の Java Bean (または Blox) に対する JavaScript オブジェクトを生成します。上の例で <blox:clientBean> を使用するには、クライアント Bean タグを Blox ヘッダーに組み込みます。その時点で、開発者は以下のように通常の JavaScript メソッドを呼び出すことができます。

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

<script>

// Use ClientBean to invoke method on the bean

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

}
</script>
```

重要: この例では、<blox:clientBean> タグに指定されたメソッドはありません。その結果、Bean 内のパブリック・メソッドごとに JavaScript メソッドが生成されます。これにより、メソッドの数が 2、3 個より多い Bean では、大きなオーバーヘッドが生じることがあります。そのため、アセンブラーは JavaScript が生成される対象となるメソッドを明示的にリストすること、およびアセンブラーがメソッド数を最小に保つことが勧められています。

<blox:clientBean> の使用に関する唯一の制限は、渡される引数および戻される引数が、サポートされる引数をプリミティブおよび配列に効果的に限定する JavaScript によってサポートされている必要があるということです。

<blox:clientBean> のサーバー・サイド Blox コンポーネントとの使用

<blox:clientBean> タグを使用して、クライアントからもサーバー・サイド Blox にアクセスできます。 PresentBlox の JavaScript オブジェクトを生成できる方法の例を、以下に示します。

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

重要: この例では、2 つのメソッドが公開されています。ほとんどの SSPM Blox オブジェクトで使用可能なメソッドの数が与えられている場合、使用されるメソッドがヘッダー内の clientBean タグに明示的にリストされている必要があります。

ヘッダー内でサーバー・サイド Blox を使用するには、 visible="false" を指定して Blox を定義してから、 <blox:display> タグを使用して Blox を HTML 本体に提供します。

サーバー・サイド Blox が clientBean と共に使用されると、以下の特別な処理が実行されます。

- クライアント上の Bean の名前には、最後に API が追加されます。これは、サーバー・サイド Blox の種類が何であれ行われます。上の例では、実際の JavaScript オブジェクトの名前は myPresentBloxAPI となります。これが行われるのは、ほとんどの場合、DHTML クライアントによって追加されたページ上にすでに JavaScript オブジェクトが存在するためです。
- 実用性のために、DHTML クライアントが API で終わる Blox 名の JavaScript オブジェクトを見つけた場合、開発者はメソッドをメインの DHTML クライアントの Blox オブジェクトから直接呼び出すことが可能になります。そのため、クライアント Bean が myPresentBloxAPI と呼ばれていても、 myPresentBloxAPI Blox オブジェクト上のメソッドを直接呼び出すことができます。たとえば、myPresentBlox.setChartFirst() および myPresentBloxAPI.setChartFirst() の両方がチャートを最初に設定します。
- サーバー・サイド Blox に DataBlox または他のネストされた Blox が、たとえば PresentBlox 内などにある場合、別のクライアント Bean セクションを作成しなくても、クライアント上のネストされた Blox にアクセスできます。これを行うには、data、grid、chart、dataLayout、toolbar、page が接頭部となっている親 Blox のリストにメソッドを追加します。メソッドを呼び出すには、適切な getter をメイン Blox と共に使用します。たとえば、myPresentBlox.getDataBlox().connect() などとします。

以下の例は、組み込み Blox および API サフィックスの使用方法を示す完全な JSP ページです。 data.setQuery が GridBlox のクライアント Bean セクションにあり、その DataBlox メソッドを JavaScript コードで使用可能にしていることに注意してください。

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

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

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

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

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

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

DHTML Client DOM API

Internet Explorer DOM は、DHTML Client によって広範囲に使用されます。クライアントは、ユーザーがクライアントと対話する際に DOM の各部分を更新します。これは DHTML クライアントのインプリメンテーションの一部なので、 DHTML によって作成される DOM オブジェクトおよび属性は、将来のバージョンで変更されます。

重要: インプリメンテーションが変化していく可能性があるので、開発者は、DHTML クライアントによって生成された DOM を操作または全探索するようなクライアント・サイドのコードを書くべきではありません。

DHTML Client DOM API は、315 ページの『第 26 章 DHTML Client DOM API』 に含まれています。

複数のフレームの使用

DHTML クライアントはアプリケーション内の各フレームを、別個のエンティティーとして取り扱います。つまり、<blox:header> タグを含む各フレームは、独自のクライアント API フレームワーク BloxAPI オブジェクトを持つことになります。サーバーおよびクライアント API に関しては、別のフレーム内にある Blox は別のブラウザー内にある可能性があります。

別のフレーム内の Blox は別のエンティティーとして取り扱われるため、以下の場合には予期しない結果が生じることがあります。

1. 複数の異なるフレーム内の Blox が、1 つの共通の DataBlox を参照する。この場合、1 つのフレーム内の Blox に対して実行されるドリル操作または他のナビゲーション操作によって、別のフレーム内に存在する同じ DataBlox に依存する Blox が即時に更新されることはありません。実際には、これはほとんどまたはまったく生じることがありません。
2. 1 つのフレーム内で実行されるサーバー・サイド・コードが、別のフレーム内の Blox を変更するか、またはそれに影響を与える。

どちらの場合も、変更を生じているフレーム内の Blox だけが即時に更新されます。他のフレーム内の Blox は、それらのフレームが自動ポーリングを実行するまで更新されません。

この状況が生じる場合、デフォルト状態の自動ポーリングではすべてのフレーム内の Blox を更新するために 2 分間もの時間がかかるため、十分ではありません。推奨されるいくつかのオプションは、以下のとおりです。

1. 影響を受ける Blox のあるフレームごとに、BloxAPI オブジェクトを使用して手動のポーリングを実行する。
2. ポーリング・タイマーをデフォルトの値よりも小さくして、間隔を速くする。
3. Blox を単一のフレーム内に保持するか、または異なるフレーム内の Blox が同一の DataBlox に依存しないようにして、その状況を全体的に回避する。

ページのリフレッシュ

DHTML クライアントは、HTML エレメントの内容を変更することにより、ページをリフレッシュしないで更新します。そのため、ユーザーがクライアントと対話する際に、HTML は新規の情報を表示するために常に変更を続けます。ただし、ブラウザーはこれらの HTML 変更をトラッキングしません。その代わりに、ブラウザーはページが最初に要求されたときに受け取った HTML をキャッシュに入れます。さらに、ソースの表示を実行すると、ブラウザーは初期ページの HTML も表示します。

重要: DHTML クライアントで生じる増分的なページ更新により、ブラウザーの「ソースの表示 (View Source)」オプションによって表示される HTML ソース・コードは、ブラウザーの現行の状態と通常は一致しません。これにより、デバッグが困難になる場合があります。

ユーザーがページをリフレッシュしたり、ブラウザーの「戻る」ボタンを使用してページに戻ると、ブラウザーはページのキャッシュされたバージョンを復元します。DHTML クライアントに対する変更はサーバー上で保守されているので、Blox の最新の表現がユーザーに対して表示されていることを確実にするために、クライアントおよびサーバーにはこの状況を検知して処理するためのメソッドがあります。このメカニズムの副次作用は、ページをリフレッシュしたり「戻る」ボタンを使用するときに、Blox の最初の状態が一時的に表示されてから現在の状態に更新されるということです。

第 12 章 サーバー・サイドのイベント・フィルターおよびイベント・リスナーを使用したイベントのキャプチャー

サーバーでイベントを処理する*前*か*後* のいずれかに、データ分析イベントをキャプチャーし、カスタム・アクションを実行できます。たとえば、ユーザーがメンバーをドリルダウンする場合、サーバーでイベントを処理する前に、そのユーザーにアクションを実行する権限があるかどうかの確認を行えます。そのようにすると、必要な場合にはそのアクションをキャンセルできます。このようにしない場合には、誰かがデータベースの特定の重要な部分にドリルダウンするたびに経理部門に E メールを送信することもできます。イベント・フィルターはサーバー・サイドのオブジェクトで、イベントをサーバー上で実際に処理する*前* に、ドリルダウンやピボットなどのユーザー・データ分析イベントをキャプチャーできます。イベント・リスナーを使用すると、ユーザー・イベントを処理した*後*に通知を受けることができます。

イベント・フィルターの重要な特徴の 1 つは、このフィルターは、アクションが発生したものの実際にそのイベントが処理される*前* に起動されるので、アプリケーションはアクションが発生する前にキャンセルできるという点にあります。たとえば `DrillDownEvent` は、ユーザーがドリルダウンするメンバーをクリックすると発生しますが、それはデータベースでドリルダウン・アクションが実行されて新規データがクライアントに戻る前のことです。一方、イベント・リスナーを使用すると、サーバーでイベントが正常に完了した後に追加アクションを実行できます。たとえば、選択的非表示イベントの完了後に、別の Blox を更新し、イベントの副次作用である例外を処理したり、イベントの結果に基づいてクライアントにメッセージを返送することを望むかもしれません。こうしたことは、イベント・リスナーを使用すると行えます。イベント・リスナーは、イベントがエラーなしで完了したときに限り起動します。

イベント・フィルター・オブジェクト

イベント・フィルター・オブジェクトはサーバー・サイドのオブジェクトであり、ドリルダウンやピボットなどの特定のユーザー・イベントをキャプチャーして、イベントが実際に処理される*前* にアクションを実行できます。

イベント・フィルター・オブジェクトには 2 つのタイプがあります。

- DataBlox 関連: 次のデータ分析操作をキャプチャーできます。縮小、ドリルダウン、ドリルスルー、ドリルアップ、拡張、選択的非表示、選択的保持、メンバー選択、ピボット、選択的除去、すべて表示、選択的表示、軸の交換、およびデータ照会。
- ブックマーク関連: 以下のブックマーク関連イベントをキャプチャーできます。ブックマークの削除、ブックマークのロード、ブックマークの名前変更、およびブックマークの保管。

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する必要があ

ります。イベント・フィルターを DataBlox、PresentBlox、または他のユーザー・インターフェース Blox に追加すると、対応するイベント・フィルター・オブジェクトをインプリメントする独自のクラスを作成し、イベントが実際に処理される前にとるアクションを指定できます。

イベント後に処理を実行するには、イベント・リスナーを使用してください。イベント・リスナーの詳細と、これら 2 つの使用法の比較については、『イベント・リスナー・オブジェクト』を参照してください。

イベント・リスナー・オブジェクト

イベント・リスナー・オブジェクトはサーバー・サイドのオブジェクトであり、イベントの処理後に、ドリルダウンやピボットなどの特定のユーザー・イベントについて通知して、アクションを実行できます。 イベント・リスナー・オブジェクトには 3 つのタイプがあります。

- DataBlox 関連。次のデータ分析操作の完了をキャプチャーできます。縮小、ドリルダウン、ドリルスルー、ドリルアップ、拡張、選択的非表示、選択的保持、メンバー選択、ピボット、選択的除去、すべて表示、選択的表示、軸の交換、およびデータ照会。
- ブックマーク関連。以下のブックマーク関連イベントの完了をキャプチャーできます。ブックマークの削除、ブックマークのロード、ブックマークの名前変更、およびブックマークの保管。
- ChartBlox 関連。ユーザーがページ・フィルターを変更する際にイベントをキャプチャーできます。

Blox ユーザー・インターフェースからの軸の交換などユーザー起動イベントが完了すると、対応するイベント・リスナーが通知されます。イベント・リスナーを使用するには、共通 Blox メソッド `addEventListener()` を使用して、最初に特定のイベント・リスナー・オブジェクトを追加する必要があります。イベント・リスナーを DataBlox、PresentBlox、または他のユーザー・インターフェース Blox に追加すると、対応するイベント・リスナー・オブジェクトをインプリメントする独自のクラスを作成し、イベントの完了時にとるアクションを指定できます。

イベント前の処理を実行するには、イベント・フィルターを使用してください。2 つの使用法の比較については、127 ページの『イベント・フィルターと比較したイベント・リスナー』を参照してください。

イベント・フィルターおよびイベント・リスナーの使用

イベント・フィルター・オブジェクトは、`com.alphablox.blox.filter` パッケージの一部です。このオブジェクトを使用する JSP ファイルの先頭部分で、以下の JSP インポート・ステートメントを使用しなければなりません。

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

このパッケージには、多様なイベントのフィルター用インターフェースが含まれています。キャプチャーする特定のイベントをインターセプトするため、こうしたインターフェースをインプリメントするクラスを定義する必要があります。このようなインターフェースの名前は、`BookmarkDeleteFilter`、 `DrillDownFilter`、`ExpandFilter`、および `HideOnlyFilter` などのように、すべて `Filter` という語で

終わります。これらのフィルターには、独自のアクションを指定するためにインプリメントできる、bookmarkDelete()、drillDown()、expand()、および hideOnly() などの対応するメソッドがあります。このようなすべてのメソッドには、影響を及ぼす入力として、対応するイベント・オブジェクトが必要です。こうしたイベント・オブジェクト名は、BookmarkDeleteEvent、 DrillDownEvent、ExpandEvent、および HideOnlyEvent などのように、すべて Event という語で終わります。

イベント・リスナー・オブジェクトは、com.alphablox.blox.event パッケージの一部です。このオブジェクトを使用する JSP ファイルの先頭部分で、以下の JSP インポート・ステートメントを使用しなければなりません。

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

このパッケージには、多様なイベントのリスナー用インターフェースが含まれています。イベント・リスナーの使用法は、イベント・フィルターの使用法とよく似ています。完了を通知する特定のイベントをインターセプトするため、こうしたインターフェースをインプリメントするクラスを定義する必要があります。こうしたインターフェースの名前は、BookmarkDeleteListener、DrillDownListener、ExpandListener、および HideOnlyListener などのように、すべて Listener という語で終わります。これらのリスナーには、独自のアクションを指定するためにインプリメントできる、bookmarkDelete()、drillDown()、expand()、および hideOnly() などの対応するメソッドがあります。このようなすべてのメソッドには、影響を及ぼす入力として、対応するイベント・オブジェクトが必要です。こうしたイベント・オブジェクト名は、BookmarkDeleteEvent、 DrillDownEvent、ExpandEvent、および HideOnlyEvent などのように、すべて Event という語で終わります。

たとえば、ドリルダウン操作を実行するユーザーを許可するかどうかを確認する場合、以下のようにする必要があります。

1. メソッド addEventFilter(*YourDrillDownEventFilter*) を使用して、DataBlox にサーバー・サイドのドリルダウン・イベント・フィルターを追加します。

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

   </blox:present>
   ```

   上記の例では、DDFilter はドリルダウン・イベント・フィルター・オブジェクトの名前です。
2. ドリルダウン・イベント・フィルター・オブジェクトに DrillDownFilter インターフェースをインプリメントします。

   ```
   <%!
   public class DDFilter implements DrillDownFilter
   {
      //more code here....
   }
   %>
   ```
3. drillDown メソッドが呼び出される際にとるアクションを追加します。このメソッドは、入力として DrillDownEvent オブジェクトをとります。

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

// drillDown is the method to implement to capture a drilldown
// events. It takes a DrillDownEvent object as input.
   public void drillDown( DrillDownEvent dde ) throws Exception
    {
      DataBlox blox = dde.getDataBlox();
      StringBuffer msg = new StringBuffer("DRILL DOWN event on " +
blox.getBloxName() + "\n");
      msg.append("With Axis ID: " + dde.getAxisIndex() + ", ");
      msg.append("Nest level: " + dde.getNestLevel() + ", ");
      msg.append("Member index: " + dde.getMemberIndex() + ", and ");
      msg.append("TupleMember: " + dde.getMember().getDisplayName());
      MessageBox msgBox = new MessageBox(msg.toString(), "DrillDown Filter
Message", MessageBox.MESSAGE_OK, null);
      model.getDispatcher().showDialog(msgBox);
    }
}
%>
```

Blox カスタム・タグ内での追加および除去メソッドの配置

ページが再ロードされるたびに新規イベントが追加されないようにするには、JSP ページの Blox カスタム・タグ内に、addEventFilter() メソッドを使用してコードを配置します。たとえば、以下のコードにより Blox が作成され、ユーザーがメンバーをドリルダウンするごとに呼び出されるフィルターが追加されます。

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

<blox:present id="myPresent">
   <blox:data .../>

<%
   myPresent.getDataBlox().addEventFilter(new DDFilter() );
%>

</blox:present>
```

ページが再ロードされるたびに新規イベントが追加されないようにするには、JSP ページの Blox カスタム・タグ内に、dvven(FrLtsr(F() メソッドを使用してコードを配置します。たとえば、以下のコードにより Blox が作成され、ユーザーがメンバーを (選択的) 非表示にするたびに呼び出されるリスナーが追加されます。

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

<blox:present id="myPresent">
   <blox:data .../>
...
<%
   myPresent.getDataBlox().addEventListener(new HideOnlyHandler() );
%>
</blox:present>

<%!
  public class HideOnlyHandler implements HideOnlyListener
  {
    public void hideOnly( HideOnlyEvent hoe)
    {
```

```
    ...// custom actions here
    }
  }
%>
```

完全な drillDownEventFilter の例

この完全な例は、ドリルダウン・イベントの起動時に MessageBox UI モデル・コンポーネントを使用して、ドリルダウン・アクションをインターセプトし、出力を作成する方法を示しています。

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ page import="com.alphablox.blox.filter.*,
                 com.alphablox.blox.uimodel.core.MessageBox,
                 com.alphablox.blox.uimodel.BloxModel,
                 com.alphablox.blox.DataBlox" %>
<html>
<head>
<blox:header/>
</head>

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

<%!
  public class DDFilter implements DrillDownFilter
  {
  BloxModel model;
  public DDFilter(BloxModel model) {
    this.model = model;
  }
  // drillDown is the method to implement to capture a drilldown
  // event. It takes a DrillDownEvent object as input.
  public void drillDown( DrillDownEvent dde ) throws Exception
  {
    DataBlox blox = dde.getDataBlox();
    StringBuffer msg = new StringBuffer("DRILL DOWN event on " +
blox.getBloxName() + "\n");
    msg.append("With Axis ID: " + dde.getAxisIndex() + ", ");
    msg.append("Nest level: " + dde.getNestLevel() + ", ");
    msg.append("Member index: " + dde.getMemberIndex() + ", and ");
    msg.append("TupleMember: " + dde.getMember().getDisplayName());
    MessageBox msgBox = new MessageBox(msg.toString(), "DrillDown Filter
Message", MessageBox.MESSAGE_OK, null);
    model.getDispatcher().showDialog(msgBox);
    }
  }
%>
```

addEventFilter() メソッドを Blox カスタム・タグ内に配置すると、ページを再ロードするたびに複数のフィルターを追加しなくてもよくなります。上記の例では、作成されたクラスでは、現行のドリルダウン・アクションに関する情報を含むメッセージ・ダイアログ・ボックスが、ドリルダウン・イベントが生じる前に表示されます。

同じイベントには希望どおりの数のフィルターを追加でき、イベントがキャンセルされない限りは、追加した順序でイベントが処理されます。

この例は、Blox Sampler の『データとの対話 (Interacting with Data)』セクションにあります。

完全な drillDownEventListener の例

この完全な例は、MessageBox UI モデル・コンポーネントを使用して、ドリルダウン・アクションの発生時に通知を出し、出力を作成する方法を示しています。

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

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

<body>
<blox:present id="myPresent2">
  <blox:data
    dataSourceName="QCC-Essbase"
    query="!" />
  <% myPresent2.getDataBlox().addEventListener( new
SimpleListener(myPresent2.getBloxModel()) );    %>
</blox:present>

</body>
</html>

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

    public void drillDown( DrillDownEvent event ) throws Exception
    {
      DataBlox blox = event.getDataBlox();
      StringBuffer msg = new StringBuffer("DRILL DOWN event on " +
blox.getBloxName() + "\n");
      msg.append("With Axis ID: " + event.getAxisIndex() + ", ");
      msg.append("Nest level: " + event.getNestLevel() + ", ");
      msg.append("Member index: " + event.getMemberIndex() );

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

同じイベントには希望どおりの数のリスナーを追加でき、追加した順序で処理されます。

イベント・フィルターと比較したイベント・リスナー

イベント・リスナーはイベントが正常に完了したことを通知するために使用され、イベント・フィルターはイベントが処理される前に、サーバーがイベントを受け取るとそのサーバー上のイベントをインターセプトするために使用されます。イベント・リスナーのインプリメンテーションとイベント・フィルターのインプリメンテーションはよく似ています。以下の表に、この 2 つの類似点および相違点の要約を示します。

	イベント・リスナー	**イベント・フィルター**
通知を受け取る時期	イベントの処理後	イベントの処理前
パッケージ	`com.alphablox.blox.event`	`com.alphablox.blox.filter`
パッケージ内のインターフェース	すべてのインターフェースは、`DrillDownListener`、および `RemoveOnlyListener` などのように、`Listener` という語で終わる。	すべてのインターフェースは、`DrillDownFilter`、および `RemoveOnlyFilter` などのように、`Filter` という語で終わる。
インプリメントするメソッド	リスナーには、対応するイベント・オブジェクトを引数として取る (例: `drillDown(DrillDownEvent)` `removeOnly(RemoveOnlyEvent)`) `drillDown()`、および `removeOnly()` などの対応するメソッドがあります。	フィルターには、対応するイベント・オブジェクトを引数として取る (例: `drillDown(DrillDownEvent)` `removeOnly(RemoveOnlyEvent)`) `bookmarkLoad()`、 `drillDown()`、および `removeOnly()` などの対応するメソッドがあります。
イベント	同一のイベント・オブジェクトは、イベント・フィルター内でも同じ名前が付けられます。	同一のイベント・オブジェクトは、イベント・リスナー内でも同じ名前が付けられます。しかし、そのようなイベントには、イベント・リスナー内のイベントにはない `cancelEvent()` および `isCanceled()` メソッドがあります。

指定のイベントに対してイベント前およびイベント後のいずれかの処理を行うイベント・ハンドラーを作成できます。以下に例を示します。

```
<%!
  public class DDHandler implements DrillDownFilter, DrillDownListener
  {
    public void drillDown(DrillDownEvent event) throws Exception {
      // actions to take before the event is processed
    }

    public void drillDown(com.alphablox.blox.event.DrillDownEvent event) {
      // actions to take after the event has been processed
    }
  }
%>
```

しかし、イベント・オブジェクトはイベント・フィルター・パッケージとイベント・リスナー・パッケージの両方で同じ名前を有しているため、イベント前とイベント後の 2 つの処理を行うために同じクラスを使用する場合には、パッケージ情報が含まれる完全なクラス名を指定してください。

イベント・フィルター用にインプリメントするメソッド

イベント・フィルターを作成するには、以下にリストされている 1 つ以上のイベント・フィルター・メソッドをインプリメントするクラスを作成する必要があります。以下の表では、キャプチャーするイベント、そのイベントをキャッチするためにインプリメントするメソッド、およびそのフィルター・イベントでサポートされるメソッドをリストしています。

(ユーザーのアクション実行時に) キャプチャーするイベント	インプリメントするインターフェース	使用可能なイベント・メソッド
ブックマーク: 削除	BookmarkDeleteFilter 内の `bookmarkDelete(BookmarkDeleteEvent)`	BookmarkDeleteEvent メソッド
ブックマーク: ロード	BookmarkLoadFilter 内の `bookmarkLoad(BookmarkLoadEvent)`	BookmarkLoadEvent メソッド
ブックマーク: 名前変更	BookmarkRenameFilter 内の `bookmarkRename(BookmarkRenameEvent)`	BookmarkRenameEvent メソッド
ブックマーク: 保管	BookmarkSaveFilter 内の `bookmarkSave(BookmarkSaveEvent)`	BookmarkSaveEvent メソッド
縮小	CollapseFilter 内の `collapse(CollapseEvent)`	CollapseEvent メソッド
データのソート	DataSortFilter 内の `dataSort(DataSortEvent)`	DataSortEvent メソッド
ドリルダウン/すべて拡張	DrillDownFilter 内の `drillDown(DrillDownEvent)`	DrillDownEvent メソッド
ドリルスルー	DrillThroughFilter 内の `drillThrough(DrillThroughEvent)`	DrillThroughEvent メソッド
ドリルアップ	`DrillUpFilter` 内の `drillUp(DrillUpEvent)`	DrillUpEvent メソッド
拡張	ExpandFilter 内の `expand(ExpandEvent)`	ExpandEvent メソッド
選択的非表示	HideOnlyFilter 内の `hideOnly(HideOnlyEvent)`	HideOnlyEvent メソッド
選択的保持	KeepOnlyFilter 内の `keepOnly(KeepOnlyEvent)`	KeepOnlyEvent メソッド
メンバーの選択 (たとえば、メンバー・フィルター内の)	MemberSelectFilter 内の `memberSelect(MemberSelectEvent)`	MemberSelectEvent メソッド
ピボット	PivotFilter 内の `pivot(PivotEvent)`	PivotEvent メソッド
データ照会	QueryFilter 内の `query(QueryEvent)`	QueryEvent メソッド
選択的除去	RemoveOnlyEvent 内の `removeOnly(RemoveOnlyEvent)`	RemoveOnlyEvent メソッド
すべて表示	ShowAllFilter 内の `showAll(ShowAllEvent)`	ShowAllEvent メソッド
選択的表示	ShowOnlyFilter 内の `showOnly(ShowOnlyEvent)`	ShowOnlyEvent メソッド
軸の交換	SwapAxisFilter 内の `swapAxis(SwapAxisEvent)`	SwapAxisEvent メソッド

イベント・リスナー・オブジェクト用にインプリメントするメソッド

イベント・リスナーを作成するには、以下にリストされている 1 つ以上のイベント・リスナー・メソッドをインプリメントするクラスを作成する必要があります。以下の表では、キャプチャーするイベント、そのイベントをキャッチするためにインプリメントするメソッド、およびそのフィルター・イベントでサポートされるメソッドをリストしています。

(ユーザーのアクション実行時に) キャプチャーするイベント	インプリメントするインターフェース	使用可能なイベント・メソッド
ブックマーク: 削除	BookmarkDeleteFilter 内の `bookmarkDelete(BookmarkDeleteEvent)`	BookmarkDeleteEvent メソッド
ブックマーク: ロード	BookmarkLoadFilter 内の `bookmarkLoad(BookmarkLoadEvent)`	BookmarkLoadEvent メソッド
ブックマーク: 名前変更	BookmarkRenameFilter 内の `bookmarkRename(BookmarkRenameEvent)`	BookmarkRenameEvent メソッド
ブックマーク: 保管	BoomarkSaveListener 内の `bookmarkSave(BookmarkSaveEvent)`	BookmarkSaveEvent メソッド
ChartBlox でのフィルター変更	ChartPageListener 内の `changePage(ChartPageEvent)`	ChartPageEvent メソッド
縮小	CollapseListener 内の `collapse(CollapseEvent)`	CollapseEvent メソッド
データのソート	DataSortListener 内の `dataSort(DataSortEvent)`	DataSortEvent メソッド
ドリルダウン/すべて拡張	DrillDownListener 内の `drillDown(DrillDownEvent)`	DrillDownEvent メソッド
ドリルスルー	DrillThroughEvent 内の `drillThrough(DrillThroughEvent)`	DrillThroughEvent メソッド
ドリルアップ	DrillUpListener 内の `drillUp(DrillUpEvent)`	DrillUpEvent メソッド
拡張	ExpandListener 内の `expand(ExpandEvent)`	ExpandEvent メソッド
選択的非表示	HideOnlyListener 内の `hideOnly(HideOnlyEvent)`	HideOnlyEvent メソッド
選択的保持	KeepOnlyListener 内の `keepOnly(KeepOnlyEvent)`	KeepOnlyEvent メソッド
メンバーの選択 (たとえば、メンバー・フィルター内の)	MemberSelectListener 内の `memberSelect(MemberSelectEvent)`	MemberSelectEvent メソッド
PDF へのデータのエクスポート	PdfListener 内の `pdf(PdfEvent)`	PdfEvent メソッド
ピボット	PivotListener 内の `pivot(PivotEvent)`	PivotEvent メソッド
データ照会	QueryListener 内の `query(QueryEvent)`	QueryEvent メソッド
選択的除去	RemoveOnlyListener 内の `removeOnly(RemoveOnlyEvent)`	RemoveOnlyEvent メソッド
すべて表示	ShowAllListener 内の `showAll(ShowAllEvent)`	ShowAllEvent メソッド
選択的表示	ShowOnlyListener 内の `showOnly(ShowOnlyEvent)`	ShownOnlyEvent メソッド

(ユーザーのアクション実行時に) キャプチャーするイベント	インプリメントするインターフェース	使用可能なイベント・メソッド
軸の交換	SwapAxisListener 内の `swapAxis(SwapAxisEvent)`	SwapAxisEvent メソッド

第 13 章 データへの接続

DB2 Alphablox アプリケーションを実用的に使用するためには、まず最初のタスクとして、データ・ソースに接続する必要があります。このセクションでは、データ・ソースの作成、データ・ソースへの接続、およびデータ・ソースへのアクセスの管理方法について詳しく学びます。

データ・ソースの作成

分析アプリケーションを実用的に使用する前に、ユーザーによって表示および分析可能なデータにアクセスする必要があります。最初に行う必要のあるタスクの 1 つは、DB2 Alphablox 管理ページでデータ・ソース定義を作成することです。これらのデータ・ソース定義は接続先となるリレーショナルまたはマルチディメンション・データベースを指定して、それらに素早く接続し、結果セットを検索することを可能にします。

データ・ソース定義の作成は管理用タスクに近いものですが、サーバー管理者または開発者のどちらでも、管理権限があれば行うことができます。DB2 Alphablox データ・ソースを定義するために開発者または管理者が行う必要のあるタスクについて、以下に簡潔に説明します。

注: 「開発者用ガイド」および Blox Sampler アプリケーションで使用されるすべての例では、QCC-Essbase (DB2 OLAP Server および Essbase 用) または QCC-MSAS (Microsoft Analysis Services 用) のどちらかの QCC データベースが使用されます。QCC をインストールして構成するには、DB2 Alphablox CD で次のサンプル・データ・ディレクトリーの下にある `readme.txt` ファイルを参照してください。

```
<cdromDir>/sampledata/qcc/
```

データ・ソースの定義

データ・ソースを定義することには、以下のステップが含まれます。

1. 「スタート」メニューを使用するか、または Web ブラウザーに以下の URL を入力して、 DB2 Alphablox ホーム・ページにアクセスします。

   ```
   http://<serverName>/AlphabloxAdmin/home/
   ```

2. 管理者権限のあるユーザー名およびパスワードを使用して、ログインします。3 つのタブのある DB2 Alphablox 管理ページが表示されて、「アプリケーション」ページがデフォルトで示されます。
3. 「管理」タブをクリックします。その後、「データ・ソース」をクリックして、使用可能なデータ・ソース定義のリストを表示します。
4. データ・ソースを定義するには、既存のデータ・ソース定義の下にある「作成」ボタンをクリックします。(アプリケーションに必要なデータ・ソース定義がすでに存在する場合、これら残りのステップは省略できます。)

5. 「データ・ソースの作成 (Create Data Source)」パネルの項目を完成させます。援助が必要な場合は、このページにある「ヘルプ」ボタンをクリックします。
6. 「保管」をクリックして、新規の定義を保管します。新規に定義されたデータ・ソース名が、使用可能なデータ・ソース定義のリストに表示されるはずです。

データ・ソースについての完全な説明と、サポートされるマルチディメンションおよびリレーショナル・データベース用にそれらを定義するためのステップについての詳細は、「*管理者用ガイド*」の『新規データ・ソースの定義』セクションを参照してください。

DataBlox dataSourceName プロパティーの定義

DataBlox は、スタンドアロンとして使用される場合でもネストされた Blox として使用される場合でも、プレゼンテーション Blox と適切なデータ・ソースとの間の接続を管理するために使用されます。 DataBlox は、照会のサブミット、およびデータ・ソースからの結果セットの検索も行います。 DB2 Alphablox 管理ページでデータ・ソースを定義した後に、適切なデータベースにアクセスするための情報を入手できる場所について DataBlox に知らせる必要があります。 DataBlox にデータ・ソースを位置指定するには、 DataBlox `dataSourceName` プロパティーを使用します。

DataBlox にデータ・ソースを位置指定するためには、次の 2 つの手法を使用できます。

- DataBlox `dataSourceName` 属性の設定。
- DataBlox `setDataSourceName` メソッドの設定。サーバー・サイド Java メソッド、または (DHTML Client API を使用して) サーバー・サイド・メソッドを呼び出す JavaScript を使用します。

dataSourceName 属性の設定

データ・ソースを定義するための最も頻繁に使用される技法は、 `dataSourceName` を属性として追加して、その値を設定することです。その値は、 DB2 Alphablox 管理ページで定義済みのいずれかのデータ・ソースの名前でなければなりません。

たとえば、以下のコード例で、ネストされた DataBlox はデータ・ソースを `QCC-Essbase` に設定します。

```
<blox:present id="myPresent" ...>
...
  <blox:data
    dataSourceName="QCC-Essbase"
    query='<SYM <ROW("All Products")
      <COLUMN ("All Time Periods") "2000" Sales !' />
</blox:present>
```

注: `dataSourceName` 属性を DataBlox に追加していない場合、データ・プレゼンテーション Blox は「使用可能なデータはありません」のメッセージを表示します。または、データ・ソースが定義されていない場合、JSP ページは正常にコンパイルされないので、例外が生成されることになります。

setDataSourceName() JavaScript メソッドの使用

たとえばユーザーがボタンをクリックしたときなどに、JavaScript または Javaを使用して、プログラマチックにデータ・ソースを変更したい場合があります。以下の例は、 JavaScript setDataSourceName メソッドを使用する方法を示しています。

DataSourceSelectFormBlox を使用する異なるデータ・ソースの設定

以下のステップに従って、DB2 OLAP Server および Essbase データ・ソースの選択リストを持つ JSP ページを作成します。データ・ソースが選択されると、使用可能なディメンションを DataLayout パネルにロードしてユーザーが随時分析を実行できるようにする、デフォルトのクエリーが実行されます。この例の完全なバージョンは、Blox Sampler の『FormBlox の使用 (Using FormBlox)』セクションの下にある『随時分析の例 (Ad Hoc Analysis example)』にあります。以下の例では DB2 OLAP Server および Essbase バージョンを使用しますが、 Microsoft Analysis Services バージョンも同様に機能します。

1. ページの上部に、使用可能にする必要のある Java クラスを指定する JSP page ディレクティブを追加します。

```
<%@ page import="com.alphablox.blox.form.FormEventListener,
    com.alphablox.blox.DataBlox,
    com.alphablox.blox.form.FormEvent" %>
```

2. page ディレクティブの下に、このページで使用される Blox タグ・ライブラリーのための taglib ディレクティブを追加します。この例では、それらのライブラリーは標準 Blox タグ・ライブラリーおよび Blox Form タグ・ライブラリーです。

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

3. 使用される DataBlox を指定することにより、別名のメンバー名を使用可能にして (DB2 OLAP Server および Essbase のみ)、開始時にデータ・ソースに接続しないように DataBlox に指示します。

```
<blox:data id="AdHocDataBlox"
  connectOnStartup="false"
  useAliases="true" />
```

4. PresentBlox を指定します。

```
<blox:present id="AdHocPresentBlox"
  visible="false"
  width="600"
height="350">
  <blox:grid noDataMessage="Select a data source" />
  <blox:chart noDataMessage="Select a data source" />
  <blox:data bloxRef="AdHocDataBlox" />
</blox:present>
```

   共通の noDataMessage 値は、デフォルトの「使用可能なデータはありません」メッセージよりも良い「 データ・ソースを選択してください (Select a data source)」に設定しました。さらに、ネストされた DataBlox タグは、事前に定義済みの DataBlox を使用するように指示しています。

5. ここで、使用可能な DB2 OLAP Server および Essbase データ・ソースのリストを自動的に生成する DataSourceSelectFormBlox を追加できます。

```
<bloxform:dataSourceSelect id="dataSourceSelector"
  type="MDB"
  adapter="IBM DB2 for OLAP"
  visible="false"
  nullDataSourceLabel="Select the data source">
<%
  dataSourceSelector.addFormEventListener(new
     DataSourceFormBloxEventListener(AdHocDataBlox));
%>
</bloxform:dataSourceSelect>
```

type 属性は、マルチディメンション・データ・ソースだけを指定していることを示し、 adapter 属性設定は、データ・ソースを DB2 OLAP Server または Essbase だけに限定します。さらに、ページのロード時にデータ・ソースが指定されるようにするのではなく、 nullDataSourceLabel オプションを追加して、ユーザーに「データ・ソースを選択してください (Select the data source)」と通知できます。

また、ネストされた Java スクリプトレットは DataSourceSelectFormBlox に、ページの下部に含まれる FormEventListener を追加する必要があることを指示します。このイベント・リスナーによって、ユーザーが選択するまでデータ・ソースを指定することなく、 DataBlox を作成できるようになります。

6. 以下の <blox:display> タグを使用して、ページのレイアウトを行い、DataSourceSelectFormBlox および PresentBlox が表示される場所を指定します。

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

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

ページのレイアウトを行う完全なコードについては、Blox Sampler の例を参照してください。

7. 最後に、FormBloxEventListener クラスを追加します。

```
<%!
  public class DataSourceFormBloxEventListener
     implements FormEventListener {

    private DataBlox dataBlox;

    public DataSourceFormBloxEventListener(DataBlox dataBlox) {
      this.dataBlox = dataBlox;
}

    public void valueChanged(FormEvent event) throws Exception {

      String dataSourceName = event.getFormBlox().getFormValue();
      dataBlox.setDataSourceName(dataSourceName);

      if (dataSourceName != null) {
        dataBlox.setQuery("!");
        dataBlox.updateResultSet();
      }
      else
        dataBlox.disconnect(true);
      }
   }
%>
```

この DataSourceFormBloxEventListener クラスは、データ・ソースの FormBlox 値を取得して、デフォルトのクエリーを設定します。これにより、PresentBlox の DataLayout パネルに使用可能なすべてのディメンションが取り込まれます。

注: FormBlox または DataBlox プロパティーおよびメソッドの構文および使用方法について詳しくは、「*開発者用リファレンス*」を参照してください。

データ・ソースとの接続および切断

スタンドアロンまたはネストされた DataBlox がインスタンス化されると、暗黙の `connect` メソッドが呼び出されます。 DataBlox にクエリーが指定されている場合、そのクエリーは実行されて、結果セットが生成されます。DataBlox `connectOnStartup` プロパティーが `false` に設定された場合 (デフォルトは `true`)、`connect` メソッドは呼び出されないので、後にプログラマチックに接続することが必要になります。

Blox がそのデータ・ソースとの接続を確立すると、その接続は現行セッションを通して存続します。このデフォルトの動作はパフォーマンスに関して最適のものであり、アプリケーションがクエリー (初期クエリーおよびユーザーが Blox と対話することから生じるクエリーを含む) ごとにデータベース接続のオープンとクローズを繰り返すことを防止します。

タスクに応じて、データ・ソースと DataBlox プロパティーおよびメソッドとの接続を管理するために使用可能な、いくつかの異なるオプションがあります。それらについて、以下に要約します。

ゴール	DataBlox プロパティーおよびメソッド	結果
データ・ソースに接続するが、クエリーは実行しない	`<blox:data ... connectOnStartup="true" ... </blox:data>` [注: `query` 属性は設定されていない]	• 結果セットは取得されない • ユーザーには、カスタマイズ可能な、共通の Blox `noDataMessage` プロパティーのメッセージ (デフォルト: 「使用可能なデータはありません」) が表示される
	`connect(false);`	• 接続がすでに存在する場合、切断してから再接続する • 接続が確立される • 定義されたクエリーは、実行されない • ユーザーには、カスタマイズ可能な、共通の Blox `noDataMessage` プロパティーのメッセージ (デフォルト: 「使用可能なデータはありません」) が表示される

ゴール	DataBlox プロパティーおよびメソッド	結果
データ・ソースに接続して、クエリーを実行する	`connect();` または `connect(true);` [注: クエリーは設定済みと想定する]	• 定義されたテキスト形式のクエリーが実行される • 結果セットが取得される
	`setQuery(); updateResultSet();`	• クエリーが設定され、接続が確立されて、結果セットが取得される

接続プロパティーの変更に基づいて結果セットを更新する	`// Change properties first`
	
	`updateResultSet();`
		• 結果セットのプロパティーの変更を適用した後 (`useAliases` を `true` または `false` に設定した後など) に、結果セットを更新する
		
		[注: 接続プロパティー (`dataSourceName`、`username`、`schema`、および `password` など) を適用する場合は、代わりに `connect` メソッドを使用します。]

注: これらの DataBlox プロパティーおよびメソッドの構文および使用方法について詳しくは、「*開発者用リファレンス*」を参照してください。

クエリーを設定してから接続するための Java スクリプトレットの例を、以下に示します。

```
<%
  String query = "<ROW (\"All Products\") <ICHILD \"All Products\" "+
  "<COLUMN (\"All Time Periods\") <CHILD \"All Time Periods\" "+
(Measures) Sales !";

  PresentBlox3.getDataBlox().setQuery(query);
  PresentBlox3.getDataBlox().connect();

%>
```

注: Blox Sampler の『データの取り出し (Retrieving Data)』セクションにある『JSP スクリプトレットを使用する初期クエリー (Initial Query Using JSP Scriptlet)』の例では、この技法を実際に示します。

Blox がデータ・ソースに対していつ接続または切断するかを、プログラマチックに制御したい場合もあります。たとえば、ユーザーが選択リスト、ラジオ・ボタン、チェック・ボックスなどを使用していくつかの選択を行ってからボタンをクリックして表示要求をサブミットするような、ページを設計することがあります。これをインプリメントする方法は多くあります。たとえば、デフォルトのビューをロードして HTML フォーム要素または FormBlox にデフォルト値をプリセットする方法や、ユーザーが選択を完了するまで Blox にビューをロードしない方法などです。これを行う方法を示す例については、以下の 137 ページの『自動接続および自動切断』を参照してください。

自動接続および自動切断

以下に説明するように、DataBlox autoConnect および autoDisconnect プロパティーを特定の状況下でリレーショナルおよびマルチディメンション・データ・ソースと共に使用して、 DB2 Alphablox 分析アプリケーションのパフォーマンスおよびスケーラビリティーをより良く管理することができます。

autoConnect および autoDisconnect 属性の構文および使用法について詳しくは、「*開発者用リファレンス*」の『DataBlox』セクションを参照してください。

リレーショナル・データ・ソース

使用可能なポート数が限られていて、リレーショナル・データ・ソースを使用している場合、autoConnect および autoDisconnect プロパティーを DataBlox に設定することにより、アプリケーション接続の使用を管理できます。以下の表は、autoConnect および autoDisconnect プロパティーの可能な設定値の組み合わせすべてと、その結果となる動作について要約しています。

autoConnect	autoDisconnect	動作
false	false	これらは DataBlox 上でのデフォルト設定です。定義されたクエリーは、定義されたデータ・ソースに対して、暗黙の connect メソッドを使用して実行されます。接続が確立された後、それは現行のブラウズ・セッションを通して維持されます。
true	true	使用可能なデータベース・ポートの数が限定されている場合にのみ推奨されます。初期データベース接続が確立されてクエリーが実行された後に、結果セットが戻されて表示され、データベースから自動的に切断します。 Blox 上のユーザー対話の多くはこのサイクルを繰り返すので、接続を再確立する必要が生じることに注意してください。
true	false	これは実際にはデフォルト動作と同じです。
false	true	初期の結果セットが表示された後、ユーザーは結果セットに対して操作を行うことができません。設定値のこの組み合わせは可能ですが、通常は推奨されません。

マルチディメンション・データ・ソース

DataBlox autoConnect プロパティーはマルチディメンション・データベースに影響を与えませんが、autoDisconnect プロパティーは Microsoft Analysis Services データ・ソースと共に使用して、分析アプリケーションのスケーラビリティーおよびパフォーマンスを管理することができます。

Microsoft Analysis Services データ・ソースのみに関しては、 autoDisconnect プロパティーを true に設定すると、クエリーの実行、ドリルアップとドリルダウン、ピボット、「選択的保持」と「選択的除去」の使用を含む、クエリーの実行の直後にデータ・ソース接続が切断されます。メタデータ呼び出しは、影響を受けませ

ん。各切断の後に、 PivotTable Services キャッシュ・メモリーが java.exe プロセスからクリアされて、 DataBlox は直前の接続情報を使用して即時に再接続します。

autoDisconnect プロパティーを Microsoft Analysis Service と共に使用することを検討する必要があるのは、PivotTable Services キャッシュ・メモリーの過剰な消費によるスケーラビリティーの問題を経験している場合だけです。維持されている MSAS 接続ごとに、最大で約 250 MB のメモリーを使用することがあり、使用可能なサーバー・メモリー・リソースが急速に消費されてしまいます。 autoDisconnect を true に設定すると、PivotTable Services メモリーの消費は防止されます。 autoDisconnect を false (デフォルト値) に設定すると、PivotTable Services キャッシュが維持されて、頻繁にアクセスされるデータがユーザーにより速く表示されることがあります。

Essbase および DB2 OLAP Server のシングル・サインオン

DB2 Alphablox アプリケーションは、Hyperion Essbase データ・ソースおよび DB2 OLAP Server データ・ソースにアクセスするための Essbase シングル・サインオン証明書オブジェクトの使用をサポートしています。この機能を用いると、Essbase ユーザーは一度サインオンして、生成された証明書を使用すると、DB2 OLAP Server と Essbase データ・ソース間を移動できます。

シングル・サインオンによって、ユーザーは一度ログインすると、複数の DB2 OLAP Server データ・ソースおよび Hyperion Essbase データ・ソースに接続できます。サポートされるデータ・ソースでユーザーが認証されると、(ユーザー名、構成によってはユーザー・パスワードも含まれる) ユーザー証明書が組み込まれた暗号化されたトークンが生成され、複数の Essbase データ・ソース間で受け渡すことができます。 DB2 Alphablox アプリケーションを作成し、 Hyperion Common Security Service によって作成される証明書オブジェクトを使用して、このユーザー情報を DataBlox を介して受け渡すことができます。 Essbase シングル・サインオンを DB2 Alphablox アプリケーションで使用すると、ユーザー名およびパスワードは DB2 Alphablox リポジトリーに保管する必要はありません。

DataBlox credential 属性を使用したユーザー証明書の受け渡し

DB2 OLAP Server ユーザー証明書および Hyperion Essbase ユーザー証明書は、DataBlox credential 属性を使用して、サポートされるデータ・ソースに受け渡すことができます。

以下のステップでは、シングル・サインオンをサポートする DB2 OLAP Server データ・ソースまたは Hyperion Essbase データ・ソースにアクセスできるものと想定します。

証明書オブジェクトを受け渡すには、以下のようにします。

1. ユーザー証明書トークンを取得する変数を指定する JSP スクリプトレットを追加します。

   重要: このスクリプトレットは、ユーザー証明書を受け取るデータ・ソースの DataBlox タグの上に現れなければなりません。

2. DataBlox タグで、credential 属性を追加し、その値を、指定した変数の値を取り出す JSP 式に設定します。

次の例では、JSP ページ・ディレクティブを使用して必要な Java パッケージを使用可能にし、JSP スクリプトレットは、ユーザー証明書トークンを取り出すのに使用する変数 (userCredential) を指定します。その後 DataBlox の credential 属性がこの情報を取り出し、指定の Essbase データ・ソースにアクセスします。

```
<%@ 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 omitted here.
}
%>

<html>
<head>
  <blox:header/>
</head>
<body>
<%
    String credential = request.getSession().getAttribute("SSOToken");

   if (credential == null)
   {
      HashMap css_context = new HashMap();
      String user = request.getParameter("username");
      String password = request.getParameter("password");

      MyCssApp myApp = new MyCssApp();
      CSSSystem system = CSSSystem.getInstance();
      CSSAPIIF css = system.getCSSAPI();

      css_context.put(CSSAPIIF.LOGIN_NAME, user);
      css_context.put(CSSAPIIF.PASSWORD, password);
      css.initialize(css_context, myapp);

      CSSUserIF css_user = css.authenticate(css_context);

      credential = css_user.getToken();
      request.getSession().setAttribute("SSOToken", user1.getToken());
  }
%>



  <blox:data id="myDataBlox"
    credential="<%= credential %>"
    dataSourceName="EssbaseSSO"
    ...
  </blox:data>

  <blox:present id="myPresentBlox"
    width="700" height="500"
```

```
    <blox:data bloxRef="myData" />
  </blox:present>
</body>
</html>
```

DataBlox credential 属性について詳しくは、「*開発者用リファレンス*」の『DataBlox タグ・リファレンス』セクションを参照してください。

Blox API を使用したユーザー証明書の受け渡し

Hyperion Essbase ユーザー証明書は、Blox API を使用して Essbase データ・ソースに受け渡され、DataBlox `setCredential()` メソッドを呼び出すことができます。

以下のステップでは、シングル・サインオンをサポートする Hyperion Essbase データ・ソースにアクセスできるものと想定します。

Blox API を使用して証明書オブジェクトを受け渡すには、以下のようにします。

1. JSP スクリプトレットを追加して、ユーザー証明書トークンを取得する変数を指定します。

   重要: このスクリプトレットは、ユーザー証明書を受け取るデータ・ソースの DataBlox タグの上に現れなければなりません。
2. DataBlox タグを追加しますが、`credential` 属性は追加しません。
3. JSP スクリプトレットを DataBlox タグの下に追加して、ユーザー証明書を設定します。

次の例では、JSP スクリプトレットは、ユーザー証明書トークンを取り出すのに使用する変数 (`usercredential`) を指定します。 DataBlox `setCredential()` はユーザー情報を取り出してから、指定の DataBlox にそれを適用します。

```
<%
  String userCredential = myGetCSSToken();
%>
...
<blox:data id="myDataBlox"
  datasourceName="EssbaseSSO"
  ...
</blox:data>

...
<%
  myDataBlox.setCredential(userCredential);
%>
```

ユーザー証明書の設定を制御するために Blox API を使用すると、ユーザー証明書を取り出す前に、実行する必要のある他の操作を行うことができます。
setCredential() メソッドについて詳しくは、「*Blox API Javadoc*」資料を参照してください。

シングル・サインオンの制限事項

Hyperion Essbase データ・ソースまたは DB2 OLAP Server データ・ソースでシングル・サインオンを使用する場合、ご使用のアプリケーションに影響を与える可能性のある既知の制限事項および他の問題に留意してください。

シングル・サインオン・サポートは、Hyperion Common Security Services 2.6 および 2.7 (Hyperion Essbase および Hyperion Essbase Deployment Services 7.1.3、7.1.2、および 7.1.1) を使用するデータ・ソースについて、DB2 Alphablox で使用可能です。対応する DB2 OLAP Server 8.2 バージョンも、DB2 Alphablox によってサポートされています。

- 外部認証に LDAP サーバーを使用する場合、Essbase 7.1.3 で作動しているユーザー証明書トークンは Essbase 7.1.1 または 7.1.2 では作動しませんし、その逆の場合も作動しません。対応する DB2 OLAP Server 8.2 バージョンも同様に動作します。
  - Essbase 7.1.3 は、ユーザー ID (UID) を使用してユーザーを検出しようとします。トークンがユーザーの UID で生成される場合、ユーザー証明書トークンは Essbase 7.1.3 データ・ソースでは作動しますが、Essbase 7.1.1 および 7.1.2 に適用しようとすると障害が生じ、「不明ユーザー (Unknown User)」メッセージが表示されます。
  - Essbase 7.1.1 および 7.1.2 は、共通名 (CN) を使用してユーザーを検出しようとします。トークンがユーザーの CN を使用して生成される場合、トークンは 7.1.1 および 7.1.2 に受け渡されると作動しますが、Essbase 7.1.3 に適用しようとすると障害が生じ「不明ユーザー (Unknown User)」メッセージが表示されます。
- ユーザー証明書トークンを DataBlox に適用すると、DataBlox に関連付けられた他のユーザー名またはパスワードは無視されます。 DataBlox タグ属性を使用して、または DB2 Alphablox リポジトリー内でユーザー名あるいはパスワードを指定した場合、ユーザー証明書トークンが使用されるとユーザー名とパスワードは無視されます。

第 14 章 データの取り出し

データ・ソースに接続したら、次の作業は、サブミットした照会から生成された結果セットのデータを取り出すことです。このような照会は、データベース管理者やデータ分析者によって提供されることがあります。しかし、照会ステートメントを独自で作成したり、他のユーザーとの共同作業で作成することのほうが多いことでしょう。アクセスしようとするデータ・ソースに精通すればするほど、独立して作業することが容易になります。このセクションでは、DB2 Alphablox アプリケーションで表示するために、さまざまなデータ・ソースからデータを取り出すときの基礎だけを学習します。ここでの目標は、頻繁に生じる一部の問題を参考に理解を深めることです。

アクセスしようとしているデータ・ソースによって、アプリケーション照会に使用する構文が著しく異なる場合があります。DB2 Alphablox アプリケーションでは、照会ストリングは、以下のいずれかになります。

- Essbase レポート・スクリプト: IBM DB2 OLAP Server および Hyperion Essbase データ・ソースの場合
- マルチディメンション式 (MDX): Microsoft SQL Server Analysis Services および DB2 Alphablox Cube Server の場合
- SQL ステートメント: リレーショナル・データ・ソースの場合

特定のデータ・ソースに精通していて、データを取り出すための照会の作成方法が分かる場合、そのほとんどの知識は、DB2 Alphablox アプリケーションですぐに活用できます。しかし、DB2 Alphablox で作業を行うときには、知っておく必要のある役立つヒントや技法がいくつかあるので、このセクションに収められている作業予定のデータ・ソースに関する該当するトピックを必ずお読みください。

特定のデータ・ソースに精通していない場合、以下のセクションをお読みになれば、構文の概要と、データ・ソースで作業を行うときに生じる可能性のある DB2 Alphablox 問題の概要を理解できます。データ・ソースで作業するときの詳細は、下記の詳細情報の該当するセクションを参照してください。

Application Studio Workbench に付属する照会ビルダーは、分析アプリケーションで使用予定のデータ・ソースに対して照会を入力してテストするために使用できます。このツールは、DB2 Alphablox で定義された任意のデータ・ソースに接続します。照会ビルダーは、照会を開発するために以下のいくつかの方法で使用できます。

- テキスト・ストリングを入力し、結果の分析表示を確認する
- データ・ソースに対する最新の照会を呼び出し、結果の分析表示を確認する
- データ・ソースのデフォルト照会 (あれば) を実行し、結果の分析表示を確認する
- GridBlox ユーザー・インターフェースを使用し、軸間でディメンションを移動する (データのピボット、ドリル、およびフィルター)。そして、アプリケーションの必要な分析表示に到達するための他のアクションを実行します。次に、分析表示を生成するのに必要な照会ストリングを取り出します。

該当する照会ストリングに達したら、カット・アンド・ペーストして、DataBlox の照会値にするか、後で使用するためにテキスト・ファイルで保管します。詳しくは、167 ページの『照会ビルダー』を参照してください。

アプリケーションの照会を指定する場所は、アプリケーション設計に応じて異なります。DB2 Alphablox アプリケーションは、以下に基づいて照会要求を発行できます。

- (DataBlox 照会プロパティーによる) Blox インスタンス化
- (おそらく HTML フォーム・ボタンによる) 定義済み照会のリストからのユーザー選択
- ユーザー・プロファイルに関連付けられた、照会ストリングを含むカスタム・プロパティー

DataBlox query プロパティーの設定

DataBlox の query プロパティーにより、DataBlox またはネストされた DataBlox がロードされた後に、データベースで実行する必要のある初期照会が決まります。照会が定義されない場合、デフォルトの照会は空ストリングになります。デフォルトでは、結果セットなしでロードする Blox は、「使用可能なデータがありません」というメッセージを表示します。

Blox の初期照会を定義するために、以下の 2 つのオプションがあります。

- DataBlox の query 属性の値に初期照会を定義する
- Java メソッドを使用して照会を設定してから実行する

DataBlox query 属性を使用して照会ストリングを定義する場合、 DataBlox に query 属性を追加するだけです。 DataBlox query 属性は、次のように入力する必要があります。

```
query="queryString"
```

ここで *queryString* は、dataSourceName 属性を使用して定義されたデータ・ソースに対して実行する照会を定義するストリングです。

次の例では、GridBlox のネストされた DataBlox は、QCC-Essbase データ・ソースに対して、定義された *queryString* を実行します。

```
<blox:grid id="myGrid">
  <blox:data
    dataSourceName="QCC-Essbase"
    query='<ROW("All Products") <CHILD "All Products"
      <COLUMN("All Time Periods") <CHILD "2000" Sales !'/>
</blox:grid>
```

読みやすくすることとコーディングを容易にするため、query プロパティーを Java 関数で定義することが役立つこともあります。以下の 2 つの作業では、このことを実行する方法が示されています。 query 属性を使用する場合、Blox は、デフォルトでは、定義されたデータ・ソースに注意深く接続し、照会を注意深く実行します。メソッドを使用する場合、定義された照会を実行するために、DataBlox

connect() メソッドを使用する必要があります。次のタスクでは、Java メソッドを使用して、定義されたデータ・ソースから結果セットを取り出す方法の例が示されています。

JSP スクリプトレットを使用した照会の設定および実行

パフォーマンスが問題である場合、JSP スクリプトレットを使用するこの簡単な仕掛けを使用して、表示される表示の応答速度を改善できます。この場合、結果セットを生成するのに、2 つのサーバー側のメソッドが使用されます。DataBlox setQuery() メソッドは初期照会を定義するのに使用でき、その後 connect() メソッドで、その照会が実行されて含まれる Blox に結果セットが戻されます。

1. JSP ページの先頭で (ただし、taglib ディレクティブの後)、該当する Blox タグを追加してプレゼンテーション Blox を定義しますが、データが使用可能になる前に Blox がレンダリングされないように、visible 属性を false に設定します。ネストされた DataBlox に dataSourceName 属性を組み込み、データ・ソースを定義しますが、query 属性は組み込みません。

   たとえば、次の Blox タグは、false に設定された visible と、QCC に設定された dataSourceName を使用して、PresentBlox を定義しています。

   ```
   <blox:present id="PresentBlox3"
     visible="false"
     width="550"
     height="350">
     <blox:data
       dataSourceName="QCC"/>
     <blox:grid
       bandingEnabled="true"/>
     <blox:chart
       chartType="Bar"/>
   </blox:present>
   ```

2. プレゼンテーション Blox を定義している Blox タグの下に、以下の 3 つのサブステップを実行する JSP スクリプトレットを追加します。
   - 照会変数を宣言する
   - 照会変数で定義された照会を設定する
   - Blox をデータ・ソースに接続する (結果セットが戻される)

   次のスクリプトレットの例では、 Java メソッドを使用して定義および実行される照会が示されています。

   ```
   <%
    String query="<ROW(\"All Products\") <ICHILD \"All Products\""+
                 "<COLUMN(\"All Time Periods\") <CHILD 2000 "+
                 "<PAGE(Measures) Sales !";

    PresentBlox3.getDataBlox().setQuery(query);
    PresentBlox3.getDataBlox().connect();
   %>
   ```

   この例では、照会変数は (変数宣言の前に String を置くことにより) ストリングとして宣言され、その変数を希望する照会ステートメントに等しいものとして設定します (この例では、照会は Essbase レポート・スクリプト)。照会は、連結ストリングを使用して、読みやすく保守しやすいようにレイアウトされていることに注意してください。照会が宣言されたら、2 つの DataBlox メソッド

setQuery() および connect() が使用されます。 setQuery() メソッドは、照会を DataBlox に設定します (この例では、引数として query を配置することで、定義された照会をメソッドの引数に置き換えられることに注意してください)。次に connect() メソッドを使用すると、DataBlox は、データ・ソースに接続し、設定された照会を実行するよう指示されます。

3. JSP ページの <body> のさらに下では、Blox をレンダリングして表示する箇所に、 Blox 表示タグを置きます。 bloxRef 属性を使用してプレゼンテーション Blox を参照し、その値をレンダリングする Blox の名前に設定します。

   この例では、<body> タグ内に、次のタグを置きます。

   ```
   <blox:display bloxRef="PresentBlox3"/>
   ```

この例で分かるように、この技法は Java の知識が少なくても使用できます。

マルチディメンション・データ・ソース

マルチディメンション・データベースの概要は、「*管理者用ガイド*」に示されています。以下の詳細については、以降のセクションを参照してください。

- OLAP の用語および概念
- マルチディメンション分析
- OLAP データベース用語

IBM DB2 OLAP Server および Hyperion Essbase

IBM DB2 OLAP Server および Hyperion Essbase は、分析用に最適化されたマルチディメンション・データベースです。通常は、照会に対して 1 秒以内で応答を生成します。

データを DB2 OLAP Server または Essbase キューブから取り出すには、Essbase Report Specification Language を使用して、レポート・スクリプトを生成する必要があります。これらは、DB2 Alphablox アプリケーションの照会値に使用できます。

次に示すのは、Essbase レポート・スクリプトを作成する方法の基本的な要約です。DB2 Alphablox 機能と共にレポート・スクリプトを使用する方法の重要なヒントが示されています。

DB2 OLAP Server または Essbase および Essbase レポート・スクリプトの使用についての詳細は、DB2 OLAP Server または Essbase の資料を参照してください。

Essbase レポート・スクリプトの作成

照会を IBM DB2 OLAP Server または Hyperion Essbase データ・ソースに渡すには、Essbase Report Specification Language を使用して、レポート・スクリプトを作成します。

ヒント: Report Script Specification Language についての詳細は、DB2 OLAP Server または Essbase インストール・ディレクトリーにある /docs/techref/RPTIND.HTM のオンライン資料を参照してください。ワーク

ステーションに DB2 OLAP Server または Essbase Application Manager をインストールしてある場合、「ヘルプ」メニューからこの資料にアクセスできます。

次の DataBlox の例は、以下のことを指定します。

- Market および Accounts ディメンションが列の軸に示される。
- Scenario および Product ディメンションが行の軸に示される。
- 4 つのディメンションすべての子を結果セットに含める必要がある。
- 未使用のディメンションがあれば、“Other” 軸に示される。

```
<blox:data ...
  query="<SYM <ROW (Scenario,Product)
     <ICHILD Scenario <ICHILD Product <COLUMN (Market, Accounts)
     <ICHILD Market <ICHILD Accounts !"/>
```

DB2 Alphablox でサポートされている Essbase レポート・スクリプト・コマンド

次の表は、ほとんどの Essbase レポート・スクリプト・コマンド、DB2 Alphablox でサポートされているかどうか (つまり、レポート・スクリプトに入力したときに機能するかどうか)、同等または近い DB2 Alphablox 機能、そしてこれらのコマンドを使用したレポート・スクリプトの例をリストしています。

レポート・スクリプト・コマンド	レポート・スクリプトの例およびコメント
!	これは、レポート・スクリプト照会を実行するのに必要です。それ自体では、「bang」照会により、グリッドまたはチャートの 1 つのディメンション、そして DataLayout パネルで使用可能なすべてのディメンションのリストが戻されます。DB2 Alphablox では、複数の bang レポート出力コマンドはサポートされていません。この表の下にある注を参照してください。
&1	&CurrentMonth IBM DB2 OLAP Server または Hyperion Essbase で定義される場合、サーバー置換変数をレポート・スクリプトに追加できます。これは、主にスクリプトの保守を単純化するために使用されます。
ALLINSAMEDIM	<ROW (Scenario) <ALLINSAMEDIM Actual !
ALLSIBLINGS	<ROW (Scenario) <ALLSIBLINGS Actual !
ANCESTORS	<ROW (Measures) <ANCESTORS "Marketing" !
ASYM	<ASYM <COL (Scenario, Year) Actual Jan Budget Feb !
ATTRIBUTE	<ROW (Product) <ATTRIBUTE Bottle !
BOTTOM	<ROW (Year) <DIMBOTTOM Year <BOTTOM (6, @DataCol(1)) !
CALCULATECOLUMN	例については、DB2 OLAP Server または Essbase の資料を参照してください。
CALCULATEROW	例については、DB2 OLAP Server または Essbase の資料を参照してください。
CHILDREN	<ROW (Market) <CHILDREN Market !

レポート・スクリプト・コマンド	レポート・スクリプトの例およびコメント
CLEARALLROWCALC	例については、DB2 OLAP Server または Essbase の資料を参照してください。
CLEARROWCALC	例については、DB2 OLAP Server または Essbase の資料を参照してください。
COLUMN	<COLUMN (Year) <CHILD Year !
DESCENDANTS	<ROW (Year) <DESCENDANTS Year !
DIMBOTTOM	<ROW (Year) <DIMBOTTOM Year !
DUPLICATE	<ROW (Year) <Child Year <DUPLICATE Qtr1 !
FIXCOLUMNS	<COL (Year) {FIXCOLUMNS 3} <DIMBOTTOM Year !
GEN	<ROW (Product) gen2,Product !
IANCESTORS	<ROW (Year) <IANCESTORS Jan !
ICHILDREN	<ROW (Product) <ICHILDREN Colas !
IDESCENDANTS	<ROW (Product) <IDESCENDANTS Product !
INCMISSINGROWS	{SUPMISSINGROWS} {INCMISSINGROWS} <PAGE (Market) "New York" <ROW (Product) lev0,Product !
INCZEROROWS	{SUPZEROROWS} {INCZEROROWS} <PAGE (Market) "New York" <ROW (Product) lev0,Product !
IPARENT	<ROW (Year) <IPARENT Jan !
LATEST	<LATEST Aug <ROW (Year) <CHILD QTR3 Q-T-D !
LEV	<ROW (Product) lev0,Product !
LINK	<ROW (Year) <LINK(<DIMBOTTOM(Year) AND <DESCENDANTS(Qtr1)) !
MATCH	<ROW (Market) <MATCH (Market, C*) !
NAMESON	{SUPNAMES} {NAMESON} <ROW (Market) <CHILD East !
NOROWREPEAT	{NOROWREPEAT} <ROW (Market, Product) <CHILD East <CHILD Product !
OFFCOLVCALCS	例については、DB2 OLAP Server または Essbase の資料を参照してください。
OFFROWCALCS	例については、DB2 OLAP Server または Essbase の資料を参照してください。
OFSAMEGEN	<ROW (Market) <OFSAMEGEN East !
ONSAMELEVELAS	<ROW (Market) <ONSAMELEVELAS East !
ORDER	{ORDER 0 5 4 3 2 1} <COL (Product) <CHILD Product !
ORDERBY	<ROW (Product) <DIMBOTTOM Product <ORDERBY ("Product", @DATACOL(1) ASC) !
PAGE	<PAGE (Market) East <ROW (Product) <CHILD Product !
PARENT	<ROW (Year) <PARENT Jan !
REMOVECOLCALCS	例については、DB2 OLAP Server または Essbase の資料を参照してください。
RESTRICT	<ROW (Product) <DIMBOTTOM Product <RESTRICT (@DATACOL(1) > 10000) !

レポート・スクリプト・コマンド	レポート・スクリプトの例およびコメント
ROW	<ROW (Year) <PARENT Jan !
SCALE	{SCALE 100} <ROW (Product) <CHILD Product !
SETROWOP	例については、DB2 OLAP Server または Essbase の資料を参照してください。
SINGLECOLUMN	<SINGLECOLUMN <COL (Year) Year <ROW (Product) <CHILD Product !
SORTALTNAMES	<ROW (Product) <SORTALTNAMES <DIMBOTTOM Product !
SORTASC	<ROW (Market) <SORTASC <DIMBOTTOM Market !
SORTDESC	<ROW (Market) <SORTDESC <DIMBOTTOM Market !
SORTGEN	<ROW (Product) <SORTGEN <DESCENDANTS Product !
SORTLEVEL	<ROW (Product) <SORTLEV <DESCENDANTS Product !
SORTMBRNAMES	<ROW (Product) <SORTMBRNAMES <SORTDESC <DIMBOTTOM Product !
SORTNONE	<ROW (Product) <SORTMBRNAMES <SORTDESC <SORTNONE <DIMBOTTOM Product !
SPARSE	<SPARSE <ROW (Product, Market) <DIMBOTTOM Product <DIMBOTTOM Market !
SUPEMPTYROWS	{SUPEMPTYROWS} <PAGE (Market) "New York" <ROW (Product) lev0,Product !
SUPMISSINGROWS	{SUPMISSINGROWS} <PAGE (Market) "New York" <ROW (Product) lev0,Product !
SUPSHARE	<SUPSHARE <ROW (Product) lev0,Product !
SUPSHAREOFF	<SUPSHARE <SUPSHAREOFF <ROW (Product) lev0,Product !
SUPZEROROWS	{SUPZEROROWS} <COL (Measures) Sales <ROW (Year) Jan Feb Mar !
SYM	<SYM <COL (Measures, Year) Sales COGS Jan Feb !
TOP	<ROW (Market) <DIMBOTTOM Market <TOP(5, @DATACOL(1)) !
UDA	<ROW (Market) <UDA (Market, "Major Market") !
WITHATTR	<ROW (Product) <WITHATTR(Caffeinated,"<>",True) !

注:

1. DB2 Alphablox カスタム・プロパティーを使用できます。
2. DB2 Alphablox は、selectableSlicerDimensions を使用してページ表示を制御しますが、<PAGE コマンドは、データをスライスするレポート・スクリプト内で機能します。
3. suppressMissingOnRows、suppressMissingOnColumns、および suppressZeros は、同様の効果を得るために DB2 Alphablox で使用できます。
4. DB2 Alphablox で suppressMissingOnRows および suppressMissingOnColumns を使用すると、行および列の値が欠落することを避けられます。
5. suppressZeros は DB2 Alphablox で使用できますが、このプロパティーでは行と列の両方でゼロが表示されなくなります。

注: 複数の bang (!) レポート出力コマンドを含む複数 bang 照会は、DB2 Alphablox ではサポートされていません。複数の bang レポート出力コマンドを採用するいくつかの選択レポート・スクリプトでも、Blox 内の結果を表示できることにお気付きかもしれませんが、これは各自のリスクで使用してください。

DB2 Alphablox 同等機能でサポートされないレポート・スクリプト・コマンド

以下の Essbase レポート・スクリプト・コマンドは、DB2 Alphablox ではサポートされません。リストされている Essbase レポート・スクリプト・コマンドではなく、リストされている DB2 Alphablox 同等機能を使用してください。

レポート・スクリプト・コマンド
DB2 Alphablox 同等機能

AFTER defaultCellFormat(GridBlox)

BEFORE defaultCellFormat(GridBlox)

COMMAS defaultCellFormat(GridBlox)

DECIMAL
defaultCellFormat(GridBlox)

EUROPEAN
defaultCellFormat(GridBlox)

MISSINGTEXT
missingValueString(GridBlox)

NOINDENTGEN
rowIndentation(GridBlox)

OUTALT useAliases (DataBlox)

OUTALTNAMES
useAliases (DataBlox)

OUTALTSELECT
aliasTable (DataBlox)

OUTMBRNAMES
useAliases (DataBlox)

SUPBRACKETS
defaultCellFormat(GridBlox)

SUPCOMMAS
defaultCellFormat(GridBlox)

DB2 Alphablox の同等機能がないためにサポートされないレポート・スクリプト・コマンド

BLOCKHEADERS BRACKETS COLHEADING	SAVEANDOUTPUT SAVEROW SETCENTER SETROWOP
CURHEADING CURRENCY DIMEND DIMTOP	SKIP SKIPONDIMENSION STARTHEADING SUPALL
DIMBOTTOM ENDHEADING FEEDON	SUPCOLHEADING SUPCURRHEADING SUPEUROPEAN
FORMATCOLUMNS HEADING IMMHEADING	SUPFEED SUPFORMATS SUPHEADING SUPMASK
INCEMPTYROWS INCFORMATS INCMASK INDENT	SUPNAMES SUPOUTPUT SUPPAGEHEADING
INDENTGEN LMARGIN MASK NAMESCOL	TABDELIMIT TEXT TODATE UCHARACTERS
NAMEWIDTH NEWPAGE NOPAGEONDIMENSION	UCOLUMNS UDATA UNAME UNAMEONDIMENSION
NOSKIPONDIMENSION PAGEHEADING PAGELENGTH	UNDERLINECHAR UNDERSCORECHAR WIDTH
PAGEONDIMENSIONS PRINTROW	ZEROTEXT

Calc スクリプト

Calc (計算) スクリプトは、DB2 OLAP Server または Essbase キューブのデータを計算するための指示を含むテキスト・ファイルです。Calc スクリプトは、以下の DataBlox メソッドを使用して DB2 Alphablox アプリケーションで呼び出せます。

- `executeCustomCalc`
- `executeNamedDBCalcScript`
- `substituteCalcScriptTokens`
- 書き戻し

これらのメソッドの使用法の詳細は、「*開発者用リファレンス*」を参照してください。 DB2 OLAP Server または Essbase キューブでの Calc スクリプトの使用法について詳しくは、DB2 OLAP Server または Hyperion Essbase の資料を参照してください。

置換変数

IBM DB2 OLAP Server または Hyperion Essbase キューブでは、置換変数は、周期的に変更する情報のためのグローバルなプレースホルダーとして機能します。それぞれの変数は、割り当てられた値を持ち、いつでもデータベース管理者側で変更できます。置換変数を使用すると、DB2 Alphablox アプリケーションの個々のレポート・スクリプトを手動で変更する必要がなくなるので、レポート・スクリプトの保守が軽減されます。

たとえば、多くのレポート・スクリプトは、現在の月または現在の四半期など、レポート期間を参照します。`CurrentMonth` や `CurrentQuarter` など、IBM DB2 OLAP Server または Hyperion Essbase サーバーに設定された置換変数を使用することにより、特定の時点での割り当てられた値を変更でき、該当するレポート・スクリプトの実行時に、そのレポート・スクリプトは動的に更新されます。

レポート・スクリプトの特定の置換変数を参照するには、その変数名の前にアンパーサンド (&) を付けます。たとえば、置換変数 `CurrentMonth` を参照するには、レポート・スクリプトで `&CurrentMonth` を使用します。次の DataBlox の例は、query 属性での `&CurrentMonth` の使用法を示しています。

```
<blox:data
  dataSourceName="QCC-Essbase"
  query='<ROW("All Products") <COLUMN("All Time Periods")
  &CurrentMonth <PAGE(Measures) Sales !'/>
```

照会が実行されると、&CurrentMonth は、IBM DB2 OLAP Server または Hyperion Essbase サーバーで定義された値に置き換えられます。

置換変数は、レポート・スクリプトでの保守を軽減しますが、IBM DB2 OLAP Server または Hyperion Essbase サーバーの値をだれかが手動で変更する必要は残ります。DB2 Alphablox アプリケーションでの代替手段として、JSP ページで Java メソッドを使用して、現在の月または他のレポート期間の値を自動的に計算してから、レポート・スクリプトでその値を置き換えることができます。

DB2 OLAP Server または Essbase 別名の使用

DB2 OLAP Server または Essbase の別名、すなわち DB2 OLAP Server または Essbase キューブで定義されたメンバーの代替名を使用して、分析表示を読みやすくできます。別名は、外国語での名前などの代替メンバー名を参照するときや、製品の識別値を参照するときに使用できます。DB2 Alphablox では、さまざまなプロパティーでのレポート・スクリプトおよび値での別名の使用をサポートしています。

デフォルトでは、DB2 Alphablox アプリケーションは固有のメンバー名を表示し、別名は表示しません。分析表示で別名を表示する場合、DataBlox useAliases プロパティーを true に設定します。詳しくは、「*開発者用リファレンス*」の『DataBlox リファレンス』セクションを参照してください。

小数部の処理

数値データの値を、指定した小数点以下の桁数で表示する場合、{DECIMAL} レポート・スクリプト・コマンドを、DB2 OLAP Server または Essbase 照会ステートメントに追加する必要があることがあります。詳しくは、「*開発者用リファレンス*」のこれらのプロパティーの詳細な説明を参照してください。

使用する Blox タグ属性	対象 Blox	使用するレポート仕様ディレクティブ
<blox:grid id="myGrid" ... defaultCellFormat="#,###.00; -#,###.00" > <blox:cellFormat scope="{Sales}" format="#,###.##"/> <blox:cellAlert background="#3333ff" format="#,###.##;(#,###.##" scope="{Scenario}"/> </box:grid>	GridBlox	{DECIMAL 2}

Microsoft Analysis Services

Microsoft SQL Server には、Microsoft Analysis Services が含まれています。これは、Microsoft SQL Server、Oracle、およびその他を含む、複数のリレーショナル・データ・ソースからデータを取り出すときに使用できます。機能面では IBM OLAP Server および Hyperion Essbase と似ていますが、Microsoft では、Microsoft

Analysis Services マルチディメンション・データ・キューブを照会するのに、マルチディメンション式言語 (MDX) を使用します。

Microsoft Analysis Services について詳しくは、以下のリソースを参照してください。

書籍

Spofford, George. 2001. *MDX Solutions: With Microsoft SQL Server Analysis Services.* New York: John Wiley & Sons.

意思決定支援向けにデータにアクセスして分析するときの MDX の使用方法を、分かりやすく徹底的に説明したチュートリアル/解説書です。基本および上級の MDX ステートメントを網羅しており、ほとんどの一般的に生じる問題をすっきり解決します。本格的な理解を求める開発者の方にぜひお勧めします。

Jacobsen, Reed. 2000. *Microsoft SQL Server Analysis Services Step by Step.* Redmond, Washington: Microsoft Press.

データベース管理、データベース作成、および基本的な MDX の使用法を含む、Microsoft Analysis Services のすべての面を取り上げた良質な入門書です。

ニュースグループ

上記の書籍や Microsoft の資料で疑問の答えを得られない場合、別のすばらしいリソースである、インターネット・ニュースグループを試してみることができます。MSAS については、疑問の答えを得られるニュースグループは、`microsoft.public.sqlserver.olap` の 1 つだけです。このピアツーピアのニュースグループを利用して、他の管理者および開発者と MSAS について論じ合います。このニュースグループに加わる多くの投稿者が、最も有用なコンピューター・ニュースグループを作り上げています。「MDX Solutions」の著者である George Spofford 氏は、頻繁に投稿しており、たくさんの厳しい質問を提出しています。さらに、複数の Microsoft Analysis Services チーム・メンバーがこのニュースグループに携わっており、どこにも見つけられないような洞察を与えてくれます。

加入するには、ニュース・サーバーを次に移動します。

```
news:msnews.microsoft.com/microsoft.public.sqlserver.olap
```

別の方法として、次のリンクから OLAP ニュースグループにアクセスできます。ここでは、すべての Microsoft SQL Server 関連のニュースグループがリストされています。

```
http://www.microsoft.com/sql/support/newsgroups/
```

また、このニュースグループのアーカイブを検索するには、Web ブラウザーを Usenet ニュースグループの検索エンジンである Google の「グループ」に移動します。

```
http://groups.google.com/
```

Google の「グループ」のホーム・ページの検索フィールドで、次のように入力します。

```
microsoft.public.sqlserver.olap
```

そして、「Google 検索」ボタンをクリックします。すぐに、最新の投稿が表示されます。検索をこのニュースグループに絞り、キーワードを検索することで、検索をさらに絞ることもできます。これは、緊急に回答が必要な場合に優れたリソースになります。

MDX 照会ステートメントの作成

照会を Microsoft® SQL Server 2000 Analysis Services に渡すには、有効な MDX SELECT ステートメントを使用します。MDX 構文は、いくらか SQL 構文に似ています。次の簡単な照会の構文で、SELECT、FROM、および WHERE キーワードの使用方法に注意してください。

```
SELECT axis specification ON COLUMNS,
axis specification ON ROWS
FROM cube_name
WHERE slicer_specification
```

注: 行に対する MDX ステートメント照会は、列が定義されていないと無効です。

次の式は、Sales キューブを照会し、California および Washington の全 Store の Measures ディメンションの要約を戻します。Measures ディメンションは、列の軸に示され、Store ディメンションは行の軸に示されます。

```
SELECT Measures.MEMBERS ON COLUMNS, {[Store].[Store
State].[CA], [Store].[Store State].[WA]} ON ROWS
FROM [Sales]
```

各州のメンバー (Store) の詳細を入手するには、CHILDREN キーワードを追加します。

```
SELECT Measures.MEMBERS ON COLUMNS, {[Store].[Store State].
[CA].CHILDREN, [Store].[Store State].[WA].CHILDREN} ON ROWS
FROM [Sales]
```

固有のメンバー名を入手する方法は、ディメンション階層を下方向にカスケードすることである点に注意してください。たとえば、次のような階層の Stores というディメンションを想定してください。

```
All Stores
   Canada
   USA
     CA
     OR
   Mexico
```

以下は、その階層内で有効な固有のメンバー名です。

```
[Store].[All Stores]
[Store].[All Stores].[USA]
[Store].[All Stores].[USA].[CA]
```

DB2 Alphablox がサポートする MDX 構文のサブセットについて詳しくは、「*DB2 Alphablox Cube Server 管理者用ガイド*」を参照してください。

マルチディメンション式 (MDX) の概要は、Microsoft サイト: http://msdn.microsoft.com でオンラインで学ぶことができます。ここでは、この例に加えてさらに多くの例が示されています。

autoDisconnect プロパティーを使用した PivotTable Services Cache の消去

大規模な MSAS キューブを使用していて、MDX 照会が大量の結果セットを戻すので、PivotTable Services メモリー・キャッシュ使用量が過大になってしまい、スケーラビリティーの問題が生じている場合、DataBlox `autoDisconnect` プロパティーを使用して、MSAS ベースの分析アプリケーションのスケーラビリティーとパフォーマンスの管理に役立てることができます。このプロパティーの使用法の詳細は、137 ページの『自動接続および自動切断』を参照してください。

DB2 Alphablox Cube Server

DB2 Alphablox Cube Server は、MDX コマンドの限定サブセットを使用して生成された照会をサポートします。このようなコマンドおよびその使用方法の詳細は、「*DB2 Alphablox Cube Server 管理者用ガイド*」を参照してください。

DB2 Alphablox での SAP Business Information Warehouse (SAP BW) の使用

DB2 Alphablox で SAP BW を使用する前に、説明されている前提条件を満たしている必要があります。

SAP のエンタープライズ・ビジネス・インテリジェンス・ソリューションである SAP Business Information Warehouse (SAP BW) は、DB2 Alphablox 分析アプリケーションからアクセスできます。 SAP BW データ・ソースには、DB2 Alphablox によって提供されている OLE DB for OLAP データ・アダプターを使用してアクセスできます。 DB2 Alphablox を使用すると、高度の分析アプリケーションをカスタムビルトして、SAP BW 環境でのビジネス・ユーザーのニーズを満たすことができます。 DB2 Alphablox は既存の OLE DB for OLAP インフラストラクチャーを使用して、SAP OLE DB for OLAP プロバイダーを介して SAP BW に接続します。

SAP BW を用いて DB2 Alphablox を使用できるようにするには、以下の前提条件を満たす必要があります。

- SAP BW 3.5 Frontend コンポーネントおよび Microsoft Data Access Components (MDAC) を、DB2 Alphablox と同じマシンにインストールする必要があります。SAP BW 3.5 Frontend は、SAP NetWeaver ソフトウェア・パッケージに含まれます。
- Microsoft の MDAC 2.7.1、2.8.0、および 2.8.2 は、DB2 Alphablox でテスト済みです。
- SAP Logon Frontend コンポーネントは、SAP Basis Administrator がインストールしてください。

上記の条件を満たすと、SAP BW データ・ソース用に DB2 Alphablox データ・ソース定義を定義できます。

SAP BW データ・ソース定義の作成

SAP BW で使用可能な分析アプリケーションを作成する前に、DB2 Alphablox にデータ・ソース定義を作成する必要があります。

SAP BW データ・ソース定義を DB2

Alphablox に作成するには、 155 ページの『DB2 Alphablox での SAP Business Information Warehouse (SAP BW) の使用』で取り上げられている前提条件を満たしていることを確認します。

DB2

Alphablox データ・ソース定義を作成するには、次のようにします。

1. DB2 Alphablox 管理ページで「管理」タブをクリックし、次いで「**データ・ソース**」メニュー項目をクリックします。
2. リストの下にある「**作成**」ボタンをクリックします。 「**データ・ソースの作成 (Create Data Source)**」ウィンドウが表示されます。
3. 「データ・ソース」フィールドに SAP BW データ・ソースの名前を入力します。
4. 「**アダプター**」選択リストで、OLE DB for OLAP オプションを選択します。画面では、このアダプターに関連するフィールドがリフレッシュされます。
5. 「**OLAP Server**」フィールドで、「**フロントエンド (Frontend)**」構成システムの「**説明フィールド (Description Field)**」に値を入力します。
6. 「**デフォルト・データベース (Default Database)**」フィールドには、InfoCube の名前を入力します。
7. 「**デフォルト・スキーマ**」フィールドは空白のままにします。
8. 「**プロバイダー (Provider)**」フィールドには、SAP OLE DB for OLAP プロバイダーの値を入力します。 SAP Basis Administrator に、SFC_CLIENT 値と SFC_LANGUAGE 値を尋ねてください。 プロバイダー・ストリングは、以下のようにします。

   ```
   MDrmSAP;SFC_CLIENT=clientValue;SFC_LANGUAGE-languageValue
   ```

   ここで、clientValue および languageValue は SAP Basis Administrator から教わる値です。
9. 「**デフォルト・ユーザー名 (Default Username)**」および「**デフォルト・パスワード (Default Password)**」フィールドに、SAP Basis Administrator から提供されたユーザー ID とパスワードを記入します。
10. 「**保管**」ボタンをクリックします。

これで、DB2 Alphablox 分析アプリケーションで使用可能な DB2 Alphablox データ・ソースが作成されました。

SAP BW で使用する MDX 照会の作成

SAP BW でサポートされる MDX 構文を使用して、DB2 Alphablox アプリケーションの構築を開始できます。

このステージでは、以下のとおりに設定する必要があります。

- Blox API を使用し、DataBlox setQuery() メソッドを用いて SAP BW MDX 照会を設定します。 このメソッドの使用方法について詳しくは、「*Blox API Javadoc*」資料を参照してください。 以下は、setQuery() メソッドを使用した MDX 照会の設定例です。

```
DataBlox.setQuery("SELECT DISTINCT( crossjoin ( {[0CALYEAR].[2001]},
{[0D_SALE_ORG].[All]})) ON AXIS(0), DISTINCT( {[Measures].[0D_COST],
[Measures].[0D_INV_QTY], [Measures].[0D_NETVLINV], [Measures].[0D_TAXAMOUN]})
ON AXIS(1) FROM [$0D_DECU]");
```

- DB2 Alphablox API を使用する場合には、照会および引数では固有名のみを使用してください。SAP OLE DB for OLAP プロバイダーで固有名が必要だからです。 SAP BW データ・ソースの固有名を使用する DB2 Alphablox メソッドの例は、次のとおりです。

  ```
  ODBOMetaData.resolveMember("[Measures].[0D+NETVLINV]");
  ```

- SAP BW および MDX 式言語の使用方法に関する詳細は、SAP Help Portal Web サイト (http://help.sap.com/) またはご使用の SAP BW 資料を参照してください。
- DB2 Alphablox は、SAP BW におけるネイティブ・ドリルスルーおよび書き戻しは現在サポートしていません。

(EIS を使用した) DB2 OLAP Server および Hyperion Essbase でのドリルスルー・サポート

OLAP データ・ソースを使用すると、ビジネス分析者や基幹業務ユーザーは、データからトレンドについての深い洞察を得られますが、基礎となるデータ・ソースへドリルスルーするいくつかのメカニズムがなければ、ユーザーはロー・データへすぐにアクセスできません。OLAP データ・ソースで使用可能である場合にドリルスルー・サポートを使用すると、ユーザーは、OLAP データベースの選択したセルに関して、基礎となるファクト表レコードに含まれるロー・データの深い部分に達することができます。

DB2 OLAP Server または Essbase Integration Services を使用すると、DB2 OLAP Server または Essbase 管理者は、マルチディメンション・データを詳細にリレーショナル・データにマップすることができます。Integration Services は、Microsoft Excel により、EIS サーバーで使用できる事前定義されたドリルスルー・レポートを表示するためにすぐに使用できます。Excel を使用して生成されたレポートは、以下のような基本的なレポートです。

- 行および列のみ
- データ・フォーマットなし
- ユーザー対話に対する制御なし
- ユーザーは複数のレポートまたはシートを同時に表示できない

EIS のために DB2 Alphablox ドリルスルー・サポートを使用することにより、ユーザーは、DB2 OLAP Server (または Essbase) に保管された要約および計算済みデータから、(スター・スキーマを使用した) リレーショナル・ウェアハウスに保管された詳細なデータに到達できます。Integration Services の DB2 Alphablox ドリルスルー・サポートは、事前定義された Integration Services ドリルスルー・リレーショナル・レポートを強化し、簡単に使用して構成でき、強力な機能および柔軟なカスタマイズ性を備えています。

すぐに使用できる Integration Services ドリルスルー・サポート

GridBlox の `drillThroughEnabled` プロパティーを true (デフォルトは `false`) に設定するだけの最小限の労力で、ネイティブの DB2 OLAP Server または Hyperion

Essbase Integration Services ドリルスルー・サポートを使用し始めることができます。いったん使用可能にしたら、グリッドのデータ・セルを右クリックすると表示されるコンテキスト (右クリック) メニューに、「ドリルスルー」メニュー・オプションが追加されます。DB2 Alphablox は、(EIS 管理者によって事前定義された) 使用可能な Integration Services ドリルスルー・レポートのリストが示されたダイアログ・ウィンドウを自動的に生成します。ユーザーがレポートを選択すると、組み込まれたデフォルトの JSP ページは、個別のブラウザー・ウィンドウ内に、基本的な対話式の「Alphablox Relational Reporting」ビューに表示されたレポート・データを戻します。

DB2 Alphablox Relational Reporting 機能の詳細は、「*Relational Reporting 開発者用ガイド*」に掲載されています。

EIS ドリルスルー・ウィンドウ・スタイルの制御

デフォルトのドリルスルー・ウィンドウが目的に適している場合もありますが、DB2 Alphablox では、ネストされた GridBlox `<blox:drillThroughWindow>` タグを使用して表示ウィンドウをカスタマイズすることもできます。このタグが、ドリルスルー・サポートが使用可能な GridBlox 内にネストされる場合、デフォルトのすぐに使用可能な動作はオーバーライドされ、カスタム・ブラウザー・ウィンドウ・プロパティーを定義できます。

`<blox:drillThroughWindow>` のタグ属性は、ブラウザー・ウィンドウをオープンするのに使用される JavaScript `window.open(url,windowName,features)` メソッドの `features` 引数で定義された、最も一般的に使用されるウィンドウ定義プロパティーの後にモデル化されます。サポートされるプロパティーには、以下のものがあります。

タグ属性
: **説明**

`url` ドリルスルー・ウィンドウの JSP のロケーションを定義します

`name` ドリルスルー・ウィンドウの名前

`height` ウィンドウの高さ

`width` ウィンドウの幅

`resizable`
: ドリルスルー・ウィンドウをユーザー側でサイズ変更できるかどうかを決定するブール・プロパティー。デフォルトでは True です。

`statusbarVisible`
: ドリルスルー・ブラウザー・ウィンドウのステータス・バーを表示するかどうかを決定するブール・プロパティー。デフォルトでは True です。

`scrollbarVisible`
: ドリルスルー・ブラウザー・ウィンドウのスクロール・バーを使用可能にするかどうかを決定するブール・プロパティー。デフォルトでは True です。

locationbarVisible
ドリルスルー・ブラウザー・ウィンドウのロケーション・バー (アドレス・バー) を表示するかどうかを決定するブール・プロパティー。デフォルトでは True です。

toolbarVisible
ドリルスルー・ブラウザー・ウィンドウのツールバーを表示するかどうかを決定するブール・プロパティー。デフォルトでは True です。

menubarVisible
ドリルスルー・ブラウザー・ウィンドウのメニュー・バーを表示するかどうかを決定するブール・プロパティー。デフォルトでは True です。

`<blox:drillThroughWindow>` タグとその属性の詳細は、「*開発者用リファレンス*」の『GridBlox リファレンス』セクションを参照してください。

DB2 Alphablox Relational Reporting を使用したカスタム EIS ドリルスルー・サポート

「*Relational Reporting 開発者用ガイド*」で詳しく説明されている DB2 Alphablox Relational Reporting Blox は、Microsoft Excel を使用したネイティブの EIS ドリルスルー・サポートでは不可能な、多くの望ましい機能を備えたカスタム・ドリルスルー・サポートを生成するのに使用できます。 DB2 Alphablox の柔軟性により、ドリルスルーの動作方法をカスタマイズできます。DB2 Alphablox EIS ドリルスルー・サポートを使用して、開発者は以下のことを行えます。

- 結果セットで列を永続的に隠すことにより、ユーザーまたは役割レベルでのセキュリティーを実現する
- 結果セット列の順序を制御する
- 計算済みの列を結果セットに追加する
- ブレーク・グループおよび合計を作成する
- 列を名前変更する
- データをフォーマットする
- 複数のレポートを同時にオープンする

RDBREsultSetDataBlox および RDBResultSetTag の使用

Relational Reporting を使用してカスタム・レポートを表示する場合、以下の点を考慮してください。

- `RDBResultSetDataBlox` および `RDBResultSetTag` を使用すると、DataBlox を参照し、その中の `RDBResultSet` を Relational Reporting パイプラインのデータ “producer”とすることができます。
- `RDBResultSetDataBlox` は、リレーショナル・データ・ソースを示す DataBlox や、ドリルスルー可能な DB2 OLAP Server または Essbase データ・ソースを示す DataBlox と共に機能します。 `rowCoordinate` および `columnCoordinate` が指定される場合、`RDBResultSet` のデータは、`bloxRef` 属性によって参照される DataBlox で実行されたドリルスルーからのものでなければなりません。 DB2 OLAP Server または Essbase データ・ソースで機能するドリルスルーの場合、レポート名も設定する必要があります。 `rowCoordinate` または `columnCoordinate`

が指定されない場合、RDBResultSet のデータは、bloxRef 属性によって参照される DataBlox での getResultSet 呼び出しからのものでなければなりません。どちらの場合も、アクション (drillthrough または getResultSet) を正しく実行できない場合、ヌルが戻されます。

複数のレポートのサポート

各セルで複数のレポートをサポートするには、アプリケーションで複数のレポートを処理できなければなりません。たとえば、レポートごとに、異なる列に対してグループ化できます。また、各レポートでは、異なる CSS スタイル・シートを使用することもできます。レポートは、controller.jsp ファイルを使用して、該当する JSP ページに配布できます。または、レポートが複雑ではない場合、1 つの JSP ファイルを使用して、すべてのレポートを処理することも可能です。どちらの場合でも、RDBResultSetDataBlox が機能するにはレポート名が必要であるため、レポート名を JSP ページに渡す必要があります。

カスタム・メニュー・オプションの追加

Blox UI モデルを使用することにより、独自のカスタム・メニュー・オプション (たとえば、“Drill to Relational”) を作成して、データ・セルを右クリックするときに表示させることができます。この場合、DB2 Alphablox のすぐに使用可能な付属メニュー・オプションの代わりに、独自のカスタム・オプションを使用できます。

次の Java コード例では、“Drill to Relational”オプションがグリッドの右クリック・メニューに追加されます。

```
Menu cellMenu = new Menu();
cellMenu.add(new MenuItem("cellItem","Drill To Relational"));
grid.setCellsRightClickMenu(cellMenu);
// Add a dedicated controller to the cell menu
DataBlox db = presentBlox.getDataBlox();
cellMenu.setController(new DrillingController(db,grid,
    abSessionName,appName));
```

レポートが戻すデータの行が多すぎる場合など、必要に応じて、ドリルスルーを処理するために DrillController を作成し、特定のセルでのドリルスルーを使用不可にすることができます。または、1 つのセルから 2 つの別個のレポートを同時にオープンすることも可能です。

他のカスタム EIS ドリルスルー・サポート

独自のカスタム・ドリルスルー・サポートを作成することを決定し、DB2 Alphablox Relational Reporting を使用してデータを表示しないことにする場合、以下の DataBlox メソッドを使用する必要があります。

- RDBResultSet drillThrough(String reportName, Tuple[] coordinates);
- RDBResultSet drillThrough(String reportName, int columnCoordinate, int rowCoordinate);
- String[] getDrillThroughReportNames(int columnCoordinate, int rowCoordinate);

戻される RDBResultSet を使用して、独自のカスタム・レポートを作成し、行および列を繰り返してデータを入手します。さらに、getDrillThroughReportNames を使

用して、特定セルで使用可能なドリルスルー・レポートのリストを戻してから、特定のレポートおよびセルに対してドリルスルーを呼び出します。

RDBResultSet オブジェクトおよび DataBlox メソッドについての詳細は、「*開発者用リファレンス*」の『DataBlox リファレンス』を参照してください。

Microsoft Analysis Services でのドリルスルー・サポート

OLAP データ・ソースを使用すると、ビジネス分析者や基幹業務ユーザーは、データからトレンドについての深い洞察を得られますが、基礎となるデータ・ソースへドリルスルーするいくつかのメカニズムがなければ、ユーザーはロー・データへすぐにアクセスできません。 OLAP データ・ソースで使用可能である場合にドリルスルー・サポートを使用すると、ユーザーは、OLAP データベースの選択したセルに関して、基礎となるファクト表レコードに含まれるロー・データの深い部分に達することができます。

MDX DRILLTHROUGH ステートメントは、指定したセルや Microsoft Analysis Services キューブでのタプルの作成に使用されたファクト表 (リレーショナル・データ・ソース) から、ソース行セットを取り出すときに使用できます。ここに示すのは、Microsoft Analysis Services に付属するサンプル OLAP データベースの Foodmart 用の、DRILLTHROUGH ステートメントの例です。

```
DRILLTHROUGH SELECT FROM [Inventory] WHERE (
[Product].[ByManufacturer].[All Product].[Acme],  [Warehouse].[Whse 8],
[Time].[Aug. 2000] )
```

MSAS 用の DB2 Alphablox ネイティブ・ドリルスルー・サポートでは、次の DRILLTHROUGH ステートメントを使用して、基礎となる結果セットを取り出します。

```
DRILLTHROUGH [<Max_Rows>] [<First_Rowset>] <MDX SELECT>
```

ここで <Max_Rows> は MDX MAXROWS 値と同じで、<First_Rowset> は MDX FIRSTROWSET 値と同じで、<MDX SELECT> は自動的に生成される SQL 照会です。<Max_Rows> の値は、DB2 Alphablox データ・ソース定義の設定から取られます。

DRILLTHROUGH ステートメントを使用して、基礎となるデータ・ソースから行セットを取り出せるようになる前に、まず MSAS キューブを使用可能にして、「ドリルスルー・オプション (Drillthrough Options)」ダイアログでドリルスルーを使用できるようにして、戻す予定の列を指定する必要があります。また、クライアント・アプリケーションでドリルスルーをサポートしている必要もあります。[キューブでのドリルスルー・オプションの構成についての詳細は、「Microsoft SQL Server」メニューから選択できる、Microsoft の「SQL Server 資料オンライン (SQL Server Books Online)」の資料を参照してください。]

一部のクライアント・アプリケーションは、限定されたドリルスルー・サポートを提供していますが、MSAS ドリルスルーの DB2 Alphablox サポートは、構成しやすく、他のクライアント・アプリケーションでは提供されていないカスタマイズされたドリルスルー動作も可能です。DB2 Alphablox ドリルスルー・サポートでは、以下のことが可能です。

- 結果セット列の順序を制御する

Microsoft Analysis Services では、ドリルスルー操作で表示する列を選択できますが、結果のレコードの列を並べ替えることはできません。DB2 Alphablox ReportBlox 機能を使用すると、表示する列の順序を定義できます。 OrderBlox では、結果セット列を表示する順序を指定するよう構成できます。

- 結果セットで列を永続的に隠すことにより、ユーザー/役割レベルでのセキュリティーを実現する

  ドリルスルーを使用可能にしたり使用不可にすることができるのは、MSAS の組み込みセキュリティーだけです。ユーザー役割に基づいて使用可能なドリルスルー列を制御することはできません。ReportBlox のネストされた MembersBlox を使用することにより、永続的に列を隠すことができます。エンド・ユーザーは、UI を使用して列を再表示することはできませんが、開発者は、該当する API 呼び出しを使用して列を表示できます。

- 計算済みの列を結果セットに追加する

  ReportBlox の CalculateBlox を使用することにより、元々 MSAS では計算された列を戻すことになっていない場合でも、リレーショナル・レポートで計算を複製することができます。

- 複数のレポートを個別のウィンドウでオープンする

  DB2 Alphablox で提供されるすぐに使用可能なドリルスルー・サポートでは、エンド・ユーザーは、複数のレポート・ウィンドウをオープンし、グリッド内の複数のセルからのドリルスルー・データを比較できます。また、複数のウィンドウをオープンするために、独自のソリューションをカスタム・コーディングすることもできます。Microsoft Analysis Services の下の『データの取り出し (Retrieving Data)』セクションの Blox Sampler 例には、このことを実現するための 1 つの方法が示されています。

すぐに使用できる Microsoft Analysis Services ドリルスルー・サポート

GridBlox の `drillThroughEnabled` プロパティーを true (デフォルトは `false`) に設定するだけの最小限の労力で、ネイティブの Microsoft Analysis Services ドリルスルー・サポートをすぐに使用し始めることができます。いったん使用可能にしたら、グリッドのデータ・セルを右クリックすると表示されるコンテキスト (右クリック) メニューに、「ドリルスルー」オプションが追加されます。 DB2 Alphablox は、自動的に適切な `DRILLTHROUGH` ステートメントを生成し、照会を実行します。戻される結果セットは、個別のブラウザー・ウィンドウ内の対話式の ReportBlox に表示されます。

ドリルスルー・ウィンドウ・スタイルの制御

デフォルトのドリルスルー・ウィンドウが目的に適している場合もありますが、DB2 Alphablox では、ネストされた GridBlox `<blox:drillThroughWindow>` タグを使用して表示ウィンドウをカスタマイズすることもできます。このタグが、ドリルスルー・サポートが使用可能な GridBlox 内にネストされる場合、デフォルトのすぐに使用可能な動作はオーバーライドされ、カスタム・ブラウザー・ウィンドウ・プロパティーを定義できます。

<blox:drillThroughWindow> のタグ属性は、ブラウザー・ウィンドウをオープンするのに使用される JavaScript window.open(url,windowName,features) メソッドの features 引数で定義された、最も一般的に使用されるウィンドウ定義プロパティーの後にモデル化されます。サポートされるプロパティーには、以下のものがあります。

タグ属性
: **説明**

url ドリルスルー・ウィンドウの JSP のロケーションを定義します

name ドリルスルー・ウィンドウの名前

height ウィンドウの高さ

width ウィンドウの幅

resizable
: ドリルスルー・ウィンドウをユーザー側でサイズ変更できるかどうかを決定するブール・プロパティー。デフォルトでは True です。

statusbarVisible
: ドリルスルー・ブラウザー・ウィンドウのステータス・バーを表示するかどうかを決定するブール・プロパティー。デフォルトでは True です。

scrollbarVisible
: ドリルスルー・ブラウザー・ウィンドウのスクロール・バーを使用可能にするかどうかを決定するブール・プロパティー。デフォルトでは True です。

locationbarVisible
: ドリルスルー・ブラウザー・ウィンドウのロケーション・バー (アドレス・バー) を表示するかどうかを決定するブール・プロパティー。デフォルトでは True です。

toolbarVisible
: ドリルスルー・ブラウザー・ウィンドウのツールバーを表示するかどうかを決定するブール・プロパティー。デフォルトでは True です。

menubarVisible
: ドリルスルー・ブラウザー・ウィンドウのメニュー・バーを表示するかどうかを決定するブール・プロパティー。デフォルトでは True です。

<blox:drillThroughWindow> タグとその属性の詳細は、「*開発者用リファレンス*」の『GridBlox リファレンス』セクションを参照してください。

DB2 Alphablox Relational Reporting を使用したカスタム・ドリルスルー・サポート

「*Relational Reporting 開発者用ガイド*」で詳しく説明されている DB2 Alphablox Relational Reporting Blox は、ネイティブの Microsoft Analysis Services ドリルスルー・サポートでは不可能な、多くの望ましい機能を備えたカスタム・ドリルスルー・サポートを生成するのに使用できます。 DB2 OLAP Server (および Essbase) と Microsoft Analysis Services 両方の『データの取り出し (Retrieving Data)』の下にある Blox Sampler には、カスタム・ドリルスルー・サポートの一例があります。ここでは、コードをウォークスルーし、最も重要なコード部分を説明します。

1. JSP ファイルで、drillThroughEnabled 属性を <blox:grid> タグに追加し、値を true に設定することにより、「ドリルスルー」コンテキスト (右クリック) メニュー・オプションを使用可能にします。

```
<blox:grid ...
      drillThroughEnabled="true" ... />
```

2. <bloxui:actionFilter> タグを PresentBlox に追加することにより、「ドリルスルー」コンテキスト (右クリック) メニュー・オプションでの右クリック・イベントのインターセプトと処理をセットアップします。

```
<bloxui:actionFilter
  className="<%= MyDrillThroughClass.class.getName() %>"
  componentName="dataAdvancedDrillThrough" />
```

componentName 属性は dataAdvancedDrillThrough の値に設定され、 className 属性は右クリック・イベントの処理のために追加するクラスの名前に設定する必要があります。

3. JSP page ディレクティブをページの先頭に追加し、それぞれのページで必要なクラスを指定します。

```
<%@ page import="com.alphablox.blox.uimodel.ModelConstants,
com.alphablox.blox.uimodel.tags.IActionFilter,
com.alphablox.blox.DataViewBlox,
com.alphablox.blox.uimodel.core.Component,
com.alphablox.blox.uimodel.core.MessageBox,
com.alphablox.blox.uimodel.GridBrixModel,
com.alphablox.blox.uimodel.PresentBloxModel,
com.alphablox.blox.uimodel.core.grid.GridCell,
com.alphablox.blox.uimodel.GridBrixCellModel,
com.alphablox.blox.uimodel.core.ClientLink" %>
```

4. actionFilter のハンドラー・クラスを追加します。

```
<%!
  public static class MyDrillThroughClass implements IActionFilter
  {
    public void actionFilter( DataViewBlox blox, Component component )
      throws Exception {
        GridBrixModel grid =
           ((PresentBloxModel)blox.getBloxModel()).getGrid();
        GridCell[] cells = grid.getSelectedCells();

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

        }
        GridBrixCellModel cell = (GridBrixCellModel)cells[0];
        int rowIndex = cell.getNativeRow();
        int colIndex = cell.getNativeColumn();
        String bloxName = blox.getBloxName();
        String urlStr = "someReportBlox.jsp?bloxRef="+bloxName;
        urlStr += "&colIndex=";
        urlStr += colIndex;
        urlStr += "&rowIndex=";
        urlStr += rowIndex;
```

```
          String timestamp = String.valueOf(System.currentTimeMillis());
           urlStr += "&reportName=";
            urlStr = urlStr + "reportBlox"+timestamp;
              ClientLink link =
                  new ClientLink(urlStr,"reportBlox"+timestamp);
                component.getDispatcher().showBrowserWindow( link );
    }
  }
%>
```

5. カスタム・リレーショナル・レポートをセットアップします。この例では、someReportBlox.jsp です。

```
<%
  String reportName = request.getParameter("reportName");
  if(reportName == null) {
  reportName = "defaultName";
  }
%>
```

6. Blox レポート・タグ・ライブラリーの <blox:RDBResultSetData> または <bloxreport:RDBResultSetData> タグを使用して、カスタム・リレーショナル・レポートをセットアップします。

```
<blox:report id="drillThrough"
  bloxName="<%= reportName %>"
  interactive="true">
  <blox:rdbResultSetData
  bloxRef="<%= request.getParameter(\"bloxRef\") %>"
  columnCoordinate="<%= request.getParameter(\"colIndex\") %>"
  rowCoordinate="<%= request.getParameter(\"rowIndex\") %>">
</blox:rdbResultSetData>

...
```

   [オプション] 複数のレポートをオープンすることをサポートする場合、デフォルトの ReportBlox 名をセットアップします。

   [オプション] 複数のレポートを使用する場合、ReportBlox の bloxName 属性を設定します。 bloxName 属性の使用方法の詳細は、「*開発者用リファレンス*」の『共通 Blox リファレンス』セクションにあります。

   [オプション] メンバーを表示から永続的に除外する場合、MembersBlox を定義します。

   [オプション] ブレーク・グループおよび集約を使用してクリーナー・レイアウトを作成する場合、GroupBlox を定義します。

   [オプション] MSAS で元々サポートされていない複製された計算を追加するには、CalculateBlox を定義します。

   [オプション] 列を並べ替えるには、OrderBlox を定義します。元々は、MSAS ドリルスルー・サポートでは、列を再度並べ替えることはありません。

Blox Sampler 例のカスタム・ドリルスルーを実行するには、完全なコードを検討してください。

Relational Reporting ビューの作成の詳細は、「*Relational Reporting 開発者用ガイド*」を参照してください。次の例では、ドリルスルーで Relational Reporting を使用することに関係した、いくつかの重要な点を網羅します。この例では、取り出され

るページ someReportBlox.jsp を呼び出し、<blox:RDBResultSetData> タグを使用して、ドリルスルー・データを入手してリレーショナル・パイプラインに入れます。

他のカスタム・ドリルスルー・サポート

独自のカスタマイズを実現する場合、1 つの選択肢は、DB2 Alphablox によって提供されるドリルスルー・サポートを制御する右クリック・イベントをインターセプトしてから、サーバー側の ClickEvent を使用してドリルスルーのカスタマイズを管理することです。

独自のソリューションを使用してドリルスルー・レポートを開発する場合、Blox レポート・タグ・ライブラリーを使用する代わりに、以下のいずれかの DataBlox メソッドを使用する必要があります。

- RDBResultSet drillThrough(Tuple[] coordinates)
- RDBResultSet drillThrough(int columnCoordinate, int rowCoordinate)

これらのメソッドは、指定した座標で実行したドリルスルーのリレーショナル・データを含む、RDBResultSet を戻します。その後、リレーショナル・ドリルスルー・データを希望どおりに表示できます。

RDBResultSet オブジェクトについての詳細は、Javadoc で com.alphablox.blox.data.rdb パッケージを参照してください。

リレーショナル・データ・ソース

DB2 Alphablox では、プレゼンテーション Blox を使用した、リレーショナル結果セットの表示をサポートしています。サポートされている他のデータベースと同様、データ・ソースおよび照会を指定する必要があります。リレーショナル結果セットが表示されたら、標準のプレゼンテーション Blox で、DB2 Alphablox 機能の限定されたサブセットが使用できます。ReportBlox は、リレーショナル・データを表示するためにも使用できるもので、多くのレポートのためのサポートが提供されています。リレーショナル・レポートを表示するための新しい Relational Reporting 機能の使用に関する詳細は、「*Relational Reporting 開発者用ガイド*」を参照してください。

以下に示すのは、SQL ステートメントと、それらを DB2 Alphablox で使用する方法についての概要およびサマリーです。

SQL ステートメントの作成

照会をリレーショナル・データ・ソースに渡すには、それぞれの RDBMS でサポートされている SQL SELECT ステートメント構文を使用します。たとえば、次の SQL 構文は、複数のリレーショナル・データ・ソースでサポートされています。

```
SELECT... FROM... WHERE... ORDER BY... GROUP BY...
```

- SELECT (ALL|DISTINCT) [COLUMNS] は、結果セットに含めるデータ列を示します
- FROM [TABLELIST] は、データの入手元である各データベース表の名前を示します

- WHERE [PREDICATE EXPRESSION] は、データに対するフィルターおよび結合を指定します
- ORDER BY [COLUMNNAMES] は、ソート・シーケンスを指定します
- GROUP BY [COLUMNNAMES] は、グループ・リストを指定します

注: 関数はサポートされて**いません**。

次の例では、以下のことを指定します。

- SalesQty および ProductID という列が、 (Actual および Projected という) 2 つの表から選択されること
- 実際の販売量が計画の販売量よりも少ない行だけが選択されること

```
<blox:data query="SELECT Actual.SalesQty, Actual.ProductID,
  Projected.SalesQty, Projected.ProductID FROM Actual,
  Projected WHERE Actual.SalesQty < Projected.SalesQty".../>
```

照会ビルダー

照会ビルダーを使用すると、照会構文を簡単に開発してテストすることができます。このツールでは、ポイント・アンド・クリック・インターフェースを使用し、データ・ソースの照会言語についての深い理解は求められていません。

照会ビルダーは、対話式の照会開発をサポートしています。適切なデータ表示になるまで、デフォルトの結果セットを表示するグリッドを操作するだけです。「**現行クエリーの取得**」ボタンをクリックすると、照会ステートメントが正しい構文で表示されます。そのステートメントを後で使用するためにテキスト・ファイルにコピー・アンド・ペーストするか、アプリケーション・テンプレート内の照会の値へ直接にコピー・アンド・ペーストできます。

照会ビルダーの別の強力な機能は、「**Blox タグの生成**」ボタンです。これを使用すると、同じ表示および結果セットを表示する PresentBlox を複製するのに必要な Blox タグ構文を取り出すことができます。同様に、そのタグを後で使用するためにテキスト・ファイルにコピー・アンド・ペーストするか、アプリケーションへ直接にペーストすることができます。

照会ビルダーの使用

次の作業は、照会ビルダーを使用して照会ステートメントについての学習を開始する方法と、DB2 Alphablox アプリケーションの表示に与える影響を示しています。

照会ビルダーにアクセスするには、次のようにします。

1. 「Application Studio」ページで、「**ワークベンチ (Workbench)**」をクリックします。「ワークベンチ (Workbench)」ページがオープンします。
2. 「**照会ビルダー**」リンクをクリックします。
3. 「**接続設定**」ボタンをクリックして、ドロップダウン・リストをオープンします。照会を作成する対象のデータ・ソースを選択します。「カタログ」、「スキーマ」、「ユーザー名」、「パスワード」フィールドの値は、データ・ソース定義から取られ、テキスト・ボックスに表示されます。

4. 必要であれば、「カタログ」、「スキーマ」、「ユーザー名」、および「パスワード」の値を変更します。接続時にデフォルトの照会を実行する場合、「**デフォルトの照会を実行**」チェック・ボックスをチェックします。
5. 「接続 (Connect)」ボタンをクリックします。正常に接続したら、「状況」フレームに確認インディケーターが表示されます。
6. 以下のいずれかを行います。
   - テキスト・ボックスに照会ストリングを入力し、「**照会の実行**」ボタンをクリックします。結果が、ページの最下部の PresentBlox に表示されます。

     「**現行照会の取得**」ボタンをクリックして、このデータ・ソースに対する成功した最新の照会を取り出します。照会ストリングが、テキスト・ボックスに表示されます。
   - 「**デフォルトの照会 (Default Query)**」ボタンをクリックします。このデータ・ソースに関連付けられたデフォルトの照会ストリングが、テキスト・ウィンドウに表示されます。データ・ソースが、Alphablox キューブまたは Microsoft Analysis Services キューブである場合、キューブ名が正しいことを確認してください。

     注: リレーショナル・データベース接続プール機能を使用する場合、リレーショナル・キューブでのデフォルトの照会を入手する「**デフォルトの照会 (Default Query)**」ボタンを使用することはできません。
   - 照会の結果を表示するには、「**照会の実行**」ボタンをクリックします。結果セットが、ページの最下部の PresentBlox に表示されます。
   - データがページの最下部の PresentBlox に表示されたら、標準のユーザー・インターフェースを使用して、軸を交換したり、上下にドリルしたり、軸間でディメンションを移動したりします。
   - 「**Blox タグの生成**」ボタンをクリックします。これにより、PresentBlox に示された表示および結果セットを複写する完全なタグを含むテキスト・ボックスがオープンします。
   - 「**非対称照会ビルダー (Asymmetrical Query Builder)**」ペインを展開し、除去する列を選択解除し、「**列設定の適用**」ボタンをクリックします。

     注: 「非対称照会ビルダー (Asymmetrical Query Builder)」ペインは、DB2 OLAP Server または Essbase データ・ソースでのみ機能し、列ヘッダーに対してのみ機能します。
7. 適切なデータ・ビューを開発したら、「**現行照会の取得**」ボタンをクリックします。テキスト・ボックスには、現行のデータ・ビューを開発するのに必要とされた照会ストリングが表示されます。

   適切な照会ストリングを導出したら、それを後で使用するためにテキスト・ファイルにコピー・アンド・ペーストするか、照会の値へ直接にコピー・アンド・ペーストできます。照会ビルダーを終了するには、ブラウザー・ウィンドウをクローズします。
8. 「**Blox タグの生成**」ボタンをクリックします。PresentBlox レイアウトおよび結果セットを複製するタグを示すテキスト・ボックスがオープンします。このタグを後で使用するためにテキスト・ファイルにコピー・アンド・ペーストするか、アプリケーションへ直接にペーストすることができます。

JDBC データ・ソースでの作業

DataBlox を使用すると、マルチディメンション・データ・ソースやリレーショナル・データ・ソースに接続できます。リレーショナル・データ・ソースを DB2 Alphablox 管理ページで DB2 Alphablox に対して定義すると、DataBlox を使用してデータ・ソースに接続し、データを取り出すことができます。しかし JDBC データ・ソースの場合、特定の JDBC 呼び出しを実行し、JDBC URL 接続ストリングを取得するか、接続プールまたはストアード・プロシージャーの利点を生かす必要がある場合があります。DB2 Alphablox では JDBCConnection Bean が備えられていて、DB2 Alphablox JDBC データ・ソースから JDBC 接続ストリングを構成することができます。また DB2 Alphablox では、リレーショナル・データベースでストアード・プロシージャーを使用するための StoredProceduresBlox も提供されています。

JDBCConnection Bean の使用

JDBCConnection Bean は Java Bean で、DB2 Alphablox リレーショナル・データ・ソースに関する情報を取得できます。JDBCConnection Bean を使用すると、Blox を作成しなくても、JDBC URL 接続ストリングを取得して JDBC 呼び出しを実行できます。さらに、この Bean を使用して、DB2 Alphablox に定義されているリレーショナル (JDBC) データ・ソースのプロパティーをオーバーライドすることもできます。

JDBCConnection Bean は `com.alphablox.blox.data.rdb` パッケージ内の 1 つのクラスであり、この Bean で任意の API を使用する JSP ファイルの先頭部分で、以下の JSP インポート・ステートメントを使用しなければなりません。

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

JDBCConnection Bean の例

以下は、JDBCConnection Bean を使用して JDBC URL 接続ストリングを印刷する JSP ファイルの例です。

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

<html>
<head>
<title>JDBC Connection Bean Example</title>
</head>

<body>

<%
  String ds = (String)request.getParameter( "ds" ) ;
%>

<form name=form method=get>
Enter data source name:&amp;nbsp;
<input name="ds" value="<%= ds == null ? "" : ds %>"><br />
<input type=submit value="Go"><br />
</form>

<%-- Create the Bean --%>
<jsp:useBean id="jbean"
   class="com.alphablox.blox.data.rdb.JDBCConnection"
   scope="session" />
```

```
<%-- Put in try statement to catch errors --%>
<% try { %>

<%--Test if there is a data source --%>
<% if ( ds != null ) { %>

<%
jbean.setDataSourceName( ds );
%>

<%-- Use the Alphablox bean to get the connection JDBC string --%>
<%= "URL = " + jbean.getURL() %><br />
Properties = <%= jbean.getConnectionProperties( ) %><br />
<%
Connection connection = jbean.createConnection( );
%>
Connection = <%= connection  %><br />
<br />

<%-- If no data source, prompt for one --%>
<% } else { %>
<br />
<b>Please enter a relational data source name!</b>
<br />
<% } %>
<%-- Catch the exception --%>
<% } catch ( Exception e ) {
   out.write( "<br />An error has occured: <b>"
      + e.getMessage() + "</b>" ); } %>
</body>
</html>
```

StoredProceduresBlox の使用

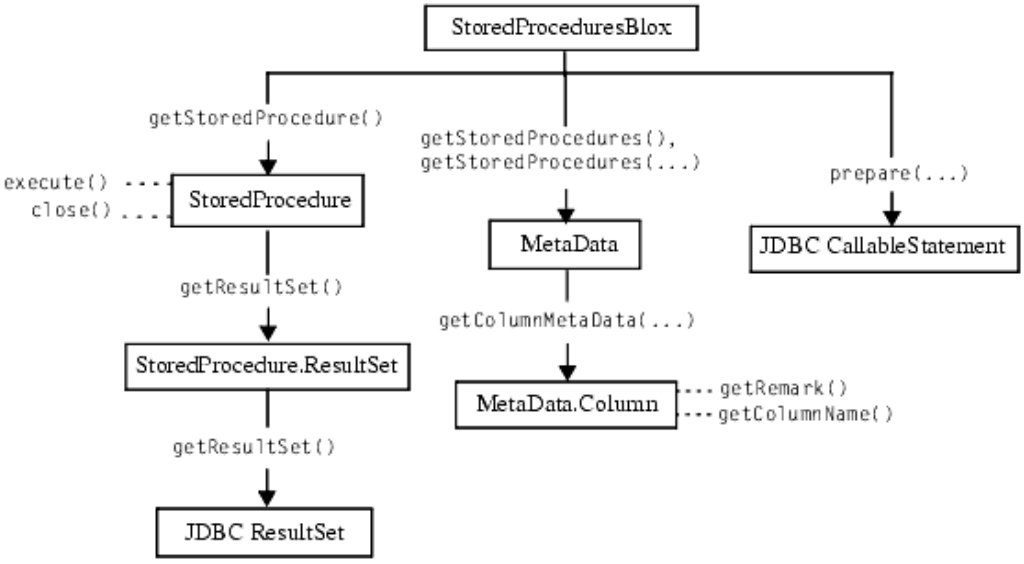
StoredProceduresBlox は、リレーショナル・データベースのストアード・プロシージャーの使用に関する開始点となります。データベースへの接続を作成し、ストアード・プロシージャー・ステートメントの準備を可能にします。 適切な DB2 Alphablox データ・ソースおよび他の接続パラメーターを設定すると、以下のことが可能になります。

- `prepare(...)` メソッドを使用すると、JDBC CallableStatement オブジェクトを戻し、ストアード・プロシージャーを実行するのに必要なストアード・プロシージャー・パラメーターをセットアップするために使用できます。
- `getStoredProcedure()` メソッドを使用すると、現行の StoredProcedure オブジェクトにアクセスできます。それから、ストアード・プロシージャーを実行して、その実行したストアード・プロシージャーの ResultSet を取得するか、JDBC ResultSet にアクセスできます。
- `getStoredProcedures()` または `getStoredProcedures(...)` メソッドを使用すると、1 つ以上の MetaData オブジェクトを戻し、個々のパラメーターにアクセスできます。

StoredProcedure オブジェクトと MetaData オブジェクトは、`com.alphablox.blox.data.rdb.storedprocedure` パッケージ内の別個のクラスです。StoredProceduresBlox から StoredProcedure および MetaData のオブジェクトを分離することにより、一度ストアード・プロシージャーを準備すると、複数回それを実行できます。ストアード・プロシージャーは実行ごとに変更することが可能で

すが、実行のたびにストアード・プロシージャーを作成しないようにすることによって、パフォーマンスを向上させられます。

以下の図は、ストアード・プロシージャー関連オブジェクトのオブジェクト階層を示しています。



StoredProcedure および MetaData オブジェクトは別個のパッケージ内にあるため、こうしたオブジェクトの API を使用する JSP ファイルの先頭部分で以下の 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。

準備されたストアード・プロシージャーを実行するために StoredProcedure オブジェクトを使用する際、以下の点に注意してください。

- DataBlox をストアード・プロシージャーからの情報を表示するために使用する場合、DataBlox は StoredProceduresBlox と同じデータ・ソースに別個に接続する必要があります。
- DataBlox をストアード・プロシージャーからの情報を表示するために使用し、そのストアード・プロシージャーに出力パラメーターもある場合、その出力パラメーターを取得する前に結果セットを最初に使用する必要があります。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。このオブジェクトを使用すると、ストアード・プロシージャーで入出力パラメーターを取得して設定できます。
- ストアード・プロシージャーを実行し、任意の出力パラメーターまたは結果セットを使用したなら、`StoredProceduresBlox.disconnect()` を呼び出し、切断して

リソースを解放する必要があります。オープンされたデータベースとの接続を保持する場合には、 StoredProceduresBlox.close() を呼び出して、使用したリソースを解放します。

- DataException がスローされる場合、DataException.getNestedException() を調べると SQLException として追加情報が利用可能な場合もあります。

ストアード・プロシージャーを実行すると StoredProcedure.ResultSet オブジェクトが戻され、このオブジェクトにより JDBC ResultSet オブジェクトにアクセスできます。JDBC ResultSet オブジェクトを直接使用することが必要な場合には、ResultSet.getResultSet() メソッドを使用してこのオブジェクトを取得します。

ストアード・プロシージャーで作業する際には java.sql パッケージもインポートすることをお勧めします。そのようにすると、JSP ファイルは 2 つのパッケージをインポートします。

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

StoredProceduresBlox の例

このセクションには、StoredProceduresBlox の使用法を示す 6 つの例が含まれています。別の例については、Javadoc 資料を参照してください。

- 『DataBlox なしでのデータ・ソースへの接続』
- 173 ページの『DataBlox で使用するデータ・ソースに接続するための StoredProceduresBlox の使用』
- 173 ページの『指定のパターンと一致する名前のストアード・プロシージャーのリストの取得』
- 173 ページの『各ストアード・プロシージャーのパラメーターすべてのリストの取得』
- 175 ページの『1 つの入力パラメーターと 2 つの出力パラメーターを持つストアード・プロシージャーの実行』
- 175 ページの『DataBlox へのストアード・プロシージャーの結果セットの設定』

DataBlox なしでのデータ・ソースへの接続

この例は、パラメーターのみを取得したい場合や DataBlox を必要としない INSERT SQL ストアード・プロシージャーを実行したい場合など、DataBlox なしでデータ・ソースに接続する方法を示しています。

```
<%@ 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();
%>
```

DataBlox で使用するデータ・ソースに接続するための StoredProceduresBlox の使用

この例は、ストアード・プロシージャーからの情報を表示するために使用されている DataBlox が、StoredProceduresBlox と同じデータ・ソースに別個に接続する方法について示しています。

```
<%@ 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();
%>
```

指定のパターンと一致する名前のストアード・プロシージャーのリストの取得

この例は、getStoredProcedures(...) メソッドを使用して、「procedure」という名前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ています。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 オブジェクトを介すると、指定のストアード・プロシージャーの個々のパラメーターにアクセスできます。

各ストアード・プロシージャーのパラメーターすべてのリストの取得

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そ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た MetaData オブジェクトの戻りを既に持っていると仮定します。

```
MetaData procedures[] =
mySP.getStoredProcedures("procedure%");
```

これで、各ストアード・プロシージャー、そのカタログ、スキーマ、名前、および注釈情報がテーブルにリストされます。

```
<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>
```

各ストアード・プロシージャーのそれぞれの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>
<% }
%>
```

1 つの入力パラメーターと 2 つの出力パラメーターを持つストアード・プロシージャーの実行

この例は、prepare() メソッドを使用して、入出力パラメーターを持つストアード・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ています。

```
<%@ 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();
%>
```

DataBlox へのストアード・プロシージャーの結果セットの設定

この例は、DataBlox へのストアード・プロシージャーの結果セットの取得方法を示しています。

```
<%@ 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);
%>
```

第 15 章 データの提示

一般のユーザー・インターフェース Blox (GridBlox、ChartBlox、および PresentBlox) には、開発者向けに分析表示をカスタマイズするための多くの代替手段があります。このトピックは、使用する Blox や、ユーザー・インターフェース Blox を効果的に使用する方法を決定するときに役立ちます。

データの提示のための Blox の選択

DB2 Alphablox アプリケーションでは、任意の一般ユーザー・インターフェース Blox (GridBlox、ChartBlox、および PresentBlox) を使用して、エンド・ユーザーに結果セットを表示できますが、使用する特定の Blox の決定は、それぞれのユーザー要件によって異なります。

ユーザー・インターフェース Blox は、一度に 1 つのデータ・ソースの結果だけを表示できます。同じページ上で複数のデータ・ソースの結果を同時に表示する必要がある場合、以下のオプションがあります。

- それぞれが異なるデータ・ソースを指す複数の Blox を 1 つのページに置く。
- サーバー側の Java API またはクライアント側の Blox クライアント API を使用してデータ・ソースを変更することにより、1 つのページに 1 つの Blox を使用して、複数のデータ・ソースの結果を表示する。例については、133 ページの『DataSourceSelectFormBlox を使用する異なるデータ・ソースの設定』を参照してください。

データ・プレゼンテーション Blox コンポーネントの選択

本書の前の部分で説明されているように、ユーザーに関係する主な Blox は、データを表示する Blox (ChartBlox、GridBlox、および PresentBlox) です。本書では、DHTML クライアントで使用可能なユーザー・インターフェース Blox の使用方法について主に扱います。

以下は、各データ・プレゼンテーション Blox の利点と欠点を要約した表です。

Blox	利点	欠点
GridBlox	• ユーザーは小さな差異を示す数値をすぐに比較できます • 特定セルの情報を強調表示するアラートを作成できます • (cellLink および cellAlert プロパティーを使用した) 情報リンクを個々のセルまたはセル群に追加できます • (アプリケーション定義で定義されたヘッダー・リンクを使用した) 情報リンクを使用して、行および列ヘッダーの情報にリンクを追加できます	• ユーザーが差異をすぐに表示しようとしても、値の大幅な差異を突き止めるのに手間が掛かることがあります • DataLayout パネルへアクセスできません
ChartBlox	• マルチディメンションまたはリレーショナル・データをさまざまなチャート・フォーマットで表示します	• データを視覚的に表示すると、値の差異がわずかである場合に、値の差異が目立たない場合があります • チャートで情報リンク (GridBlox の利点を参照) にアクセスできません • ユーザーはデータに結び付けられたアラートを確認できません • DataLayout パネルへアクセスできません
PresentBlox	• データのグリッド表示とチャート表示の両方を組み合わせて 1 つの Blox にまとめられます • ユーザーはグリッド表示とチャート表示を切り替えたり、両方のビューを同時に表示できます (分割ペインが使用可能な場合) • 上級者は DataLayout パネルを使用してディメンションの表示を操作できます • 1 つのグリッドまたはチャートだけが使用可能な場合、DataLayout パネルまたは Page パネルにアクセスできます (これらは単独の GridBlox および ChartBlox では使用できません)	• データ密度が高すぎる場合、1 つのグリッドの表示だけをユーザーに許可する場合があります (1 つのチャートに含まれるデータ点が多すぎる場合、解釈が難しくなる可能性があります)。PresentBlox chartEnabled プロパティーを false に設定できますが、そうすると PresentBlox の利点の 1 つが失われます

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

DHTML クライアントのレンダリングは、使用可能な各種 Blox レンダリング・フォーマットの 1 つです。すべての新しいアプリケーションは、DB2 Alphablox レンダリング・ページで DHTML フォーマットで作成されています。その他のレンダリング・フォーマットは、プリンター、Excel、PDF、および XML です。使用する特定のレンダリング・フォーマットに応じて、レンダリング・フォーマットには、一般の Blox render プロパティー、render URL 属性、またはこの項で説明されている他のメカニズムを使用してアクセスできます。

DHTML フォーマット (render=dhtml)

デフォルトでは、DB2 Alphablox で作成されるアプリケーションは DHTML を使用してレンダリングされます。このフォーマットでは、標準の DHTML テクノロジー (HTML、CSS、DOM、および JavaScript) とサーバー側の Java テクノロジーを使用して、強力な Java クライアントに適合する高度な対話式のフォーマットでデータのレンダリングを作成します。ただし、Java プラグインや Java 対応のブラウザーは必要ありません。DHTML レンダリング・フォーマットの別の利点は、Blox UI モデルによって定義されるユーザー・インターフェースです。これは、組み込まれた DHTML フォーマットのサポートを超えて、拡張およびカスタマイズできます。

DHTML レンダリング・フォーマットは、共通 Blox の render プロパティーと、render URL 属性のどちらかを使用してサポートされています。

注: 基本的な JavaScript ファイルがロードされず、ページが別のフォーマットでレンダリングするよう設定されている場合、Blox を DHTML でレンダリングすることはできません。

プリンター・フォーマット (render=printer)

プリンター・フォーマットでは、ブラウザーの組み込まれた印刷機能を使用して、印刷用に最適化された Blox のデータのビューを生成します。Blox ビューは、HTML 表および CSS スタイルを使用して生成されます。さらに、選択可能なページ・フィルターを、ディメンション名およびその選択メンバーを含む、選択されたフィルターのリストに変換します。プリンター・フォーマットでは、DHTML クライアントが作成されたのと同じ Blox UI モデルに基づいてビジュアル表現が生成されます。

プリンター・レンダリング・フォーマットは、一般の Blox render プロパティーとしても、render URL 属性を使用するものとしてもサポートされています。

一般に、このフォーマットを使用するには、Blox ビューを使用するページに HTML ボタンかリンクを組み込み、ユーザーが印刷可能なコピーを要求するときにクリックできるようにします。ユーザーの要求に対し、DB2 Alphablox は、純粋な HTML でページをレンダリングしてからクライアントに送達します。ブラウザーにページが表示されたら、ブラウザーの「印刷」ボタンをクリックして、ページをプリンターに送信できます。詳細は、181 ページの『Blox 出力の印刷』を参照してください。

PDF フォーマット (render=pdf)

将来の参照用として、または他のユーザーと共用するために、PDF ファイルにアクセスできると良い場合があります。DB2 Alphablox には、別の印刷送達メカニズム (pdf フォーマット) が用意されています。これを使用して、ページ上のユーザー・インターフェース Blox を PDF にエクスポートできます。共通 Blox render プロパティーが pdf に設定されているか、または JSP ページの render URL 属性値が pdfに設定された状態で呼び出されると、JSP ページは直接 Adobe Acrobat にロードされます。

より高性能な「**PDF にエクスポート**」機能が提供されていて、エンド・ユーザーはこの機能を使用してグリッドやチャート内のデータを PDF フォーマットにエクスポートできます。「**PDF にエクスポート**」オプションは、メニュー・バーの「ファイル」メニューで選択できます。ツールバーにも「**PDF にエクスポート**」ボタンがあります。このオプションを選択すると、ダイアログがポップアップ表示され、ユーザーはページの向き、PDF に追加するヘッダーまたはフッターのテキスト、およびグリッドまたはチャートを表示する形式を指定できます。DB2 Alphablox には、アプリケーション開発者がプログラマチックにページ・レイアウトをカスタマイズできるオプションが提供されており、改ページがあるときのチャート・サイズを指定したり、ヘッダーやフッターにロゴと特定のテキストが表示されるようにしたりできます。

PDF レポートの機能の詳細を学習し、この機能をアプリケーションに追加する場合には、276 ページの『PDF にエクスポート』を参照してください。

「Excel にエクスポート」フォーマット (render=xls)

render Blox プロパティーまたは render URL 属性が xls に設定されている場合、DB2 Alphablox は出力を Microsoft Excel スプレッドシートにロードされる HTML フォーマットで送信します。このフォーマットを使用する場合、戻されるページの MIME タイプは、application/vnd.ms-excel に設定され、標準の MIME タイプは、Microsoft Excel に設定されます。

このレンダリング・フォーマットは単に HTML 出力に MIME タイプを設定するだけなので、データは通常の Excel フォーマットではありません。グリッド内のデータはそのままロードされ、チャートはグラフィックとしてロードされます。

より高性能な「**Excel にエクスポート**」機能が提供されており、エンド・ユーザーはこの機能を使用してグリッドやチャート内のデータを通常の Excel フォーマットにエクスポートできます。「**Excel にエクスポート**」オプションは、メニュー・バーの「ファイル」メニューで選択できます。ツールバーにも「**Excel にエクスポート**」ボタンがあります。このオプションを選択すると、ダイアログがポップアップ表示され、ユーザーは使用する Excel テンプレートを選択し、さらに Excel で開くかまたはスプレッドシートをシステムに保存するかを選択できます。

Excel にエクスポートするこの拡張機能についての完全な説明は、271 ページの『Excel へのデータのエクスポート』を参照してください。

XML フォーマット

単独の DataBlox (つまり、他の Blox 内にネストされていないもの) で、render URL 属性を XML (render=xml) に設定することにより、アプリケーション・データ・ソースからの照会結果セットを XML フォーマットでレンダリングできます。XML フォーマットの使用についての詳細は、286 ページの『XML へのエクスポート』で説明されています。

送達フォーマットの指定

デフォルトでは、アプリケーション・ページは DHTML フォーマットで送達されます。複数のアプリケーション送達フォーマットを使用可能にするのに、特別なステップは必要ありません。アプリケーションの URL に追加された属性が、指定したフォーマットでページを送達するよう DB2 Alphablox に通知します。たとえば、次の URL は、MyApplication ページを DHTML フォーマットで送達するよう要求するものです。

```
http://<server>/applications/ThisView.jsp?render=dhtml
```

また、ページをプリンター・フォーマットでレンダリングする場合、URL は次のようになります。

```
http://<server>/applications/ThisView.jsp?render=printer
```

DHTML クライアントで render URL 属性を使用するときの有効な値には、以下の値があります。

Render 属性
説明

dhtml デフォルト。クライアントで DHTML テクノロジーを使用して Blox をレンダリングします。

none ページからすべての Blox を除去します。

printer
対話性なしでプリンター対応フォーマットでレンダリングします。

xls ページを HTML フォーマットでレンダリングし、ページの MIME タイプを application/vnd.ms-excel に設定します。ページは Microsoft Excel にエクスポートされます。

xml XML フォーマットにレンダリングします (明示的な DataBlox にのみ該当)

Blox 出力の印刷

プリンター・フォーマットおよび PDF レポート・フォーマットという 2 つの選択オプションは、Blox 結果を表示する印刷可能ページを生成するときに使用可能です。プリンター・フォーマットは、 PNG イメージおよび HTML 表を使用して、Web ページ上の Blox コンポーネントをレンダリングします。また、任意の PageBlox ページ・フィルターは (スタンドアロンであっても PresentBlox 内にネストされていても)、対話式の HTML 選択リストから、ディメンションおよび各ディメンションの選択スライサーの静的リストに変換されます。

PDF オプションでは、PDF ファイルが生成され、アセンブラーおよびユーザーによって生成された印刷可能ファイルに対してより柔軟に対応できます。PDF レポート・フォーマットの使用の詳細については、276 ページの『PDF にエクスポート』を参照してください。

HTML ベースの印刷による印刷

HTML ページ上の Blox は、ブラウザーの「印刷」ボタンを使用して印刷できますが、DB2 Alphablox を使用すると、下記の例で説明されているように、Blox 出力をプリンターにより良く対応したフォーマットでレンダリングできます。レンダリングされた出力は、ブラウザー・ウィンドウに表示され、対話式の Blox は印刷可能なバージョンに置き換えられます。

注: デフォルトでは、Microsoft Internet Explorer ブラウザーは、背景色およびイメージは印刷しません。 Internet Explorer で背景色およびイメージを印刷するには、使用しているブラウザーを手動で構成する必要があります。そのため、大多数のユーザーは、この設定を使用可能にしないと思われます。こちらでブラウザー設定を制御することはできないため、ほとんどのユーザーがこの設定を変更していないという想定の下で (ほとんどのユーザーは、この設定について知らないか、ブラウザー・オプションで操作できることを知りません) 印刷可能ページを設計する必要があります。

このプロパティーを設定するには、ブラウザーのメニュー・バーで、「表示」をクリックしてから「インターネット オプション」をクリックします。「詳細設定」タブを選択し、「印刷」セクションにスクロールダウンします。次に、「背景の色とイメージを印刷する」オプションをチェックして、「適用」をクリックします。

render=printer URL 属性を使用した印刷可能ページの作成

印刷可能ページをレンダリングする最も簡単な方法は、`render=printer` URL 属性を使用することです。ページの URL の最後に `?render=printer` を付加することにより、そのページの Blox は、プリンターで扱いやすいフォーマットでレンダリングされます。 Blox が PresentBlox である場合、Blox ビューの最上部にあるリストに、選択可能なページ・フィルターが表示されます。ここでは、印刷可能コピーが生成されたときに、アクティブであったスライサーを示します。通常は、URL 属性を既存のページに適用するのではなく、印刷用のビューを表示した新しいブラウザー・ウィンドウをオープンするほうが望ましい方法です。

この方法を使用して印刷可能ページを作成するには、以下のステップに従ってください。

1. Blox を使用したページ上で、印刷可能ページを生成するためのボタンまたはリンクを追加します。たとえば、以下の HTML コードは JSP ページの本文に “Print Preview”というラベルのボタンを作成します。このボタンは、現在のページのビューを新しいウィンドウで開き、`printer` モードでレンダリングします。

```
<form>
<input type="button" value="Print Preview"
   onclick="window.open('mView.jsp?render=printer','_new')">
</form>
```

2. ここで、ビューをオープンしてボタンをテストします。新しいウィンドウがオープンし、印刷可能フォーマットでレンダリングされた現在のビューが表示されるはずです。

このページには、追加したボタンが含まれています。一般に、このことは望ましいことではないため、たとえば、フレーム・セットの特定フレームに (「印刷」ボタンおよび「Excel にエクスポート」ボタンを含む) 共通サービス・ボタンを置き、分析ビューを個別のフレームで表示できます。そうすると、ボタンを押すときに、ビュー・ページを個別の印刷可能ページとしてオープンできると同時に、そのボタンがページに表示されることを避けることができます。

さらに良い方法は、新しいページを個別のウィンドウでオープンすることです。カスタム印刷ページには、適切な印刷可能ページとして、さらに大きな可能性があります。次の作業では、カスタム印刷ページを生成するための 1 つの方法を紹介します。

<blox:display> タグを使用したカスタム印刷ページの作成

以下のステップは、<blox:display> タグを使用するページを作成し、同じデータ・ビューを含む新しいページを印刷可能フォーマットでレンダリングする方法を示しています。

1. 分析ビューを作成し、このビューのカスタム印刷ページをオープンするのに使用されるボタンを追加します。次のコード部分では、MyView-print.jsp が表示される新しいウィンドウをオープンするボタンが作成されます。

```
<form>
  <input type="button" value="Print View"
  onclick="window.open('MyView-print.jsp','_new')">
</form>
```

2. ここで、カスタム印刷ページを作成します。ビューをレンダリングするための <blox:display> タグに加えて、以下の項目を含めることを考慮できます。
   - 印刷されるビューのタイトル
   - 内容の要約
   - 会社名またはロゴの図
   - ページが生成された日付
   - 使用上の警告 (たとえば、内部や機密)
   - 著作権表示

   次のコード部分は、簡単な印刷ページの一例です。タイトル、printer モードでレンダリングされる Blox、および印刷された日付を含む印刷可能ページが生成されます。

```
<h2>My Grid View</h2>
<blox:display bloxRef="MyGridBlox2" render="printer"/>
<p>
  Printed: <script>document.write(new Date());</script>
</p>
```

3. カスタム印刷ページを保管します。

この方法を使用することにより、カスタマイズされた多くの印刷オプションを提供できます。

上記の例を実際に運用した例は、『データの提示』の下の『Blox Sampler』にあります。生成される印刷ページでは、ブラウザーの「ファイル」メニューを選択してページを印刷できます。

CSS テーマ

DB2 Alphablox では、Cascading Style Sheets (CSS) テーマを使用して、レンダリングされるページのレイアウトと外観の特徴を制御します。テーマは、アプリケーションのルック・アンド・フィールを拡張したものです。カスタム・テーマを作成することにより、それぞれの組織は、DB2 Alphablox アプリケーションで会社固有の様式を採用したり、既存の Web ベース・アプリケーションまたはポータルと調和した様式を作成できるようになります。

テーマの指定

Blox の外観のテーマは、次の 4 とおりの方法で指定できます。

- DB2 Alphablox 管理ページの「**デフォルト HTML クライアント・テーマ**」設定は、この DB2 Alphablox インスタンスのすべてのアプリケーションにデフォルト・テーマを設定します。インストールしたばかりのときは、coleman テーマが選択されています。異なるシステム・デフォルト・テーマを指定するには、DB2 Alphablox 管理ページの「**管理**」タブをクリックします。次に、左側のメニューの「**一般プロパティー**」セクションから、「**システム (System)**」を選択します。「**デフォルト HTML クライアント・テーマ**」オプションには、テーマを選択するドロップダウン・リストがあります。この方法で設定する場合、テーマはシステム・レベルで設定されます。
- DB2 Alphablox 管理ページの各アプリケーションの定義には、「**デフォルト・テーマ (Default Theme)**」設定が含まれています。この方法で設定する場合、テーマはアプリケーション・レベルで設定され、システム・レベルの設定をオーバーライドします。「**デフォルト・テーマ (Default Theme)**」を「デフォルト (default)」に設定した場合には、システム・レベルのデフォルト・テーマが適用されます。
- <blox:header> タグには theme 属性があります。この方法で設定する場合、テーマはページ・レベルで設定され、アプリケーション・レベルまたはシステム・レベルのテーマ設定をオーバーライドします。
- ページが呼び出されるときに URL に追加される theme URL 属性は、それ以外のすべてのテーマ設定をオーバーライドします。

theme プロパティーを URL 属性として指定するには、次のフォーマットを使用します。

```
http://.../application/view.jsp?theme=financial
```

URL 属性を使用してテーマを定義する場合、定義された theme プロパティーが該当ページのすべての Blox に適用されます。

CSS テーマ・ファイル

DB2 Alphablox CSS テーマのファイルは、DB2 Alphablox リポジトリーの theme ディレクトリーにあります。

```
<alphabloxDirectory>/repository/theme
```

DB2 Alphablox テーマは、以下の構造で構成されます。

<themeName>.properties
: ファイル場所、レイアウト、および色についてのテーマ仕様が含まれます。<themeName>.properties ファイル設定は、下記のように指定されます。

<themeName>_dhtml.css
: DHTML レンダリング・フォーマットに固有のテーマ仕様が含まれます。

i
: テーマに固有のイメージを含むフォルダー。

DB2 Alphablox テーマ・ファイルは、属するテーマに基づいて名前が付けられます。たとえば、financial.properties および financial_dhtml.css は、finanical テーマ・オプションのテーマ・ファイルです。これらのファイルは、やはりテーマ・オプションに基づいて名前を付けられたテーマ・ディレクトリーにあります。たとえば、financial テーマ・ファイルは、次のディレクトリーにあります。

```
<alphabloxDirectory>/repository/theme/financial
```

注: ポートレット開発の場合、テーマの作成にはポータル・テーマ・ユーティリティーを使用してください。このユーティリティーは選択したポータル・テーマを DB2 Alphablox テーマと結合します。このツールは、DB2 Alphablox ホーム・ページの「管理」タブから使用できます。

themeName.properties ファイルで定義される CSS テーマ・プロパティー

テーマ・プロパティー・ファイル (<themeName>.properties) は、以下の DHTML 関連テーマ・プロパティーを定義するプレーン・テキスト・ファイルです。

プロパティー
: **説明**

name =*<themeName>*
: テーマの名前 (たとえば、coleman または financial)

description =*description*
: サーバー・コンソールに表示されるテーマの説明。

bgcolor = #3d3d5f
: <themeName>.css で指定されない ChartBlox 領域の背景色。

fgcolor = white
: *themeName*.css で指定されない ChartBlox 領域の前景色 (テキスト)。

css = <themeName>.css
: *themeName*.css ファイルの名前

background = true
: Blox の背景を表示します。

privateimages = true
: このテーマで使用されるすべてのイメージで、テーマの専用イメージ・ディレクトリー (<repository>/theme/<*themeName*>/i/) を使用します。

windowbgcolor = #655973
PresentBlox 内のチャート表示領域の背景色。

windowfgcolor = white
PresentBlox 内のチャート表示領域の前景色 (テキスト)。

bkgrd_image_chart = *imageFile*
チャートに示される背景イメージ。

chart_color_series
線、棒、扇形スライスなどを含め、チャートの色を作成する際に使用される 18 の色。

.css ファイルで定義される CSS クラス

themeName.css ファイルは、下記の CSS クラスを定義するプレーン・テキスト・ファイルです。特定テーマのクラスに設定された値を確認するには、次の場所にあるテーマの *themeName*_dhtml.css ファイルをオープンします。

```
<alphabloxRepository>/theme/<themeName>/themeName_dhtml.css
```

ヒント: Cascading Style Sheets 仕様は、World Wide Web Consortium (http://www.w3c.org/style/css) にあります。

以下の表は、アプリケーション全体のスタイル、レガシー・スタイル、および *themeName*_dhtml.css ファイルで使用可能なオーバーライドをリストしたものです。

アプリケーション全体のスタイル

csApBg アプリケーションの背景色 - Blox の背景全体

csCmpBg
コンポーネントの背景色 - 個々の Blox のデータ領域の背景

csCmpBrdr
コンポーネントの枠 - 個々の Blox およびコントロールの枠

csThmClr
基本テーマの色 - テキスト・ラベル、情報テキスト、装飾エレメントで使用

csFntClr
デフォルトのフォントの色 - データ、メッセージ、機能テキストで使用

csFntSpc
デフォルトのフォントの仕様 - 主にメニュー、ツールバー、およびボタンで使用

csLblFnt
デフォルトのラベル・フォントの仕様 - GUI 対話式コントロールのラベルで使用

csGrdFnt
デフォルトのグリッド・フォントの仕様 - すべてのグリッド・セルで使用

csSlctBg
デフォルトの選択項目の背景色

csSlctClr
デフォルトの選択項目フォントの色

csDsbldClr
デフォルトの使用不可フォントの色

csThrDBrdrRsd
3D 隆起の枠

csThrDBrdrLwrd
3D 陥没の枠

csBckClr
背景色 (Blox コンポーネント)

csWndwClr
「ウィンドウ」の色 (Blox コンポーネント)

グリッドのスタイル

csDmnsnHdrs
ディメンション・ヘッダー

csClmnHdrs
列ヘッダー

csRwHdrs
行ヘッダー

csRwHdrsNnBnd
行ヘッダー - バンドなし

csRwHdrsBnd
行ヘッダー - バンド付き

csDtCl データ・セル

csDtClBnd
データ・セル - バンド付き

csDtClNnBnd
データ・セル - バンドなし

オーバーライド

スタイル・クラス
説明

csRwHdrs
行ヘッダー

csRwHdrsNnBnd
行ヘッダー - バンドなし

csRwHdrsBnd
行ヘッダー - バンド付き

csDtCl データ・セル

csTdClBnd
データ・セル - バンド付き

csDtClNnBnd
データ・セル - バンドなし

定義されたスタイルのオーバーライド

<blox:header> タグを JSP ページの head セクションに追加した結果の 1 つは、適切な CSS ファイルへの自動リンクが設定されることです。

注: データ・セルおよびセル・アラートにスタイルを適用することについて説明した、 197 ページの『フォーマット・マスクを使用したデータの強調』および 199 ページの『セル・アラートを使用したデータの強調』も参照してください。

定義されたスタイルをオーバーライドするため、まったく新しいテーマを作成できます。

1. 既存のテーマ・ディレクトリーをコピーして、新しい名前を付けます。
2. そのディレクトリー内の *themeName*.cssおよび *themeName*.propertiesを名前変更します。
3. これらのファイルに、適切な内容変更を行います。
4. アプリケーション URL に、新しいテーマの名前を指定します。view.jsp というページであれば、URL は次のようになります。

```
/<applicationName>/view.jsp?theme=MyTheme
```

注: 新しいテーマをロードするか変更されたテーマを再ロードするには、DB2 Alphablox コンソールで load theme コマンドを使用します。このコマンドは、特定のテーマ名をパラメーターとして受け入れないことに注意してください。このコマンドは、リポジトリーからすべてのテーマをロードするだけです。

注: Web ブラウザーは、ボタンやアイコンの GIF ファイルなど、ファイルを頻繁にキャッシュに入れます。新しいテーマや変更されたテーマをテストする場合、行なった変更を表示するときには、まずブラウザー・キャッシュに入っているキャッシュをクリアしなければならないことがあります。

セル・アラートへのスタイルの適用

セル・アラートのスタイルは、CSS ファイルでは定義されません。代わりに、HTML 内で DataBlox cellAlert プロパティーに基づいてインラインで定義されます。 CSS にはカスケード影響があるため、インライン・スタイルが、ブラウザーでレンダリングされるスタイルになります。データ・セルに定義された他のスタイルはオーバーライドされます。

ユーザー・インターフェースの外観

Blox は、アプリケーション・ページの外観、すなわち「ルック・アンド・フィール」を補足できる、異なる色スキーム、フォント、およびバンド特性を表示するよう構成できます。Cascading Style Sheet を使用して Web ページのフォントの色や背景色を制御できるように、Blox プロパティーおよび HTML テーマを使用して、

アプリケーション・ページでの Blox の外観も制御できます。以下のセクションでは、一般的な外観プロパティーについて説明します。

下記の表に示されているのは、プレゼンテーション Blox での最も一般的な変更対象外観の機能をリストしたものです。

Blox　一般的な外観プロパティー

GridBlox

- `missingValueString`: デフォルトの `#MISSING` の代わりに表示する内容を決定します
- `rowHeadingsVisible`: データ値の左側の行ヘッダーをグリッド上に表示するかどうかを指定します
- `gridLinesVisible`: グリッド行をオン/オフします

ChartBlox

- `chartType`: チャート・タイプを変更します
- `depthRadius`: 標準の棒グラフで微妙な 3D 効果を作成するのに使用できます
- ラベルおよび配置

PresentBlox

- `dividerLocation`: splitPane が使用可能な場合、PresentBlox のロード時に仕切りを表示するかどうかを決定します
- `splitPane`: グリッドおよびチャートを同時に表示できるようにするスプリッター・バーを使用可能にします
- `splitPaneOrientation`: スプリッター・バーが水平なのか垂直なのかを決定します

DataBlox

- `suppressMissing`: データのない行または列を抑制します
- `suppressNoAccess`: ユーザーがデータへのアクセス権を持たない行および列の表示を抑制します
- `suppressZeros`: すべてがゼロである行または列を抑制します

ToolbarBlox

- `removeButton`: 定義されたボタンをツールバーから除去します
- ボタンの色およびサイズ

ReportBlox

- 「*Relational Reporting 開発者用ガイド*」を参照してください。ほとんどの HTML エレメントで、CSS スタイル設定は変更できます

外観プロパティーの完全なリストについては、「*開発者用リファレンス*」の各『Blox タグ・リファレンス』セクションの『カテゴリー別』セクションにある外観プロパティーおよびタグのリストを参照してください。

次のセクションでは、GridBlox、ChartBlox、および PresentBlox で使用される一般的な外観プロパティーの一部を詳しく説明します。

グリッドの外観

グリッドは、重要な情報源になる場合がありますが、それを読んだり、重要な情報に注意を向けるのが難しいこともあります。開発者として、グリッドの外観プロパティーの多くを変更できますが、次に示すのは、最も一般的な変更対象プロパティーです。

- `missingValueString`: デフォルトの `#MISSING` の代わりに表示する内容を決定します
- `rowHeadingsVisible`: データ値の左側の行ヘッダーをグリッド上に表示するかどうかを指定します
- `gridLinesVisible`: グリッド行をオン/オフします

下記のセクションでは、このようなプロパティーの一部と、それらを使用してさらに使いやすいグリッドを作成する方法について説明します。

行のバンディング

多数の行があるグリッドでは、行のバンディングがデフォルトで使用可能です。美観上の理由のため、またはクラスに選択した可能性のあるセル・アラートの色が付けられるのを避けるため、必要であれば、CSS テーマ・プロパティーを変更することにより、行のバンディングで使用されている背景色および前景色を変更できます。変更可能な CSS テーマのリストを含め、CSS テーマの詳細は、184 ページの『CSS テーマ』を参照してください。

セルの外観

結果セットを表示するグリッド・データ・セルは、別の前景色、背景色、およびフォントで表示するようカスタマイズすることができます。このオプションを制御するには、CSS テーマを使用します。変更可能な CSS テーマのリストを含め、CSS テーマの詳細は、179 ページの『プリンター・フォーマット (render=printer)』を参照してください。

チャートの外観

DB2 Alphablox アプリケーション内のチャートの外観は、ユーザーの特定のニーズを満たすように、さまざまな方法でカスタマイズできます。ここでは、一般的な変更対象 ChartBlox プロパティー (`chartType`、`depthRadius`、および `chart_color_series`) について説明します。

チャート・タイプ

ChartBlox で最も頻繁に変更されるプロパティーは、`chartType` です。これは、`Vertical Bar, Side-by-Side, 3D Effect` をデフォルトとして取ります。ChartBlox には、他にもほとんどすべてのユーザーのニーズを満たせる多くのチャート・タイプがありますが、最も一般的に使用されるチャート・タイプは、`Bar`、`Line`、`3D Bar`、および `Pie` の 4 つです。`chartType` プロパティー設定では、これらの短い名前を使用することもできますし、完全なチャート・タイプ名を使用することもできます。サポートされるすべてのチャート・タイプの有効な名前の完全なリストは、「*開発者用リファレンス*」の『ChartBlox タグ・リファレンス』セクションを参照し

てください。次の表は、使用可能な 4 つのチャートのショートカット名および完全なチャート・タイプ名をリストしています。

ショートカット名
完全なチャート・タイプ名

Bar 垂直バー、横並び

3D Bar 3D 棒グラフ

Line 縦線、絶対

Pie 円グラフ

チャートへの 3D 外観の追加

ChartBlox のデフォルトの chartType プロパティーは、Vertical Bar, Side-by-Side, 3D Effect です。これにより、微妙な 3D 効果が付けられた 2 次元の棒グラフが生成されます。標準的な棒グラフ (Bar または Vertical Bar, Side-by-Side) または線グラフ (Line または Vertical Line, Absolute) は平面で 2 次元ですが、depthRadius プロパティーを設定することにより、それぞれが選択した微妙な 3D 効果をこれらのチャートに追加し、違った外観にすることができます。平面の棒グラフおよび線グラフにわずかな深さを追加するには、たとえば、次のようにして、 depthRadius プロパティーと chartType プロパティーを組み合わせます。

```
<blox:chart
  ...
  chartType="Bar"
  depthRadius="5"
  ... />
```

depthRadius の許容値は、0 から 100 の間の整数です。 depthRadius の設定は、他の 2D チャートの外観にも影響します。

チャートの色

線、棒、および扇形スライスの作成に使用するチャートの色は、使用される CSS テーマに基づいて設定できます。特定のテーマで使用されているデフォルト色以外のチャート色を指定する場合、18 の異なる色で構成される独自のチャート色系列を定義できます。テーマで使用する別の色系列を指定するには、次のディレクトリーにある *themeName*.properties ファイル (たとえば、coleman.properties) をオープンします。

<alphabloxRepository>/theme/*themeName*/

chart_color_series プロパティーを探し、色の定義に使用される 18 の色を再定義します。チャート色は、Web ページで一般に使用されている値のような、16 進コード (たとえば、 #E0CB68) を使用して指定されます。

PresentBlox の外観

以下のトピックでは、PresentBlox の外観を簡単に変更できる方法をいくつか説明します。

ペインの分割

デフォルトでは、PresentBlox は 2 つのペインの表示をインスタンス化します。一方はグリッドになり、もう一方はチャートになります。splitPane プロパティー (デフォルトでは true に設定) を使用すると、データを表およびグラフィカル表現で一度に表示できます。また、ユーザーが一方のペインでデータを処理すると、他方のペインも同時に更新されて、最初のペインでの変更が反映されます。たとえば、ユーザーがチャートでデータをドリルダウンすると、グリッド・ペインでもその新しい結果セットが反映されます。

ペインの分割が使用可能な場合、仕切りはデフォルトで vertical に設定されています。グラフの Y 軸に多数の項目が表示される場合、dividerLocation を horizontal に変更して、PresentBlox の幅全体でグリッドとチャートの両方を表示することを考慮できます。 horizontal (水平) の設定を使用しているときには、グリッドで表示するよりも、チャートで表示したほうが見栄えが良いことに気が付くこともよくあります。このことは、chartFirst プロパティーを true に設定して、デフォルトをオーバーライドすることで指定できます。

有効な画面スペースが足りないか、初期結果セットのデータ密度のせいで初期グラフが読めないため、 splitPane を false に設定するほうがよいと考えられる場合があります。その方が妥当な選択であっても、(このオプションは使用可能な「ツールバー」オプションでは変更できないことを考慮して) ペインの分割オプションをユーザー側が使用できるようにしておくことができます。ただしこの場合、ペイン分割の仕切りの初期表示の場所を変更し、仕切りを一方の側に設定すると同時に、使用可能なままにしておきます。

dividerLocation プロパティーを使用すると、スプリッター・バーの初期の場所を設定できます。許容値は、0 から 1 です。0 の値は、右側 (または、splitPaneOrientation の設定に応じて下部) の表示だけが現れることを意味し、1 の値は、左側 (または上部) の表示だけが現れることを意味します。いくつかの異なる設定を試してみて、デフォルト値の 0.5 から変更することが有効かどうかを確認してください。

splitPane、splitPaneOrientation、dividerLocation、および chartFirst プロパティーの詳細は、「*開発者用リファレンス*」のPresentBlox タグ・リファレンスを参照してください。

DataLayout プロパティーの変更

デフォルトでは、DataLayoutBlox または DataLayout パネルを使用できますが、エンド・ユーザーには示されません。大部分の上級者が使用できる分析ビューでは、PresentBlox のロード時に DataLayout パネルを表示することを決定できます。PresentBlox のロード時に DataLayout パネルを表示するには、次の例に示されているように、ネストされた DataLayoutBlox の visible 属性を true に設定する必要があります。

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

ユーザーが DataLayout パネルを使用できないようにする場合、たとえば、臨時ユーザーだけが PresentBlox を表示して使用できるようにする場合、PresentBlox dataLayoutAvailable 属性を false に設定することにより、DataLayout パネルを使用不可にできます。これにより、DataLayout ボタンが自動的にツールバーに表示されることはなくなります。次のコード部分は、このプロパティーの適切な使用法を示しています。

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

DataLayout パネルの可用性は、PresentBlox dataLayoutAvailable プロパティー設定によって決定され、<blox:present> タグ上の属性になります。 DataLayout パネルの可視性は、DataLayoutBlox オブジェクト自体のプロパティーであるため、<blox:dataLayout> タグの visible 属性を使用して制御されます。

メニュー・バー・プロパティーの変更

メニュー・バーは、Blox の最上部に表示されるテキスト・ベースのメニューであり、Blox が、GridBlox、ChartBlox、または PresentBlox であるかどうかに応じて、関係するメニューを自動的に取り込みます。デフォルトでは、<blox:grid>、<blox:chart>、および <blox:present> の menubarVisible タグ属性は true に設定されています。いずれかの Blox からメニュー・バーを除去するには、menubarVisible タグ属性を Blox タグに追加し、値を false に設定します。

メニュー・バーの表示/非表示以外には、外観を制御するためのタグまたはタグ属性はありません。上級開発者は、Blox UI モデルの拡張性を使用して、メニュー・バーを独自にカスタマイズできます。

ツールバー・プロパティーの変更

デフォルトでは、ユーザーはツールバーを PresentBlox 上で使用可能です。開発者によって頻繁に変更される一般的な外観設定がいくつかあります。ツールバーはいつでも使用可能であると見なされますが、ToolbarBlox タグの visible 属性を使用することにより、ユーザー側からは見えないように設定できます。

ツールバーを表示したままにする場合でも、一部のツールバー・ボタンを、使用不可にするか除去することを選択できます。ボタンをツールバーから除去する場合、ネストされた ToolbarBlox の removeButtons プロパティーを使用し、不要であると考えるすべてのボタンを除去します。たとえば、次の例では、ToolbarBlox から Help が除去されます。

```
<blox:present ...>
  <blox:toolbar removeButtons="Load,Save,Help"/>
</blox:present>
```

注: Load および Save は、このプロパティーのデフォルト値ストリングに含まれているため、それらを忘れずに removeButtons タグ属性に追加しなければならないことに注意してください。ストリングに追加することを忘れると、「Load」および「Save」ボタンがツールバーに表示されます。

データの外観

データ・ソースから結果セットを取り出す場合、データは、あらかじめフォーマットされている状態、またはフォーマットされていない状態で戻すことができます。結果を DB2 Alphablox アプリケーションで取り出したら、外観を制御し、データをフォーマットするためのいくつかの方法を使用できます。DataBlox プロパティーを使用することにより、ゼロ、欠落データ (または NULL 値)、重複データ、またはユーザーにアクセス権のないデータを含む行または列を抑制できます。GridBlox では、記号 ($、% など) や (コンマまたはピリオドを示す) グループ分け記号を含め、数値をフォーマットできます。さらに、GridBlox では、数値を、千単位、百万単位、またはそれぞれのデータに適切と思われるグループ分け記号で表示できます。

GridBlox プロパティー

データ値のフォーマットを変更し、グリッドでのデータの外観を変更するために、defaultCellFormat、cellFormat、および formatMaskの 3 つの GridBlox プロパティーを使用できます。以下のタスクでは、これらのフォーマット・マスク・プロパティーを使用して、頻繁に生じる一部の作業を解決する方法を学習します。これらのプロパティーの使用法についての完全な詳細は、 GridBlox タグ・リファレンスを参照してください。

千単位および百万単位での値のフォーマット

エンド・ユーザーからのよく寄せられる要求は、関係のない数値を除去する、すなわち実質的に丸めることで、データに含まれる不必要なノイズを除去するということです。ユーザーが数千ドル単位の予算で作業している場合、セント単位やドル単位を表示しても役に立たないことがあり、データ・セルで不必要なスペースを占めるだけのことがあります。開発者としては、GridBlox の defaultCellFormat プロパティーを変更することにより、このユーザーの役に立てます。たとえば、すべてのグリッド値を千単位で表示する場合、以下の 2 つの方法があります。千の単位を表す“K”を後に付けて、値を千単位で表示するには、次の設定を入力します。

```
defaultCellFormat="#,###K"
```

値の K は、数値が千単位であることを DB2 Alphablox に通知するものです。ただし、K は値の最後に付加します。

K は、値が千単位で表示されていることを示すものですが、多くのユーザーは、すべてのフィールドで K が表示されることを望みません。グリッド値を千単位で表示しますが、K 接尾部を使用しない場合、 DB2 Alphablox アプリケーションのすべてのフォーマット・マスクで使用できる機能を使用し、次の設定を入力することにより、数値を千単位で計算できます。

```
defaultCellFormat="#,###/1000"
```

一部の状況では、たとえば、百万を表す“B”のような特殊文字を最後に表示できます。この値の最後に“B”を追加するには、前の設定を次のように変更します。

```
defaultCellFormat="#,###/1000'B'"
```

defaultCellFormat および他のフォーマット・マスク・プロパティーの使用法についての詳細は、GridBlox タグ・リファレンスを参照してください。

特定メンバーのパーセンテージの表示

GridBlox defaultCellFormat プロパティーを使用する場合、値のグリッド全体が影響を受けます。しばしば、値のフォーマットを特定の行または列に制限したい場合や、defaultCellFormat プロパティーは、特定のメンバーの値を除くすべての値に使用したい場合があります。特定メンバーに対する数値フォーマットを制限し、1 つの行または列だけに影響を与えるようにするため、cellFormat プロパティーを使用できます。 defaultCellFormat プロパティーとは異なり、cellFormat は、索引付きプロパティーで、独自の Blox タグを使用する必要があります。これは個別のタグのことであり、GridBlox プロパティーを表しているため、cellFormat タグは GridBlox タグの本体内にネストする必要があります。ここに示すのは、メンバーのそれぞれの値の後にパーセント記号 (%) を付けた Variance % 値を表示するだけの場合に、ネストされていない GridBlox の中での cellFormat タグの内容を示すコード部分です。

```
<blox:grid id="myGridBlox">
  <blox:cellFormat
    format="#,###.00%"
    scope="{Scenario: Variance %}" />
</blox:grid>
```

この例では、Variance % は、値をパーセントで小数点以下 2 桁まで示します。

小数部の外観の制御

小数部がデータの外観に与える影響がすぐに明らかになるとは限りませんが、ここに示すのは、データの表示のためにグリッドが読み難くなっている例です。次のグリッドで、配置内に示されたどの場所の値も、列の数値が行内で整列されていないことに注意してください。

7.654
3.21
43.21
543.2
3
3.2

ここから分かるように、数値の小規模なサンプルであっても、小数部の位置に基づいて意識的に数値を整列してみる必要があるため、値を比較するということは難しいことです。このような値のすべての小数部が列内で整列するように、フォーマット・マスクを定義できます。たとえば、次のように <blox:cellFormat> タグを使用して、 Variance % 列を挙げることができます。

```
<blox:cellFormat
  format="#,###.000"
  scope="{Scenario:Variance %}"/>
```

これで、Variance % 列はデータを次のように表示するようになります。

7.654
3.210

43.210
543.200
3.000
3.200

この例で学習したように、適切なフォーマット・マスクを使用するなら、データはより読みやすく、より分かりやすく、そして見栄えがさらに良くなります。フォーマット・マスクは、負の値を括弧で囲むか赤字で表示したり、通貨記号を表示したり、パーセント記号を表示するためにも使用できます。他のフォーマット・マスク・オプションの詳細は、「*開発者用リファレンス*」のGridBlox タグ・リファレンスを参照してください。

第 16 章 情報の強調とコメント

残りのデータとは大きく異なる情報に注意を引くにはどのようにできるでしょうか。「例外報告」または「トラフィック・ライト」として一般に言及されるものがありますが、共通の目的は、決定する際に重要になり得る情報をユーザーに知らせることです。このトピックでは、この問題を解決するための、DB2 Alphablox セル・アラートおよび情報リンクの使用方法を説明します。セル・アラートは、データ・セルのスタイルを変更することで情報を強調するのに使用できますし、特定の基準に基づいてリンクを表示することにも使用できます。ヘッダーやセル・リンクを含む情報リンクでも、セルを強調する手段が備えられていて、ユーザーの注意をさらに多くの情報に向けています。

ユーザーがグリッド内の特定データに関するコメントを追加して表示する機能は、情報を強調するための別の強力な手段です。このセクションのトピックでは、CommentsBlox を使用してこの機能をアプリケーションに追加する方法を扱っています。

概説

会社のデータベースに保管されている膨大な情報にアクセスできるようになると、ユーザーが重要な情報を迅速に見付けられるよう支援する方法が問題になります。DB2 Alphablox 開発者は、グリッドでのセル・アラートやハイパーリンクのような技法を使用して、特定のビジネス基準に基づいて情報を強調したり、使用しているアプリケーションに関連した他の情報にユーザーをリンクしたりすることができます。以下のセクションでは、セル・アラート、セル・リンク、およびセル・アラート・リンクを使用して、重要な情報や補助的な情報に注意を向ける方法を学習します。

ユーザーがグリッド内の特定データに関するコメントを追加して表示する機能は、情報を強調するための別の強力な手段です。このトピックでは、ユーザーがマルチディメンション・データベース内のデータ・セルにコメント (または注釈) を追加して表示できるようにする方法についても学習します。

フォーマット・マスクを使用したデータの強調

グリッド内の負の値は、デフォルトで値の前に負符号 (-) を表示します。別の方法としては、`defaultCellFormat` または他のフォーマット・マスク・プロパティーを使用して、負の値を括弧で囲むか赤字にして表示できます。 Microsoft Excel のように、DB2 Alphablox アプリケーションのフォーマット・マスクを使用して、負の値を赤字で表示できます。

負の値を赤字で強調する

グリッドのすべての負の値を赤字で強調するには、負の値を赤字で表示するいずれかのフォーマット・マスクを使用して defaultCellFormat を設定します。値はすべて、cellStyle の値 (デフォルトではすべての値が黒で表示される) に従ってフォーマットされます。

次の例では、正の値はすべての値が小数点以下 2 桁まで表示され、整数と小数はコンマで区切られます。 (セミコロンの右側のフォーマット・マスクで示される) 負の値はすべて、正の値と同じフォーマットで表示されますが、値が赤字で表示されるという点だけが異なります。

```
<blox:grid ...
  defaultCellFormat="#,###.00;[red]#,###.00"
</blox:grid>
```

特定メンバーの負の値だけを赤字で表示するのであれば、cellFormat プロパティーを使用する必要があります。次の例では、メンバー Actual の負の値が赤字で表示されます。

```
<blox:grid ...>
  <blox:cellFormat
    format="#,###.00;[red]#,###.00"
    scope="{Scenario:Actual}"/>
</blox:grid>
```

defaultCellFormat および cellFormat プロパティーの使用法についての詳細は、GridBlox タグ・リファレンスを参照してください。

負の値を括弧で強調する

別の方法は、金融業界の一般的な慣行を反映した、負の値を括弧内に入れて表示することです (この場合、負符号 (-) は使用しない)。このことは、数値フォーマットの一般的な慣行ですが、グリッド内の負の値に注意を向けるのに役立つこともあります。次の例では、負の値は括弧で囲まれます。

```
<blox:grid ...
  defaultCellFormat="#,###.00;(#,###.00)"
</blox:grid>
```

必要であれば、これら 2 つの強調方式を組み合わせることができます。次の設定では、負の値は括弧内に赤字で表示されます。

```
<blox:grid ...
  defaultCellFormat="#,###.00;[red](#,###.00)"
</blox:grid>
```

defaultCellFormat および cellFormat プロパティーの使用方法の詳細は、「*開発者用リファレンス*」の『GridBlox』セクションを参照してください。

セル・アラートを使用したデータの強調

情報の分析は、大規模なグリッド内のすべての数値を注意深く調べて、さらに注意を向けるに値する偏差や傾向にスポットを当てることをせずに行うことは非常に困難です。分析者がたった 1 つの値における重要な偏差を見逃した場合でも、企業にとってはコスト的に大きな影響を受けることがあります。時間は限られているため、作業のスピードを上げるために実行できるどのようなことであっても、生産性を後押しする重要な変化を見逃すことのないように助けるものとなります。フラッシュ・レポートやエグゼクティブ・スコアカードを含む多くの分析アプリケーションでは、いくつかのデータに注意を向ける必要があることを通知するために、例外報告またはトラフィック・ライトを使用します。DB2 Alphablox には、この重大な情報を強調するのに使用できる Blox プロパティーおよびメソッドが用意されています。

DB2 Alphablox アプリケーションでは、ユーザーが関心を向ける可能性のある重要な情報を強調するのに、GridBlox `cellAlert` プロパティーを使用して、以下のアラートを出すことができます。

- 予測された値からの大きな逸脱
- 利益問題につながる可能性を指摘する負の値
- 受け入れ可能範囲の境界を超えた比率

情報を強調するために `cellAlert` プロパティーを使用する 2 つの方法、すなわち、セル・フォーマットおよびセル・アラート・リンクについて、以下で説明します。

セル・フォーマット

作業しているアプリケーションによっては、値に対して特定のビジネス・ロジックに基づいて「トラフィック・ライト」を実行することが要件となる場合があります。トラフィック・ライトは、一般的に使用される用語で、それぞれの値の範囲をユーザーに強調するときの例外報告の形態を記述するために使用されます。これはちょうど、車で街をドライブするときに遭遇する実際の信号機の赤信号、黄信号、および青信号に簡単にたとえられたものです。トラフィック・ライトは、生活のある分野 (車でドライブしたり、道路を横断する) の知識を、別の分野 (ビジネス・インテリジェンス) に当てはめた、理解しやすい技法です。分析アプリケーションでのトラフィック・ライトの典型的な適用例は、特定の基準に従ってデータ値の背景を強調することです。この基準は、標準的な交通信号の 3 色で表されます。次の表は、標準の交通信号の 3 色をリストし、標準的な使用時の意味を説明します。

背景色 説明

赤 値は、ユーザーにとって要注意の危険なレベル。

黄 値は、受け入れられる望ましい範囲内にはなく、注意を払う必要がある。

緑 値は、「安全」または「良好」な範囲であるため、必ずしも注意を払う必要はない。

次の作業では、簡単なトラフィック・ライト報告システムを作成して、データ内での重要な変更についてユーザーにアラートを出す方法を説明します。

簡単なトラフィック・ライト報告システム

報告のためのトラフィック・ライト通知システムを作成する場合、さまざまなバリエーションが可能です。このトピックの前の部分では、GridBlox defaultCellFormat または cellFormat プロパティーを使用して、グリッド内の負の値を強調する方法を学習しました。これは多くの状況で良い解決策ですが、負の値が実際には「適切な」値である事例もあるため、フォーマット・マスクを使用してその値を赤字で強調することが適切でない場合があります。 GridBlox の cellAlert プロパティーを使用すると、以下の手段によるアラートをカスタマイズできます。

- セルの背景色
- フォントおよび色を含むデータ値スタイル
- 基準が満たされたときに表示されるセル・リンク

cellAlert プロパティーで使用できるすべての属性についての完全な詳細は、GridBlox タグ・リファレンスを参照してください。グリッドの特定メンバーのために簡単なトラフィック・ライト・システムを作成するには、以下のステップに従ってください。

1. 値の範囲を強調する対象メンバーを選択します。

   この例では、トラフィック・ライトを実行する対象メンバーは、Variance % です。グリッドには 4 つの列 (Actual、Forecast、Variance、および Variance %) が示されますが、Variance % メンバーについてのみ、関係するレベルを示す背景色が表示されます。

2. <blox:cellAlert> タグを追加して、背景を赤で表示する値を定義します。

```
<blox:cellAlert
   scope="{Scenario:Variance %}"
   condition="LT"
   value="0"
   background="red">
</blox:cellAlert>
```

3. <blox:cellAlert> タグを追加して、背景を黄色で表示する値を定義します。

```
<blox:cellAlert
   scope="{Scenario:Variance %}"
   condition="between"
   value="0"
   value2="10"
   background="yellow">
</blox:cellAlert>
```

4. 背景を青で表示する値のために <blox:cellAlert> タグを追加します。

```
<blox:cellAlert
   scope="{Scenario:Variance %}"
   condition="GT"
   value="10"
   background="green">
</blox:cellAlert>
```

5. レポートを実行します。

例: この報告システムの作業例を見るには、 Blox Sampler の『データの強調表示 (Highlighting Data)』セクションの『トラフィック・ライト (Traffic Lighting)』の例を参照してください。

トラフィック・ライトおよび例外報告を使用する場合、明記しておく必要のある重要な点がいくつかあります。

- 範囲はすべての値を網羅するようにしてください。たとえば、青で示されるゼロより大きい値があり、黄色で示されるゼロより小さい値がある場合、値がゼロであると強調は行われません。
- 極端な場合、値の範囲に固定された制限がない場合、範囲の一方の端で GT または GTEQ を使用し、値の範囲のもう一方の端で LT または LTEQ を使用することを考慮してください。これにより、値が事前定義された範囲を超過した場合でも、後で余分な保守を実行しなくても済みます。
- 行バンディング、生成スタイル、およびセル・スタイルで使用される色スキームは、セル・アラートに干渉することがあり、その結果、重要なアラートが見逃されることがあります。たとえば、デフォルトの行バンディングが、代替行の背景を黄色で表示する場合で、アラートも背景色として黄色を使用している場合、ユーザーはアラートを非常に見逃しやすくなります。
- アラートで使用される色スキームは、印刷されるページの結果に影響することがあります。背景が赤であるセルは、プリンターによっては、印刷時に黒になってしまったり、非常に濃くなってしまったりする可能性があり、そのようなセルの値はあいまいになることがあります。
- オンラインでは、値の色と背景色のコントラストは低いため、アラートの背景色を使用したセルを読みやすくすることは難しい場合があります。背景が赤で値が黒の場合は、特に読み難くなることがあります。また、色および色コントラストのバリエーションは、モニター装置によって異なることがあることにも留意してください。

セル・アラート・リンク

セル・アラートの基準に一致した結果として、セルの外観を変更することに加え、特定の基準が満たされたときに表示されるリンクを定義することもできます。たとえば、特定の値が条件に一致する場合、その値についての説明やこのケースでセル・アラートが適用された理由を伝えるテキスト・ウィンドウをポップアップすることができます。

セル・アラートのアラート・メッセージの作成

この作業での目標は、ユーザーがセルのリンク・アイコンをクリックするときに、テキスト・ウィンドウをポップアップすることです。以下のステップに従ってください。

1. アラート時にポップアップされるウィンドウを作成します。

   たとえば、そのウィンドウは、短いメッセージが記され、ナビゲーション・エレメントがなく、そして「Close Window」ボタンがある簡単な HTML ウィンドウにすることができます。次のようになります。

```
<html>
<head>
<title>Alert Message</title>
</head>
<body>
Your alert message here
</body>
</html>
```

この例では、ファイルは alertMessage.html として保管されるものとします。

2. GridBlox cellAlert タグを GridBlox タグ内にネストします。

```
<blox:grid id="myGridBlox"
   ...>
   <blox:cellAlert
      condition="LT"
      value="0"
      link="http://<serverName>/<appName>/notes/alertMessage.html"/>
</blox:grid>
```

注: cellAlert タグ内で使用されるリンクは、ページへの完全なパスを示す絶対 URL か、アプリケーション・コンテキストからの相対 URL (先頭にスラッシュを付ける) のいずれかにします。リンクが絶対 URL である場合、URL 内にはサーバー名が含まれていなければなりません。以下の 2 つの例は、どちらも有効です。

```
link="http://<serverName>/<appName>/notes/alertMessage.html"
link="/notes/alertMessage.html"
```

上記の URL では、serverName は、DB2 Alphablox アプリケーションが実行されているサーバーにする必要があります。

3. アプリケーション・ページをテストします。

注: セル・リンクと共に使用する場合で、両方にイメージが含まれている場合には、セル・アラートがセル・リンクに優先することに注意する必要があります。セル・アラートがデータ・セルに適用され、イメージまたはイメージ配置が定義された定義済みセル・リンクも存在する場合、セル・リンクは表示されません。しかし、リンクまたはイメージを含まないセル・アラートである場合、両方が適用されます。

チャート系列色を使用したデータの強調

DB2 Alphablox アプリケーションでは、GridBlox cellAlert プロパティーを使用してグリッド内のデータを強調表示するほかに、チャートに異なるデータ系列色を指定することによっても、注目させるデータを強調することができます。たとえば、データ値が指定したしきい値に達しない場合のバーの色を赤に設定することも可能です。

チャートにトラフィック・ライト効果を作り出すには、2 つの一般的なタスクが必要です。

- データ系列色を設定する。
- それぞれの色が何を示すかを説明するために、各凡例項目に色と表示ラベルを設定する。

データ系列色の設定

チャート内の各データ系列には多数のデータ・ポイントが含まれており、各データ系列は単一の凡例項目として表されます (円グラフを除く)。すべてのデータ系列は com.alphablox.blox.uimodel.core.chart パッケージの AbstractDataSeries クラスを拡張します。AbstractDataSeries にはデータ系列名を設定および取得するための基本機能が備わっており、選択したデータ・ポイントの状態を保管し、エンド・ユーザーの

データ・ポイント選択イベントのための基本的なイベント処理の一部を行います。SingleValueDataSeries オブジェクトは AbstractDataSeries を拡張し、BarChart、LineChart、または PieChart などの各データ・ポイントの単一値を含むすべてのデータ系列のスーパー・クラスとなります。

データ系列の色は、ChartBlox `setSeriesColorList()` メソッドを使用して設定することができます。しかし、このメソッドを使用する場合、系列の順序に基づいて色を設定することしかできず、そのようにしてアプリケーションのテーマやカラー・スキームに合う色をカスタマイズすることになります。データ値に基づいて色を設定するには、AbstractDataSeries または SingleValueDataSeries の `setColor()` メソッドを使用する必要があります。

凡例項目の色と表示ラベルの設定

com.alphablox.blox.uimodel.core.chart.common パッケージの Legend クラスを使用することにより、チャートの凡例の外観を操作することができます。デフォルトでは、凡例項目のテキストにはプロットされているデータ系列の名前が入ります。たとえば、1 つの棒グラフに 4 つのバーがプロットされている場合、それぞれのバーは BarDataSeries であり、それぞれが系列名を持ちます。この系列名が、凡例項目として表示されるものです。Legend オブジェクトを通して各 LegendItem へのアクセスと制御が可能になりますが、それら LegendItem によって凡例内に表示する項目、塗りつぶしの色やパターン、および表示ラベルの内容を個別に制御することができます。

トラフィック・ライト・チャートの場合、通常は凡例項目ごとに色と表示テキストを設定します。LegendItem の `setFillColor()` および `setText()` メソッドを使用することによって、たとえば、緑のバーが販売目標を達成したまたは超えたデータ系列を、黄色のバーが四半期の目標達成が危ぶまれているデータ系列を表すように指定できます。

トラフィック・ライト・チャートの完全な例については、『トラフィック・ライト・チャートの例』を参照してください。

トラフィック・ライト・チャートの例

この例は、次のように設定された簡単なトラフィック・ライト棒グラフ (`chartType = "Bar"`) を示したものです。

- データ値が 3000000 より小さい場合は、バーの色を赤に設定する。
- データ値が 6000000 より大きい場合は、バーの色を緑に設定する。
- データ値が上の条件のいずれをも満たさない場合は、バーの色を黄色に設定する。
- チャート内の凡例項目に色と表示テキストを設定して、それぞれの色の意味を説明する。

この例では、いくつかの手法が示されています。

- 使用不可に設定された GridBlox、DataLayoutBlox、および PageBlox がネストされている PresentBlox を使用します。ツールバーとメニュー・バーの可視性は `false` に設定されているので、300 x 300 ピクセルの表示域全体をチャートが占めます。

- 凡例上での DoubleClickEvent 用にカスタム・コントローラー MyController が作成されています。ユーザーが凡例の項目をダブルクリックすると、その凡例項目に固有の URL が新規ブラウザー・ウィンドウにロードされます。
- 棒グラフ内のデータ系列は SingleValueDataSeries オブジェクトです。棒グラフ内のすべてのデータ系列を取得するには、次のようにします。

```
BarChart bChart = (BarChart) chart;
SingleValueDataSeries series[] = bChart.getAllDataSeries();
```

  次に、それらのデータ系列に操作を繰り返し、データ値に基づいてバーの色を設定します。
- 各データ系列の LegendItem は、色が表す内容の記述に変更されます。

```
LegendItem items[] = new LegendItem[3];
items[0] = new LegendItem();
items[0].setFillColor(Color.green);
items[0].setSymbolType(LegendItem.SHAPE_BAR);
items[0].setText("Meeting the quarterly goal");
items[0].setEventIndex(0);
```

SingleValueDataSeries と LegendItem オブジェクトの詳細については、202 ページの『チャート系列色を使用したデータの強調』を参照してください。

完全なコード例は次のようになります。

```
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ page import="com.alphablox.blox.uimodel.BloxModel,
                 com.alphablox.blox.uimodel.ChartBrixController,
                 com.alphablox.blox.uimodel.ChartBrixModel,
                 com.alphablox.blox.uimodel.ModelConstants,
                 com.alphablox.blox.uimodel.core.Component,
                 com.alphablox.blox.uimodel.core.Controller,
                 com.alphablox.blox.uimodel.core.chart.BarChart,
                 com.alphablox.blox.uimodel.core.chart.Chart,
                 com.alphablox.blox.uimodel.core.chart.SingleValueDataSeries,
                 com.alphablox.blox.uimodel.core.chart.common.ChartComponent,
                 com.alphablox.blox.uimodel.core.chart.common.Legend,
                 com.alphablox.blox.uimodel.core.chart.common.LegendItem,
                 com.alphablox.blox.uimodel.core.event.ComponentRebuiltNotify,
                 com.alphablox.blox.uimodel.core.event.DoubleClickEvent,
                 com.alphablox.blox.uimodel.core.event.IEventHandler"%>
<%@ page import="java.awt.*"%>
<%@ page import="java.util.ArrayList"%>
<%@ taglib uri="bloxtld" prefix="blox" %>
<html>
<head>
   <blox:header/>
</head>
<%
class MyController extends Controller{
  public boolean handleDoubleClickEvent(DoubleClickEvent event) throws Exception {
    Chart theChart = (Chart) event.getComponent();
    ChartComponent chartComponent = theChart.getSelectedChartComponent();
    if (chartComponent instanceof Legend){
      Legend legend = (Legend) chartComponent;
      int selected =legend.getSelectedIndex();
      String color = ((String[])legend.getUserObject())[0];
      String url = "http://webexhibits.org/pigments/indiv/color/"+ color + ".html";
      theChart.getDispatcher().sendClientCommand("window.open('"+url+"',
'"+color+"')");
      return true;
      }
      return false;
  }
```

```
}
%>
<body>
<blox:present id="TrafficLight"
 gridAvailable="false"
 dataLayoutAvailable="false"
 pageAvailable="false"
 menubarVisible="false"
 toolbarVisible="false"
 width="300"
 height="300">
  <blox:dataLayout visible="false" />
  <blox:chart
   chartType="Bar"
   legendPosition="bottom"/>
  <blox:data
   dataSourceName="QCC-Essbase"
   query="<COLUMN (\"All Time Periods\") (\"All Time Periods\")
         <ICHILD \"2001\" <ROW (\"Measures\") Sales !"
   parentFirst="true"
   useAliases="true" />
<%
BloxModel model = TrafficLight.getBloxModel();
model.addEventHandler( new IEventHandler() {
  public boolean handleComponentRebuiltNotify(ComponentRebuiltNotify event)
throws Exception {
    Component component = event.getComponent();
    if (component instanceof ChartBrixModel) {
      Chart chart=(Chart)((ChartBrixModel) component).searchForComponent(
  ModelConstants.CHART);
      if (chart instanceof BarChart) {
        //Gets the data series in the bar chart
        BarChart bChart = (BarChart) chart;
        SingleValueDataSeries series[] = bChart.getAllDataSeries();
        for (int i = 0; i < series.length; i++) {
          for (int j = 0; j < series[i].size(); j++) {
            Number val = series[i].get(j);
            if (val != null && val.doubleValue() < 3000000){
              series[i].setColor(j, Color.red);
            }
            else if (val != null && val.doubleValue() > 6000000){
              series[i].setColor(j, Color.green);
            }
            else {
              series[i].setColor(j, Color.yellow);
            }
          }
        }
      }
      LegendItem items[] = new LegendItem[3];
      items[0] = new LegendItem();
      items[0].setFillColor(Color.green);
      items[0].setSymbolType(LegendItem.SHAPE_BAR);
      items[0].setText("Meeting the goal");
      items[0].setEventIndex(0);
      items[1] = new LegendItem();
      items[1].setFillColor(Color.yellow);
      items[1].setSymbolType(LegendItem.SHAPE_BAR);
      items[1].setText("Potential problem areas");
      items[1].setEventIndex(1);
      items[2] = new LegendItem();
      items[2].setFillColor(Color.red);
      items[2].setSymbolType(LegendItem.SHAPE_BAR);
      items[2].setText("Not meeting the goal");
      items[2].setEventIndex(2);
      chart.setController(new MyController());
      Legend legend = chart.getLegend();
```

```
      legend.setLegendItems(items);
      String desc[] = new String[]{"greens", "yellows", "reds"};
      legend.setUserObject(desc);
      chart.changed();
      component.changed();
    }
    return false;
  }
}
);
((ChartBrixController)(model.getChart().getController())).synchronize();
%>
</blox:present>
</body>
</html>
```

この例は、トラフィック・ライト・チャートを作成するために不可欠なメソッドと作成の手法を示しています。指定した色としきい値は、ユーザーがドリルダウンなどのデータ・ナビゲーション操作を実行するときに今後もチャート・データに適用されるため、色が付いていると紛らわしい場合があります。データ・ナビゲーション操作を制御したり、データ系列の色をリセットしたり、しきい値を変更したりする追加のカスタム・コードが必要かもしれません。あるいは、 Blox ユーザー・インターフェースをクリック不能に設定するという方法もあります (`<bloxui:component>` タグの `clickable` 属性を使用する)。

情報リンク

分析アプリケーションでは、グリッド内のデータについての詳細に対するリンクを含めなければならないことがあります。リンクには、以下のようなさまざまな使用方法があります。

- ヘッダーについての詳細へのリンク
- 特定のセルについての情報へのリンク
- ビジネス・ロジックに基づくセルのアラート情報へのリンク

DB2 Alphablox には、ヘッダー・リンク、セル・リンク、およびセル・アラート・リンクの 3 タイプの情報リンクが備えられています。以下に、各リンク・タイプの利点と欠点をリストします。

リンク・タイプ
使用方法の要約

ヘッダー・リンク

- リンクは、行または列ヘッダーに示されるときに、ディメンション・メンバーの右側に表示されます
- DB2 Alphablox 管理ページのアプリケーション定義ページで定義されます
- メンバーに関連付けられたリンクがあるときに常に表示されます
- すべてのヘッダー・リンクに 1 つのアイコン・イメージだけが使用可能です

セル・リンク

- リンクは有効範囲を使用して定義できます
- GridBlox の `cellLink` プロパティーを使用して定義されます
- `cellLink` に基づいて別のイメージを定義できます

- 通知ウィンドウがオープンされるか、 JavaScript 関数が実行される可能性があります

セル・アラート・リンク

- cellAlert プロパティーの一部として定義されます
- セル・リンクと共に使用できますが、イメージが両方に存在する場合、セル・アラート・リンクが優先します
- 条件ロジックまたは有効範囲に基づいて表示できます
- 通知ウィンドウがオープンされるか、 JavaScript 関数が実行される可能性があります

下記の、各情報リンク・タイプの使用方法についての詳細を参照してください。

ヘッダー・リンクの使用

ヘッダー・リンクは、グリッド内の行または列メンバーの横に表示される (内部に白の “i” が示された青の円で表される) 情報アイコンをクリックするときに、Web ページを表示したり、JavaScript 関数を起動したりするよう定義できる、アプリケーション固有の情報リンクです。ヘッダー・リンクは、アプリケーション定義ページの「ヘッダー・リンク」テキスト・ボックスで定義されたメンバーのヘッダーにのみ示されます。

特定アプリケーションでヘッダー・リンクを追加するには、DB2 Alphablox ホーム・ページでアプリケーション定義ページをオープンします。ページの下の方に、「ヘッダー・リンク」テキスト・ボックスがあり、次の構文を使用してヘッダー・リンクを定義できます。

```
memberName = URL
```

ここで、memberName は、データ・ソースで定義された固有のメンバー名で、*URL* は、ページへのパス全体を示す絶対 URL か、アプリケーション・コンテキストからの相対 URL (先頭にスラッシュを付ける) です。リンクが絶対 URL である場合、URL 内にはサーバー名が含まれていなければなりません。たとえば、Diet Cola の商品ページへの情報リンクを作成するには、次のヘッダー・リンク定義を使用できます。

```
Diet Cola = http://productServer/products/dietcola.html
```

または

```
Diet Cola = /<pathTo>/dietcola.html
```

注: JavaScript プロトコル・メソッドはサポートされていません。

セル・リンクの使用

ヘッダー・リンクを使用して行および列ヘッダーにリンクを配置できるように、GridBlox の cellLink プロパティーを使用してデータ・セルに対するハイパーリンクを定義できます。ただし、ヘッダー・リンクと異なる点は、セル・リンクを使用すると、JavaScript プロトコル・メソッドを使用した JavaScript メソッドを呼び出

すこともできるという点です。他の索引プロパティーのように、セル・リンクは、実行時に動的に生成されるか開発者によって定義される、それぞれの索引値に従って評価されます。

セル・リンクの数値は、評価される順序を指示するものです。評価は、索引値が 1 である cellLink から開始されます。最初に定義されたセル・リンクで、セルの条件および有効範囲に一致するものは、そのセルに適用される唯一のリンクです。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。また、セル・アラート・リンクは、cellLink を使用して作成されたリンクよりも優先します。つまり、特定のデータ・セルに定義されたセル・アラート・リンクとセル・リンクがある場合、セル・アラート・リンクはセルに表示されますが、セル・リンクは表示されません。

同じデータ・セルでセル・アラートとセル・リンクの両方を持つことは可能ですが、両方にイメージ・エレメント (image、image_align、または link) が定義されている場合、セル・アラート・リンクはセル・リンクよりも優先します。そのため、1 つのセルでは 1 つのアイコンとリンクだけが使用可能であり、(リンク付きの) セル・アラートは、セル・リンクよりも重要であると見なされます。

以下は、GridBlox タグ内にネストされるセル・リンク・プロパティー・タグの一例です。

```
<blox:grid id="cellLinkGridBlox">
   <blox:cellLink
   scope="{Scenario:Variance %}"
   description="Opens information window"
   link="/<applicationDirectory>/links/Manhattan.html"/>
   <blox:data dataSourceName="QCC-Essbase"
      query='<ROW("All Products") "All Products"
         <COLUMN(Scenario) <CHILD Scenario
         <PAGE("All Locations") Manhattan Sales !'/>
</blox:grid>
```

上記で定義されたセル・リンクは、Variance % の下の Manhattan データ・セルにのみ表示されます。オープンするページは、アプリケーション・フォルダーの links サブディレクトリーにあります。

指定される URL は、ページへのパス全体を示す絶対 URL か、アプリケーション・コンテキストからの相対 URL (先頭にスラッシュを付ける) のいずれかにします。リンクが絶対 URL である場合、URL 内にはサーバー名が含まれていなければなりません。上記の例では、<serverName> はサーバーを表し、<applicationDirectory> はファイルが存在するアプリケーション・ディレクトリーの名前を表します。

cellLink プロパティーとそれに関連したメソッド getCellLink() および setCellLink() の構文と使用法の詳細は、「*開発者用リファレンス*」を参照してください。

セル・アラート・リンクの使用

201 ページの『セル・アラート・リンク』で説明されているように、cellAlert ではリンクも表示できます。 cellAlert プロパティーで作成されたリンクと cellLink

プロパティーで作成されたリンクの両方を組み合わせると、ユーザーが知りたい情報を強調する方法について別の手段が実現します。

別のオプションは、UI モデルの拡張性を使用することにより、ユーザー・インターフェースを拡張して、1 つのセルに複数のイメージを表示することです。

`cellAlert` リンクと、`cellLink` で作成されたリンクの両方の詳細は、GridBlox タグ・リファレンスを参照してください。

グリッド・データ・セル内のコメント

1 つの組織内で情報を共有することには、データに対するコメントが関係する場合があります。しかし、これらのコメントは E メール・メッセージかどこかで消失することもあります。 DB2 Alphablox には、これらの重要なコメントをコメント・データベースに追加して、ユーザー分析のコンテキストで表示する機能が組み込まれました。この機能を使用し、そのようなセルに対するコメントを検索したり、コメントをアプリケーション・ページの個別のリストに表示したりすることにより、ユーザーは特定のデータ・セルに関連付けられたコメントを表示できます。

CommentsBlox コンポーネントを使用することでサポートされるコメントには、セル・レベル・コメントと名前付きコメントの 2 つのタイプがあります。セル・レベル・コメントは、特定のデータ・セルに添付されてグリッド内に表示されるコメントです。これらは、一群のディメンションで定義できます。名前付きコメントは、コメントの有効範囲を定義するのに使用できるストリング・アドレスを持つコメントです。

グリッド内のデータ・セルにコメントが追加され、グリッドでコメントが使用可能にされた場合、コメント・インディケーターが表示されます。デフォルトでは、コメント・インディケーターは、コメントが関連付けられたデータ・セルの右上角に表示される、小さな赤い三角です。ユーザーがコメント・インディケーターが示されたセルを右クリックすると、コンテキスト (右クリック) メニューでコメント・オプションが表示されます。 2 つのサブメニュー・オプションが使用可能です。「コメントの追加」では、選択したセルに新しいコメントを追加でき、「コメントの表示」では、選択したセルで使用可能なコメントを表示できます。

注: 「コメント管理」ダイアログを使用するには、リレーショナル表を作成してドロップするための権限が必要です。カスタム・コメント・アプリケーションの開発時に CommentsBlox API を使用する場合、表を選択、挿入、更新、削除、作成、およびドロップする権限が必要な場合があります。

鍵となる用語

説明

コメント・コレクション

1 つのマルチディメンション・キューブでのコメント群のリポジトリー。リレーショナル・データベースに格納されています。

CommentsBlox

ページ上のコメント・コレクションを表します。DataBlox 内にネストされる一連のタグを含みます。

CommentsSet

1 つのデータ・セル用に存在するか同じアドレスまたは名前で存在するコメント群。 1 つの有効範囲またはアドレス (たとえば、Product:100、Year:Qtr1、Scenario:Actual のコメント) を持つすべてのコメントを含みます。

コメントのエレメント

それぞれのコメントは、以下の部分で構成されます。

コメント・エレメント

説明

作成者 必須。コメントの作成者です。デフォルトでは、このフィールドは、コメントの作成時に、DB2 Alphablox によって現在ログインしているユーザーに設定されます。

タイム・スタンプ

必須。コメントが作成された時刻です。コメントが最初にコメント・セットに保管されるときに、サーバーによって自動的に設定されます。

コメント・テキスト

必須。コメントのテキストです。ハイパーリンクを含めたり、(HTML を使用した) フォーマット済みテキストを含めることもできます。文字サイズに制限はありません。

カスタム・フィールド

最大の柔軟性を実現するため、コメント・セットのコメントに追加のカスタム・フィールドを定義できます。たとえば、件名、重要度、セル値などです。

アドレス

各コメントにはアドレスがあります。セル・レベル・コメントの場合、アドレスは、コメントが添付されるセルを固有に識別する <dimension, member> のリストになります。名前付きコメントの場合、アドレスは、開発者によって意味が定義されるストリングだけです。たとえば、一群の Blox レベル・コメントでは、コメントが関連付けられている Blox の名前で構成されるアドレスになる場合があります。アプリケーション・レベル・コメントでは、アドレスは、アプリケーションの名前になる場合があります。アセンブラーは、このストリングを使用して、コメントのネーム・スペースを定義し、個別設定機能を支援できます。

セル・レベル・コメントには、ディメンションのサブセットを特定のキューブに統合するアドレッシング・スキームを持つものがあります。コレクションの定義に含まれないディメンションは無視されます。一例として、キューブに 3 つのディメンション (Time、Measures、および Product) がある場合で、現在のコレクション内のコメントが Time および Measures の値を使用して指定されることを定義する場合、定義されるコメントはすべて Product の値を求めます。一般に、レポートの一部として扱われる可能性のあるコメント定義に、すべてのディメンションを含める必要があります。

このセル・レベル・コメントでのアドレッシング・スキームは、2 つの目的で役立ちます。第 1 に、特に大規模な一括表示で、コメントの管理が容易になります。 2 番目に、キューブおよびデータ・ソース全体でのコメント

の共有が容易になります。 Product、Time、および Measures で定義されたコメント・セットは、多数のデータ・ソースに適用できますが、一括表示のそれぞれのディメンションで定義されたコメント・セットは、キューブおよびデータ・ソース固有になってしまうリスクがあります。

コメント・コレクションの定義

コメント・コレクションを定義するには、リレーショナル・データ・ソースを作成する必要があります。このデータ・ソースでは、複数のコレクションを保持できます。コメント・コレクションを作成するには、以下の 2 つのステップが必要です。

1. データ・ソースを作成し、DB2 Alphablox データ・ソース定義ページで定義します。
2. コメント・コレクション・リポジトリーを作成します。

   コメント・コレクションを定義するため、DB2 Alphablox 管理ページをオープンし、「管理」タブをクリックします。左側のメニューの「実行時管理 (Runtime Management)」の下にある「コメント管理」をクリックして、「コメント管理」ウィンドウをオープンします。

   コレクションでは、コレクション名、キューブから選択されたディメンション、および使用するフィールドを作成することが必要です。作成者、タイム・スタンプ、およびコメント・テキスト・フィールドは自動的に定義されますが、カスタム・フィールドを作成することもできます。

   コメント・コレクションを構成するときのヘルプは、「コメント管理」ウィンドウでも使用可能です。

セル・コメントの使用可能化

グリッド内のデータ・セルのコメントを使用可能にするには、以下のステップに従う必要があります。

1. 独立したまたはネストされた GridBlox で、`commentsEnabled` 属性を追加し、`true` に設定します。
2. 上記のグリッドの独立したまたはネストされた DataBlox で、CommentsBlox タグを追加し、コメント・コレクションに `collectionName` および `dataSourceName` 属性を指定します。データ・ソースおよびコレクション名は、DB2 Alphablox 管理ページの「管理」タブの「コメント管理」で定義されます。

以下は、コメントをサポートするように設定された PresentBlox の一例です。

```
<blox:present id="myPresentBlox">
   <blox:grid commentsEnabled="true" />
   <blox:data dataSourceName="QCC-Essbase" query="<%=query%>">
      <blox:comments
        collectionName="sales_comments"
        dataSourceName="CommentsCollection" >
      </blox:comments>
   </blox:data>
</blox:present>
```

これを完了したら、ユーザーはデータ・セルを右クリックして、コメントを追加または表示できます。基本的なコメントのサポートでは、開発者側でさらに必要なステップはありません。

カスタム・コメント・サポートの追加

ユーザーが独自のコメントを追加し、他のユーザーのコメントを表示する機能は、強力なコラボレーションおよび情報共用メカニズムです。すぐに使用可能なコメントの使用可能化機能は、構成しやすく使いやすいものです。しかし、CommentsBlox 機能は強力で柔軟性があるので、開発者はコメント作成の使用方法をカスタマイズできます。以下は、CommentsBlox を使用してアプリケーションをさらに拡張できる可能性のあるカスタマイズのいくつかの例です。

注: CommentsBlox の構文と使用方法についての詳細は、CommentsBlox タグ・リファレンスを参照してください。

しばしばユーザーは、特定のトピックまたは特定の分析ビューについてのコメントを、特定のデータ・セルに関連付けずに追加したい場合があります。このことは、CommentsBlox タグおよびサーバー側の Java API を使用して簡単に実現できます。Blox Sampler アプリケーションの『データのコメント付け (Commenting on Data)』セクションの下で、『ページ上の一般的コメント (General Comments on a Page)』の例には、グリッドの下に示される一般コメントを、個別のコメント・ウィンドウに追加して表示する例が示されています。

ユーザーは、特定のグリッドに関連付けられたすべてのコメントを印刷したい場合もあります。 Blox Sampler の『データのコメント付け (Commenting on Data)』セクションに含まれる『グリッド内のコメントの印刷 (Printing Comments in a Grid)』の例には、新しいブラウザー・ウィンドウをオープンし、グリッド内のすべてのコメントを表示するボタンが含まれています。

第 17 章 データとの対話

このトピックは、 Blox を使用した場合のユーザーの動作と対話性を取り上げています。説明される主な問題には、Blox の動作を制御して変更する方法、動作を制限するのに使用される基本的な技法、ユーザー・アクションをキャプチャーして対話性およびアクションを制御する方法が含まれます。

対話性についての考慮事項

対話式のビジュアルなプレゼンテーション Blox を使用すると、ユーザーに提示されたビューを操作し、データ内をドリルダウンまたはドリルアップしたり、チャート・タイプを変更したり、他の多くのオプションを変更したりできます。使用しているアプリケーション、対象者、およびそのスキル・レベルに応じ、アプリケーションの制御を除去したり制限することを決定できます。以下の項では、Blox を使用した対話を制限するときに考慮する必要のある問題を説明します。

対話性の制限または対話性なしの許可

ユーザーが重要なデータを簡単なビューで必要としており、より深い分析のためにデータを操作することまでは望んでいない場合、GridBlox、ChartBlox、または PresentBlox で十分です。対話性のない Blox ビューで、データの一部を表示することも考慮できます。

Blox の使用時に対話性をなくすには、たとえば、Blox UI コンポーネント・タグ (<bloxui:component>) を追加し、 clickable タグ属性を false に設定できます。たとえば、次のコードは、ユーザー対話を使用不可にした PresentBlox を生成する、ネストされた <bloxui:component> タグの使用を示したものです。

```
<blox:present id="myPresentBlox"
   width="80%"
   height="70%"
   menubarVisible="false">
   <blox:toolbar
      visible="false" />
   <blox:data
      bloxRef="dataBlox" />
   <bloxui:component name="myPresentBlox"
      clickable="false" />
</blox:present>
```

対話性を使用不可にすることは、「忙しすぎて」データを掘り下げられないユーザーや、データの操作方法を学習することには関心のないユーザーにとって最善の解決策である場合があります。たとえば、会社の上級管理職は、その会社の全体の動向を示すスナップショット・ビューを見ることだけに関心があり、詳細な分析は、ビジネス・アナリストや金融アナリストに任せることがあります。

次の作業は、Blox 全体を使用不可にする、または他の Blox 内にネストされた選択 Blox を使用不可にする方法を示しています。

列でのピボット作成およびドリルの使用不可化

Blox Sampler の『Blox UI タグ』セクションで、バタフライ・レポートの例には、列でピボット作成したりドリルしないようにするイベント・フィルターが含まれています。このどちらのユーザー操作でも、適切に示されていない非対称の報告が表示されます。ユーザーが混乱して抜け出すのが難しい状態になるのを避けるため、UI モデルを使用して、ユーザーがピボットを作成したりドリルしたりすることを妨げるイベント・フィルターを追加し、ビューを使用不可の状態にしておくことができます。

次のコード部分は、列でのピボット作成およびドリルを防ぐために、グリッドで使用されるイベント・ハンドラーを示しています。

```
<%
  GridBloxModel model =
    butterflyReportGridBlox.getGridBloxModel();
  model.populateDataNavigationButton();
  model.getGrid().getController().addEventHandler(
    new IEventHandler() {
  public boolean handleGridDataActionEvent(GridDataActionEvent
      event ) throws Exception {
   GridBrixCellModel cells[] = event.getGridCells();
// If any of the cells is a header cell, then ignore the data action

  for ( int i=0; i < cells.length; i++ )
  if ( cells[i].isColumnHeader() )
  return true;
  return false;
}
} );
%>
```

コード例の全体は、Blox Sampler を参照してください。

Blox UI モデルの DHTML 拡張機能でイベント・ハンドラーを使用して、この例のようにアプリケーションをカスタマイズすることについては、86 ページの『Blox UI Model のイベント』を参照してください。

Blox プロパティーを使用した対話性の変更

ユーザー対話は、Blox プロパティーおよびメソッドを使用しても制御できます。データ・プレゼンテーション Blox で、「ツールバー」、「DataLayout」、および「ページ」パネルが使用可能であれば、ユーザーはデータとより多く対話できます。これが一部のユーザーには役立つものの、他のユーザーは、特にデータの構造に精通していない場合に、すぐにデータを見失ってしまうことがあります。上記の `<bloxui:component>` タグおよび Blox `visible` 属性を使用した技法に加えて、他の Blox プロパティーを使用してビューの対話性を調整し、一部のパネルを使用可能にし、ユーザーが混乱状態になり得る方法の数を制限することもできます。さらに、個別設定の技法を使用して、サーバー側の Java、JavaServer Pages、および JavaScript メソッドを使用して、ユーザー・ログインに基づく対話性をカスタマイズできます。

次の表は、データとのユーザー対話に影響する可能性のある、一般に使用される Blox プロパティーの一部をリストしています。

Blox	プロパティー (または関連付けられたメソッド)	説明およびコメント
GridBlox	cellAlert	セル・アラート・リンクを使用して、通知ウィンドウをオープンしたり、JavaScript 関数を起動できます
	cellLink	セル・リンクを使用して、通知ウィンドウをオープンしたり、JavaScript 関数を起動できます
	expandCollapseMode	Windows Explorer のようなプラス・アイコンおよびマイナス・アイコンの使用方法で、グリッド内でドリルアップおよびドリルダウンできるようにします
	writebackEnabled	有効範囲内のセルに基づいて、許可ユーザーがデータを直接グリッド内に入力できるようにします
DataBlox	drillDownOption	ドリル対象が、次の世代、すべての子孫、最下部の世代、兄弟、または同じ世代なのか決定します
	drillKeepSelectedMember	ドリル対象として選択されたメンバーを保持します
	drillRemoveUnselectedMembers	ドリル時に選択されないメンバーを除去します
	enableKeepRemove	「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かどうかを指定します
	enableShowHide	「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かどうかを指定します

Blox	プロパティー (または関連付けられたメソッド)	説明およびコメント
DataLayoutBlox	`interfaceType`	ユーザーが、ディメンションを選択するときに、ドロップ・リストによるインターフェースを使用するのか、ドラッグ・アンド・ドロップによるツリー・インターフェースを使用するのかを指定します
ReportBlox	「*Relational Reporting 開発者用ガイド*」を参照してください。	

注: プロパティーを変更することで対話での動作が変更されるために、通常の慣れている動作が予測どおりに動作しないと、ユーザーは混乱したり驚いたりすることになります。たとえば、`drillKeepSelectedMember` および `drillRemoveUnselectedMembers` を `true` に設定すると、ユーザーがグリッドやチャートに表示される情報の量を効果的に管理するのに役立つときがあります。重要な考慮事項は、ユーザーがドリルするときに、アプリケーションまたは複数のアプリケーションのすべてのビューが同じように振る舞わない場合で、特定の Blox ビューが他のすべてのビューの動作とは別の動作方をする場合、混乱が生じる可能性があるという点です。このような状態のユーザーに役立つ 1 つの方法は、ページ上に、予測しておく必要のある動作をはっきりと示しておくことです。別の方法は、ラジオ・ボタンかチェック・ボックスを使用して、ユーザーがドリルの 2 つの動作方を切り替えられるようにすることです。

グリッド

グリッドは、独立した GridBlox としても、PresentBlox 内でネストされた GridBlox としても使用可能です。いずれかのモードで、ユーザーはデータをドリル、ピボット作成、ソート、および探索できます。ただし、PresentBlox で使用されるグリッドには、明示的な GridBlox では使用できないいくらかの追加機能があります。次の表は、独立した GridBlox とネストされた GridBlox との間の主な相違点を要約しています。

機能	独立した GridBlox	(PresentBlox 内に) ネストされた GridBlox
DataLayoutBlox	同じ DataBlox を使用した独立した DataLayoutBlox が必要です	PresentBlox `dataLayoutAvailable` を使用して使用可能になります
PageBlox	同じ DataBlox を使用した独立した PageBlox が必要です	`pageAvailable` を `true` に設定することで使用可能になります
ChartBlox	同じ DataBlox を使用した独立した ChartBlox が必要です	グリッドは自動的にチャートと同期化します。チャートを使用可能にできます

上記の機能は、PresentBlox を使用して使用可能になるものなので、大抵は、この機能へのアクセス権をユーザーに付与することになります。

チャート

グリッドの場合のように、チャートで表示されるデータも、ユーザーはドリルアップまたはドリルダウンできます。しかし、グリッドの場合と違うのは、チャートと対話できるという感覚が認識されない点です。これは、ユーザーがチャートと直接に対話していることを理解するのに役立つ、グリッドでの上/下矢印やプラス/マイナス・アイコンのような視覚的合図が存在しないためです。ユーザーがチャートをドリルしていると初めて認識できるのは、誰かがそうしている場合か、偶然にそうしようとしている場合か、またはチャート・エレメントを右クリックしてメニュー・オプションが表示される場合です。ほとんどのユーザーは、チャートのバーおよびデータ点に移動するときに、データ値およびラベルが表示されることに気付きます。必要であれば、ChartBlox プロパティー (たとえば、円グラフの場合には pieFeelerTextDisplay、棒グラフの場合には dataTextDisplay) を使用して、ユーザーがカーソルをチャート・エレメント上に移動させなくても値またはラベルを表示することができます。

ユーザーが、リモート・ロケーションからインターネット経由でアプリケーションにアクセスしているか、オフィス内の近くの席でアプリケーションを使用しているかに関係なく、それらのユーザーはトレーニングを受けていないか、またはアプリケーションとその使用方法に精通していない可能性があります。開発者またはアプリケーション設計者として、分析アプリケーションをできるだけ使いやすくする方法を考慮する必要があります。チャートのページを提示しても指示を提示しないなら、多くのユーザーが、チャートと対話せず、提示されたものを表示させているだけという使い方をしてもおかしくありません。設計するときには、ユーザーが成功し、チャートをより生産的に使用することを学習する可能性を高められるよう考慮してください。 DB2 Alphablox ヘルプまたはカスタム・ヘルプ・ページにアクセスするための「ヘルプ」または「ヒント」ボタンを追加するなら、アプリケーションの操作について学習するのに役立ちます。別の方法としては、ページにいくつかの簡単なヒントを直接載せることもできます。状態によっては、ChartBlox footnote プロパティーを使用して、ユーザー情報をチャートの脚注に直接載せることが役立つこともあります。

表示される世代のユーザー制御の許可

チャートに対話性を追加する簡単な方法は、totalsFilter 属性を ChartBlox に追加し、プロパティーを 2 に設定することです。値を 2 に設定することにより、合計フィルター・スライダー・パネルをチャートの下部に表示できます。そして使用される照会に応じて、ユーザーは表示されるディメンション・レベルを制御できます。次の ChartBlox の例では、totalsFilter 属性が 2 に設定されています。

```
<blox:chart id="totalsFilterChartBlox"
   width="90%"
   height="90%"
   chartType="Bar"
   totalsFilter="2"
   title="Truffle Sales for 2001">
   <blox:data
```

```
        dataSourceName="QCC-Essbase"
        query='<ROW ("All Products") <ICHILD "All Products"
         <COLUMN ("All Time Periods") <DESCENDANTS "2001" Sales !'/>
</blox:chart>
```

ページ上にレンダリングする際、チャートの下のパネルに 2 つの世代セレクターが表示されます。左側のセレクターでは、ユーザーは全商品のディメンションの世代レベルを制御でき、右側のセレクターでは、2001 年のいくつかの期間における世代レベルを選択できます。

例: Blox Sampler の『データとの対話 (Interacting With Data)』セクションの下にある『使用可能なチャート totalsFilter セレクター (Chart totalsFilter Selector Enabled)』の例では、`totalsFilter` スライダー・パネルの使用方法を示しています。

DataLayout インターフェース

PresentBlox で DataLayout パネル (DataLayoutBlox) が使用可能な場合、そこにアクセスして、ディメンションを「行」、「列」、および「その他」(ページ・フィルター) 軸間で移動し、表示したいグリッドおよびチャート内でのデータのレイアウトを作成できます。デフォルトでは、DataLayout パネルは、ドロップ・リスト (または選択リスト) にディメンションを表示します。この場合、ユーザーはディメンション名をクリックして、そのディメンションの移動用のオプションを選択できます。別の方法としては、開発者は、DataLayout パネルでドラッグ・アンド・ドロップによるツリー・インターフェースを使用するように設定できます。この場合、動作はより Windows Explorer に近くなります。DataLayout パネルのインターフェース・タイプを明示的に設定するには、DataLayoutBlox `interfaceType` 属性を、`dropLists` (デフォルト) または `tree` の 2 つのいずれかの値に設定します。次の例は、ツリー・インターフェースを表示するよう設定された DataLayoutBlox を示しています。

```
<blox:present ...>
   ...
   <blox:dataLayout
      interfaceType="tree" />
   ...
</blox:present>
```

注: インターフェース・タイプは、開発者だけが設定できます。ユーザーがこのインターフェース・オプションを選択するための、ユーザー・インターフェース・オプションはありません。デフォルトでは、ドロップ・リスト・インターフェースがユーザーに表示されます。

グリッドおよびチャート間の対話

グリッドおよびチャートは、GridBlox および ChartBlox コンポーネントを使用して個別に表示することもできますし、PresentBlox コンポーネント内に同時にネストすることもできます。スタンドアロンの Blox として示される場合、各 Blox では、(ネストされた DataBlox で定義された) 暗黙的なデータ・ソースまたは (スタンドアロンの DataBlox コンポーネントを使用した) 明示的で独立したデータ・ソースを使用できます。さらに、グリッドおよびチャート・ビューでは、データ・ソースと

して共通の独立した DataBlox を共用できます。PresentBlox 内にネストされた GridBlox および ChartBlox では、常にこのようになります。

GridBlox および ChartBlox コンポーネントが (独立した DataBlox を使用した) 共通のデータ・ソースを共用する場合、GridBlox に対する操作は ChartBlox に反映されます。そのため、ユーザーが GridBlox のメンバーをドリルダウンすると、同じ DataBlox を共用する ChartBlox でも、同じドリル操作が実行されて表示されます。グリッドとチャートは同じデータ・ソースを共用しているために、PresentBlox 内でそれらの間のこのような同期が生じます。

ヘッダー・リンク、セル・アラート、セル・リンク、および他の GridBlox 機能は、チャートでは使用できません。チャートとこれらの機能を両方とも使用する場合は、PresentBlox を使用したいと思われるでしょう。また、グリッドが PresentBlox ビューに表示されない場合、ユーザーは、グリッド・コンポーネント経由でアラートまたはグリッド・ベースのリンクにアクセスしない限り、アラートを表示したりグリッド・ベースのリンクにアクセスできません。

留意する必要のある別の重要な点は、データのフォーマットは、グリッドおよびチャートとは別に設定されるということです。そのため、グリッドとチャートで同じフォーマットの値を使用したい場合、忘れずに以下の Blox プロパティーをすべて設定しなければなりません。

Blox　プロパティー

ChartBlox

`y1FormatMask y2FormatMask`

GridBlox

`defaultCellFormat cellFormat`

グリッドおよびチャートでの「使用可能なデータはありません」メッセージの設定

データ・ソースが使用不可であるか、結果セットを取り出していないと、DB2 Alphablox グリッドおよびチャートは、「使用可能なデータはありません」というデフォルトのメッセージを表示します。

結果セットの取り出しに予想よりも長くかかっている場合、デフォルトの「使用可能なデータはありません」メッセージは正しくない可能性があります。その時点では、使用可能なデータがないことは正確ですが、ユーザーがもう少し待機すると、通常はデータが表示されます。取り出しに 2、3 秒以上かかる場合、ユーザーはアプリケーションが正しく機能していないことを疑い、データの表示を長く待つのではなく、アプリケーションまたはページを再ロードしてみるでしょう。このことが問題となる場合、多くの DB2 Alphablox 開発者は、`noDataMessage` に、`Please wait for data...` または `Waiting for data...` のいずれかのようなメッセージを設定します。

これは、実際に使用可能なデータがない場合を除き、通常は良い解決策となります。実際に使用可能なデータがない場合、使用可能なデータがないことを示すメッセージには変わりません。 `noDataMessage` の変更を決定する前に、その影響を考慮

してください。しかし、一般には、メッセージをより明確にする利点は、データ・ソースが実際に使用可能ではない可能性を補って余りあります。

グリッドおよびチャートに表示されるメッセージを変更するには、noDataMessage 属性を PresentBlox、GridBlox、または ChartBlox に追加します。 PresentBlox に noDataMessage 属性が追加されると、ネストされた GridBlox および ChartBlox の両方の表示に新しいメッセージが示されます。ネストされた GridBlox と ChartBlox とで値を別々に設定することを望む場合、ネストされた各 Blox で別々に noDataMessage 属性を設定できます。 PresentBlox のネストされた GridBlox および ChartBlox で以下のように設定すると、グリッドとチャートに異なるメッセージが表示されます。

```
<blox:present ...>
  <blox:grid
     noDataMessage="Grid not available"/>
  <blox:chart
     noDataMessage="Chart not available"/>
</blox:present>
```

注意深く使用すれば、noDataMessage 属性の変更により、アプリケーションを改良することが可能です。 noDataMessage の詳細は、「*開発者用リファレンス*」の『共通 Blox』セクションを参照してください。

HTML フォーム・エレメントおよび FormBlox コンポーネント

PresentBlox、GridBlox、および ChartBlox を使用して作成されたグリッドおよびチャート・ビューには、ツールバー、ページ・フィルター、およびコンテキスト (右クリック) メニューなどの組み込み機能を含めることができます。ツールバーが使用可能な場合、チャート、グリッド、およびデータの外観と動作を変更するために、オプション・メニューを使用できます。しかし、使用可能なオプションが多いと、初心者や少し使いたいだけのユーザーにとっては、圧倒するものとなる場合があります。最良のオプションは、固有の必要に応じ、選択項目の数を制限して提供することです。 HTML フォーム・エレメント、JavaScript、Java、そして豊富な Blox API を組み合わせて使用することで、ユーザーの必要とスキルに応じて調整したり対話をカスタマイズしたりしたオプションを含む、カスタム分析アプリケーションを作成できます。 Blox Sampler には、HTML フォーム・エレメント (ボタン、チェック・ボックスなど) を使用して、制御された対話性を実現したり、データ・ビューを変更するためのオプションを提供したりするいくつかの例が掲載されています。

しかし、さらに多くの場合には、やむを得ず Blox フォーム・タグ・ライブラリーの使用時に使用可能になる FormBlox コンポーネントを使用して、HTML フォーム・エレメントを管理することになります。 FormBlox コンポーネントとその使用方法の詳細は、本書の 55 ページの『Blox Form Tag Library の使用』および Blox Sampler の例で詳しく説明されています。

さらに、Blox Sampler には、HTML エレメントおよび FormBlox コンポーネントを使用して制御された Blox との対話性の例がたくさん載せられています。特に、『FormBlox および Logic Blox の使用 (Using FormBlox and Logic Blox)』セクションまたは『データとの対話 (Interacting with Data)』セクションの例を調べてください。

以下のサブセクションでは、一部の標準 HTML フォーム・エレメントおよび FormBlox 同等機能が特に扱われます。

選択リスト

Blox API プロパティーおよびメソッドを使用することにより、PageBlox ページ・フィルターを、(fixedChoiceLists を使用した) 固定された選択リスト、(dimensionRoot を使用した) 仮想ルート、および (moreChoicesEnabled および moreChoicesEnabledDefault を使用した) メンバー・フィルターでカスタマイズすることができます。しかし、PageBlox ページ・フィルターではすべての要件を解決できない場合もあります。

グリッドおよびチャート・ビューのツールバーをオフにする場合、使用できなくなったメニューの代わりに、ドロップダウン・メニュー項目を作成できます。ハードコーディングされた選択リストを使用すると、エンド・ユーザー向けに制御されたオプションを実現できると同時に、分析ビューも使いやすくなります。たとえば、「チャート」ボタンで使用可能なチャート・タイプ・リストは、特定のビューには過剰であることがあるため、選択リストには限定されたチャート・タイプ群を用意できます。このように、ユーザーはデータの表示方法でいくらかの選択ができると同時に、特定のビューでは意味がないかもしれない選択項目は提供されません。

カスケード選択メニューは、エンド・ユーザーが特定のリストからオプションを選択できるようにしてから、その選択に基づいて他のリストで 2 次的なメニュー選択項目を提供するときに便利です。 HTML フォーム・エレメントおよび JavaScript を使用して独自のカスケード・メニューを作成することは、大掛かりな作業になることがあります。しかし、Blox フォーム・タグ・ライブラリーで使用可能な MemberSelectFormBlox を使用し、コーディングを大幅に少なくすることにより、すぐにカスケード・メニューを作成してデータ・ビューに結び付けることができます。使用される FormBlox コンポーネントは、持続性にも対応します。つまり、ユーザーのセッション中に動的に生成された HTML フォーム・エレメントでは、ユーザーがページをそのままにして後で戻ってくる場合に、ユーザーの選択は維持されます。実際にこのことを確認するには、Blox Sampler の『FormBlox および Logic Blox の使用 (Using FormBlox and Logic Blox)』セクションを調べてください。ここでは、3 つの選択リストを備えた MemberSelectFormBlox の例があります。これらの選択リストは、ユーザーの選択に応じて動的に変更されます。

Blox 内のページ・フィルターは、一般に、事前に記述された順序で、ディメンションのメンバーだけを表示します。しかし、カスタム・コーディングするか MemberSelectFormBlox を使用して、ページ・フィルターの選択項目を HTML フォーム・エレメントに移行すると、データ・ソースに対してカスタマイズされた照会に基づく、動的に生成されたページ・フィルターを作成できます。

チェック・ボックスおよびラジオ・ボタン

グリッドまたはチャート・ビューにツールバーを含めない場合、FormBlox コンポーネント (CheckBoxFormBlox および RadioButtonFormBlox) を利用してカスタム・コーディングまたは作成されたチェック・ボックスおよびラジオ・ボタンを使用して、メニュー・バーまたはツールバーが使用できないときにはアクセスできない選択項目を提供できます。ユーザーは、ツールバーで使用可能なさまざまなダイアログ・ボックスへ進むことを面倒がる場合があり、一部のオプションが使用可能であ

ることを知らない可能性があります。ビューでツールバーおよびメニュー・バーを使用しない別の利点は、限定された明確なオプション群をページ上でより目立たせることができるという点です。たとえば、「欠落抑制」および「ゼロ抑制」チェック・ボックスを含め、チェック・ボックスがチェックされているかどうかに応じて状態を切り替えることができます。ラジオ・ボタンは、相互に互換性のないオプションを提供するのに便利です。 Blox Sampler には、全体的に FormBlox コンポーネントを使用して、チェック・ボックスおよびラジオ・ボタンを管理する例があります。

標準の HTML ボタン

標準の HTML ボタンは、照会を実行したり、照会をリセットしたり、ビューを生成するアプリケーションで便利です。しかし、このボタンは、別のビューを表示するときには望ましくありません。それは、ページのタイトルが変更されない限り、現在のビューを表示するのにどのボタンを使用したのか、ユーザーには分からないためです。ラジオ・ボタンであれば、選択された特定の項目のラジオ・ボタンが常に強調表示されるので、良い方法になります。

テキスト・フィールド

テキスト入力フィールドは、分析アプリケーションでは、他の HTML フォーム・エレメントに比べてあまり使用されませんが、特殊な状況では役立つことがあります。ほとんどの場合、固定された選択オプション (たとえば、ラジオ・ボタン、チェック・ボックス、および選択リストで使用可能なオプション) は、入力を求められる値で生じる可能性のあるつづりの誤りや問題 (たとえば、ユーザーが数値の代わりに文字を入力するなど) を避けるのに役立つので望ましいものです。特定の状況では、実際的な最良または唯一のオプションは、ユーザーが自分で値を入力できるようにすることです。たとえば、分析アプリケーションで、テキスト・フィールドを使用することにより、ユーザーは、データ・ソースへの書き戻しのためにデータを入力したり、セル・アラートに独自のしきい値レベルを設定することでアプリケーションを個別設定できるようになります。テキスト・フィールドおよびテキスト域を使用することにより、ユーザーはアプリケーションにコメントを追加できます。 DB2 Alphablox アプリケーションでは、分析アプリケーションで使用するテキスト・フィールドを作成するために、カスタム・コーディングされた HTML テキスト・フィールドを使用するか、EditFormBlox コンポーネントを使用できます。EditFormBlox には、他の Blox コンポーネントのプロパティーを自動的に変更する組み込み機能があります。

ツールバー・ボタンの使用

ユーザーが対話する各 Blox コンポーネントには、Blox 機能にアクセスするためのツールバーを含めることができます。 DB2 Alphablox には、調整された Blox ツールバーを作成するためのいくつかの方法が用意されています。これにより、ユーザーは経験を積むことができます。

デフォルトでは、各 Blox はツールバーを表示します。特定の Blox でツールバーを非表示にするには、Blox の `visible` プロパティーを `false` に設定します。

false に設定された Blox の clickable プロパティーと組み合わせて使用すると (213 ページの『対話性の制限または対話性なしの許可』を参照)、対話式 UI ではなく、静的なデータ・プレゼンテーションになります。これは、即時スナップショットやエグゼクティブ報告に適していることがあります。さらに、Blox ツールバーを非表示にすることは、アプリケーションがカスタム HTML ユーザー・インターフェースを使用してツールバー機能を置き換えるときにも適している場合があります。

デフォルトでは、テキスト・ベースのメニュー・バーは、Blox ツールバーの上に表示されます。表示するには、Blox menubarVisible プロパティーを true に設定します。

デフォルトでは、ツールバーの各ボタンには、説明のためのテキスト・ラベルは表示されません。このテキストをオンにするには (この場合、一部のボタンで必要な表示スペースが増加する)、textVisible プロパティーを true に設定します。ツールバーのテキストをオンにすると、ツールバーが大きくなり、より多くの Blox 領域が必要になるので注意してください。

ネストされた ToolbarBlox の removeButtons プロパティーを使用することで、ツールバーから除去するボタンを指定できます。この機能や他の ToolbarBlox 機能の詳細は、「*開発者用リファレンス*」の『ToolbarBlox』セクションを参照してください。

イベント

DB2 Alphablox にはイベントを処理するためのプロパティーとメソッドが用意されています。*イベント* とは、別の処理をトリガーするのに使用できる通常のアクションのことです。

Blox は、以下のユーザー・アクションをキャプチャーして、イベントとして処理できます。

- ドリルダウンまたはドリルアップ
- ピボット
- ヘッダーまたはセル・メニュー項目の選択
- ページ・フィルターの変更
- ブックマークのロードまたは保管
- グリッド・セルでのデータ値の変更
- 選択的保持または選択的除去
- 選択的非表示または選択的表示

Blox UI Model イベントの説明は、86 ページの『Blox UI Model のイベント』から始まります。さらに、UI Model は、クライアントで発行できる多数のイベント (たとえば、ClickEvent) も公開します。それぞれのイベントごとに、DHTML Client API は JavaScript オブジェクトを定義します。その結果、JavaScript を使用してイベント・オブジェクトを作成し、イベントをサーバーに送信できるようになります。詳しくは、111 ページの『イベントの送信』、112 ページの『イベントのインターセプト』、および 113 ページの『ユーザー・インターフェースからの JavaScript の直接の呼び出し』を参照してください。

第 18 章 データの入力および変更

DB2 Alphablox を使用してデータをデータ・ソースに入力または書き戻すことができます。さらに、算出メンバーを使用して、データ・ソースから取得したデータから派生する新しいデータを作成することもできます。

マルチディメンション・データ・ソースへの書き戻し

DB2 Alphablox Java メソッドを使用して、結果セットを変更し、その基礎となるデータ・ソースを更新することができます。データ・セルの値を更新して、その後にそれらの値を基礎となるデータ・ソースに書き戻す機能をユーザーに備えるためには、プロパティーと関連メソッドのセットを使用して、以下の効果と結果を実現する必要があります。

- グリッドを編集できるようにする
- グリッド内に編集可能なセルを定義する
- 編集可能なセルを表示するためにスタイル (通常は前景色または背景色) を指定する
- 編集したセルを表示するためにスタイルを指定する
- オプションで、ユーザーがセルを編集するときに発生する処理を指定する

GridBlox プロパティーおよび関連する書き戻しメソッド

書き戻しアプリケーションの設計および管理には、以下の GridBlox プロパティーおよび関連 Java メソッドが必要または使用可能です。

プロパティーおよび関連メソッド	説明
• writebackEnabled • isWritebackEnabled() • setWritebackEnabled()	書き戻しを使用可能にするために必要です。ユーザーがグリッド内のセルを編集することを許可します。
• cellEditor • getCellEditor() • setCellEditor()	書き戻しを使用可能にするために必要です。データ・セルの編集可能領域を定義および強調表示するための規則を指定します。

プロパティーに関連づけられていないその他の関連メソッドについては、『GridBlox Java 書き戻しメソッド』で説明されています。ユーザーが編集した、または編集できるセルの外観を指定するには、CSS テーマを使用できます。

GridBlox Java 書き戻しメソッド

関連プロパティーがないすべての GridBlox Java メソッドは、以下のとおりです。

getWritebackValue(); setWritebackValue()
グリッド内で変更された特定のデータ・セルの値を設定または戻します。

listCellEditorIds()
すべての定義済みセル・エディターの ID のリストを整数の配列として戻します。

getChangedCellList()
編集済みセルのストリングを戻します。

getChangedCellValues()
編集済みセル値のストリングを戻します。

データ書き戻しのための GridBlox コンポーネントの使用可能化

以下の例には、GridBlox コンポーネントで書き戻し機能を使用可能にするために必要な最小限のプロパティー、およびその他の一般的に使用されるプロパティーが含まれています。

```
<blox:grid id="Grid1"
  width="800"
  height="500"
  writebackEnabled="true">
  <blox:data
     bloxRef="Data1"/>
  <blox:cellEditor
     scope="{Market:New_York}"/>
</blox:grid>
```

上記の GridBlox プロパティーは、書き戻し機能の使用可能化、編集可能なセルの定義、および書き込み可能なデータ・セルの外観の変更のために使用されます。データ・ソースにデータを書き戻すために、使用する DataBlox の書き戻し関連のメソッドのセットがあります。『書き戻し関連の DataBlox メソッド』を参照してください。

書き戻し関連の DataBlox メソッド

GridBlox を構成して書き戻し機能を使用可能にした後、サーバー・サイドの DataBlox メソッドを使用して書き戻し操作を実行する必要があります。書き戻しメソッドは、DB2 OLAP Server、Essbase、または Microsoft Analysis Services 2000 のいずれかのデータ・ソースにデータを書き戻すアプリケーション用に設計されています。いくつかのメソッドは DB2 OLAP Server または Essbase のみに固有のものです。使用可能な DataBlox Java 書き戻しメソッドは、以下のとおりです。

writeback()
3 つの引数を取る便利な書き戻しメソッドで、以下のメソッドを使用します。

- `lockCurrentDataSet()`
- `setDataValues()`
- `commitData()`
- `unlockAll()`
- `executeCustomCalc()`
- `refresh()`

lockCurrentDataSet()
呼び出された結果セットをロックします。データベース全体はロックしません。

setDataValues()
指定された座標にある結果セット内のデータ値を変更します。

commitData()
現在のデータ・セットをデータベースに書き戻します。

unlockAll()
DB2 OLAP Server データベースまたは Essbase データベースで以前にロックされたすべてのデータをアンロックします。

executeCustomCalc()
DB2 OLAP Server データベースまたは Essbase データベース上で計算スクリプトを実行します。

refresh()
現在のデータ・セットを最新表示します。

executeNamedDBCalcScript()
指名された DB2 OLAP Server 計算スクリプトまたは指名された Essbase 計算スクリプトを実行します。

マルチディメンション・データベースへの書き戻し機能の使用可能化

サーバー・サイドの Java API を使用して、マルチディメンション・データベースにデータを書き戻すことがユーザーに許可されるアプリケーション・ページを作成することができます。 Blox Sampler には 3 つの例が含まれています。1 つはカスタム Java クラスを使用するもの (推奨されるアプローチ) で、2 つは Blox UI コントローラーを使用するもの (汎用およびカスタム) です。

制約事項: DB2 OLAP Server および Hyperion Essbase の照会では、書き戻し操作での属性ディメンションの使用をサポートしていません。

Blox Sampler で使用可能なカスタム Writeback クラスには、以下のステップが組み込まれています。

1. 必要なクラスをインポートする JSP `page` ディレクティブを追加します。

```
<%@ page import="bloxsampler.writeback.Writeback,
                  com.alphablox.internal.PresentBlox" %>
```

2. ページで使用される Blox タグ・ライブラリーの JSP `taglib` ディレクティブを追加します。

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

3. ページのヘッド・セクションに、必要な CSS および JavaScript メソッドを自動的に追加する必須の `<blox:header/>` タグを忘れずに追加してください。

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

4. Blox UI Model を使用した JavaScript 関数を追加して、あたかもユーザーが PresentBlox メニュー・バーの 「**データ**」 → 「**Writeback**」メニュー・オプションをクリックしたかのように、サーバー上でシミュレートされたクリック・イベントを実行します。

```
<script language="JavaScript">
  function wb() {
    bloxAPI.sendEvent(new ClickEvent('wb3PresentBlox',
        <%= wb3PresentBlox.getBloxModel().searchForComponent(
        "dataWriteback").getUID() %>));
    }
</script>
```

5. グリッド writebackEnabled 属性を true に設定した PresentBlox を追加し、カスタム Writeback クラス (Blox 名と有効範囲の 2 つの引数を持つ) を参照するスクリプトレットを組み込みます。

```
<blox:present id="wb3PresentBlox"
   height="400"
   width="600"
   pageAvailable="true"
   chartAvailable="false"
   dataLayoutAvailable="false">

   <blox:data
      dataSourceName="QCC-Essbase"
      query='<SYM <ROW("All Products") <CHILD Truffles
        <COLUMN(Scenario) "Initial Budget" Manhattan
        <ICHILD "Jan 01" "Units Sold" !' />
<%
  new Writeback(wb3PresentBlox,"{Scenario:Initial Budget}");
%>
</blox:present>
```

6. データをデータ・ソースに書き戻すときにユーザーがクリックするボタンを、ページ上に追加します。

```
<form>
   <input type="button" value="Submit Changes"
      onclick='setTimeout( "wb();", 1 );'>
</form>
```

JavaScript wb 関数は短いタイムアウトの後に呼び出されますが、この関数が呼び出されるまでの間にサーバー上でデータを更新することができます。JavaScript setTimeout 関数が使用されていない場合は、正しいデータがデータ・ソースに書き戻されない可能性があります。

この例では、この時点で書き戻し機能が PresentBlox に取り込まれました。この書き戻し手法の作業例は、Blox Sampler の『データの入力および変更 (Inputting and Changing Data)』セクションに含まれています。 Java クラスのソース・コードは、アプリケーションの WEB-INF/src/ ディレクトリーにあります。このソース・コードを変更してその他の必要な変更を加えた後、アプリケーションで使用するためにソース・コードをコンパイルできます。

Microsoft Analysis Services へのデータの書き戻し

MicrosoftAnalysis Services では、リーフ・レベルのメンバーにしかデータを書き戻せません。非リーフ・メンバーのデータを更新するには、DataBlox setQuery メソ

ッドまたは executeQuery メソッドで MDX UPDATE CUBE コマンドを使用することが必要です。 UPDATE CUBE コマンドについて詳しくは、Microsoft Analysis Services の資料を参照してください。

リレーショナル・データ・ソースの更新

DB2 Alphablox は、リレーショナル・データ・ソースの更新のための、標準 SQL ステートメントをサポートします。これらのステートメントには、insert、update、create、および delete が含まれますが、これらには限定されていません。Java メソッドを使用して適切な SQL ステートメントを構成した後、そのステートメントをアプリケーションの DataBlox に渡すことができます。データをリレーショナル・データ・ソースに書き戻してもユーザーのデータのビューには影響しませんが、変更されたデータの影響をユーザーが調べる前に照会を再実行する必要があります。照会を再実行するには、DataBlox setQuery() メソッドを呼び出してから、connect() メソッドを呼び出す必要があります。

書き戻し機能を使用したリレーショナル・データソースの更新

DataBlox Java メソッドを使用してリレーショナル・データ・ソースを更新するには、現行日付を含む新規の列を表に挿入します。

1. query1 という名前の SQL 照会ストリングを作成します。このストリングは、現在日付を「review_data」という名前の表に挿入します。

   ```
   String query1 = "insert into review_data values(TO_CHAR(SYSDATE,
   'HH:MM:SS-MMDD'))";
   ```

2. PresentBlox 用のデータ・ソースに対して該当する setQuery メソッドおよび connect メソッドを呼び出して、新規の列を挿入する SQL 照会を渡します。この例では、PresentBlox には Present1 という名前が付いています。

   ```
   Present1.getDataBlox().setQuery(query1);
   Present1.getDataBlox().connect();
   ```

カレンダー・コントロールの作成

Blox UI モデルでは DateChooser コンポーネントが提供されていて、これによりグラフィカル・カレンダー・コントロールを作成し、分析ビューの生成や他のアプリケーションで使用する日付をユーザーが選択できるようなります。このコンポーネントにより、Web ページ上に、テキスト・フィールドとその横に小さなカレンダー・アイコンが追加されます。ユーザーがそのアイコンをクリックすると、小さなカレンダーがオープンし、ユーザーは日付を選択して、テキスト・フィールドに適切な形式の日付を追加することができます。

DateChooser コンポーネントには、以下のサブコンポーネントが含まれています。

サブコンポーネント	説明
テキスト・フィールドのグラフィカル・ユーザー・インターフェース	DateChooser コンポーネントは Edit コンポーネントを拡張し、insertText()、setValue()、getValue()、および clear() などの Edit API のほとんどをサポートします。

サブコンポーネント	説明
カレンダーの起動用イメージ	DateChooser コンポーネントには、テキスト・フィールドの横に、カレンダー起動用に表示される Image サブコンポーネントが含まれます。 `getIcon()` メソッドは、この Image コンポーネントへのアクセスを提供します。たとえば、`getIcon().setImageURL("myCalendar.gif")` はアイコンで使用するイメージ・ファイルを設定します。デフォルトでは、イメージはテーマ・ベースです。イメージ・ファイルは、DB2 Alphablox リポジトリー内のテーマのイメージ・ディレクトリー内に配置します。イメージを別のロケーションに配置するには、Image の `setThemeBasedImage(boolean)` メソッドを呼び出して `false` に設定します。詳しくは、Blox API Javadoc 資料を参照してください。
カレンダー・アダプター	この CalendarAdapter オブジェクトは、com.alphablox.blox.uimodel.ICalendar インターフェースをインプリメントするラッパーです。このインターフェースは、`getTimeInMillis()` および `getFirstDayofWeek()` などの java.util.Calendar API のサブセットに準拠します。ICalendar インターフェースを使用すると、独自のカレンダー・オブジェクトを提供して、CalendarAdapter オブジェクトをインスタンス化できます。ご自分のカレンダー・オブジェクトが機能するためには、ICalendar のすべてのメソッドを必ずインプリメントする必要があります。 デフォルトでは、DateChooser コンポーネントは英語ロケールでグレゴリオ暦カレンダーを作成します。 ICU (International Components for Unicode) ライブラリーには、非グレゴリオ暦カレンダーを簡単に作成できるようにする、カレンダー・オブジェクトと API のセットが備えられています。
日付形式アダプター	このアダプターは、com.alphablox.blox.uimodel.IDateFormat インターフェースをインプリメントするラッパーです。 IDateFormat インターフェースは、`getCalendar()`、`setCalendar()`、および `format()` などの java.text.DateFormat API のサブセットに準拠します。日付形式は、選択した日付を表示するテキスト・フィールドに適用されます。以下の日付形式がサポートされます。 • FULL (デフォルト) • LONG • MEDIUM • SHORT 異なる日付形式の例については、http://java.sun.com/j2se/1.4.2/docs/api/java/text/DateFormat.html を参照してください。 IDateFormat インターフェースを使用すると、ロケール固有の日付形式で非グレゴリオ暦カレンダーを作成できます。作成する日付フォーマッター・オブジェクトは、IDateFormat インターフェース内のすべてのメソッドをインプリメントする必要があります。

関連概念

235 ページの『非グレゴリオ暦カレンダーの作成』

ICU (International Components for Unicode) は、Unicode サポートのために広く

使用されている Java ライブラリーの集合です。 ICU が提供する機能および拡張性の 1 つは、非グレゴリオ暦カレンダーおよび異なる日付形式に対するサポートです。 ICU パッケージを使用すると、独自のカレンダー・オブジェクトおよび日付フォーマッターを簡単に作成できます。

グレゴリオ暦カレンダーの作成

カレンダー・コントロールの作成には、CalendarAdapter、DateFormatAdapter、および DateChooser の作成が含まれます。次の例では、英語ロケールの日付形式でグレゴリオ暦を作成する方法を示します。ユーザーがカレンダー・アイコンをクリックして、表示されたカレンダーから日付を選択すると、選択した日付は DateFormat.SHORT 日付形式でテキスト・フィールドのデータとして設定されます。同じ概念と手順は、その他のタイプのカレンダーにも適用されます。

JSP にグレゴリオ暦を作成するには、次のようにします。

1. 次のパッケージとクラスをインポートします。

```
<%@ page import="com.alphablox.blox.uimodel.core.*,
                 com.alphablox.blox.uimodel.*,
                 java.util.Calendar,
                 java.util.Locale"%>
```

2. Blox Tag Library をインポートします。

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

3. ContainerBlox を追加して、DateChooser コンポーネントを組み込みます。

```
<blox:container id="dateChooserContainer">
```

4. コンテナーの BloxModel を取得します。

```
<%
   BloxModel model = dateChooserContainer.getBloxModel();
   ...
%>
```

5. ロケールを設定します:

```
Locale locale = Locale.US;
```

6. CalendarAdapter を作成します:

```
java.util.Calendar calendar = java.util.Calendar.getInstance( locale );
ICalendar calendarAdapter = new CalendarAdapter( calendar );
```

7. DateFormatAdapter を作成して、ロケールを適用します:

```
java.text.DateFormat dateFormat =
java.text.DateFormat.getDateInstance( java.text.DateFormat.FULL, locale );
IDateFormat dateFormatAdapter = new DateFormatAdapter( dateFormat );
```

8. DateChooser オブジェクトを作成し、DateChooser.getInstanceWithLocale() メソッドを使用して CalendarAdapter、DateFormatAdapter、およびロケールを指定します:

```
<%
DateChooser datechooser =
DateChooser.getInstanceWithLocale( calendarAdapter, dateFormatAdapter, locale );

%>
```

9. DateChooser をコンテナーのモデルに追加します。

```
   <%
      ...
      model.add(datechooser1);
   %>
```

完全な例は、次のとおりです。

```
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ page import="com.alphablox.blox.uimodel.core.*,
                 com.alphablox.blox.uimodel.*,
                 java.util.Calendar,
                 java.util.Locale"%>
<%@ taglib uri="bloxtld" prefix="blox"%>

<html>
<head>
<blox:header/>
</head>
<body>
<blox:container id="dateChooserContainer">
<%
  BloxModel model = myContainer.getBloxModel();
  Locale locale = Locale.US;

  java.util.Calendar calendar = java.util.Calendar.getInstance( locale );
  ICalendar calendarAdapter = new CalendarAdapter( calendar );

  java.text.DateFormat dateFormat =
java.text.DateFormat.getDateInstance( java.text.DateFormat.FULL, locale );
  IDateFormat dateFormatAdapter = new DateFormatAdapter( dateFormat );

  DateChooser datechooser =
DateChooser.getInstanceWithLocale( calendarAdapter, dateFormatAdapter, locale );

  01model.add( datechooser );%>
</blox:container>
</body>
</html>
```

ロケールを変更することで、英語以外のグレゴリオ暦カレンダーを簡単に作成できます。ロケールや初期選択日付をどのように設定するかに応じて、DateChooser を作成するファクトリー・メソッドは他にもあります。 234 ページの『カレンダーの起動時の選択日付の指定』 の例は、DateChooser を作成する別のファクトリー・メソッドを示したものです。CalendarAdapter、DateFormatAdapter、DateChooser にそれぞれ別々のロケールを設定することも可能です。たとえば、和暦のカレンダーを使用し、DateFormat (選択した日付をテキスト・ボックスに表示するときに使用するフォーマット) を別の言語にすることができます。

Java の Calendar または ICU (International Components for Unicode) Java ライブラリーの Calendar コンポーネントのどちらかを使用できます。複数のロケールをサポートするには、国際化対応のサポートに優れた ICU を使用してください。詳しくは、 233 ページの『マルチロケール・サポート用に ICU を使用したグレゴリオ暦カレンダーの作成』を参照してください。

DateChooser の使用例については、『UI 拡張性 (UI Extensibility)』セクションにある Blox Sampler の「DateChooser コンポーネント」の例を参照してください。

関連概念

235 ページの『非グレゴリオ暦カレンダーの作成』

ICU (International Components for Unicode) は、Unicode サポートのために広く

使用されている Java ライブラリーの集合です。 ICU が提供する機能および拡張性の 1 つは、非グレゴリオ暦カレンダーおよび異なる日付形式に対するサポートです。 ICU パッケージを使用すると、独自のカレンダー・オブジェクトおよび日付フォーマッターを簡単に作成できます。

関連タスク

234 ページの『カレンダーの起動時の選択日付の指定』
デフォルトでは、カレンダーの起動時には現行月のカレンダーに現行日付が選択されて表示されます。現行日付以外の別の初期日付を指定することができます。

マルチロケール・サポート用に ICU を使用したグレゴリオ暦カレンダーの作成

この例では、英語ではない言語のサポートに優れた ICU を使用します。Blox コンテキストからロケールを取得して、DateChooser がクライアントのロケール設定を反映する様子も示されています。

JSP にグレゴリオ暦を作成するには、次のようにします。

1. 次のパッケージとクラスをインポートします。

```
<%@ page import="com.alphablox.blox.uimodel.core.*,
                 com.alphablox.blox.uimodel.*,
                 java.util.Locale"%>
```

2. Blox Tag Library をインポートします。

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

3. ContainerBlox を追加して、DateChooser コンポーネントを組み込みます。

```
<blox:container id="container">
```

4. コンテナーの BloxModel を取得します。

```
<%
   BloxModel model = container.getBloxModel();
   ...
%>
```

5. Blox コンテキストからロケールを取得します:

```
Locale locale = bloxContext.getLocale();
```

6. CalendarAdapter を作成して、ロケールを適用します:

```
com.ibm.icu.util.Calendar calendar =
new com.ibm.icu.util.GregorianCalendar( locale );
ICalendar calendarAdapter = new CalendarAdapter( calendar );
```

7. DateFormatAdapter を作成します:

```
com.ibm.icu.text.DateFormat dateFormat =
calendar.getDateTimeFormat( com.ibm.icu.text.DateFormat.FULL, -1, locale );
IDateFormat dateFormatAdapter = new DateFormatAdapter( dateFormat );
```

8. DateChooser オブジェクトを作成し、`DateChooser.getInstanceWithLocale()` メソッドを使用して CalendarAdapter、DateFormatAdapter、およびロケールを指定します:

```
DateChooser datechooser = DateChooser.getInstanceWithLocale( calendarAdapter,
dateFormatAdapter, locale );
datechooser.setName("datechooser");
datechooser.setWidth(300);
```

9. DateChooser をコンテナーのモデルに追加します。

```
   model.add(datechooser1);
```

完全な例は、次のとおりです。

```
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ page import="com.alphablox.blox.uimodel.core.*,
                 com.alphablox.blox.uimodel.*,
                 java.util.Locale"%>
<%@ taglib uri="bloxtld" prefix="blox"%>

<html>
<head>
<blox:header/>
</head>
<body>
<blox:container id="container">
<%
  BloxModel model = container.getBloxModel();
  Locale locale = bloxContext.getLocale();

  com.ibm.icu.util.Calendar calendar =
new com.ibm.icu.util.GregorianCalendar( locale );
  ICalendar calendarAdapter = new CalendarAdapter( calendar );

  com.ibm.icu.text.DateFormat dateFormat =
calendar.getDateTimeFormat( com.ibm.icu.text.DateFormat.FULL, -1, locale );
  IDateFormat dateFormatAdapter = new DateFormatAdapter( dateFormat );

  DateChooser datechooser = DateChooser.getInstanceWithLocale( calendarAdapter,
dateFormatAdapter, locale );
  datechooser.setName("datechooser");
  datechooser.setWidth(300);

  model.add( datechooser );

%>
</blox:container>
</body>
</html>
```

関連概念

235 ページの『非グレゴリオ暦カレンダーの作成』

ICU (International Components for Unicode) は、Unicode サポートのために広く使用されている Java ライブラリーの集合です。 ICU が提供する機能および拡張性の 1 つは、非グレゴリオ暦カレンダーおよび異なる日付形式に対するサポートです。 ICU パッケージを使用すると、独自のカレンダー・オブジェクトおよび日付フォーマッターを簡単に作成できます。

関連タスク

『カレンダーの起動時の選択日付の指定』

デフォルトでは、カレンダーの起動時には現行月のカレンダーに現行日付が選択されて表示されます。現行日付以外の別の初期日付を指定することができます。

カレンダーの起動時の選択日付の指定

デフォルトでは、カレンダーの起動時には現行月のカレンダーに現行日付が選択されて表示されます。現行日付以外の別の初期日付を指定することができます。

別の初期日付を指定するには、選択日付の指定をサポートする `DateChooser.getInstanceWithDateLocale()` メソッドを使用します。次の例では、選択日付を 2005 年 1 月 1 日に設定したグレゴリオ暦カレンダーを作成します。

```
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ page import="com.alphablox.blox.uimodel.core.*,
                 com.alphablox.blox.uimodel.*,
                 java.text.DateFormat,
                 java.util.Date,
                 java.util.Calendar,
                 java.util.Locale,
                 java.util.GregorianCalendar"%>
<%@ taglib uri="bloxtld" prefix="blox"%>

<html>
<head>
<blox:header/>
</head>
<body>
<blox:container id="dateChooserContainer">
<%
  BloxModel model = dateChooserContainer.getBloxModel();
  Locale locale = Locale.US;

  java.util.Calendar calendar = java.util.Calendar.getInstance( locale );
  ICalendar calendarAdapter = new CalendarAdapter( calendar );

  java.text.DateFormat dateFormat =
java.text.DateFormat.getDateInstance( java.text.DateFormat.FULL, locale );
  IDateFormat dateFormatAdapter = new DateFormatAdapter( dateFormat );

  GregorianCalendar mydate = new GregorianCalendar(2005, Calendar.JANUARY, 01);
  Date d = mydate.getTime();
  DateChooser datechooser = DateChooser.getInstanceWithDateLocale( d,
calendarAdapter, dateFormatAdapter, locale );

    model.add( datechooser );%>
</blox:container>
</body>
</html>
```

非グレゴリオ暦カレンダーの作成

ICU (International Components for Unicode) は、Unicode サポートのために広く使用されている Java ライブラリーの集合です。 ICU が提供する機能および拡張性の 1 つは、非グレゴリオ暦カレンダーおよび異なる日付形式に対するサポートです。 ICU パッケージを使用すると、独自のカレンダー・オブジェクトおよび日付フォーマッターを簡単に作成できます。

アプリケーションが複数のロケールをサポートする必要がある場合は、java.util.Calendar および java.text.DateFormat パッケージを使用する代わりに、ICU の Calendar および DateFormat オブジェクトを使用するようにします。

非グレゴリオ暦カレンダーの例

グレゴリオ暦以外のカレンダーを作成する前に、231 ページの『グレゴリオ暦カレンダーの作成』で説明されているステップに従って、基本的なカレンダー・コントロールをページ上に追加します。

以下のステップは、JSP ファイルに和暦のカレンダー・コントロールを作成する方法を示しています。イスラム暦、ユダヤ暦、または中国暦のカレンダー・コントロールを作成する際にも同じ手順を当てはめることができます。

グレゴリオ暦ではない和暦のカレンダー・コントロールを作成するには、次のようにします。

1. 次のパッケージをインポート・ステートメントに追加します。
   - java.util.Date
   - java.util.Locale

   Calendar クラスと DateFormat クラスは ICU パッケージのものを使用します。ICU は DB2 Alphablox に同梱されているので、パッケージをインポートする必要はありません。インポート・ステートメントは、ここでは以下のようになります。

   ```
   <%@ page import="com.alphablox.blox.uimodel.core.*,
                    com.alphablox.blox.uimodel.*,
                    java.util.Date,
                    java.util.Locale"%>
   ```

2. 日本語の CalendarAdapter を作成します。CalendarAdapter オブジェクトは、ICalendar 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。

   ```
   com.ibm.icu.util.Calendar japaneseCalendar =
   new com.ibm.icu.util.JapaneseCalendar( Locale.JAPAN);
   ICalendar japaneseCalendarAdapter =
   new CalendarAdapter( japaneseCalendar );
   ```

3. DateFormatAdapter オブジェクトを作成します。カレンダーから日本語データ・フォーマットを取得して、DateFormatAdapter に適用します。

   ```
   com.ibm.icu.text.DateFormat japaneseDateFormat =
   japaneseCalendar.getDateTimeFormat( com.ibm.icu.text.DateFormat.FULL,
   -1, Locale.JAPAN );
   japaneseDateFormat.setCalendar( japaneseCalendar );
   IDateFormat japaneseDateFormatter = new DateFormatAdapter( japaneseDateFormat );
   ```

   同じカレンダーをこのフォーマッターに適用しないと、テキスト・フィールドはデフォルトの英語形式になります。

4. 日本語ロケールを DateChooser に適用します。

   ```
   DateChooser datechooserJAPAN =
      new DateChooser( japaneseCalendarAdapter,
         japaneseDateFormatter, Locale.JAPAN );
   ```

完全な例は、次のとおりです。

```
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ page import="com.alphablox.blox.uimodel.core.*,
                 com.alphablox.blox.uimodel.*,
                 java.text.DateFormat,
                 java.util.Date,
                 java.util.Locale"%>
<%@ taglib uri="bloxtld" prefix="blox"%>

<html>
<head>
<blox:header/>
</head>
<body>
<blox:container id="dateChooserContainer">
<%
   BloxModel model = dateChooserContainer.getBloxModel();

   // Create your Japanese CalendarAdapter
   com.ibm.icu.util.Calendar japaneseCalendar =
     new com.ibm.icu.util.JapaneseCalendar();
   ICalendar japaneseCalendarAdapter =
     new CalendarAdapter( japaneseCalendar );
```

```
   // Apply the Japanese date format to your DateFormatAdapter
   com.ibm.icu.text.DateFormat japaneseDateFormat =
japaneseCalendar.getDateTimeFormat( com.ibm.icu.text.DateFormat.FULL,
-1, Locale.JAPAN );
   japaneseDateFormat.setCalendar( japaneseCalendar );
   IDateFormat japaneseDateFormatter = new DateFormatAdapter( japaneseDateFormat );

   // Apply the same locale to the chooser
   DateChooser datechooserJAPAN = new DateChooser(japaneseCalendarAdapter,
japaneseDateFormatter, Locale.JAPAN );
   datechooserJAPAN.setName("datechooserJAPAN");
   model.add(datechooserJAPAN);
%>
</blox:container>
</body>
</html>
```

重要:

- 中国暦カレンダーに ICU パッケージを使用する場合には、ミリ秒を明示的に設定する必要があります。そのようにしないと、各フィールドを正しく評価することはできません。

  ```
  com.ibm.icu.util.Calendar chineseCalendar =
    new com.ibm.icu.util.ChineseCalendar();
  chineseCalendar.setTimeInMillis((new Date()).getTime());
  ICalendar chineseCalendarAdapter =
    new CalendarAdapter(chineseCalendar);
  ```

- デフォルトでは、表示方向は左から右に設定されます。ヘブライ語やアラビア語などの双方向言語の場合、次のように DateChooser の方向を設定しなければなりません。

  ```
  myDateChooser.setBidiDirecton(DateChooser.RTL);
  ```

カレンダー・コントロールのフォント

カレンダー・コントロールのテキスト用のフォントは、DB2 Alphablox Repository の各テーマの CSS ファイルで指定されます。指定されるフォントは、Lucida Sans Unicode、Arial、Helvetica、および sans-serif です。Lucida Sans Unicode は複数のロケールをサポートする点で最適なフォントであるため、このフォントがデフォルトに設定されます。このフォントは、Unicode 標準のバージョン 2.0 で定義された一般的に使用されるほとんどの文字をサポートするように設計されており、Windows 2000 と Windows XP に組み込まれています。

まれなケースとして、ユーザーのシステムにこのフォントがインストールされておらず、そのシステムのロケールが Arial、Helvetica、および sans-serif のサポート範囲外の Unicode コードを使用する言語に設定されている場合、カレンダー・コントロールのテキストは正しく表示されないことがあります。 1 つの解決策は、それらのシステムに Lucida Sans Unicode フォントをインストールすることです。さらに、Lucida Sans Unicode を検索してそれを置き換えることによって、CSS ファイルでのフォント指定を変更することができます。しかし、必ず Unicode をサポートするフォントに設定する必要があり、ユーザーのシステムにそのフォントをインストールする必要があります。

算出メンバー

算出メンバーとは、結果セットに実際に存在するメンバーに対して実行された計算から派生する動的に生成されたデータを含むデータ・メンバーで、新規に作成された行または列にその後表示されます。DB2 OLAP Server、Hyperion Essbase、Microsoft Analysis Services などのいくつかのデータ・ソースでは、独自の照会言語、Essbase レポート仕様言語、および Microsoft Multidimensional Expression (MDX) 言語を使用して、算出メンバーを生成できます。ただし、通常これらの算出メンバーは相互作用しません。例えば、照会から派生した算出メンバーを使用してキューブをドリルアップまたはドリルダウンすると、算出メンバーの名前がデータベースに再サブミットされます。これらのメンバーはデータ・ソースに初めから存在しないので、そうした照会は失敗します。

DB2 Alphablox には、結果セットがデータ・ソースから戻された後に、算出メンバーを作成する機能があります。さらには、ユーザーがデータと対話したり、これらの算出メンバーを実際のメンバーであるかのようにして使用したりできるように、対話を管理します。

DB2 Alphablox での算出メンバーの作成

算出メンバーは、DataBlox の `calculatedMembers` プロパティーを使用して DB2 Alphablox アプリケーション内で作成されます。 DB2 Alphablox の算出メンバーを使用する 1 つの重要な利点は、実際のデータ・ソースを変更せずに新しいデータを結果セットに追加できるという点です。これは、データベースの変更を待てない場合や、既存のデータから派生する新しい指標を使用して少しだけ実験してみたい場合に、特に便利です。ユーザーにとって算出メンバーが便利であるいくつかの例を挙げてみます。

- 2 つのメンバー (予算と実際の額など) の値の間の分散および分散比率
- 指定されたディメンション (ドルの売り上げなど) のすべてのメンバーの平均
- 合計売り上げに対する比率

DataBlox `calculatedMembers` プロパティーの構文および使用法の詳細については、「*開発者用リファレンス*」を参照してください。以下のセクションには、算出メンバーに関するいくつかの重要なトピックがあります。

カスタム計算のガイドライン

カスタム計算の作業をするときには、以下のガイドラインおよび制約事項を知っておく必要があります。

カスタム計算の定義

カスタム計算の定義を行うときには、以下のガイドラインが適用されます。

- 算出メンバーの定義は、数学的優先順位における通常規則に従って評価される算術式です。
- 参照メンバーを指定して算出メンバーを配置します。その算出メンバーは、グリッド内でその参照メンバーより前に配置されます。それ以外の場合には、算出メンバーは既存のディメンションの末尾に現れます。

- 同じ参照メンバーの前に複数の算出メンバーが配置されるように定義されている場合には、算出メンバーは定義内で最後にある算出メンバーが参照メンバーに最も近くなるように配列されます。
- 算出メンバーは、そのディメンションの既存の表示されたメンバーに倣って定義される必要があります。
- ディメンション上での位置決めをせずに複数の算出メンバーが定義される場合には、メンバーは、定義順にディメンションに追加されます。
- 算出メンバーの定義式では、以前に定義された算出メンバーを使用することができますが (後方参照)、未定義の算出メンバーを使用する (前方参照) ことはできません。

カスタム計算の制約事項

- 算出メンバーをチャートにするため、「世代フィルター」は、すべての世代を表示するように設定する必要があります。これは、`totalsFilter` プロパティーを 0 に設定するか、DHTML クライアントの「チャート・オプション」ダイアログ・ボックスで、「世代フィルター」を「すべての世代を表示」に設定することにより行えます。
- 算出メンバーの保管およびリストアは、通常のブックマーク操作により行えます。
- メンバー名に、等号 (=)、中括弧 ({})、または二重引用符 (″) を含めることはできません。
- Microsoft Analysis Services データ・ソースまたは DB2 Alphablox Cube データ・ソースを使用するときには、固有のメンバー名が必要です。
- DB2 OLAP Server または Essbase では、算出メンバーの式では固有のメンバー名を使用することが強く勧められています。DataBlox `useAliases` プロパティーが false に設定されていたり、ユーザーが UI の別名を使用不可にした場合に、計算に失敗する場合があります。
- 特定のメンバーについて、指定した数値として欠落した値または NULL 値を処理したい場合には、式で `ifNotNumber` 関数を使用して特殊ケース・ロジックを規定します。

  ヒント: `ifNotNumber` 関数で値を指定する場合には、計算結果の値がどのようになるかを理解しておく必要があります。欠落値または NULL 値を別の値に置換すると、計算において意味をなさなくなる場合があります。

適切なデータ表示を妨げる条件

以下の表では、意味のあるデータが表示できなくなる条件と、その条件が発生したときのグリッドの表示内容をリストしています。

条件	グリッド・セルの表示
ゼロ除算	算出メンバーはグリッドに出力されません
メンバーへの参照が結果セットにない	空ストリング (または `missingValueString` プロパティーで指定された値)
無効な計算式	`#Error` (さらにコンソールまたはログにエラーが出力されます)

条件	グリッド・セルの表示
欠落値または非数値への参照	空ストリング (または missingValueString プロパティーで指定された値)

算出メンバー・プロパティーの構文

メンバーに 1 つ以上のカスタム計算を指定するには、DataBlox calculatedMembers プロパティー (または setCalculatedMembers メソッド) を使用します。タグ属性構文では、以下のように単一のステートメントに複数のカスタム計算を組み込むことができる点に注目してください。

```
calculatedMembers="dim1:calc1{refMember1:gen:missingIsZero}=
   expr1{scopeDim:scopeMember}, dim2:calc2{refMember2} =
   expr2{scopeDim:scopeMember},..., dimn:calcN
      {refMemberN:gen:missingIsZero} =
   exprN{scopeDim:scopeMember}"
```

ここで、各パラメーターは以下のとおりです。

- dimN 値は、算出メンバーを作成するディメンションの名前です。
- calcN 値は、算出メンバーの名前です。
- [オプション] refMemberN 値は、算出メンバーの計算がグリッド内で先に来る既存のメンバーの名前です。 refMemberN は、別の算出メンバーにすることはできません。
- [オプション] 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は、definitionString の missingIsZero コンポーネントを使用できます。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。[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。]
- exprN 値は、dim のメンバーに関係した算術式です。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、各パラメーターは以下のとおりです。

  – 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。

算出メンバーで使用可能な関数

使用する計算式に関連して、以下の表では、使用可能な関数に関連した算術計算、検索、特殊計算、および欠落値を要約しています。構文および使用法の詳細については、「*開発者用リファレンス*」の『DataBlox』セクションを参照してください。

算術関数

説明

Abs メンバー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。

Average
定義内のすべての数値の平均を戻します。これは、総和を個数で除算したものです。

Count 定義に含まれる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 セットの中央にある数値の値を戻します。

Min 定義内のすべての数値の中の最低値を戻します。

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。

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。

Stdev 定義内のすべての数値の標準偏差を戻します。

Sum 定義内のすべての数値の和を戻します。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。

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。

Descendants
指定されたメンバーのすべての子孫を戻します。

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。

特殊計算関数
説明

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。

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計値を戻します。

関数に関連する欠落値
説明

ifNotNumber
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ために使用することができます。

関数に関連したこれらの算術計算、検索、特殊計算、および欠落値の構文および使用法の詳細は、「*開発者用リファレンス*」の『DataBlox』セクションにある calculatedMembers プロパティーの説明に示されています。

算出メンバーの例

以下に、算出メンバーの一般的な使用法のいくつかの例を示します。

- 実際の値と予算値の間の分散比率を示すセル値を持つカスタム計算を定義します。

```
calculatedMembers = "Scenario:Variance % = (Actual-Budget)/Budget*100"
```

- 実際の値と予算値の間の分散比率を示すセル値を持つカスタム計算を定義し、Actual を 1,000,000 の値に置き換え、Budget を 5,000 の値に置き換える (数値を指定するときにコンマは使用しない) ためのロジックを規定します。

```
calculatedMembers="Scenario:Variance %=(ifNotNumber(Actual,1000000)-
   ifNotNumber(Budget,5000))/ifNotNumber(Budget,5000) * 100"
```

- 年の最初の 2 つの四半期における今までの売り上げを表示するカスタム計算を定義します。

```
calculatedMembers = "Year: YTD = \"Q1,2000\" + \"Q2,2000\""
```

- 単一属性において 2 つのカスタム計算を結合させます (ここで Scenario が 1 つのディメンションにあり、Year が別のディメンションにあります)。

```
calculatedMembers="Scenario:Variance % =(Actual-Budget)/Budget*100,
   Year: YTD = \"Q1,2000\" + \"Q2,2000\""
```

- 単一属性において 2 つのカスタム計算を結合させ、異なる式で使用されている同じメンバーを異なる値に置き換えます。

```
calculatedMembers ="Scenario:Variance % =
   (Actual-ifNotNumber(Budget, 10000))/ifNotNumber(Budget, 10000) * 100,
   Scenario: Difference = Actual-ifNotNumber(Budget, 0)"
```

- 実際の値と予算値の間の分散比率を示すセル値を持つカスタム計算を定義します。算出メンバー Variance % が、メンバー Actual の前になるようにグリッド上に配置します。

```
calculatedMembers = "Scenario: Variance % {Actual} =
   (Actual-Budget)/Budget * 100"
```

- 各グループ内で別個のランキングを追加するには、この例に示されているように、Rank 関数を使用できます。

```
calculatedMembers="All Products:Rank =
   Rank(All Products,All Locations,2,DESC,GROUPDIM)
```

注: 算出メンバーを消去するには、空ストリングを setCalculatedMembers に渡します。

同じ軸上でネストされたディメンションを持つ計算を実行したり、計算の内部で計算を実行したりすることもできます。算出メンバーの有効範囲を指定するか、算出メンバーの位置決めを補助するために計算に世代番号を割り当ててください。calculatedMembers プロパティーと関連メソッドの使用法を示す詳細および例については、「*開発者用リファレンス*」の『DataBlox』セクションを参照してください。

Essbase レポート・スクリプト・コマンドを使用する算出メンバー

算出メンバーは、Essbase レポート・スクリプト・コマンドを使用して作成することもできます。ある状態ではこれは便利になりますが、結果として表示されるデータは、対話するときに必ず DB2 OLAP Server エラー・メッセージまたは Essbase エラー・メッセージを生成します。そのエラー・メッセージは、(DB2 OLAP Server キューブまたは Essbase キューブに実際に存在しない) 算出メンバーが存在しないことを示します。例えば、算出メンバーが存在するグリッド上でドリルする場合には、ドリルおよび他の操作は使用不可になります。望ましい代わりの方法として、可能なかぎり、DB2 Alphablox を使用して算出メンバーを作成するようにします。

ESSbase レポート・スクリプト・コマンドを使用した算出メンバーの作成の詳細については、DB2 OLAP Server 資料または Essbase 資料を参照してください。さらに、 147 ページの『DB2 Alphablox でサポートされている Essbase レポート・スクリプト・コマンド』も必ず参照してください。

第 19 章 データのフィルタリング

このトピックでは、ユーザーがデータをフィルタリングする上でのヒントおよび手法を説明します。これは、大きな結果セットをより効率的に処理したり、情報へのアクセスを制限したり、ユーザーに表示される情報を個別設定したりする助けになります。

ディメンションおよびメンバーの非表示

データにアクセスすることにより、ユーザーは有意義な仕方で質問をしたりデータを比較したりすることが可能になりますが、時には使用可能な情報量に圧倒されてしまうことがあります。経験の浅いエンド・ユーザーは、あまりにも多くのデータが表示されると、どこから手を付けたら良いのかわからなくなることがあります。経験が豊富なユーザーにとっては、多くのデータは実際問題として「ノイズ」になってしまい、肝要なデータに注意を向ける上での妨げとなります。ですから開発者は、対象者をいつも考慮に入れ、ユーザーに適切な情報量を提供する分析アプリケーションを開発する必要があります。「データ・レイアウト」パネル、「メンバー・フィルター」ダイアログ、およびデータの完全なリスト表示における全アクセス権限を持つユーザーは、非表示/表示機能および保持/除去機能を使用して情報をふるいにかけて必要な情報を取り出すために、多くの時間を費やす結果になります。しかし開発者は、DataBlox プロパティーを使用して、当面のタスクには関係のないデータや、あまり使用されないデータをフィルター操作により除去し、ユーザーを助けることができます。

DB2 OLAP Server および Hyperion Essbase の属性ディメンション、および Microsoft Analysis Services の仮想ディメンションを使用して、ビジネスでは以前では不可能であったような仕方でデータをスライス操作しています。 QCC サンプル・データ・ソースを例に取ると、1 つのパッケージごとの個数、パッケージごとのオンス数、またはナッツが入っているかどうかによりグループ化してチョコレートの売り上げを分析したり、発表日ごとに商品をリストしたり、店舗床面積を考慮して販売店を分析したりすることができます。これは、この情報を分析で使用したい場合には、すばらしい機能です。ところが使用しない場合には、そうした情報がビューにあると邪魔であるだけです。幸いにも、DB2 Alphablox には、DataLayout `hiddenDimensionsOnOtherAxis` プロパティーおよび DataBlox `hiddenMembers` プロパティーという、ディメンションおよびメンバーを容易に非表示にできる 2 つのプロパティーが用意されています。

DataLayoutBlox `hiddenDimensionsOnOtherAxis` プロパティーは、「データ・レイアウト」パネルに表示したくないディメンションをリストするために使用できます。一例として、Blox Sampler の『データのフィルタリング (Filtering Data)』セクションを参照してください。 `hiddenDimensionsOnOtherAxis` の構文および使用法の詳細については、「*開発者用リファレンス*」の『DataLayoutBlox リファレンス』セクションを参照してください。

ある場合には DataBlox `hiddenMembers` プロパティーを使用して、表示する意味がないメンバーを非表示にすることができます。シナリオのディメンションを例に取

ると、最上位のメンバーはシナリオですが、このメンバーは実際のところ、その下に子としてグループ化されているメンバーを保持するためのバケットの役割を持つにすぎません。さらに、 DB2 OLAP Server および Hyperion Essbase では、シナリオは、その下の最初の子のデータを実際に表示しますが、シナリオというメンバーにはデータはありません。 Blox Sampler の『データのフィルタリング (Filtering Data)』セクションでは、(DB2 OLAP Server、Essbase、および Microsoft Analysis Services において) シナリオ・メンバーを非表示にする実例の中でシナリオの子について示されています。その子の 1 つをドリルアップするときに、(シナリオは非表示のメンバーとしてリストされているにもかかわらず) シナリオが実際に表示されるという点に注意してください。しかし、シナリオの中にドリルダウンして戻るときには、再びそれは表示されなくなります。この動作は、ドリル操作の実行方法に制限があるため、回避できません。 `hiddenMembers` の構文および使用法の詳細については、「*開発者用リファレンス*」の『DataBlox リファレンス』のセクションを参照してください。

dimensionRoot プロパティーの使用

情報のフィルタリング (または情報へのアクセスの制御) の最も単純な方法の 1 つは、DataBlox `dimensionRoot` プロパティーを使用して、ユーザーの仮想ルートとして使用される、ディメンション上の特定のメンバーを指定する方法です。特定のディメンション・ルート・メンバーが新規「仮想」ルートとして定義されると、ユーザーは、定義されたルートより上のメンバーにはドリルアップできなくなります。この設定は、ページ・フィルター、行と列、および「メンバー・フィルター」に表示されるディメンション・メンバーのリストに適用されます。仮想ルートは、他人にアクセスしてほしくない自分のデータの領域へのアクセスを制限する上で役立つかもしれません。このプロパティーは、限定されたセキュリティー用法として、またはユーザーがデータの中で迷子にならないようにする助けとして使用できます。

アクセス権限のないデータ値をユーザーが見ることを防ぐために、おそらくデータベース・セキュリティーを使用していると思いますが、時には、メンバー名を見るだけでも情報共有のし過ぎであるということもあり得ます。このような限定されたケースでは、`dimensionRoot` が役立つかもしれません。例えば、国内の地域全体に住む見込み客をリストするデータベースは、会社をまさに辞めて競争相手になろうとしている不満を持つ従業員にとっては役立つ情報を提供してしまう可能性があります。このような情報へのアクセスを最小化することが重要である場合には、`dimensionRoot` プロパティーは役立つオプションとなるかもしれません。

注: ユーザーがプライベート情報にアクセスすることを防ぐ手段として、`dimensionRoot` プロパティーのみを使用することはしないでください。DataBlox で定義された仮想ルートを使用してブロックした情報にアクセスするために、同じデータ・ソースに対して他のデータベース・ツールが使用される可能性が常にあります。

特定ユーザーのニーズに関係のない情報へのパスをブロックすることによりユーザビリティーを向上させることも主な目標です。これにより、データの中で迷子になる可能性のある情報パスの周辺をユーザーがドリル操作することを防ぎます。ある情報が、ユーザーが実行する必要のあるタスクに関係がない場合には、`dimensionRoot` を使用すると、この「ナビゲーション・スペースで迷子になる」問題を最小限に抑える助けになります。

ユーザーのための仮想ルートの設定

以下の例では、ユーザーは、国内の特定の地域についての情報にアクセスする機能が制限されます。この単純なステップにより、ユーザーが作業する国内の特定の地域と結びついた「仮想」ルートをユーザーのために作成するように、DataBlox を構成することが可能になります。

1. スタンドアロンまたはネストされた DataBlox に dimensionRoot 属性を追加します。

```
<blox:data id="myDataBlox"
   dimensionRoot=""
   ...
</blox:data>
```

2. ディメンションを追加し、ユーザーの「仮想」ルートとして使用したい単一メンバーをそのディメンションに追加します。

   例えば、QCC データベース内の East 地域へのアクセスを制限したい場合には、dimensionRoot 設定は次のようになります。

```
dimensionRoot="All Locations:East"
```

3. ページを保管してそれをテストします。

これは、ハードコーディングされた dimensionRoot 設定が含まれる非常に単純な例です。 dimensionRoot は、DB2 Alphablox リポジトリーに保管されている値を基にして dimensionRoot を設定することにより、ページ・ロード時に動的に設定することもできます。カスタム・ユーザー・プロパティーは、データ・ソースに表示される領域の名前をその値として持つことができます。

注: セキュリティー上の理由で、このような状態でユーザーが編集することを禁止したいでしょうし、ユーザーが自分の領域を変更して他の領域へのアクセス権限を取得することを防ぎたいことでしょう。

注: ユーザーおよびアプリケーションのためにカスタム・ユーザー・プロパティーを作成する方法の説明については、「*管理者用ガイド*」を参照してください。

DB2 Alphablox を使用してカスタム・ユーザー・プロパティーを構成すると、サーバー・サイドの Java メソッドを使用してその値にアクセスできます。 DataBlox を持つ JSP ページでは、dimensionRoot 設定は、DataBlox タグ内または関連した setDimensionRoot() メソッドを使用して動的に構成できます。 JSP ページにおいて、DataBlox タグの前に RepositoryBlox を組み込んだ場合には、JSP 式ステートメントを使用して DataBlox タグ内にある値を動的に組み込むことができます。そのコードは、次の例のようになります。

```
dimensionRoot="<%= myRB.getUserProperty("userRegion") %>"
```

ここで、myRB は、RepositoryBlox の名前を表します。これは、DB2 Alphablox リポジトリーから値を取得できるこの設定の上に定義されています。この例では、カスタム・ユーザー・プロパティーは userRegion です。この非常に単純な手法は、他のプロパティーにも適用できます。

固定選択リスト

通常、DB2 Alphablox アプリケーションのページ・フィルターを使用すると、ユーザーはあるディメンション内で制限なく情報を遡ったり掘り下げたりすることが可能になり、メンバー・フィルターへのアクセスが制限されます。 dimensionRoot プロパティーを使用すると、ユーザーが掘り下げることができないルートを定義することにより、情報アクセスを制限することができます。とはいえ、ユーザーが選択できる選択項目の数を制限する方がより良いオプションとなる場合があります。これは、PageBlox fixedChoiceLists プロパティー、moreChoicesEnabledDefault プロパティー、および moreChoicesEnabled プロパティーを設定して固定選択リストを作成することにより行います。

例: これらの PageBlox プロパティーの中の 2 つの使用を示す「固定選択リスト」の例が、Blox Sampler アプリケーションの『データのフィルタリング (Filtering Data)』セクションから入手できます。

fixedChoiceLists プロパティーの使用

PageBlox の fixedChoiceLists プロパティーは、ページ・フィルターのドロップ・リストに名前が付いたディメンションおよびメンバーを配置し、ユーザーはそこから選択することができます。通常のページ・フィルターとは異なり、固定選択リストは、ユーザーのオプションを開発者が指定したものに限定します。
fixedChoiceLists のデフォルト値は空ストリングで、ディメンションとそれらのメンバーに対する全アクセス権限がユーザーに与えられます。このプロパティーを使用してディメンションと特定のメンバーが指定されると、ユーザーは定義されたメンバーにしかアクセスできなくなります。例えば、2 つの地域、Central および East のみをユーザーが参照するように制限するには、PageBlox fixedChoiceLists 属性は次の例のようになります。

```
fixedChoiceLists="All Locations:Central,East"
```

開発者の初期の照会に固定選択リストのメンバーの 1 つが含まれない場合には、固定選択リストで指定されたディメンションの最上位メンバーも、初期にそのリストに表示されます。固定選択リストのメンバーの 1 つをユーザーが選択すると、最上位メンバーはそれ以降リストには表示されなくなります。

固定選択リストが PageBlox に表示されるようにするには、DataBlox の selectableSlicerDimensions プロパティー内で、そのリストにあるディメンションを指定することも忘れないでください。この例では、DataBlox (PageBlox ではない) 属性は次のようになります。

```
selectableSlicerDimensions="All Locations"
```

注: 開発者の初期の照会に、固定選択にあるディメンションのメンバーが複数含まれている場合には、固定選択リストのページ・フィルターは PageBlox に表示されません。例えば、初期の照会が次のとおりだったとします。

```
query='<SYM <ROW ("All Products") <ICHILD "All Products"
<COL(Scenario) <CHILD Scenario "2001" Central East !'/>
```

このとき、固定選択リストは PageBlox に表示されません。固定選択リストにデフォルトで表示させたいメンバー (Central または East のいずれか) を照会に組み込むだけで、この照会を成功させることができます。

moreChoicesEnabledDefault および moreChoicesEnabled プロパティーの使用

DataBlox の moreChoicesEnabledDefault プロパティーを使用すると、デフォルトでは、ページ・フィルターでユーザーが「他の選択項目 (More Choices)」オプションを選択することが可能になります。このデフォルトの機能を使用不可にするには、次に示されているように、そのプロパティーを false に設定します。

```
moreChoicesEnabledDefault="false"
```

あるいは、このプロパティーのより選択肢が多いバージョンである moreChoicesEnabled プロパティーを使用できます。このオプションでは、「他の選択項目 (More Choices)」メニュー・オプションを表示したくないディメンションを個別に指定することが必要です。

moreChoicesEnabled プロパティーの 2 つの変種の詳細については、「*開発者用リファレンス*」を参照してください。

MemberSecurityBlox を使用したメンバーのフィルタリング

Blox Logic Tag Library に組み込まれている MemberSecurityBlox を使用すると、アクセス許可を基にしてディメンション・メンバーのリストをフィルタリングできます。 MemberSecurityBlox は、指定された MemberSecurityFilter の値を基にして DataBlox の suppressNoAccess プロパティーを使用してメンバーへのアクセスを抑制します。このプロパティーは、複数のルート・メンバーを取ることができ、フィルタリング用の複数の「ディメンションとメンバー」の対を指定することを可能にします。

MemberSecurityBlox の使用例については、64 ページの『MemberSecurityBlox を使用するキューブ・メンバーのリスト』を参照してください。 MemberSecurityBlox の構文および使用法の詳細については、「*開発者用リファレンス*」の『ビジネス・ロジック Blox』および『TimeSchema DTD リファレンス』セクションを参照してください。

HTML フォーム・エレメントおよび FormBlox コンポーネントの使用

DB2 Alphablox は、プレゼンテーション Blox コンポーネント内で情報配信を構成およびフィルタリングするための多くのプロパティーを備えていますが、この機能のいくつかをこれらの Blox から取り出して Web ページ上に移動することもできます。 HTML フォーム・エレメント (選択リスト、チェック・ボックス、およびラジオ・ボタンに加えて FormBlox が含まれる) を使用して、情報にアクセスして選択を行うことができます。 Blox メニュー・バーおよびツールバーを除去し、DHTML Client API の JavaScript メソッドを使用してサーバー・サイドのロジックを呼び出す HTML フォーム・エレメントを取り込むことにより、すべての Web ユーザーが使用できる、強力でありながらも使いやすいアプリケーションを作成できます。

標準 HTML フォーム・エレメントおよび Blox クライアント API を使用する以外に、Blox Form Tag Library (FormBlox コンポーネントが含まれる) を使用して、分

析アプリケーションを作成することもできます。 FormBlox は、分析アプリケーションを作成するために必要とされる一般的に使用される機能を持つ、カスタマイズされたフォーム・エレメントを生成します。例えば、DimensionSelectFormBlox および MemberSelectFormBlox は、自動的にディメンション名およびメンバー名が取り込まれた HTML 選択リストを生成しますが、ユーザーが選択するための単純なリストを作成するためにも使用することができます。 FormBlox コンポーネントの別の利点は、ユーザーがブラウザー・セッションでページから出た後にそこに戻ってきた場合でも状態を持続および維持するという点です。

標準 HTML フォーム・エレメントと FormBlox コンポーネントの両方を使用すると、ユーザー対話の対話性を制限したり、ユーザーに公開したくないオプションをフィルターに掛けて除去することができます。使用可能な FormBlox コンポーネントの詳細については、55 ページの『第 6 章 Blox Form Tag Library』および「*開発者用リファレンス*」の『Blox Form タグ・リファレンス』セクションを参照してください。

照会の使用

データがユーザーの所に行く前にフィルタリングするための最も効果的な方法は、有効な照会ステートメントを構成することかもしれません。即時タスクに関係のあるデータのみを戻す照会を使用すると、データベース・サーバー上での実行に長い時間がかかりネットワーク・トラフィックに影響を与える、項目の多い結果セットが作成されることを避けることができます。

特定のデータ・ソースでの照会を最適化する方法を説明するのは、このガイドの範囲外です。照会ステートメントを使用したデータのフィルタリングの詳細については、ご使用のデータベース資料およびその他のリソースを確認してください。また、このガイドの 143 ページの『第 14 章 データの取り出し』には、サポートされるマルチディメンション・データベースおよびリレーショナル・データベースから情報をフィルタリングする上で役立つ照会手法についての限定された情報があります。

Blox プロパティーを使用したデータの抑制

DB2 Alphablox は、分析アプリケーションのユーザビリティーおよびパフォーマンスを向上できるいくつかのプロパティーを備えています。以下のセクションでは、DataBlox の `suppressMissingOnRows`、`suppressMissingOnColunns`、suppressZeros、`suppressDuplicates`、および `suppressNoAccess` プロパティーの使用法を簡単に説明しています。これらのプロパティーの構文および使用法の詳細については、「*開発者用リファレンス*」の『DataBlox リファレンス』のセクションを参照してください。

ゼロまたは欠落データで全体が埋まる多くのデータ行または列があるときに、データを抑制することには多くの場合意味がありますが、この情報を抑制すると、例えば、データが欠落している場合にそれを実際に検知して処置する必要があるビジネス・ユーザーには誤解を与えることがあります。ユーザーに Blox のメニュー・バーへのアクセス権限がある場合には、「データ・オプション」ダイアログでこの設定を手動で変更できますが、ユーザーはこの設定のことを知らないかもしれませんし、欠落していることを知らなかったデータを抑制解除することを考えないかもし

れません。メニュー・バーを使用できない状態でデータを抑制したい場合には、ページ上にフォーム・エレメント (チェック・ボックス、ラジオ・ボタンなど) を配置することを考慮し、この設定を制御することをユーザーに許可することにより、データの抑制を使用中であることに気付かせてください。

suppressMissingOnRows および suppressMissingOnColumns プロパティーの使用

suppressMissingOnRows および suppressMissingOnColumns プロパティーは、戻された行または列にデータがまったくない場合に、グリッドから行および列を除去します。データ・セット内の 1 つの行または列のいずれかのセルに値が入っている場合には、その行または列全体が可視になります。

この機能を使用可能にするには、DataBlox に suppressMissingOnRows 属性および suppressMissingOnColumns 属性を追加し、その値を次のように true に設定します。

```
suppressMissingOnRows="true"
suppressMissingOnColumns="true"
```

「*開発者用リファレンス*」の『DataBlox リファレンス』セクションにリストされている関連 Java メソッドを使用して、プログラマチックにこの機能を制御することもできます。

DB2 OLAP Server データ・ソースおよび Essbase データ・ソースの場合、suppressMissingOnRows または suppressMissingOnColumns が DataBlox 内でプロパティーが true に設定されて使用可能になっているときには、抑制はデータベース・サーバーおよび DB2 Alphablox 内部の両方で実行されます。 DB2 OLAP Server または Essbase を使用するときに予期すべき動作を以下に要約します。

- 初期照会が (ブックマークの代わりに) レポート・スクリプトである場合には、DB2 Alphablox は欠落データの抑制を行います。
- 照会がブックマーク設定、ドリル操作、またはピボット操作の結果である場合には、DB2 OLAP Server または Essbase サーバーに対して、欠落値を持つ行を抑制するように指示されます。

Essbase レポート・スクリプト・コマンド <SUPPRESSMISSING のみに頼るのは一般的に最適なソリューションではありません。その場合 DB2 Alphablox は、ドリル操作またはその他の操作の結果として生成される欠落データを除去しないからです。

エンド・ユーザーが DHTML クライアントを使用している場合には、Essbase <SUPPRESSMISSING コマンドを初期レポート・スクリプト・コマンドに追加しても、照会が (1000 行を超える) 項目の多い結果セットを戻す場合であっても、パフォーマンスに目立った影響はおそらくないでしょう。 DHTML クライアントは、取得する結果セットを、特定のインスタンスで表示できるものに制限して最適化します。

注: GridBlox の missingValueString プロパティーまたはその関連メソッドを使用して、値のないセルに何を表示するかを指定してください。このプロパティーは、DataBlox suppressMissing プロパティーまたは Essbase <SUPPRESSMISSING レポート・スクリプト・コマンドで行または列全体が抑制されない場合に便利です。

missingValueString プロパティーの詳細については、「*開発者用リファレンス*」の『DataBlox リファレンス』セクションを参照してください。

suppressZeros プロパティーの使用

DataBlox の suppressZeros プロパティーが true に設定されている場合には (デフォルトは false)、ゼロのみを含むすべての行または列が抑制されます。データ・セット内の 1 つの行または列のいずれかのセルにゼロ以外の値が入っている場合には、その行または列全体が表示されます。

この機能を使用可能にするには、DataBlox に suppressZeros 属性を追加し、その値を次のように true に設定します。

```
suppressZeros="true"
```

「*開発者用リファレンス*」の『DataBlox』セクションにリストされている関連 Java メソッドを使用して、プログラマチックにこの機能を制御することもできます。

suppressDuplicates プロパティーの使用

DataBlox の suppressDuplicates プロパティーは、true (デフォルト設定) に設定されると、グリッド内の行または列からすべての重複したヘッダー値を除去します。

重複したヘッダー値を抑制したくない場合には、DataBlox に suppressDuplicates 属性を追加し、その値を次のように false に設定します。

```
suppressDuplicates="false"
```

「*開発者用リファレンス*」の『DataBlox リファレンス』セクションにリストされている関連 Java メソッドを使用して、プログラマチックにこの機能を制御することもできます。

注: 初期照会における重複した DB2 OLAP Server 共有メンバーまたは Essbase 共有メンバーを抑制するには、照会ステートメント内で <SUPSHARE レポート・スクリプト・コマンドを使用してください。 DB2 DB2 OLAP Server 共有メンバーまたは Essbase 共有メンバーの抑制について詳しくは、DB2 OLAP Server 資料または Hyperion Essbase 資料を参照してください。

第 20 章 データの持続およびブックマークの設定

データおよびビューの持続性は、分析アプリケーションにおける重要な考慮事項です。アクセスを受けるデータのほとんどはデータベースに保管されていますが、DB2 Alphablox を使用して、アプリケーションの状態、ブックマーク、およびカスタム・プロパティーのデータ持続性を管理できます。セッション中にデータ値を持続させるための JavaServer Pages 手法の使用についての簡潔な説明と実例が含まれています。

DB2 Alphablox でのデータ持続性

DB2 Alphablox アプリケーションは、リソースの集合です。DB2 Alphablox は、さまざまなブックマークおよびアプリケーション状態をユーザーが保管およびリストアできるようにするために使用できる、組み込み機能を備えています。例えば、ドリル操作およびピボット操作の後、または希望するチャート・レイアウトを選択した後、ユーザーはその後再呼び出しするために現在のビューにブックマークを付けることができます。さらに、ユーザーがブラウザー・ウィンドウを閉じるかまたは現行セッションがタイムアウトになるかのいずれかの場合、開発者が定義したアプリケーション設定に応じてアプリケーション状態が保守され、セッション終了時に自動的に保管されます。

ブックマークおよび保管済みアプリケーション状態は、パブリックにまたはプライベートに保管できます。公開ブックマークおよび公開アプリケーション状態は、そのアプリケーションへのアクセス権限があるすべてのユーザー間で共有できます。Blox ユーザー・インターフェースを介したこの機能の使用についての情報は、ユーザー・ヘルプ・ページを参照してください。以下のセクションでは、ブックマークおよびアプリケーション状態についての詳細を説明し、ブックマークおよびアプリケーション状態を管理するために開発者が使用できる Blox プロパティーおよびメソッドをリストしています。

アプリケーション状態

アプリケーション状態は、DB2 Alphablox がアプリケーションに情報を保管するためのもう 1 つの方法です。ユーザーがアプリケーションにアクセスするセッションを開始すると、DB2 Alphablox はアプリケーションのインスタンスを作成します。セッションが活動状態にある限りは、このインスタンスはユーザーの現行アプリケーション・セッションの状態を保持します。それには、照会の結果セット、グリッドおよびチャートの外観、およびユーザーによるソートや他の変更などのアプリケーション・リソースの状態が含まれます。

DB2Alphablox アプリケーションの**アプリケーション状態**とは、そのアプリケーション内にあるすべての Blox の、特定の時点での状態を表します。 DB2 Alphablox アプリケーションの使用プロセス時には、DB2 Alphablox はユーザーの現行アプリケーション状態をトラッキングして維持します。この状態が、**現行アプリケーション状態**として定義されます。一方、**保管済みアプリケーション状態**は、アプリケーション状態が保管された際のアプリケーション内のすべての Blox を表します。

ブックマークは個々の Blox の状態を保管しますが、アプリケーション状態はアプリケーション全体の状態を保管します。アプリケーション状態は、保管して、必要に応じてあとでリストアできます。さらにアプリケーション状態は、アプリケーションにアクセスするすべてのユーザー間で共用するためにパブリックに保管するか、各ユーザーごとにプライベートに保管できます。

アプリケーション状態管理は DB2 Alphablox によって自動的に処理されます。以下の場合に、DB2 Alphablox はリポジトリーにアプリケーションの現行状態を自動的に保管します。

- (ブラウザーをクローズして) ユーザーがアプリケーションを終了するとき
- ユーザー・セッションのタイムアウト時 (デフォルトでは、非アクティブになってから 15 分後)

次にユーザーがアプリケーションにアクセスすると、アプリケーション定義の「**保管済みアプリケーション状態のリストア (Restore Saved Application State)**」設定が yes (デフォルトは no) に設定されている場合には、DB2 Alphablox はアプリケーションの最新の保管済み状態にリストアします。「**保管済みアプリケーション状態のリストア (Restore Saved Application State)**」が no に設定されている場合には、ユーザーは元のデフォルト・アプリケーション状態にアクセスします。ユーザーが最後に保管した状態にリストアできるよう、アプリケーション内のページ上にカスタム・ボタンを組み込むことができます。

注: セッションのタイムアウト後にユーザーがブラウザー・ウィンドウでセッションを処理しようとすると、「最新表示 (Refresh)」(または「再ロード (Reload)」) ボタンを押して DB2 Alphablox に再接続するようユーザーに指示するメッセージが表示されます。

注: DB2 Alphablox は、アプリケーション状態にデータは保管しません。保管済みアプリケーション状態がリストアされると、アプリケーションはデータベースから新しいデータを取り出します。

以下の表には、アプリケーション状態の管理に関連する RepositoryBlox メソッドがリストされています。

Javaメソッド
delete() deleteApplicationState() exists() getApplicationStateNameAndDescription() list() load() rename() renameApplicationState() restoreApplicationState() save() saveApplicationState() search()

DB2 Alphablox リポジトリー内のカスタム・プロパティー

DB2 Alphablox リポジトリーを使用すると、標準 JSP メソッドを使用して取り出したり変更したりすることが可能な、カスタム・ユーザー・プロパティー、カスタム・グループ・プロパティー、およびカスタム・アプリケーション・プロパティーを作成できます。こうしたカスタム・プロパティーをユーザー、グループ、およびアプリケーション定義内に作成してから、RepositoryBlox を使用して保管して取り出すことができます。 JavaServer Pages テクノロジーを使用すると、こうしたプロパティーの値を Java コードおよびご使用の Blox タグ属性内のランタイム式値として置換できます。カスタム・プロパティーは、DB2 Alphablox プロパティー継承階層に応じて使用可能です。

注: Blox をポータル・アプリケーションに追加する場合には、ポートレットは、ユーザー・プロファイル情報へのアクセスをポータルのインフラストラクチャーに依存しているという点を銘記してください。ですから、ユーザー情報については DB2 Alphablox を介するのではなく、Portlet API を使用してください。

カスタム・ユーザー・プロパティーの作成

カスタム・プロパティーの使用法を理解するために、ChartBlox `chartType` プロパティーを考慮します。デフォルト値は `3D Bar` ですが、金融アプリケーションのセットには、CFO は折れ線グラフを希望するかもしれません。開発者は、カスタム・プロパティーを定義して、このユーザーに異なるデフォルトを指定することができます。カスタム・プロパティー名は Blox プロパティー (`chartType`) と同じで、`Line` の値を取ります。

この例を基にして、次のステップにより必要なカスタム・ユーザー・プロパティーが作成されます。

1. DB2 Alphablox ホーム・ページから、「管理」タブを選択します。
2. 「サーバー (Server)」リンクをクリックした後、「カスタム・プロパティー (Custom Properties)」の下の「ユーザー定義 (User Definitions)」をクリックします。「ユーザー定義 (User Definitions)」ページが開きます。
3. ページの下部にある「作成 (Create)」ボタンをクリックします。「ユーザー・カスタム・プロパティーの作成 (Create User Custom Property)」ページが表示されます。
4. 以下の項目をすべて入力します。
   - プロパティー名 (Property Name): `chartType`
   - デフォルト値 (Default Value): `Line`
   - 値リスト (Value List): `Line, 3D Bar`
5. 「保管」をクリックして新規プロパティーを保管します。次にこのカスタム・プロパティーをユーザーの定義に割り当てます。
6. DB2 Alphablox ホーム・ページから、「管理」タブを選択します。
7. 「ユーザー (Users)」リンクをクリックします。「ユーザー定義 (User Definition)」ページが開き、既存のユーザー定義が表示されます。
8. 新規ユーザーを定義するには、「作成 (Create)」ボタンをクリックします。「ユーザーの作成 (Create User)」ページが表示され、「一般プロパティー」パネル

が表示されます。 (カスタム・プロパティーの値を既存のユーザー定義に割り当てるには、リストからユーザー名を選択して「編集」ボタンをクリックしてください。)

9. 「ユーザー名」、「パスワード」、および「パスワードの確認」の項目を入力 (または編集) します。
10. chartType プロパティーが「一般プロパティー」パネルの下部に表示されます。プロパティーの値が Line に設定されていることを確認してください。
11. 「保管」をクリックします。
12. 今作成したユーザーとしてログインしてプロパティーをテストします。

JavaServer Pages テクノロジーおよびデータ持続性

JavaServer Pages テクノロジーは、ユーザー・セッションが行われている間、およびセッションとセッションの間にデータ値を管理するためのいくつかのメソッドを備えています。 JavaServer Pages テクノロジーは、データ値を保管および取得できるいくつかの異なる方法を備えています。これには、URL 再書き込み、隠蔽フォーム値、request オブジェクト・メソッド、および session オブジェクト・メソッドが含まれます。 JSP ベースのアプリケーションで使用可能な手法の説明については、JavaServer Pages 資料またはその他の JSP リソースを参照してください。

次のタスクで、それらの手法の 1 つである request オブジェクトの getParameter メソッドの使用例を学びます。

URL 属性値を取得するための要求パラメーターの使用

Web ページがサブミットされると、URL アドレスはリンク・ページ内から取得する情報を受け渡すことができます。 DB2 Alphablox アプリケーションの一般的な使用方法は、ページ上で Blox ビューを使用してカスタム印刷ページを作成することです。 DB2 Alphablox URL render 属性を使用すると、次の行のようにして、新しい印刷ページを簡単に開くことができます。

```
window.open("view-print.jsp?render=printer","_blank");
```

この JavaScript メソッドを使用すると、印刷用に HTML にレンダリングされる現行の Blox ビューが表示された新しいブラウザー・ウィンドウが開きます。レンダリングされるページで HTML フォーム・エレメント (ボタン、チェック・ボックスなど) およびテキストを使用すると、こうしたすべてのエレメントやテキストが印刷可能なページに組み込まれます。これは、理想的なソリューションとは言えません。

代わりに、カスタム印刷ページを作成すると、Blox ビューからエレメントを取得して、それらをカスタム印刷ページに組み込むことができます。以下のステップは、getParameter メソッドを使用して JSP ページ間で値を受け渡す方法を示しています。

1. 印刷する Blox ビューのあるページで、カスタム印刷ページに情報を渡す URL を構成する JavaScript 関数を作成します。

以下は、URL アドレスを作成する JavaScript 関数の例です。時間、地域 (HTML 選択リストの選択値から)、レンダリング・モード、および HTML テーマを渡します。

```
function printPreview() {
  var   region=document.RegionForm.RegionSelectionList.
          options[document.RegionForm.RegionSelectionList,
          selectedIndex].text;

  var timestamp=new Date();

  var URL="passingValues-print.jsp?Region="+escape(region)+
           "&TimeStamp="+escape(timestamp.toString())+
           "&render=printer"+
           "&theme=printer";

  window.open(URL,"PrintPreviewWindow");
}
```

2. カスタム印刷ページで、URL 照会ストリングから値をキャプチャーし、必要に応じてご使用のページにこの値を組み込みます。

   次の例は、要求オブジェクト `getParameter` メソッドを使用して、カスタム印刷ページの本文でページ上に配置される時間および地域を示しています。

```
<h1>Sales for <%= request.getParameter("Region") %></h1>
<p>
<blox:display bloxRef="RegionPresentBlox"/>
</p>
<h3><%= request.getParameter("timestamp") %></h3>
```

注: Blox Sampler 実例セットの『持続およびブックマーク (Persisting and Bookmarking)』セクションの下の『ページ間での値の受け渡し (Passing Values Between Pages)』の例は、要求パラメーターの使用方法を示しています。

ブックマーク - 開発者の詳細情報

開発者として、ブックマークを扱う際に留意すべき重要な詳細情報が幾つかあります。

- ブックマークは、Blox の状態をリストアするのに使用される、プロパティー・セット (名前と値の対) の集合です。以下は、Blox プロパティーのプロパティー・セットの例です。
  - `dividerLocation = 0.25`
- ブックマークには、Blox プロパティーだけではなく、ブックマーク・プロパティーも含まれています。こうしたブックマーク・プロパティーには、以下のものが含まれます。
  - アプリケーション
  - Blox タイプ (Present、Chart、Grid など)
  - 説明
  - ブックマーク名
  - 非表示 (ブール)
  - Blox プロパティーへの参照
- ブックマークが保管されると、初期の Blox 状態とブックマーク保管時の Blox の現行状態とが**異なります。初期状態**には、デフォルトおよび定義済みタグ、タ

グ属性プロパティーが含まれています。 DB2 Alphablox リポジトリーには、初期状態から変更されたプロパティーのある Blox の、プロパティー・セットを持つブックマークのみが存在します。

- ブックマークの保管時には、アプリケーション名、Blox 名、ブックマーク・タイプ (パブリック、グループ、プライベート)、およびグループまたはユーザー名に応じて、DB2 Alphablox リポジトリーの特定ロケーションにブックマークは保管され、そのロケーションからリストアされます。
- ブックマーク照会ファイルには逐次化照会オブジェクトが含まれ、このオブジェクトはデータのないグリッド結果セット (つまり、メンバー・オブジェクトのタプル) によく似ています。これは、初期の Blox 状態で定義された照会を表す、テキスト形式の照会ではありません。
- ブックマーク・フィルターを使用すると、ブックマークに対してコンテキスト・フィルターを作成できます。このフィルターは、ブックマーク・メニュー項目に基づいてビューを設定し、ページ上に単一の Blox を持つアプリケーションを作成するのに役立ちます。さらに、共有ブックマークを使用すると、『発行済み』およびセルフサービス・アプリケーションを作成できます。

共通 Blox プロパティー、BookmarksBlox タグおよびタグ属性、さらには使用可能なサーバー・サイドの API の詳細については、「*開発者用リファレンス*」の『共通 Blox リファレンス』セクションおよび『BookmarksBlox リファレンス』セクションで取り上げられています。

すべてのブックマークの数の取得

この例では、以下の点を例示します。

- BookmarksBlox の使用法、および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に対するアクセスを取得する `listBookmarks()` メソッド。`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って、ブックマークの合計数を取得する方法。

```
<%@ 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).
```

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可視性に基づいてブックマークにアクセスし、そのプロパティー・セットに関する情報を取得する方法を示しています。特に示されているのは、以下の事柄です。

- 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 の使用方法

- Bookmark オブジェクトの getName()、getVisibility()、getDescription()、getBloxType()、および getBinding() メソッドの使用方法
- 個々のプロパティー (それぞれのネストされた Blox ごと) にアクセスするための Bookmark オブジェクトの getBookmarkProperties() メソッドの使用方法

生成される出力は、次のようになります。

予想されるブックマークは、次のとおりです。

1. このブックマークのリポジトリー JNDI バインディングは:
2. ブックマーク名: q2fy02WestSales
3. このブックマークが保管された Blox のタイプ: グリッド
4. ブックマークの説明: Q2 West Sales
5. ブックマーク可視性: プライベート
6. ブックマークには、リポジトリー内の Blox プロパティーが含まれます。
7. ブックマークに保管される Blox プロパティーのタイプ:
   - グリッド
   - データ

```
<%@ 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>
```

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、サーバー・サイドのイベント・フィルターを使用して、bookmarkLoad イベントの起動時にカスタム・タスクを実行する (この例では、メッセージをコンソールに追加する) 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。

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

2. それから、イベントの起動時に呼び出されることになる、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独自のクラスを作成します。

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

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

<%!
public class LoadFilter implements BookmarkLoadFilter {
   public void bookmarkLoad( BookmarkLoadEvent ble )
        throws Exception {
      Bookmark bookmark = ble.getBookmark();
      String name = bookmark.getName();
      System.out.println("A bookmark called " + name + " is
        loaded.");
      }
   }
%>

<body>
<blox:present id="myPresent" >
   <blox:data dataSourceName="TBC"
      query="<Row(Market) <ICHILD Market <Column(Year) Year !"/>
<%
   myPresent.addEventFilter(new LoadFilter());
%>
</blox:present>

</body>
</html>
```

BookmarksBlox API を使用したアプリケーションのカスタマイズ

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。例えば、時系列レポートを作成したり、ブックマークと共に保管されているデータ照会を動的に変更することにより現四半期のデータを常時取り出すレポートを作成したりできます。カスタム・ブックマーク・プロパティーを使用すると、レポート・レイアウトの各ユーザーの選択を保管したり、開発者が使用しているセキュリティーをインプリメントしたりできます。データ・ソース内のメンバー名またはアウトラインの変更がある場合には、ブックマークと共に保管されている照会を変更できます。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。

Blox Sampler に含まれているブックマーク・カスタマイズ例のうち、興味深い数例を以下に示します。

ブックマーク・イベントの使用

DB2 Alphablox アプリケーション内で使用される `load`、`save`、`rename`、および `delete` という 4 つのブックマーク・イベントが使用可能です。類似したタイプの複数のイベントを含め、これらのイベントの任意の組み合わせを Blox に登録でき

ます。登録されると、これらのイベントは実際のプロセスが開始される前に呼び出され、ブックマークの動作をカスタマイズする機会が与えられます。

典型的なイベントは、次のようなものです。

```
public class LoadFilter implements BookmarkLoadFilter {
    public void bookmarkLoad( BookmarkLoadEvent ble ) throws Exception {
        Bookmark bookmark = ble.getBookmark();
        String name = bookmark.getName();
       System.out.println("Bookmark " + name + " applied");
    }
}
```

この例では、イベントは、ロードされるブックマークの名前を取得し、その後その名前をコンソールに表示します。

登録済みイベントの使用

イベントの登録は、次のように Blox タグ内で行われます。

```
<blox:present id="myPresent3" >
   <blox:data
      dataSourceName="QCC-Essbase"
      query="!" />
<%
   myPresent3.addEventFilter(new LoadFilter());
   myPresent3.addEventFilter(new SaveFilter());
   myPresent3.addEventFilter(new RenameFilter());
   myPresent3.addEventFilter(new DeleteFilter());
%>
</blox:present>
```

上記の Blox タグは、4 つのブックマーク・イベントすべてを登録します。

ブックマークでの動的照会の使用

新規ブックマーク API を使用すると、ブックマークと共に保管した元の照会とは異なる照会を実行してその結果を結果セットとして使用するように、サーバーに指示することができます。

Microsoft Analysis Services MDX 照会ステートメントを使用する次の例では、ブックマークのロード時に使用される、テキスト形式のパラメーター化照会を保管するためにブックマークを変更する方法を示します。ブックマークにおける異なるテキスト形式の照会の強制使用には、照会をブックマーク・プロパティーと共に保管し、textualQueryEnabled プロパティーを true に設定することが含まれます。

ブックマークの save イベントで、パラメーター化照会を保管するために次のようにできます。

```
// Parameterized query (NOTE: :year and :quarter)

final String PARAM_QUERY = "SELECT {[Products].[Category].[All Products],
   [Products].[Category].[All Products].children} ON ROWS,
   {[Time].[Calendar].[All Time Periods].[:year],
   [Time].[Calendar].[All Time Periods].[:year].[:quarter]}
   ON COLUMNS FROM [QCC]";

// get the Bookmark Object from the BookmarkSaveEvent
```

```
Bookmark bookmark = bse.getBookmark();

// Find DataBlox properties for this bookmark

BookmarkProperties data =
   bookmark.getBookmarkPropertiesByType(Bookmark.DATA_BLOX_TYPE);

// If DataBlox properties not found in existing property set, create

if (data == null) {
  data = bookmark.createBookmarkProperties(Bookmark.DATA_BLOX_TYPE);
  }

// Set textualQueryEnabled to true, saving the query above to bookmark

  data.setProperty("textualQueryEnabled", true);
  data.setProperty("query", PARAM_QUERY);
```

ブックマークがロードされるとき、ブックマークの load イベントを使用して、パラメーターをユーザーにより指定された関連情報に置き換えることができます。例えば次のようになります。

```
// get the Bookmark Object from the BookmarkLoadEvent
Bookmark bookmark = ble.getBookmark();

// find a DataBlox properties for this bookmark

BookmarkProperties data =
   bookmark.getBookmarkPropertiesByType(Bookmark.DATA_BLOX_TYPE);

if (data != null) {

   // Get the parameterized query from the bookmark
   String query = data.getProperty("query");

// Replace the parameters with real information

// NOTE: replaceText simply replaces any references to the 2nd argument
// with the contents of the third argument.

   query = replaceText(query, ":year", "2002");
   query = replaceText(query, ":quarter", "Qtr2");

// set the new un-parameterized query

data.setProperty("query", query);
}
```

ブックマークがロードされるとき、パラメーターは 2002 および Qtr2 に交換され、照会が実行されます。

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。

ユーザー管理のために 5 つのブックマークがありました。

ブックマークは次のとおりです。

- 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>
```

ブックマークのロード時におけるテキスト形式の DB2 OLAP Server 逐次化照会または Essbase 逐次化照会の取得

この例では、ブックマークからテキスト形式で逐次化照会を取得する方法を示します (この照会は、DataBlox での照会と同じではありません)。この例は、DB2 OLAP Server データ・ソースおよび Essbase データ・ソースのみに適用されるという点に注意してください。 Microsoft Analysis Services を参照するには、照会を自分で保管する必要があります。

1. DataBlox の `textualQueryEnabled` プロパティーを `true` に設定します。

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

2. ブックマークのロード時にカスタム・アクションを起動するために、サーバー・サイドのイベント・フィルター、`BookmarkLoadFilter` が使用されます。サーバー・サイドのイベント・フィルターの例については、286 ページの『リモート PDF プロセッサーの使用』を参照してください。
3. ブックマークがロードされるとき、テキスト形式の逐次化照会を取得します。

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" %>

<%@ taglib uri="bloxtld" prefix="blox"%>
<html>
<head>
<blox:header/>
<%!
  public class LoadFilter implements BookmarkLoadFilter
  {
   public void bookmarkLoad( BookmarkLoadEvent ble ) throws Exception
   {
    Bookmark bookmark = ble.getBookmark();
     SerializedQuery sq = bookmark.getSerializedQuery();
     SerializedTextualQuery stq = null;
      SerializedMDBQuery smq = null;
      String query = null;

    if( sq instanceof SerializedTextualQuery )
    {
     stq = (SerializedTextualQuery)sq;
      query = stq.getQuery();
      }
       else if( sq instanceof SerializedMDBQuery )
        {
          smq = (SerializedMDBQuery)sq;
           query = smq.generateQuery();
          }

   System.out.println("query=" + query);
   }
}
%>
<body>
<blox:present id="myPresent"
  width="800"
  height="600">
  <blox:data
    dataSourceName="QCC-Essbase"
    query='<ROW ("All Locations") Central East West
       <COLUMN ("All Time Periods") 2001 !"
    useAliases="true"
    textualQueryEnabled="true" />
<%
  myPresent.addEventFilter(new LoadFilter());
%>
</blox:present>
</body>
</html>
```

カスタム・プロパティーを使用してアクセスを制限する

カスタム・プロパティーは、既存のブックマークを機能拡張したものです。ブックマークに追加のキー/値情報を置いて、ブックマークの load イベント時にブックマークがロードされるときに使用できるようになりました。

ブックマークの save イベントで、カスタム・プロパティーをブックマークに追加できます。例えば、次のようにします。

```
// get the Bookmark Object from the BookmarkSaveEvent
Bookmark bookmark = bse.getBookmark();

// add username of bookmark owner as a custom property
bookmark.setCustomProperty("Owner", "Admin");
```

注: BookmarkSaveEvent クラスのためにコンストラクターを作成したりその他のブックマーク・イベントを作成したりして、所有者名などのパラメーターを取得することができます。

ブックマークがロードされるとき、次のようにブックマークの load イベントを使用して、カスタム・プロパティーを取得して所有者が一致するかを調べることができます。

```
// get the Bookmark Object from the BookmarkLoadEvent
Bookmark bookmark = ble.getBookmark();

// get the owner custom property
String owner = bookmark.getCustomProperty("Owner");

// compare this user and the owner
if (!owner.equalsIgnoreCase(currentUser)) {

    // if user and owner do not match, stop bookmark load
    ble.cancelEvent();
}

You can also do the same to stop the deletion of a bookmark
  that the current user does not own:

// get the Bookmark Object from the BookmarkDeleteEvent
Bookmark bookmark = bde.getBookmark();

// get the owner custom property
String owner = bookmark.getCustomProperty("Owner");

// compare this user and the owner
if (!owner.equalsIgnoreCase(currentUser)) {
    // if user and owner don't match, stop bookmark delete
    bde.cancelEvent();
}
```

第 21 章 ビューの配布

分析ビューを E メールおよびブックマークによって共有できます。

分析の多くは事務所や個人用オフィス内に腰掛けて 1 人で行われますが、分析情報および結果は、重役、同僚、および顧客を含む他の人々とビジネスで共有されることがよくあります。インターネットや Web ブラウザーが広く存在するおかげで、DB2 Alphablox アプリケーションは世界中の至るところの事務所や会社の他の人々と共有することができます。以下のトピックでは、情報の配布および共有のための E メール、ブックマーク、および印刷メソッドについて説明します。

E メール Bean を使用したメール・リンクの作成

データの静的ビューを 1 人以上の E メール受信者に E メール送信するために E メール Bean を使用できます。この Bean およびサポート JSP ファイルとイメージのセットが Application Studio の下にある E メールの例で提供されています。この例で中核となるファイルは `EmailBean.class` ファイルで、これをアプリケーションに組み込む必要があります。

この Bean を使用するには、SMTP サーバーが DB2 Alphablox に指定されている必要があります。「管理」タブの「システム」リンクの下にある「DB2 Alphablox 管理ページ (DB2 Alphablox Admin Pages)」からこのことを行えます。

関係するステップの一般的な概要を以下に示します。アプリケーションにおけるファイルの構成およびカスタマイズについての詳細な段階的指示については、実例をご覧ください。

1. 最初のステップには、Java クラス・ファイルをアプリケーションにコピーすることが含まれます。特に、`EmailBean.class` ファイルおよび他の 2 つのサポート・クラス・ファイル (`HTMLFileParser.class` および `HTMLFile.class`) を、アプリケーションの `WEB-INF\classes\alphablox\` ディレクトリーにコピーする必要があります。すべての Java クラス、サーブレット、Bean、またはその他のユーティリティー・クラスは `WEB-INF\classes\` になければなりません。この事例では、`classes\` の下に `alphablox¥` というサブディレクトリーを作成します。
2. 以下のステップには、以下のファイルをアプリケーション・ディレクトリーにコピーすることが含まれます。
   - `emailSend.jsp`
   - `emailError.jsp`
   - `emailTemplate.jsp`
   - `emailDialog.html`

   各ファイルの目的は、このセクションの最後にある表にリストされています。`emailError.jsp`、`emailTemplate.jsp`、および`emailDialog.html` を変更またはカスタマイズしたいと思うかもしれません。表には提案されている変更方法も含まれています。

3. 1 つのスタイル・シートに加えて、インプリメンテーション例の一部となっているイメージがいくつかあります。これらのファイルも、変更したり使用したいと思うかもしれません。これらのイメージを使用する場合には、アプリケーション・ディレクトリーのイメージ・サブディレクトリーにコピーしてください。スタイルシート (styles1.css) は、アプリケーション・ディレクトリーに直接コピーできます。このセクションの最後にある表を参照してください。
4. 例には emailExample.jsp というファイルが含まれています。 E メール機能をアプリケーションに取り込む方法の例として、このファイルを使用できます。emailExample.jsp では、openEmailDialog() という JavaScript 関数が定義されています。この関数は、E メール・ダイアログを呼び出します。実例にあるボタンがクリックされたときに openEmailDialog 関数が呼び出されるように、コードも追加されています。

ファイル	説明	変更
emailExample.jsp	以下を含む JSP ファイルです。 • E メール送信されるユーザー・インターフェース Blox • E メール機能を起動する E メール・リンクまたはボタン	emailDialog.html を別個のサイズ変更されたブラウザー・ウィンドウに表示する JavaScript コードのブロックを、このファイルから、Blox を含む JSP ファイルにコピーします。 E メール・リンクまたはボタンに、コピーされた JavaScript 関数を呼び出す指定をします。
emailDialog.html	emailExample.jsp によって呼び出される HTML ファイル。差出人、宛先、件名、および E メール・メッセージの本文を記入するフォームを含んでいます。 フォームのサブミット時に、emailSend.jsp ファイルが、フォーム・ポストより渡されるすべてのパラメーターを付けて呼び出されます。	そのまま使用することもできますし、アプリケーションに合わせてタイトル、ロゴ、またはスタイル・シート参照を変更することもできます。
emailSend.jsp	E メールを送信するために E メール Bean とのインターフェースをとる JSP ファイル。	このファイルは変更しないでください。
emailTemplate.jsp	E メールが送信されたことをユーザーに通知する戻されたページ。	そのまま使用することもできますし、アプリケーションに合わせてタイトルまたはテキストを変更することもできます。
emailError.jsp	emailSend.jsp のエラー・ページ。 E メール送信の試行がうまくいかない場合には、このページにエラー情報が表示されます。	そのまま使用することもできますし、ご使用の環境に合わせてカスタマイズすることもできます。

ファイル	説明	変更
emailBlox.gif	emailExample.jsp で E メール・アイコンとして使用される「メールボックス」イメージ。	そのまま使用することもできますし、変更することもできます。
required.gif	E メール・ダイアログで必要フィールドを示す小さい赤色の矢印。	そのまま使用することもできますし、変更することもできます。
gridlogo-sm.gif	E メール・ダイアログの「E メールの送信」ボタンの左に表示される Alphablox ロゴ。	それを使用することもできますし、独自のイメージに置き換えることもできます。
grid-bg.gif	E メール・ダイアログの背景を形成するためにタイル表示されるイメージ。	それを使用することもできますし、独自のイメージに置き換えることもできます。
style1.css	emailDialog.html により使用されるスタイル・シート。	そのまま使用することもできますし、変更することもできます。

ブックマーク

ブックマークは、最新のデータを持つ Blox ビューのインスタンスを、定義されたグループ内の他の人との間で、またはパブリックに (アプリケーション・アクセス権限を持つ他の人と) 共用するために使用することができます。ブックマークを使用すると、アナリストおよび管理者がすぐにデータのカスタマイズされたビューを共用できるようになるため、カスタム・アプリケーション・ビューを作成する時間を開発者が取れるようになるまで待つ必要はありません。むしろ、ビューにブックマークを設定し、それを他の人と共用することにより、グループのすべてのメンバーが情報を共用できます。

ブックマークは、完全にカスタマイズされたビューとしてアプリケーションに追加される可能性がある保管されるビューをグループで共用するためにも使用できます。

注: ブックマークが設定されたビューは、数カ月後に無効になる可能性があるメンバー名の使用を維持するので、ビューを長期にわたって使用するためにブックマークを使用することはお勧めしません。さらに、新規メンバーをデータ・ソースに追加しても、ブックマークを変更しなければブックマーク・ビューには反映されない可能性があります。

注: ときおり、ブックマークに実際に保管されているものをユーザーが誤解しているため、ブックマークについて心配することがあります。ブックマークが保管されるとき、データは格納されません。ブックマーク・ビューが開くたびに、適切な照会がサーバーに再サブミットされ、最新のデータが検索されます。さらに、ブックマークが設定されたビューから、データベース・セキュリティーが遮断する情報に他の人がアクセスすることはできません。

分析アプリケーションの開発において使用できるブックマーク機能について詳しくは、 280 ページの『<blox:pdfReport> タグを使用したカスタム PDF レポートのプロパティー』を参照してください。ブックマークに関連した使用可能なプロパティーおよびメソッドの詳細については、「*開発者用リファレンス*」の『共通 Blox リファレンス』セクションおよび『BookmarksBlox リファレンス』セクションを参照してください。

印刷

DB2 Alphablox アプリケーションの利点の 1 つは、ユーザーに提示されるデータが即時に使用可能になることです。データ・ソースが更新されるときにユーザーに提示されるデータも更新され、印刷したり会社のメールで配布したりせずに共用することができます。しかし、ユーザーが他の人と共用するために印刷したいと思うことも必ずあります。印刷および PDF レンダリングにより効果的に分析ビューを配信する方法について学ぶには、このガイドの他のセクションを参照してください。

- 179 ページの『プリンター・フォーマット (render=printer)』
- 180 ページの『PDF フォーマット (render=pdf)』
- 181 ページの『Blox 出力の印刷』
- 276 ページの『PDF にエクスポート』

第 22 章 データのエクスポート

データのエクスポートは、スプレッドシートまたは他のフォーマットにデータを出力して、共用、アーカイブ、またはさらに計算を行えるようにするための 1 つの方法です。このセクションでは、Blox ビューのデータの Microsoft Excel や PDF、または XML 形式でのエクスポートをサポートするアプリケーションを作成する方法を説明します。

Excel へのデータのエクスポート

デフォルトでは、PresentBlox または GridBlox のツールバーに「**Excel にエクスポート**」ボタンが組み込まれ、メニュー・バーには「**ファイル**」 → 「**Excel にエクスポート**」オプションが組み込まれます。このオプションを使用すると、ユーザーは現在のビューのデータを通常の Excel フォーマットにエクスポートすることができます。

この「Excel にエクスポート」オプションを使用するには、次のようにします。

- ユーザーの環境に Microsoft Office 2000 または Office XP が必要です。
- マクロを有効にするようプロンプトが出された場合は、そうする必要があります。マクロが有効になっていない場合には、グリッド・データだけが表示されます。チャートは生成されません。

ユーザーが「**Excel にエクスポート**」ボタンをクリックすると、ダイアログがオープンし、テンプレートを選択するよう求められます。DB2 Alphablox には、「**デフォルト (Default)**」と「**Use Chart Data**」という 2 つのテンプレートが用意されています。独自にテンプレートを用意して、ダイアログのテンプレート選択リストからそのテンプレートを選択できるようにすることができます。ユーザーは、いずれかのテンプレートを選択することもできますし、どのテンプレートも使用しないという選択をすることもできます。

Excel 用デフォルト・テンプレート

DB2 Alphablox には、デフォルトで 「**デフォルト (Default)**」と「**Use Chart Data**」という 2 つのテンプレートが用意されています。

以下の表で、各テンプレートの動作と利点を説明します。どちらのテンプレートも使用しないという選択も可能です。その場合の動作と利点についても説明します。

テンプレート	動作	利点
デフォルト (Default)	グリッドのデータを使用し、マクロを使って Excel にチャートを生成します。ユーザーが Excel のグリッド・データを変更すると、チャートは自動的に更新されます。ただし、Excel で生成されるチャートは DB2 Alphablox で表示されるものとはいくらか異なる場合があります。 **注:** 独立型 ChartBlox 内のチャートは、そのチャートを生成するために使用できるグリッド・データがないため、エクスポートできません。	チャートは常にグリッドと同期します。
Use Chart Data	チャート・データの **chartData** ワークシートを別個に作成し、そのデータからチャートを生成します。Excel のグリッド・データが変更されても、ユーザーがその別個の **chartData** ワークシートのデータを変更しない限り、チャートは自動的には更新されません。	Excel で生成されるチャートは、DB2 Alphablox のチャートのデータを使用して生成されるので、DB2 Alphablox で表示されるものと同様です。
テンプレートなし	戻されるページの MIME タイプが `application/vnd.ms-excel` に設定されます。`application/vnd.ms-excel` は Microsoft Excel の標準 MIME タイプです。ブラウザーはこの MIME タイプを判別して Excel アプリケーションを起動し、ページをロードします。ページ自体はネイティブの `xls` フォーマットではありません。 **注:** チャートはエクスポートできません。独立したチャートと PresentBlox にネストされたチャートのどちらの場合も同様です。	組み込みマクロはありません。

独自にテンプレートを作成して、ユーザーがデータを Excel にエクスポートするときにテンプレート選択リストからそのテンプレートを選択できるようにするには、『カスタム Excel テンプレートの作成』を参照してください。

カスタム Excel テンプレートの作成

Excel にデータをエクスポートするためのテンプレートを独自に作成することができます。独自のテンプレートを作成するには、DB2 Alphablox が提供するプロパティーを使用してマクロを作成します。たとえば、列ヘッダー・スタイルやデフォルト・セル・スタイルをカスタマイズできます。カスタム Excel テンプレートを作成するには、Microsoft Visual Basic for Applications の知識が必要です。

「Excel にエクスポート」オプション用のテンプレートは、DB2 Alphablox Repository の templates ディレクトリーに保管されます。テンプレートごとに別個のフォルダーに保管する必要があります。templates ディレクトリーは以下のような構造になっています。

ファイルまたはディレクトリー	説明
templates.xml	デフォルトで選択されるテンプレートを指定する XML ファイル。
template_dir	各テンプレートは必ずディレクトリー内に置かれている必要があります。ディレクトリー内には、Excel XLT テンプレート・ファイルと template.xml ファイルの 2 つのファイルがあるはずです。 template.xml ファイルは次の 2 つの情報を記述します。 • ユーザー・インターフェースのテンプレート選択リストに表示される、テンプレートの表示名。 • 同じディレクトリーにある、このテンプレート用に使用する XLT ファイルの名前。

カスタム・テンプレートを作成するには、次のようにします。

1. リポジトリーの templates ディレクトリーの下にディレクトリーを作成します。
2. 独自のテンプレートを作成します。DB2 Alphablox には、Excel マクロで使用できるプロパティーのセットがあります。提供されているこれらのプロパティーは、提供されている 2 つのテンプレートで使用されています。これら提供されている 2 つのテンプレートを手本にして、独自のテンプレートを作成することができます。プロパティーのリストについては、 274 ページの『Excel テンプレートのプロパティー』を参照してください。
3. 作成した Excel テンプレート (.XLT ファイル) を、DB2 Alphablox Repository の templates ディレクトリーの下の自分のディレクトリーに保管します。
4. 既存の default または useChartData テンプレート・ディレクトリーから template.xml ファイルを自分のディレクトリーにコピーして、自分のディレクトリー用に template.xml ファイルを作成します。その結果、テンプレート・フォルダーにはテンプレート XLT ファイルと template.xml の 2 つのファイルが含まれるようになります。
5. 先程テンプレート・フォルダーにコピーした template.xml ファイルを変更します。
   a. `<displayName>` タグにテンプレート表示名を指定します。たとえば、`<displayName>My Custom Template<displayName>` のようにします。
   b. このテンプレート用の XLT ファイルを `<file>` タグに指定します。たとえば、`<file>myCustomTemplate.xlt</file>` のようにします。

これで、「PDF にエクスポート」ダイアログのテンプレート選択ドロップダウン・リストに自分で作成したテンプレートが表示されるようになります。提供されているオンライン・ヘルプを変更して、作成したカスタム・テンプレートに関する情報を含めるには、 296 ページの『カスタム・ユーザー・ヘルプの作成』を参照してください。作成したテンプレートをデフォルトで選択されるテンプレートとして設定するには、 274 ページの『Excel へのエクスポートに使用するデフォルト・テンプレートの設定』を参照してください。

Excel へのエクスポートに使用するデフォルト・テンプレートの設定

ユーザーがデータを Excel にエクスポートするときに選択されるデフォルト・テンプレートは、「**デフォルト**」です。これは、DB2 Alphablox Repository の templates ディレクトリーの `templates.xml` ファイルに指定されています。

- My Template という名前のカスタム・テンプレートをデフォルトに設定するには、`<default>` タグのテンプレート名を次のように変更します。

  ```
  <default>My Template</default>
  ```

- テンプレートなしをデフォルトとして設定するには、`<default>` タグのテンプレート名を次のように変更します。

  ```
  <default>noTemplate</default>
  ```

`templates.xml` ファイルに `<default>` タグがない場合、またはこのファイル自体がない場合は、この templates ディレクトリー内で XLT ファイルを含む最初のディレクトリーが使用されます。

Excel テンプレートのプロパティー

次の表に、Excel テンプレートで使用される DB2 Alphablox が提供するプロパティーをリストします。これらのプロパティーとその値は、Excel で ″properties.″ という名前の別個のワークシートにエクスポートされます。

プロパティー名	設定者	説明
`WorkSheet`	アプリケーション開発者	DB2 Alphablox がデータをエクスポートするシート索引。デフォルト値は 0 です。
`StartCell`	アプリケーション開発者	DB2 Alphablox がエクスポート処理を開始するセルの場所。デフォルト値は A1 です。
`EndCell`	DB2 Alphablox	DB2 Alphablox が書き込む最後のセル。このプロパティーはチャートの境界を設定するために使用できます。
`ColumnHeaderStyle`	アプリケーション開発者	列ヘッダーに使用するスタイル。このセルの書式設定、色、フォント、および配置が、エクスポートされるデータの列ヘッダーのスタイルとして設定されます。
`RowHeaderStyle`	アプリケーション開発者	行ヘッダーに使用するスタイル。このセルの書式設定、色、フォント、および配置が、エクスポートされるデータの行ヘッダーのスタイルとして設定されます。
`DefaultCellStyle`	アプリケーション開発者	セルに使用するスタイル。このセルの書式設定、色、フォント、および配置が、エクスポートされるデータのセルのスタイルとして設定されます。ただし、セルに別の書式設定やセル・アラートが設定されている場合を除きます。

プロパティー名	設定者	説明
ExportChartData	アプリケーション開発者	このプロパティーは、DB2 Alphablox に対して、チャートに Alphablox チャート・データから直接エクスポートされたデータを使用するか、それともグリッドのデータを使用するかを指示します。通常、DB2 Alphablox チャートはグリッド・データのサブセットです。Excel チャートの外観を DB2 Alphablox チャートのようにするには、このプロパティーを true に設定します。ただし、チャート・データは別個のワークシートにエクスポートされます。Excel グリッドのデータが変更されても、チャートには影響しません。
ChartEndCell	DB2 Alphablox	ExportChartData が true に設定されている場合に、このプロパティーはチャート・データの最後のセルをマークし、チャートの境界の設定に使用されます。
ChartTitle	DB2 Alphablox	チャートのタイトル。
ChartFootnote	DB2 Alphablox	チャートの脚注。
ChartType	DB2 Alphablox	DB2 Alphablox チャート・タイプ。このプロパティーは、DB2 Alphablox チャート・タイプから Excel チャート・タイプにマップするマクロで使用されます。
ChartY1AxisTitle	DB2 Alphablox	DB2 Alphablox でタイトルがチャート・プロパティーとして設定されている場合の y1-axis のタイトル。
ChartY2AxisTitle	DB2 Alphablox	DB2 Alphablox でタイトルがチャート・プロパティーとして設定されている場合の y2-axis のタイトル。
ChartXAxisTitle	DB2 Alphablox	DB2 Alphablox でタイトルがチャート・プロパティーとして設定されている場合の x-axis のタイトル。
ChartO1AxisTitle	DB2 Alphablox	DB2 Alphablox でタイトルがチャート・プロパティーとして設定されている場合の o1-axis のタイトル。

DB2 Alphablox から Excel へのチャート・タイプのマッピング

DB2 Alphablox で提供されているテンプレートのマクロは、DB2 Alphablox のチャートを最も近い Excel のチャート・タイプにマップします。

以下の表に、DB2 Alphablox から Excel へのチャート・タイプのマッピングを示します。

DB2 Alphablox のチャート・タイプ	Excel のチャート・タイプ
棒グラフ	棒グラフ
線グラフ	線グラフ
円グラフ	円グラフ

DB2 Alphablox のチャート・タイプ	Excel のチャート・タイプ
面グラフ	面グラフ
散布図およびバブル・チャート	散布図
レーダー・チャート、線	レーダー・チャート、線
レーダー・チャート、面	レーダー・チャート、面

それ以外のすべてのチャートは棒グラフにマップされます。3D 効果は保持されます。

PDF にエクスポート

ユーザーは、分析ビューを Blox から Adobe Acrobat PDF ファイルにエクスポートすることができます。標準の Web ベース・プリンティングと比べて、PDF レンダリング・フォーマットには以下の利点があります。

- アプリケーション開発者とユーザーの双方がレイアウトをより精密に制御できる。広幅のチャートやグリッドを、切り捨てることなく、複数のページにレンダリングできます。
- PDF ファイルを保存して、後で使用できる。
- PDF ファイルを E メールで他のユーザーに送信できる。

PDF レポートのデフォルトのユーザー・インターフェース・オプション

デフォルトでは、「PDF にエクスポート」オプションは、PresentBlox、GridBlox、および ChartBlox コンポーネントのメニュー・バーおよびツールバーで使用することができます。ユーザーがメニュー・バーで「**ファイル**」 > 「**PDF にエクスポート**」を選択するか、または Blox ツールバーで「**PDF にエクスポート**」ボタンを押した場合には、デフォルトの「**PDF レポートの作成**」ダイアログが表示されます。このダイアログ内で、ユーザーは以下の設定値を変更できます。

一般設定	オプション
方向	横長 (デフォルト) または縦長
ページ・サイズ	Letter、Legal、A3、または A4。デフォルトはクライアントのロケールによって異なります (たとえば、米国英語では Letter、フランス語では A4)。
テーマ	サーバー上で使用可能なテーマの選択リスト。デフォルト値は、サーバーのデフォルト HTML テーマ `coleman` と同じです。
ヘッダー・テキスト	空のテキスト入力フィールド
フッター・テキスト	空のテキスト入力フィールド

グリッドの設定	オプション
すべての列を収めるページ数	• グリッドが収まる横方向のページ数 • データ列が複数ページに分割される場合、行ヘッダーを繰り返すかどうか。 • データ列が複数ページに分割される場合、どの列の後に改ページを挿入するか。 指定したページ数に基づいて、列を均等に分割するにはどの列の後に改ページを挿入したらよいかが、「後ろで改ページ」パネルで自動的に提案されます。ユーザーは、「使用可能な列」パネルと「後ろで改ページ」パネルの間で列を移動して、提案されている列を変更することができます。

チャートの設定	オプション
チャート・サイズ	「**ページに合わせる**」(デフォルト) または 「**カスタマイズ**」。「**カスタマイズ**」に設定した場合、ユーザーはチャートの幅と高さをピクセル単位で指定することができます。

このダイアログをカスタマイズしたり、このダイアログが表示されない選択をすることもできます。詳しくは、280 ページの『<blox:pdfReport> タグを使用したカスタム PDF レポートのプロパティー』および 283 ページの『<blox:pdfDialogInput> タグを使用した「PDF レポートの作成」オプションのカスタマイズ』を参照してください。

グローバル・デフォルト PDF レポート・プロパティーの作成

カスタム・グローバル・デフォルト PDF レポートのプロパティーは、次のディレクトリーに置かれているオプションの PDF レポート・プロパティー・ファイル (pdfreport.properties) で定義できます。

```
<db2alphablox_dir>/repository/theme/
```

このファイルにあるすべての設定値は、デフォルトではすべての DB2 Alphablox アプリケーションで使用されます。 PDF レポート・プロパティー・ファイルの例 (example_pdfreport.properties) を、同じディレクトリーから入手できます。この例示ファイルでは、DB2 Alphablox にハードコーディングされたものと同じプロパティーを使用しています。上記の指定されたディレクトリーに pdfreport.properties ファイルを追加する場合には、このファイルはハードコーディングされた値をオーバーライドして新しいグローバル・デフォルト設定値を使用します。

デフォルト PDF レポート・プロパティー・ファイルを作成するには、例示ファイルのコピーを作成し、それを pdfreport.properties に名前変更して、そのファイルのプロパティーを自分のニーズに合うように変更します。以下のプロパティーをこのファイルで指定できます。

ヘッダー

ヘッダー (テキストとレイアウトを含む)。以下のマクロおよび XHTML に似た XML ベースのフォーマットを使用して定義します (リストの下の注を参照)。

使用可能なマクロ:

- 日付: <date/>
- 時刻: <time/>
- ページ数: <totalpages/>
- 現行ページ: <pagenumber/>
- PDF ダイアログ入力: <pdfDialogInputN/> (N は 1 から 5 の整数)

デフォルトでは、<pdfDialogInput1/> はヘッダーを定義し、<pdfDialogInput2/> はフッターを定義します。

以下に例を示します。

```
header = <table border-bottom='1px' width = '100%'><tr>
<td valign = 'middle'><img src='/AlphabloxServer/theme/i/brand.gif'/></td>
<td align = 'center' style='font: bold 30px Helvetica; color: #333333;'
valign='middle'> <span><pdfDialogInput1/></span></td>
<td align = 'right' style = 'font: 8px Helvetica; color: black;'
valign = 'top'><span/></td>
</tr></table>
```

footer フッター (テキストとレイアウトを含む)。マクロおよび XHTML に似た XML ベースのフォーマットを使用して定義します (リストの下の注を参照)。

使用可能なマクロ:

- 日付: <date/>
- 時刻: <time/>
- ページ数: <totalpages/>
- 現行ページ: <pagenumber/>
- PDF ダイアログ入力: <pdfDialogInputN/> (N は 1 から 5 の整数)。デフォルトでは、<pdfDialogInput1/> はヘッダーを定義し、<pdfDialogInput2/> はフッターを定義します。

以下に例を示します。

```
footer = <table border-top='1px' width='100%'><tr>
<td align='left' style='font: 8px Helvetica; color: black;' valign='bottom'
width='33%'> <span><date/> <time/></span></td>
<td align = 'center' style = 'font: bold 10px Helvetica; color: #333333;'
valign = 'bottom' width = '33%'><span> <pdfDialogInput2/> </span> </td>
<td valign = 'bottom' width='34%'>
<p style = 'font-size:10;align:right;valign:bottom;'>
<pagenumber/> of <totalpages/></p></td>
</tr></table>
```

headerHeight

ヘッダーの高さ。有効な単位には次のものが含まれます: 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)。指定されない場合には、ピクセル (px) が使用されます。

以下に例を示します。

```
headerHeight=50
```

footerHeight
フッターの高さ。有効な単位には次のものが含まれます: 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)。指定されない場合には、ピクセル (px) が使用されます。

以下に例を示します。

```
footerHeight=10
footerHeight=0.5in
```

margin 余白。有効な単位には次のものが含まれます: 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)。指定されない場合には、ピクセル (px) が使用されます。

以下に例を示します。

```
margin=18
```

size 用紙サイズ。用紙サイズ (A3、A4、Letter (レター)、Legal (リーガル)) および方向 (landscape (横長)、portrait (縦長)) を定義するために使用されます。有効な属性は次のとおりです: [A3 | A4 | Letter | Legal | Custom [[Portrait | Landscape] | [width | [height]]]。デフォルトのページ・サイズはロケール固有のものです。米国またはカナダでは、デフォルトは Letter で、それ以外のロケールではページ・サイズのデフォルトは A4 となります。 デフォルトの向きは Landscape です。

例:

```
size=Letter Portrait
```

```
size=A4 Landscape
```

```
size=Legal
```

```
size=Custom 15in 100mm
```

```
size=Custom 8in
```
(この事例では、デフォルトの高さが使用されます)

themeListEnabled
テーマ・リストを使用可能にします。値は true (デフォルト) または false になります。

```
themeListEnabled=true
```

pdfDialogInput1
例: `pdfDialogInput1=Header Text`

pdfDialogInput2
例: `pdfDialogInput2=Footer Text`

repeatPageFilters
先頭ページの後のページでページ・フィルターを繰り返します。

例: `repeatPageFilters=true`

theme テーマ名。DB2 Alphablox リポジトリーで使用されるテーマ名と同じです。

Example: `theme=my_own_theme`

注: ヘッダーとフッターは XHTML に似た構文で指定されますが、フォーマットは本当の XHTML ではありません。以下の制限があります。

1. はサポートされません。
2. 改行なしスペースの場合、HTML 文字 ( ) の代わりに、Unicode 文字 ( ) を使用してください。
3. CSS 略式属性は、W3C CSS 仕様に従う必要があります。

JSP タグを使用した PDF レポートのカスタマイズ

DB2 Alphablox Blox Tag Library は、JSP ページで PDF プロパティーをカスタマイズするために使用できる <blox:pdfReport> および <blox:pdfDialogInput> という 2 つのカスタム JSP タグを提供しています。 <blox:pdfReport> タグを開発者が使用すると、フッター、ヘッダー、マージン、およびページ・サイズなどのカスタム PDF レポートのプロパティーを指定することができます。入力フィールド・ラベルおよびテキスト・フィールドを「PDF レポートの作成」ダイアログに追加する指定をするために、<blox:pdfDialogInput> タグが使用されます。

<blox:pdfReport> タグを使用したカスタム PDF レポートのプロパティー

<blox:pdfReport> タグを開発者が使用すると、カスタム PDF レポートのプロパティーを、Blox レベルまたはセッション・レベルのいずれかで (ハードコーディングされた PDF レポート・プロパティーをオーバーライドすることにより) 指定することができます。単一の Blox のみに影響を与える PDF プロパティーを設定するには、PDF にレンダリングするときにそのプロパティーを適用したい Blox に、ネストされた <blox:pdfReport> タグを追加します。

同じ JSP ページ上のすべての Blox に適用される PDF プロパティーを指定するには、<blox:pdfReport> タグを JSP ページ上の Blox の外側に置くことにより、そのページ上の Blox のすべての PDF ダイアログに PDF プロパティーが適用されます。

以下の表では、<blox:pdfReport> タグで PDF プロパティーを定義するために使用できるタグ属性を説明します。

プロパティー
: **説明**

footer フッター。 XHTML タグ (表の下の注釈を参照) およびマクロを使用して定義されます。

使用可能なマクロは次のとおり。日付: <date/>時刻: <time/> ページ数: <totalpages/>現行ページ: <pagenumber/>

例:

```
footer="<table border-top='1px' width='100%'> <tr> <td align='left'
style='font: 8px Helvetica; color: black;' valign='bottom'
width='33%'> <span><date/> <time/></span></td> <td align='center'
style='font: bold 10px Helvetica; color: #333333;' valign='bottom'
width='33%'> <span> <pdfDialogInput2/> </span> </td> <td
valign='bottom' width='34%'> <p
style='font-size:10;align:right;valign:bottom;'> <pagenumber/> of
<totalpages/> </p> </td> </tr> </table>"
```

footerHeight
フッターの高さ。有効な単位には次のものが含まれます: 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)。指定されない場合には、ピクセル (px) が使用されます。

例:

```
footerHeight="10"
```

```
footerHeight="0.5in"
```

ヘッダー
ヘッダー。 XHTML タグ (表の下の注釈を参照) およびマクロを使用して定義されます。

使用可能なマクロ:

- 日付: <date/>
- 時刻: <time/>
- ページ数: <totalpages/>
- 現行ページ番号: <pagenumber/>

例: header="<table border-bottom='1px' width = '100%'> <tr> <td valign = 'middle'> </td> <td align='center' style='font: bold 30px Helvetica; color: #333333;' valign='middle'> <pdfDialogInput1/> </td> <td align='right' style='font: 8px Helvetica; color: black;' valign='top'> </td> </tr> </table>"

headerHeight
ヘッダーの高さ。有効な単位には次のものが含まれます: 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)。指定されない場合には、ピクセル (px) が使用されます。

例:

```
headerHeight="10"
```

```
headerHeight="1in"
```

margin 余白。有効な単位は次のとおりです: 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)。指定されない場合には、ピクセル (px) が使用されます。1in の値は、1 インチの余白を持つページとなります。

例:

```
margin="1in"
```

```
margin="40"
```

pageBreak
改ページの設定規則。この規則は、ディメンションとメンバーの指定のリストの後に、@ before または @ after キーワードを付けた形を取ります。たとえば、Dim1: M1, M2; Dim2: M3, M4 @ after のようにします。

上の例では、ディメンション Dim1 のメンバー M1 または M2、あるいはディメンション Dim2 のメンバー M3 または M4 が変更になった場合に、必ずデータの**後に**改ページを追加することを指定します。

ディメンションとメンバーをそれぞれ指定する場合は、セミコロンで区切ります。同じディメンションのメンバー同士は、コンマで区切ります。指定した条件のいずれかを満たすと、改ページが追加されます。キーワード `@ before` と `@ after` は大/小文字が区別されません。この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 (レター)、Legal (リーガル)) および方向 (landscape (横長)、portrait (縦長)) を定義するために使用されます。有効な属性は次のとおりです: `[ A3 | A4 | Letter | Legal | Custom [ [Portrait | Landscape] | [width | [ height]]]`。デフォルトのページ・サイズはロケール固有のものです。米国またはカナダでは、デフォルトは `Letter` で、それ以外のロケールではページ・サイズのデフォルトは `A4` となります。 デフォルトの向きは `Landscape` です。

例:

```
size="Letter Portrait"
```

```
size="A4 Landscape"
```

```
size="Legal"
```

```
size="Custom 15in 100mm"
```

`size="Custom 8in"` (この事例では、デフォルトのページ・サイズ高さが使用されます)

theme レイアウト・スタイルを定義するサーバー HTML のテーマ。値は、任意の事前定義またはカスタム DB2 Alphablox テーマとなります。

例:

```
theme="coleman"
```

themeListEnabled
テーマ・リストを使用可能にします。値は `true` (デフォルト) または `false` になります。

例:

```
themeListEnabled="false"
```

注: 以下のような XHTML タグおよび CSS の制限があります。

1. `<center>` はサポートされません。
2. 改行なしスペースの場合、XHTML 文字 (` `) の代わりに、Unicode 文字 (` `) を使用してください。
3. CSS 略式属性は、CSS 仕様に従う必要があります。

例:

```
<blox:pdfReport
   size="A3 portrait"
   margin="30mm" />

---
```

```
<blox:pdfReport
   size="Letter portrait"
   margin="0"
   theme="myTheme"
   themeListEnabled="false"/>

---

<%
 String header="<span style='color:red'>This report has
     <totalpages> pages </span>";
%>

<blox:pdfReport
header="<%=header%>"
headerHeight"50px"
footer="<%=some_xhtml_variable%>"
footerHeight"1in"
```

<blox:pdfDialogInput> タグを使用した「PDF レポートの作成」オプションのカスタマイズ

入力フィールド・ラベルおよびテキスト・フィールドを「PDF レポートの作成」ダイアログに追加する指定をするために、<blox:pdfDialogInput> タグが使用されます。それは、<blox:pdfReport> タグ内でネストされたタグとしてのみ使用できます。

以下の表では、<blox:pdfDialogInput> タグ上で使用可能なタグ属性および要旨を説明します。

index 5 つのフィールドの中のどれを定義するかを定義する 1 から 5 までの整数。

例:

index="5"

displayName

テキストのラベル。

例:

displayName="Report Header"

defaultValue

[オプション] displayName 属性により定義されるテキスト・フィールド内に表示されるデフォルトのストリング。

例:

defaultValue="2004 Revenue Report"

例:

```
<blox:pdfReport>
   <blox:pdfDialogInput index="1"
   displayName="Report Title"
   defaultValue="My Application Name" />
</blox:pdfReport>

<blox:pdfReport>
  <blox:pdfDialogInput index="1"
      displayName="Report Title"
```

```
      defaultValue="My Application Report" />
   <blox:pdfDialogInput index="2"
      displayName="Footer"
      defaultValue="My Application Report" />
</blox:pdfReport>
```

複数の Blox コンポーネントからなる PDF ファイルの作成

ページに複数の Blox を表示している Web ページ上に、すべての Blox を単一の PDF ファイルにエクスポートするオプションをユーザーに提供したい場合があるかもしれません。

以下のステップを使用して、ページ上の複数の Blox から単一の PDF ファイルを生成するためにユーザーが押す 1 つのボタンを作成できます。

1. 必要なすべての page ディレクティブおよび taglib ディレクティブをページの上部に追加します。

   ```
   <%@ taglib uri="bloxtld" prefix="blox" %>
   <%@ taglib uri="bloxuitld" prefix="bloxui" %>
   <%@ page import="com.alphablox.blox.Blox,
                    com.alphablox.blox.pdfreport.PDFReport" %>
   ```

   この例では、プレゼンテーション Blox およびそれらの Blox のネストされたタイトルを定義するために、標準 Blox Tag Library および Blox UI Tag Library が使用されます。 page ディレクティブにより、単一の PDF ファイルに複数の Blox を生成するために使用するボタンを作成するために必要な Java クラスにアクセスできるようになります。

2. [オプション] プレゼンテーション Blox 内でネストされた `<bloxui:title>` タグを使用して、個々の Blox にタイトルを追加します。

   ```
   <blox:present id="myPresentBlox"> ...
      <bloxui:title title="PresentBlox View"
         style="padding:10;font-weight:bold;"
         alignment="left" />
      ...
   </blox:present>
   ```

   この例では、`<bloxui:title>` タグは、JSP ページ上のこの PresentBlox の真上に表示されるタイトルを作成します。また、このタグは、PresentBlox タグ内でネストされるため、タイトルは PDF ファイルにも表示されます。 JSP ページに置かれるすべてのタイトルまたはその他のテキストは、PDF ファイルに表示されない点に注意してください。

3. 複数の Blox を単一の PDF ファイルへレンダリングするためのボタンを追加します。

   ```
   <blox:container id="containerName" visible='true'>
     <%
        String bloxNames="myGridBlox,myChartBlox,myPresentBlox";
        PDFReport.addButton(containerName,"buttonName",
           "Create PDF Report",bloxRequest,bloxNames);
     %>
   </blox:container>
   ```

   この例では、`bloxName` ストリングは、PDF にレンダリングされる Blox のリストを、PDF ファイルに表示される順序で定義します。

4. これらを JSP ファイルに追加すると、ユーザーは、上で定義されたプレゼンテーション Blox をすべて表示する単一の PDF ファイルを生成することができます。

PDF の保管場所とファイル名の指定

デフォルトでは、DB2 Alphablox によってアプリケーション・サーバー上で生成される PDF ファイルは、一時的にしかサーバーに保管されません。場合によっては、永続的な保管場所と固有ファイル名を指定できる必要があります。その際、ユーザーか開発者自身が、それら特定のファイルに対するリンクを持つ HTML ページを作成したり、それらの保管文書へのリンクにメールすることができます。

セッション内の Blox 用の PDF レポートを指定された場所に保管するために、com.alphablox.blox.pdfreport パッケージの PDFReport オブジェクトには `writePDFToFile(AbstractBlox[] bloxList, HttpServletRequest request, String path, Printable printJob)` メソッドがあります。

1. 以下のインポート・ステートメントを JSP ファイルに追加します:

```
<%@ page import="com.alphablox.blox.AbstractBlox"%>
<%@ page import="com.alphablox.blox.pdfreport.Printable"%>
<%@ page import="com.alphablox.blox.pdfreport.PDFReport"%>
<%@ page import="com.alphablox.blox.*"%>
```

2. 通常どおりに Blox を作成します。次のコードで作成するのは、非常に簡単な PresentBlox です。

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<blox:data id="dataBlox" dataSourceName="Qcc-Essbase"
 useAliases="true" visible="false"
    query="!">
<html>
<body>
<blox:present id="myPresentBlox"
 width="700" height="500">
 <blox:data bloxRef="dataBlox"/>
</blox:present>
```

3. `writePDFToFile()` メソッドを使用して、PDF レポートを指定された場所に書き込みます。次の例は、JSP ページが呼び出され、Blox がセッション内に作成されるときに、PDF レポートを保管するために必要なコードの主要部分を示しています。

```
<%
  BloxSession bloxSession = bloxRequest.getBloxSession();
  Printable printJob = PDFReport.getPrintable(bloxSession);
  AbstractBlox[] bloxList = new AbstractBlox[1];
  bloxList[0] =  myPresentBlox;
  PDFReport.writePDFToFile(bloxList, request, "c:\\temp\\sales.pdf", printJob);
%>
```

PDF レンダリング・エンジンにブラウザー・セッションは必要ではなく、Java コード内で上記の JSP をトリガーして、PDF レポートを指定された場所に直接保管することができます。上記のコード例を変更して、複数の Blox を PDF レポートに保管することも簡単にできます。また、com.alphablox.blox.pdfreport パッケージに提供されている Printable インターフェースのメソッドを使用して、レイアウト、フッター、ヘッダー、およびレポートの改ページ数を指定することも可能です。

別の方法として、2 つの JSP セッション属性を作成することもできます。1 つは PDF ファイル用に (PDF_FILE_NAME を使用して)、もう 1 つは PDF ファイルを格納するディレクトリー用に (PDF_DIRECTORY_NAME を使用して) 作成します。

```
<%
  session.setAttribute("PDF_DIRECTORY_NAME", "c:\\temp");
  session.setAttribute("PDF_FILE_NAME", "analysis.pdf");
%>
```

これらの属性はオプショナルであり、ご使用の特定のアプリケーションに応じて、これらのセッション属性設定の一方または両方を使用することができます。

リモート PDF プロセッサーの使用

パフォーマンス、メモリー管理、または複数の DB2 Alphablox ホストでの PDF 処理の共用のため、PDF エンジンをリモート専用サーバー上で実行することができます。リモート PDF サーバーの構成の詳細については、リモート PDF プロセッサーの使用を参照してください。

XML へのエクスポート

XML 形式にエクスポートされるデータは、アプリケーション開発者が情報を他のアプリケーションに配信するために使用したり、データのカスタム・ビューを作成するために Java、JavaScript、および JavaServer Pages テクノロジーで使用したりすることができます。以下のタスクでは、照会結果を XML 形式にエクスポートする方法を説明します。

結果セットの XML 形式へのレンダリング

アプリケーション・データ・ソースからの照会結果セットを XML 形式にレンダリングするタスクには、以下のステップが含まれます。

1. HTML ページを標準 DataBlox で定義します。

   注: DB2 Alphablox XML キューブは、明示的に定義された DataBlox の結果セットのみにアクセスできます。 PresentBlox、GridBlox、または ChartBlox の基礎となる暗黙的に定義された DataBlox の結果セットにはアクセスできません。

2. DataBlox プロパティーまたはメソッドを使用してそのデータ・ソースおよび照会ストリングを指定します。
3. アプリケーションとデータ・ソースの両方を DB2 Alphablox に定義します。
4. アプリケーションを呼び出して、次のように render 属性をアプリケーションの URL に忘れずに追加します。

   ```
   .../AppName.jsp?render=XML
   ```

   render 属性のための XML の値により DB2 Alphablox が起動して、以下の処理が実行されます。

   - DB2 Alphablox XML キューブ DTD (文書タイプ定義) へのアクセス
   - XML での DataBlox 結果セットのレンダリング (ページ上の DataBlox の置換)

- 以降の処理のために使用可能な XML 文書の作成

注: XML にレンダリングするために独立型の DataBlox を使用する場合には、Blox header タグ (<blox:header/>) はアプリケーション・ページに必要ではないため、ページが適切に表示されない可能性があります。代替措置として、render=xml URL 属性を使用する代わりに、通常の render プロパティーの値を xml に設定して、DataBlox を使用することもできます。

次のセクションでは、XML にレンダリングされた、直前のページからの結果セットの例を示します。

結果セットの XML 形式へのレンダリング: サンプル DB2 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。読み易さを考え、改行が追加されているケースがあります。

```
<?xml version="1.0" ?>

<!DOCTYPE cube SYSTEM '/AlphabloxServer/xml/dtd/cube.dtd'>

<cube>
  <bloxInfo>
    <bloxID>15</bloxID>
    <bloxName>MyDataBlox</bloxName>
    <appName>MyXMLDoc</appName>
  </bloxInfo>
  <data>
  <slicer>
    <slicerDimension name="Period">Period</slicerDimension>
    <slicerMember name="Period" gen="1"
      leaf="false">Period</slicerMember>
    </slicer>
    <slicer>
      <slicerDimension name="Accounts">Accounts
      </slicerDimension>
      <slicerMember name="Accounts" gen="1"
        leaf="false">Accounts
      </slicerMember>
    </slicer>
    <slicer>
      <slicerDimension name="Scenario">Scenario
      </slicerDimension>
      <slicerMember name="Scenario" gen="1"
        leaf="false">Scenario
      </slicerMember>
    </slicer>
    <axis name="columns" index="0">
      <dimensions>
        <dimension name="Market" index="0">Market</dimension>
      </dimensions>
    <tuple index="0">
      <member name="East" index="0" gen="2" span="1"
        spanIndex="0" leaf="false">East
      </member>
    </tuple>
    <tuple index="1">
      <member name="West" index="0" gen="2" span="1"
        spanIndex="0" leaf="false">West
      </member>
    </tuple>
    <tuple index="2">
```

```
    <member name="South" index="0" gen="2" span="1"
      spanIndex="0" leaf="false">South
    </member>
  </tuple>
  <tuple index="3">
    <member name="Market" index="0" gen="1" span="1"
      spanIndex="0" leaf="false">Market
    </member>
  </tuple>
</axis>
<axis name="rows" index="1">
  <dimensions>
    <dimension name="Product" index="0">Product
    </dimension>
  </dimensions>
  <tuple index="0">
    <member name="Audio" index="0" gen="2" span="1"
      spanIndex="0" leaf="false">Audio
    </member>
  </tuple>
  <tuple index="1">
    <member name="Visual" index="0" gen="2" span="1"
      spanIndex="0" leaf="false">Visual
    </member>
  </tuple>
  <tuple index="2">
    <member name="Product" index="0" gen="1" span="1"
      spanIndex="0" leaf="false">Product
  </member>
  </tuple>
</axis>
<cells>
  <row>
    <column>
      <cell>13438.0</cell>
    </column>
    <column>
      <cell>22488.0</cell>
    </column>
    <column>
      <cell>0.0</cell>
    </column>
    <column>
      <cell>35926.0</cell>
    </column>
</row>
<row>
    <column>
      <cell>33138.0</cell>
    </column>
    <column>
      <cell>40351.0</cell>
    </column>
    <column>
      <cell>24565.0</cell>
    </column>
    <column>
      <cell>98054.0</cell>
    </column>
</row>
<row>
    <column>
      <cell>46576.0</cell>
    </column>
    <column>
      <cell>62839.0</cell>
    </column>
```

```
      <column>
        <cell>24565.0</cell>
      </column>
      <column>
        <cell>133980.0</cell>
      </column>
    </row>
  </cells>
 </data>
</cube>
```

第 23 章 エラー処理

残念ながら、ソフトウェアの問題としてエラーは必ず発生します。とはいえ、開発者は、エラーを処理する方法を管理する能力がある程度あります。このトピックには、Blox 例外、Blox プロパティー、および Blox メソッドを使用して発生するエラーを処理する方法についての情報が含まれています。

例外

ソフトウェアにおいてエラーは発生するものですが (ないに超したことはありませんが)、エンド・ユーザーにとって重要なことは、エラーが発生した後に何が起きるかです。プログラムが作業を停止するだけでしょうか。それとも、スムーズにリカバリーするでしょうか。例外は、プログラムの通常の実行を中断するイベントです。 Java 言語は、例外をスムーズに制御された仕方で処理するために、発生する例外のキャッチまたはキャッチの試行をできるようになっています。例外全般および例外の処理方法について詳しくは、Java または JavaServer Pages の良い解説書を参照してください。

Blox Java クラスの多くは例外をスローし、Java 言語のエラー処理機能の使用を可能にしています。使用可能な Blox 例外は、DB2 Alphablox 上の次のディレクトリーに含まれている Javadoc 資料で調べることができます。

```
<db2alphablox_dir>/system/documentation/javadoc/index.html
```

ここで `<db2alphablox_dir>` は、DB2 Alphablox がインストールされているディレクトリーです。

カスタム・エラー・ページ

JSP エンジンがページのコンパイルの試行に失敗すると、エラー・メッセージおよびスタック・トレースが生成され、エンド・ユーザーに表示されます。これらのメッセージのほとんどはエンド・ユーザーにはほとんど意味を持たず、エンド・ユーザーは何が発生したかほとんど理解できません。そこで開発者は、デフォルトの標準 JSP エラー・ページを表示する代わりに、カスタム・エラー・メッセージを作成してユーザーに表示するというオプションがあります。 `errorPage` および `isErrorPage` という 2 つの page ディレクティブ属性を使用すると、JSP ページがカスタム・エラー・ページを検索する箇所を定義し、特定の JSP ページをカスタム・エラー・ページとして定義することができます。これらの属性の要旨と、独自のカスタム・エラー・ページを作成するためにそれらの属性を使用するためのステップが以下に記載されています。エラーの page ディレクティブの使用について詳しくは、JSP の基本解説書を参照してください。

errorPage 属性

page ディレクティブの `errorPage` 属性は、エラー・ページとして使用する代替ページを指定し、次のように定義されます。

```
<% page errorPage="/errorPage.jsp" %>
```

errorPage 値は、同じ Web アプリケーション内部で JSP ページを検出するための相対 URL を指定します。上記の例では、値には、指定ページの前にスラッシュ (/) が含まれています。スラッシュは必須ではありませんが、それを付けることにより、以降の URL は Web アプリケーションのルート・ディレクトリーに対して相対的なものであることをアプリケーション・サーバーに知らせます。この page ディレクティブでスラッシュを使用することにより、アプリケーション・ディレクトリー内の 1 つのロケーションにカスタム・エラー・ページを置き、JSP ファイルが複数のサブディレクトリー内に置かれていたとしてもこの同じ page ディレクティブをアプリケーションのすべての JSP ファイルで使用することができます。

isErrorPage 属性

カスタム・エラー・ページには、isErrorPage 属性を true に設定した page ディレクティブを組み込まなければなりません。 isErrorPage のブール値を true に設定した page ディレクティブは、次のようになります。

```
<%@ page isErrorPage="true" %>
```

このディレクティブは、exception 暗黙オブジェクトからの情報にページからアクセスできるようにし、ユーザーが見る情報の表示を制御できるようにします。

簡単なカスタム・エラー・ページの作成

以下のステップでは、カスタム・エラー・ページ作成のプロセスをガイドします。

1. カスタム・エラー・ページとして使用される基本 JSP ファイルを作成し、errorPage.jsp として保管します。
2. isErrorPage 属性を true に設定した page ディレクティブをページの先頭に組み込みます。以下に例を示します。

   ```
   <%@ page isErrorPage="true" %>
   <html>
   ...
   </html>
   ```

3. エラーが発生したときにエンド・ユーザーに見てほしい内容を表示するエラー・ページ本体のレイアウトを作成します。

   例えば、エラー・ページ・ヘッダーを表示して、エラーが発生したページの URL を含めたいと思うかもしれません。また、ユーザーが解釈できないと考えられるため、最上位のエラー・メッセージを表示して、例外のスタック・トレースは表示しないことにするかもしれません。

   単純な例を以下に示します。

   ```
   <%@ page isErrorPage="true" %>
   <html>
   <head>
      <title>Error Page</title>
   </head>
   <body>
   <h2>Your application has generated an error</h2>
   <h3>Please notify your help desk.</h3>
   <b>Exception:</b><br>
   <%= exception.toString() %>
   </body>
   </html>
   ```

4. カスタム・エラー・ページをテストするには、errorPage 属性値がカスタム・ページのロケーションを指すようにして、次の page ディレクティブを追加します。

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

この例では、カスタム・エラー・ページは、JSP テスト・ページと同じディレクトリー内にあります。

5. エラーを生成させてエラー・ページをテストします。

エラーを生成する 1 つの方法は、次のスクリプトレットを組み込む方法です。これにより、「ゼロ除算」実行時エラーが発生します。

```
<%
   int i = 10;
   i = i / 0;
%>
```

例: このカスタム・エラー・ページ例は、Blox Sampler 実例セットの『エラー処理 (Error Handling)』セクションに含まれています。

注: さまざまな例および Application Studio の『Basic Template』にはすべて、開発者が使用するために検討およびコピーすることができるエラー・ページが含まれています。

Blox プロパティーおよびエラー処理メソッド

エラー状態を処理するようにアプリケーションをカスタマイズするために、以下の Blox プロパティーおよび Blox メソッドが使用可能です。これらのプロパティーおよびメソッドの詳細については、「*開発者用リファレンス*」を参照してください。

noDataMessage

使用可能なデータがない場合に Blox に表示されるストリングを定義する noDataMessage プロパティーは、考えられるアプリケーション・エラーをユーザーに通知するための 1 つの手段です。 ChartBlox、GridBlox、および PresentBlox に適用されるこの共通 Blox プロパティーは、これらの Blox の 1 つがインスタンス化されたのに、データをまだ受け取っていないかまたはエラーが発生したために使用可能なデータがない場合に表示されます。デフォルトのメッセージは「使用可能なデータはありません」で、グリッドおよびチャートにはっきりと表示されます。

デフォルトの「使用可能なデータはありません」メッセージがグリッドまたはチャートに数秒以上表示されるなら、ユーザーはこれを、使用可能なデータがないことを示すエラー・メッセージとして認識するでしょう。ほとんどの場合、これは道理にかなった前提事項で、メッセージを変更する必要はありません。

時には、データ検索に予想以上の時間がかかる場合があります。これは、複雑な照会、大きなデータ・セットが戻されること、または遅い接続が原因となることがあります。このような場合には、noDataMessage ストリングを「お待ちください」またはその他の代替メッセージに変更する開発者もいます。これは、このプロパティーの道理にかなった使用法ですが、表示メッセージを変更すると、使用可能なデータが実際にはない場合にエンド・ユーザーを混乱させてしまうことがあるという点を覚えておく必要があります。使用可能なデータが実際にはない場合でも、メッセ

ージには「お待ちください」と表示されてしまう可能性があります。このメッセージの変更を考慮するとすれば、使用可能なデータがない場合にユーザーが実際には待つように指示されるという小さなリスクにくらべると、正確であることのほうが多い初期メッセージを表示することの利益のほうがまさっているかもしれません。

別の代替手段として、関連した `setNoDataMessage` メソッドをプログラマチックに使用して、発生するイベントに応じて異なるメッセージを戻す方法があります。`noDataMessage` 属性を使用するだけより作成するのは複雑ですが、このオプションを検討したいと思うかもしれません。

onErrorClearResultset

DataBlox の `onErrorClearResultSet` boolean プロパティーは、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 DataBlox から消去するかどうかを指定します。このプロパティーとその関連メソッドについて詳しくは、「*開発者用リファレンス*」の『DataBlox』セクションを参照してください。

第 24 章 ユーザー・ヘルプの追加

このセクションでは、DB2 Alphablox で作成されるアプリケーションにユーザー・ヘルプを提供することに関連した問題のいくつかを取り上げます。

アプリケーションが直観的に操作ができ、ユーザーは使用方法を理解するのに助けを必要としないことが理想です。残念ながらこうしたことはまれで、アプリケーションが複雑になればなるほど、ユーザー・ヘルプを提供する必要は高くなります。

アプリケーションにヘルプを追加することを忘れてしまうのは、アプリケーションの設計および開発におけるよくある失敗です。アプリケーションのユーザーの使用頻度が高く、熟達したユーザーで、トレーニングを受けるならば、アプリケーション全体のヘルプを追加することは重要というよりは便利になるという問題かもしれません。しかし、ユーザーがトレーニングを受けることがなかったり、臨時 (使用頻度が低い) ユーザーである場合には、ユーザー・ヘルプの提供はアプリケーションの成功における重要な決定要素となりえます。設計グループがユーザー・ヘルプを考慮し、必要であれば、ヘルプ開発のための時間と要員をアプリケーション開発サイクルのスケジュールに入れてください。

以下のセクションでは、DB2 Alphablox アプリケーションにおけるユーザー・ヘルプの可用性と動作、および設計判断の影響について説明します。

既存の DB2 Alphablox ユーザー・ヘルプの使用

アプリケーションに GridBlox、ChartBlox、または PresentBlox が含まれている場合には、ユーザーはツールバーを使用することができます。どの Blox を使用しているかにより、ツールバーの「**ヘルプ**」ボタンで、その特定の Blox についてのヘルプ・ページを持つ DB2 Alphablox ユーザー・ヘルプ・システムが開きます。例えば、PresentBlox ツールバーの「**ヘルプ**」ボタンをクリックすると、PresentBlox を説明すると共にさらに詳しいヘルプへの追加リンクを持つ「PresentBlox の使用」というタイトルのページが開きます。

ツールバーが使用できない場合には、代わりに、メニュー・バーの「**ヘルプ**」メニューから「**ヘルプ**」を選択すると、PresentBlox か、独立型の GridBlox または ChartBlox のいずれであるかによって、ユーザー・インターフェース Blox のヘルプ・ページが開きます。例えば、独立型の GridBlox では、「**ヘルプ**」メニュー・オプションを選択すると、「グリッドの使用」というタイトルのヘルプ・ページが開きます。

アプリケーションの分析ビューにツールバーを組み込まないようにすることも少なからずあるでしょう。そうした場合には、メニュー・バーを必ず使用できるようにする必要があります。デフォルトでは、これらのユーザー・インターフェース Blox でメニュー・バーを使用可能です。何かの理由でメニュー・バーおよびツールバーの両方をオフにする必要がある場合には、カスタム・ユーザー・ヘルプを備えることを考慮すると良いでしょう。

カスタム・ユーザー・ヘルプの作成

分析ビューでツールバーを使用できないようにする場合、またはユーザーにターゲット・カスタム・ユーザー・ヘルプを提供したい場合には、適切なヘルプ・リンクまたはボタンを追加することができます。特に次の状況では、カスタム・ユーザー・ヘルプを提供したいかもしれません。

- ツールバーもメニュー・バーも使用できない
- ツールバーまたはメニュー・バーがカスタム・ボタンおよびメニュー・オプションでカスタマイズされている
- Blox ビューの対話および分析を管理する組み込み Blox ユーザー・インターフェース・エレメント (ページ・フィルターやツールバー・ボタンなど) の代わりに、カスタム HTML フォーム・エレメントをページが使用する
- 用語集または他のヘルプ情報があると役立つメンバーまたはその他のラベルがビューに含まれている

ツールバーおよびメニュー・バーがカスタマイズされている場合には、既存のユーザー・ヘルプもカスタマイズする必要があるかもしれません。ヘルプ・ファイルは、次の文書ディレクトリーに置かれています。

```
<db2alphablox_dir>/system/documentation/help/dhtml/<locale>
```

ファイルを変更する前に、まずディレクトリーのコピーを作成する必要があります。このディレクトリー内のファイルは、サーバーをアップグレードするときに除去されて DB2 Alphablox ユーザー・ヘルプ・ファイルに置き換えられるという点も忘れないでください。

ツールバーおよびメニュー・バーを全部オフにする場合には、標準 HTML テクノロジーを使用してカスタム・ユーザー・ヘルプを提供することに加えて、ターゲットを絞った可視のヘルプ情報を提供する方法として、次のセクションで説明されている DB2 Alphablox 情報リンクを使用することも考慮したいと思うかもしれません。

ヘルプの情報リンクの使用

206 ページの『情報リンク』で説明されているように、Blox で使用可能な情報リンクには、ヘッダー・リンク、セル・リンク、およびセル・アラート・リンクという 3 つのタイプがあります。これらのリンク (デフォルトでは、青の円の中に白い「i」で表示される) は、行または列ヘッダーに関連する情報へのリンクまたは指定されたデータ・セルの情報へのリンクを含む、多くの目的で使用できます。これらのリンクは、例えば特定のメンバーが表すものを定義したり、特定のデータ・セルを評価する方法を定義したりして、ターゲットを絞ったユーザー・ヘルプにも使用できます。情報リンクの利点の 1 つは、よく目立ち、無視しにくいという点です。もちろん、そのことが情報リンクを含めない、あるいは少なくとも思慮深く使用する理由でもあります。

第 25 章 DB2 Alphablox FastForward での作業

DB2 Alphablox FastForward アプリケーション・フレームワークを使用すると、アプリケーション管理者 (OLAP 管理者) は、フレームワークのコピー、レポート・テンプレートの構成、および分析アプリケーションの基幹業務ユーザーへの迅速なデプロイを行うことができます。このセクションには、FastForward フレームワークの概説が記載されており、新規レポート・テンプレートを追加してアプリケーション・フレームワークをカスタマイズすることができます。

注: DB2 Alphablox FastForward は、DB2 Alphablox を WebSphere Portal Server にインストールして実行している場合には使用できません。

DB2 Alphablox FastForward の概説

DB2 Alphablox FastForward は、DB2 Alphablox にプリインストールされているサンプル・アプリケーション・フレームワークであり、開発、デプロイ、およびビジネス組織全体でのカスタム分析ビューの共用を迅速に行うために使用されます。すぐに使用可能な FastForward フレームワークは、セキュリティー、コラボレーション、カスタマイズ、および個別設定を含む共通アプリケーション・サービスを配信します。アプリケーション管理者 (通常は OLAP 管理者) は、新しいバージョンの FastForward アプリケーションを作成し、レポート・テンプレートを選択してレポート・パラメーターを構成することによりレポートをパブリッシュした後、コードをまったく見ることなく新規アプリケーションをデプロイすることができます。また、アプリケーション・フレームワークには柔軟性と拡張性があるため、JSP 開発者はそれを変更または拡張して、アプリケーション管理者が構成およびデプロイできるように新規カスタム・レポート・テンプレートを追加できます。

FastForward アプリケーション・フレームワークに組み込まれている機能には、レポート作成および分析アプリケーションに通常あるような以下のものが含まれます。

- Microsoft Excel へのエクスポート
- 印刷可能なビューの生成
- データの個人用ビューの容易な保管および共用
- 他の人へのビューの E メール送信
- 異なるビュー間の容易なナビゲーション

これらの機能を組み込むことにより、DB2 Alphablox は、このクラスのよく使用されるレポート作成および分析アプリケーションの開発を迅速に行いやすくなっています。ナビゲーション・システム、ツールバー、セキュリティー、およびレポートの保管および共用のためのメカニズムは、事前にコーディングされているため、作成する必要はありません。開発タスクと構成タスクを分離することにより、アプリケーション管理者は既存のレポート・テンプレートの構成およびデプロイに集中することができ、開発者はより挑戦となる要求に注意を集中することができます。

FastForward ユーザーの役割

DB2 Alphablox FastForward ユーザーの 3 つの主要な役割は、アプリケーション管理者の役割、テンプレート開発者の役割、およびエンド・ユーザーの役割です。これら 3 つのグループ間の密接な協同作業が、FastForward ベースのアプリケーションの成功を確かなものとします。これら 3 つの役割については、以下に要約されています。

アプリケーション管理者

アプリケーション管理者 (通常は OLAP 管理者) は、いくつかの設定を定義することにより FastForward アプリケーションの新規バージョンを作成し、使用可能なレポート・テンプレートを基にしてレポートを作成した後、ソリューションをエンド・ユーザーに迅速にデプロイすることができなければなりません。既存のレポート・テンプレートの使用ではエンド・ユーザーの必要が満たされない場合には、アプリケーション管理者はテンプレート開発者と協同して新規レポート・テンプレートを作成します。アプリケーション管理者は、OLAP データベースにおける自分の経験、Alphablox FastForward アプリケーションを管理するための資料 (「*管理者用ガイド*」を参照)、およびオンラインの「管理ヘルプ」(FastForward アプリケーションの管理タスク・モードで使用可能) を使用して作業を遂行できなければなりません。

テンプレート開発者

通常、テンプレート開発者は、既存のレポート・テンプレートを使用してアプリケーション管理者が要求されたレポートを構成することができない場合に、カスタム・レポート・テンプレートを作成する主な責任を持つ JSP 開発者です。テンプレート開発者は、アプリケーション管理者およびエンド・ユーザーと相談しながら、必要に応じて既存のレポート・テンプレートを変更したり新規のものを作成したりして新規レポート・テンプレートを作成できなければなりません。

テンプレート開発者は、自分の Web プログラミングの経験に加えて、Blox タグ・ライブラリー、サーバー・サイドの Java API、および DHTML Client API を使用して、考えられるほとんどすべてのニーズに応じてテンプレートを作成できなければなりません。開発者は、DB2 Alphablox アプリケーションおよびビューの作成に精通しているだけでなく、FastForward のユーザー・ヘルプ (ユーザー・モードの「ヘルプ」ボタンから使用可能)、管理者ヘルプ (管理者としてログインした場合に、管理タスク・モードの「ヘルプ」ボタンから使用可能)、および「*管理者用ガイド*」の『FastForward アプリケーションの管理』にも精通している必要があります。

エンド・ユーザー

エンド・ユーザー (通常、ビジネス・アナリストおよび組織内の他の基幹業務ユーザー) は、ビジネスの問題を分析するために、FastForward アプリケーションにログインしてパブリッシュされたレポートを使用できなければなりません。特定の FastForward ベースのアプリケーションで使用可能な対話性に応じて、エンド・ユーザーは、データの操作、データ階層のドリル操作、チャート・タイプの変更、コメントの追加などを行えます。特定のビジネスの質問に答えるためにビューを変更した後、保管されたレポートを後で使用するために「プライベート」タブの下に作成したり、「グループ」タブの下でアプリケーション・ユーザーの定義済みグループで共用することにより、ユーザーは現在のビューを保存できます。

それぞれのレポートごとに、レポートの上に置かれているアプリケーション・ツールバーから、ユーザーは、通常他のいくつかのオプションを使用できます。オンライン分析のためのレポートの保管のほかに、「Excel にエクスポート」オプションを使用すると、ユーザーはビューを Microsoft Excel スプレッドシートにエクスポートして、後でオフラインで分析することができます。ユーザーは「印刷プレビュー」オプションを使用して特定のビューのコピーを印刷することもできます。さらに望むなら、現行ビューへのリンクを含む E メール・メッセージを開き、コメントを追加し、それを他のアプリケーション・ユーザーに送信することができます。

必要なレポートがユーザーのアプリケーションにない場合には、通常、エンド・ユーザーはアプリケーション管理者に新規レポートを直接要求します。

Alphablox FastForward のカスタマイズ

DB2 Alphablox FastForward には、アプリケーションの作成を即時に開始できる、すぐに使用可能なサンプル・レポート・テンプレートが含まれていますが、ご使用のレポート作成および分析アプリケーションにおいて、アプリケーション管理者のためにカスタム・レポート・テンプレートを作成する必要がおそらく生じます。 DB2 Alphablox FastForward アプリケーションでの作業を開始する助けとして、このセクションでは、FastForward のアーキテクチャーおよびレポート・テンプレートの概要を取り上げます。その後、新規レポート・テンプレートの作成方法を学習します。

FastForward アプリケーションのアーキテクチャー

FastForward アプリケーションには、フレームワーク、およびそのフレームワーク内で使用されるレポート・テンプレートという 2 つの主要なコンポーネントが含まれています。 FastForward フレームワークには、JSP ページ、JavaBeans コンポーネント、およびアプリケーション・フレームワークを定義するその他の多くのファイル (アプリケーション構成、ナビゲーション、セキュリティーなどの共通アプリケーション・サービスが含まれる) が含まれます。

DB2 Alphablox FastForward サンプル・アプリケーションは、次のディレクトリーに置かれています。

```
<alphabloxDirectory>/system/ApplicationStudio/FastForward/
```

重要: このディレクトリーを削除または変更しないでください。ここにはサンプル・アプリケーションで使用されるファイルが含まれています。 DB2 Alphablox をアップグレードするときにこのディレクトリーは上書きされます。

DB2 Alphablox 管理者として (`AlphabloxAdministrator` または他の定義済み管理者役割で) ログインした場合には、「管理タスク」ボタンをクリックして新規 FastForward アプリケーションを作成した後、ダイアログ・ボックスの「作成」をクリックしてサンプル・アプリケーションの新規バージョンを作成できます。「作成」オプションをクリックすると、新規アプリケーションのコンテキスト名 (ディレクトリー名)、表示名 (「アプリケーション」ページに表示される)、簡潔なアプリケーション記述、およびアプリケーションの編集を許可されている管理者の役割を定義するようにという、ダイアログ・ウィンドウのプロンプトが出されます。

注: 新規 FastForward アプリケーションが作成されると、管理者の役割が割り当てられます。その割り当てられた役割 (デフォルトは `AlphabloxAdministrator`) のメンバーであるユーザーのみが、その特定のアプリケーションを管理できます。

「OK」をクリックした後、新規 J2EE アプリケーションが自動的に作成され、FastForward アプリケーション・フレームワークおよびサンプル・レポート・テンプレートのコピーがそのアプリケーションのディレクトリーにコピーされます。通常それらは次の場所に置かれています。

```
<alphabloxDirectory>/webapps/<applicationDirectory>
```

注: FastForward アプリケーションの構成の詳細については、「*管理者用ガイド*」の『Alphablox FastForward アプリケーションの管理』を参照してください。

FastForward アプリケーションのコピーを作成するとき、作成する Web アプリケーションに以下のディレクトリーおよびファイルが追加されます。 FastForward フレームワークに含まれるディレクトリー構造およびファイルの要約を以下に示します。

ディレクトリー
: **説明**

admin
: 管理モードで FastForward アプリケーションが使用される場合に使用されるほとんどのファイルが入っています。管理ヘルプ・ディレクトリーも含まれます。

admin/help
: アプリケーション管理固有のヘルプ・ファイル。

admin/images
: 管理モードで使用されるイメージが入っています。

help
: ユーザー・ヘルプ・ファイルが入っています。アプリケーション開発者が必要に応じてカスタマイズ可能です。

images
: ユーザー・モードで使用されるイメージ・ファイルが入っています。

templates
: アプリケーションが使用可能なレポート・テンプレートのコレクションが入っています。このディレクトリーのサブコンポーネントについては、以下の『レポート・テンプレート』セクションに説明されています。

WEB-INF
: アプリケーションを実行するために必要なファイルが含まれる標準 J2EE アプリケーション・ディレクトリー。

WEB-INF/classes
: アプリケーションで使用される JavaBeans コンポーネントのための、コンパイルされた Java クラス・ファイルが入っています。

WEB-INF/src
: JavaBeans コンポーネントのソース・ファイル。必要に応じてカスタマイズまたは拡張が可能です。

WEB-INF/tlds
Blox タグ・ライブラリー記述子ファイルのコピー。blox.tld、bloxform.tld、bloxlogic.tld、bloxreport.tld、および bloxui.tld が含まれます。

WEB-INF/ui
アプリケーションの見栄えを定義するために使用される XML ファイルが入っています。buttons.xml、toolbar.xml、および toolbarhelp.xml が含まれます。これらのファイルを編集して、必要に応じてボタンを追加または除去したり、ボタン・イメージおよびツールチップの説明を変更したりできます。アプリケーション開発者は、残りの XML ファイルを変更してはなりません。

ほとんどのアプリケーションにおいて、レポート・テンプレートが保管されている templates ディレクトリーに主に関心があるでしょう。次のセクションでは、レポート・テンプレートと templates ディレクトリーについてさらに詳細に説明します。

レポート・テンプレート

レポート・テンプレートは、FastForward アプリケーションの心臓部であり、エンド・ユーザーのためにレポートを迅速に構成するために使用される編集ページをアプリケーション管理者に提供します。レポート・テンプレートは、編集ページ、テンプレート・パラメーター・ファイル、レポート・ページ、およびヘルプ・ページという、最低 4 つのファイルで構成されます。**編集ページ**はアプリケーション管理者により使用され、レポートおよびデータのオプションを定義するために表示される選択リスト、ラジオ・ボタン、およびチェック・ボックスから選択することにより新規レポートを作成するページです。編集ページで使用可能なオプション (またはパラメーター) は、**テンプレート・パラメーター・ファイル**で定義されます。エンド・ユーザーに表示されるものは、**レポート・ページ**のレイアウトにより決定されます。特定のレポートと結合している**ヘルプ・ページ**も組み込まれます。オプションで、印刷ページおよび Excel ページをレポート・テンプレートに組み込み可能です。新規レポート・テンプレートはテンプレート開発者により作成され、使用可能なレポート・テンプレートのコレクションに追加して、レポートで頻繁に使用されるクラスのカスタム・レポート開発の必要を減らすことができます。また、その結果として、エンド・ユーザーは、後で再利用および共用するために、レポートのコピーのアクセス、変更、および保管を行えます。

各 FastForward アプリケーションにおいて、そのアプリケーションが使用可能なレポート・テンプレートのコレクションは、次のディレクトリーに置かれています。

```
<applicationDirectory>/templates/
```

templates ディレクトリーの内部には、レポート・テンプレート・サブディレクトリーのコレクションがあり、それぞれは必要なものを完備したレポート・テンプレートを構成します。新バージョンの FastForward アプリケーションが作成されるとき、サンプル・レポート・テンプレートがコピーされ、このディレクトリーに組み込まれます。使用可能なレポート・テンプレートのセットを含むディレクトリーに単にテンプレートをドロップするだけで、オンザフライで新規レポート・テンプレートを追加することもできます。 DB2 Alphablox に組み込まれているサンプル・レ

ポート・テンプレートだけでなく、組織内のテンプレート開発者の間で共用される、またはサード・パーティー開発者により作成された他のレポート・テンプレートを Alphablox からダウンロードできます。新規レポート・テンプレートが templates ディレクトリーに追加されると、それはアプリケーション管理者の使用可能なレポート・テンプレートのリストの中で即時に選択可能になります。また、templates ディレクトリーにドロップされるレポート・テンプレートの ZIP されたコピーが自動的に解凍され、メニュー内で選択可能になります。続いて、必要に応じて新規レポート・テンプレートを追加すると、アプリケーション管理者がサーバーを停止して開始する必要なく、すぐにそれらを使用可能にすることができます。

各レポート・テンプレート・ディレクトリーの内部には、少なくとも以下の 3 つの必要ファイルがあり、要約すると以下のようになります。

ファイル名
: **説明**

edit.jsp
: **編集ページ**。管理者により表示および構成され、新規レポートを作成したり、既存のレポートを編集したりします。

template.xml
: アプリケーション管理者により設定されるすべてのパラメーター (プロパティー) のリストを含む**テンプレート・パラメーター・ファイル**。

report.jsp
: **レポート・ページ**。管理者により定義された値と組み合わせると、ユーザーが表示および操作できるレポートが生成されます。

以下のテンプレート・ファイルはオプションです。

ファイル名
: **説明**

help.jsp
: **ヘルプ・ページ**は、この特定のレポートに関して、ユーザーに有用な情報を提供します。サンプル・テンプレートには、ヘルプ・ページのプロトタイプが含まれます。

excel.jsp
: Microsoft Excel に送信される HTML ページのカスタマイズを行う **Excel** ページ。

print.jsp
: **印刷ページ**は、印刷のために使用可能な HTML ページのカスタマイズを行います。

レポート・テンプレートの複雑さは、ユーザーが見たい情報 (report.jsp ファイルに表示される)、パラメーターの数とタイプ (template.xml ファイルで指定される)、およびこれらのパラメーターを構成するためのユーザー・インターフェース (edit.jsp を使用して生成される) によって決まります。簡単なレポート・テンプレートでは、多くの設定値を事前定義することにより、限られた数のパラメーター設定にすることが可能です。テンプレートをより柔軟にすると、より多くの構成オプションを提供しますが、多数のパラメーターを使用可能にする必要も生じます。

DB2 Alphablox に付属するサンプル・レポート・テンプレートの説明に従い、『カスタム・レポート・テンプレートの作成』というセクションでは、新規レポート・テンプレートの作成ステップを説明する文脈におけるこれら 3 つのファイルの詳細について説明します。

サンプル・レポート・テンプレート

DB2 Alphablox に付属するサンプル・レポート・テンプレートには、ほとんどの業務において通常遭遇する、一般的に使用されるさまざまなレポート・タイプが含まれています。これらのサンプルのいくつかは、現状のままでも役立ちますが、これらは、独自のカスタム・レポート・テンプレートを開発する方法を学習する助けとして使用することもできます。

DB2 Alphablox に含まれているサンプル・テンプレートは、レポート作成のさまざまなタイプのニーズを対象としており、それらは以下に要約されています。

サンプル・テンプレート	ディレクトリー名	説明
対話式 Present Blox	InteractiveBlox	サンプル・レポート: 対話式分析
サンプル割り振り	SampleAllocation	サンプル・レポート: ストアごとの売上高 (販売分析)
サンプル・レポート	SampleReport	サンプル・レポート: サンプル・レポート (販売分析)
サンプル・トレンド	SampleTrending	サンプル・レポート: 地域ごとの販売傾向 (販売分析)
販売分散	SampleVariance	サンプル・レポート: 販売分散 (分散分析)
対話式分散	VarianceQCC	サンプル・レポート: 随時分散分析 (分散分析)
サンプル・レポート Blox	SampleReportBlox	サンプル・レポート Blox テンプレート

サンプル・テンプレートは、特定ユーザーのすべてのニーズを網羅するものではありませんが、レポート・テンプレートの能力と柔軟性について手早く学習する上で便利ですし、特定の問題のソリューションをコード化する方法を学習するための良い例ともなります。少し修正するだけで Alphablox FastForward サンプル・アプリケーションのコピーをデプロイすることができますが、アプリケーション・フレームワークおよびレポート・テンプレートの能力は、固有のユーザー・ニーズに合わせたカスタム・レポート・テンプレートを追加してみると実感できます。次のセクションでは、簡単なレポート・テンプレートを作成する方法について説明します。

カスタム・レポート・テンプレートの作成

独自のレポート・テンプレートの開発を始める助けとして、このセクションでは、カスタム・レポート・テンプレートの作成を開始するために知っておく必要のある最も重要なステップについてガイドします。前述のように、各レポート・テンプレートには、レポート・ページ (report.jsp)、テンプレート・パラメーター・ファイル (template.xml)、および編集ページ (edit.jsp) という 3 つの重要なファイルが

含まれます。以下のステップでは、簡単な割り振りレポート・テンプレートにおいてそれらの各必要ファイルを作成する方法を説明します。

レポート・ページ (report.jsp) の作成または変更

レポート・ページ (report.jsp) は、レポートがアプリケーション管理者により構成された後に FastForward アプリケーション・ユーザーに表示されるビューを生成します。このページには、通常、レポート・タイトル、データ・ビュー、およびユーザー制御の情報が含まれます。レポート・ページは、静的表示ページからガイド付き分析レポートの範囲にわたり、高性能なものは多くのユーザー制御が含まれます。

レポート・ページを作成するには、エンド・ユーザーが使用する機能が組み込まれた代理 JSP ページを作成することにより開始します。

この例では、レポートは、タイトルと、指定した製品グループのサブカテゴリーの割合を示す基本的な円グラフ・ビューをロードします。

このビューを生成するために使用できるレポート・テンプレートを作成するには、レポートを生成するために読み込まれるパラメーターを作成する必要があります。テンプレート・パラメーター・ファイル (template.xml) で後で定義されるこれらのパラメーターには、アプリケーションを構成するために編集ページ (edit.jsp) を使用するときにアプリケーション管理者が設定できるプロパティーが含まれます。編集ページの作成方法については、レポート・ページおよびテンプレート・パラメーター・ファイルの作成を完了した後に学習します。

レポート・プロトタイプ・ファイルを開き、ファイルの上部に次の行を追加します。

```
<%@ include file="../../reportdata.jsp" %>
```

この JSP include ディレクティブにより、reportdata.jsp の内容がテンプレートのルート・ディレクトリーに置かれ、そのコンパイル時にレポート・ページに追加されます。 reportdata.jsp ファイルは、必要なクラス・ファイルをインポートし、bloxtld および bloxformtld のための taglib ディレクティブを追加し、以下のオブジェクトを report.jsp の内部で使用できるようにします。

オブジェクト
: **目的**

レポート
: レポート・パラメーター (またはプロパティー) にアクセスできるようにします

ユーザー
: ユーザー・パラメーターにアクセスできるようにします

appContext
: アプリケーション・パラメーターにアクセスできるようにします

template
: テンプレートおよびそのパラメーターにアクセスできるようにします

savedState
保管されたプライベート・レポートおよびグループ・レポートにアクセスできるようにします

これらのオブジェクトは、以下のメソッドをインプリメントします。

メソッド
String getParameter(ストリング名) - 名前付きパラメーターのパラメーター値またはヌルを戻します。名前付きパラメーターが未定義の場合には、ヌルを戻します。
String getParameter(ストリング名, ストリング・デフォルト) - 名前付きパラメーターのパラメーター値または defaultValue 引数を戻します。名前付きパラメーターが未定義の場合には、指定された defaultValue を戻します。
String[] getParameterValues(ストリング名) - 名前付きパラメーターに対して定義されたパラメーター値を戻します。パラメーター値が定義されていない場合には、空の配列を戻します。
String[] getParameterNames() - このオブジェクトに対して定義されたすべての名前付きパラメーターが含まれる配列を戻します。

レポート・プロトタイプをパラメーター化するには、ここでレポート・ページの上部で JSP スクリプトレットを定義して、使用するパラメーターを指定する必要があります。扇形スライス親メンバーと使用されるメジャーを指定した後、これら 2 つのパラメーターを読み込む照会ストリングを生成します。 DB2 OLAP Server または Essbase データ・ソースの場合には、次のようになります。

```
<%
  String pieSliceParent =
     report.getParameter("pieSliceParent","Specialties");
  String measure = user.getParameter("preferredMeasure", "Sales");
  String query = "&lt;SYM &lt;COLUMN (\"Measures\") \""+measure+"\"
     &lt;ROW (\"All Products\") &lt;CHILD \""+pieSliceParent+"\" !";
%>
```

Microsoft Analysis Services を使用している場合には、照会ストリングを指定する 3 行目は次のようになります。

```
String query = "SELECT DISTINCT( {[Measures].["+measure+"]} ) ON COLUMNS,
{"+pieSliceParent+".children} ON ROWS FROM [qcc]";
```

この例で使用される getParameter メソッドは、パラメーター名に続くオプションの 2 番目の引数を使用して両方のプロパティーのデフォルト値を設定することを可能にします。

次に、ページ上のパラメーター値を表示したいそれぞれの場所で JSP 式を置換します。 DataBlox タグで、<%= query %> を使用してユーザーの照会を読み込む必要があります。

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

さらに、2 つの JSP 式、<%= measure %> および <%= pieSliceParent %> を追加して、measure 値および pieSliceParent 値をレポート・タイトルに置換する必要もあります。次のようになります。

```
<h3>Comparing <%=measure%> for subcategories of <%= pieSliceParent %> </h3>
```

結果は、表示される円グラフに適したタイトルとなります。

ページ上部の JSP include ディレクティブを追加し、保管された値を読み込む JSP 式を追加すると、完全な report.jsp ファイルはおおよそ次のようになります。

```
<%@ include file="../../reportdata.jsp" %>
<%@ taglib uri="bloxlogictld" prefix="bloxlogic"%>
<%@ taglib uri="bloxuitld" prefix="bloxui" %>

<%
  String pieSliceParent =
     report.getParameter("pieSliceParent","Specialties");
  String measure = user.getParameter("preferredMeasure", "Sales");
  String query = "<SYM <COLUMN (\"Measures\") \""+measure+"\"
     <ROW (\"All Products\") <CHILD \""+pieSliceParent+"\" !";
%>

<blox:header/>
<body>
<blox:data id="dataBlox"
  dataSourceName="QCC-Essbase"
  useAliases="true"
  query="<%=query%>"/>
<h3>Comparing <%=measure%> for product code subcategories:
   <%=pieSliceParent%></h3>

<blox:chart id="chartBlox"
 bloxName="chart"
  height="100%"
  width="100%"
  chartType="Pie"
  totalsFilter="0" >
  <blox:data bloxRef="dataBlox" />
</blox:chart>
</body>
```

テンプレート・パラメーター・ファイル (template.xml) の作成または変更

次に、FastForward アプリケーション管理者が構成するパラメーター (またはプロパティー) を定義する必要があります。これを行うには、template.xml 実例ファイルを作成または変更する必要があります。そこには、レポート・テンプレートに必要なパラメーターのみを含めます。この処理は比較的短時間で終わるはずです。

template.xml ファイルの上部に、この XML ファイルのために必要な DTD 仕様を置きます。

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE template PUBLIC "-//Sun Microsystems, Inc.//DTD
   Web Application 2.2//EN" "../template.dtd">
```

この仕様に続き、template エレメントを組み込み、この特定のテンプレートのパラメーターを取り込みます。以下の表は、template エレメントの使用可能なネストされたエレメントをリストしています。

エレメント
説明

display-name
編集ページのレポート・テンプレート選択項目の選択リストに表示される名前が入っています。オプションの **lang** 属性を追加できます。これは、標準言語コードおよび国別サブコード (例えば、pt_BR または fr) を使用して定義されています。

description
[オプション] 編集ページに表示されるレポートの要旨。オプションの **lang** 属性を追加できます。これは、標準言語コードおよび国別サブコード (例えば、pt_BR または fr) を使用して定義されています。

report-page
レポートを生成するために使用されるレポート・ページを指定します。

edit-page
アプリケーション管理者がレポートを定義するために使用するビューを作成する編集ページのファイル名を指定します。

report-params
[オプション] 管理者が設定できるレポート・パラメーターのコレクションを定義します。各 param エレメントはこのエレメント内でネストされます。

param [オプション] このエレメントの内容がパラメーターを定義することを指定します。

param-name
[オプション] このタグは、param エレメント内でネストされ、コーディング・テンプレートで使用されるパラメーターの名前を指定します。

param-label
[オプション] このタグは param エレメント内でネストされ、パラメーターの表示名を指定し、レポート・テンプレートの編集ページでアプリケーション管理者に表示されます。オプションの **lang** 属性を追加できます。これは、標準言語コードおよび国別サブコード (例えば、en-GB または fr) を使用して定義されています。

default-value
[オプション] パラメーターのデフォルト値。レポートで値が提供されない場合には、この値が使用されます。

print-page
[オプション] レポートで使用される印刷ページを指定します。

excel-page
[オプション] レポートで使用される「Excel にエクスポート」ページを指定します。

help-page
[オプション] レポートで使用されるヘルプ・ページを指定します。

注: テンプレート・パラメーターは、編集ページで表示される順序で定義される必要があります。また、編集ページ・ファイル名およびレポート・ページ・ファイル名は、妥当な任意の名前にできます。以下の例で使用されるファイル名

は、FastForward に含まれているサンプル・レポート・テンプレートで使用される慣例に従い、妥当な命名の慣例を規定しています。これは、継続して使用することも可能なものです。

この例では、display-name、description、および report-params エレメントを変更する必要があります。 report-page エレメントおよび edit-page エレメントは、それらのページの実際のファイル名を定義します。

この例に従った、定義済みパラメーターを持つ template.xml ファイル全体の内容を以下に示します。

```
<?xml version="1.0" encoding="utf-8"?>

<!DOCTYPE template PUBLIC "-//Sun Microsystems, Inc.//DTD
   Web Application 2.2//EN" "../template.dtd">

<template>
   <display-name>Allocation (Essbase Version)</display-name>
   <description>First Allocation Template (Essbase Version)
   </description>
   <report-page>report.jsp</report-page>
   <edit-page>edit.jsp</edit-page>
   <report-params>
      <param>
         <param-name>pieSliceParent</param-name>
         <param-label>Parent Member of Pie Slices:</param-label>
      </param>
   </report-params>
</template>
```

report-params エレメントは、このテンプレートで使用されるレポート・パラメーターのコレクションを定義します。上記のネストされた param エレメント、pieSliceParent は、表示される扇形スライスの親メンバーを指定するためのパラメーターを定義します。レポート・ページおよびテンプレート・パラメーター・ファイルの作成の後、最後のタスクとなるのは、アプリケーション管理者が構成するための選択可能オプションを表示する編集ページを作成することです。

編集ページ (edit.jsp) の作成または変更

編集ページは、選択リスト、ラジオ・ボタン、およびチェック・ボックスに通常表示されるレポート・パラメーター (またはプロパティー) を定義するためにアプリケーション管理者が使用するページです。サンプル FastForward アプリケーションに含まれるサンプル・レポート・テンプレートの編集ページは、この例で必要とされる編集ページよりさらに複雑ですが、同様の肝要なステップが含まれています。

それでは編集ページの作成を始めます。 edit.jsp ファイルの上部に、インポートする必要があるすべての必要なクラスを指定する JSP page ディレクティブを追加します。

```
<%@ page import="com.alphablox.blox.form.FormEventListener,
     com.alphablox.blox.DataBlox,
     com.alphablox.blox.form.TimePeriodSelectFormBlox,
     com.alphablox.blox.logic.timeschema.TimeSchemaBlox,
     com.alphablox.blox.form.FormEvent,
     com.alphablox.blox.ServerBloxException,
     fastforward.*,
     com.alphablox.blox.form.MemberSelectFormBlox,
     com.alphablox.blox.data.mdb.Member" %>
```

次に、ページ上で使用される Blox Tag Library にアクセスするための JSP taglib ディレクティブを追加します。

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

さらに、<blox:header> タグが Blox タグにアクセスするすべての JSP ページ上に必要です。

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

次に、選択リスト・オプションを生成するために編集ページが使用する DataBlox を指定します。以下の <blox:data> タグは、QCC-Essbase データ・ソース (これは DB2 Alphablox アプリケーション定義ページですでに指定されている必要がある) を使用するために事前構成されている DataBlox を定義します。

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

加えて、必要なすべてのタグ属性を指定します。この例では、useAliases タグ属性値が true である場合、DB2 OLAP Server または Hyperion Essbase の定義済みデータ・ソースから取られた表示メンバー名 (固有メンバー名ではない) を表示させたいということをサーバーに知らせます。 Microsoft Analysis Services を使用している場合には、この例におけるデータ・ソース名は QCC-MSAS となり、useAliases タグ属性を追加しません。それは、そのタグ属性は DB2 OLAP Server データ・ソースおよび Essbase データ・ソースのみに適用されるからです。

次に pieSliceParent の ID を持つ MemberSelectFormBlox を指定して、データ・ソースから戻された扇形スライス親オプションを持つ選択リストを生成します。

```
<formblox:memberSelect id="pieSliceParent"
     visible="false"
     dataBloxRef="dataBlox"
     dimensionName="All Products"/>
```

注: ここで FormBlox を使用する上でのいくつかの重要な点を以下に示します。

- edit.jsp ページで定義される FormBlox は、template.xml ファイルで使用されるパラメーター名と完全に一致する id 属性名を持つ必要があります。この例では、MemberSelectFormBlox id は、template.xml ファイルで定義された pieSliceParent パラメーターと一致する pieSliceParent である点に注目してください。
- いずれかの処理ロジックが終了する前にレンダリングされることを防ぐため、visible タグ属性を false に設定することを忘れないでください。処理ロジックが完了した後、この例では以下の TemplateHelper クラスの renderControls メソッドは Blox をページ上にレンダリングします。この visible 属性 (visible="false") の追加を忘れた場合、または誤ってそれを true に設定した場合には、予期せずに重複した Blox がページに表示されます。

これで編集ページに表示される選択リストの定義が完了したので、ページを作成できます。

以下の JSP スクリプトレットは、以前に保管されたパラメーター値を適用し、(以前に可視にならない設定をした) ページ制御をレンダリングして、表示されるペー

ジを生成します。この JSP スクリプトレットは、期待される情報を管理者が必ず入力するようにするために使用されるバリデーターも設定します。妥当性検査のステップはオプションですが、アプリケーションの頑強性を向上させ、期待される値をユーザーが必ず入力するように助けます。

```
<%
   TemplateHelper.applySavedParameters(pageContext);
   TemplateHelper.renderControls(pageContext);
   Template template=(Template)request.getAttribute("template");
   template.setValidator(new Validator());
%>
```

次に、ReportValidator をインプリメントする Validator が定義されます。ReportValidator のインプリメントには、重要な処理を行う 1 つの関数、validate(ReportData data) を定義するクラスが必要です。バリデーターは、レポートが定義済みパラメーターへのアクセス権を取得するときと同じ方法でそれらのパラメーターへのアクセス権を与えます。つまり、検査が必要なすべてのパラメーターにおいて、データ・オブジェクト上で getParameterValue() を呼び出すことによりこれを行います。

この例では、検査により、管理者がリーフ・メンバーである (すなわち子を持たない) pieSliceParent を選択しないことを検証します。さらに、ユーザーが許可されない値の使用を試行した場合に表示される適切なエラー・メッセージが追加されます。

```
<%!
  public class Validator implements ReportValidator {
    public void validate(ReportData data) throws ServerBloxException {
      String pieSliceParent=data.getParameterValue("pieSliceParent");
        if (pieSliceParent == null){
            data.addError("Please select a member from the products.");
               return;
            }
// There is a FormBlox associated with pieSliceParent
// Use this FormBlox to get the selected member object and validate it
     MemberSelectFormBlox select =
       (MemberSelectFormBlox)data.getFormBlox("pieSliceParent");
     Member members[] = select.getSelectedMembers();
// there is only one selected member -- it cannot be a leaf
     if (members[0].isLeaf() == true) {
          data.addError("The selected member must have some children");
     }
   }
  }
%>
```

編集ページはこれで完成しました。参照用に edit.jsp ファイル全体の完全なコピーを示します。

```
<%@ page import="com.alphablox.blox.form.FormEventListener,
                 com.alphablox.blox.DataBlox,
                 com.alphablox.blox.form.TimePeriodSelectFormBlox,
                 com.alphablox.blox.logic.timeschema.TimeSchemaBlox,
                 com.alphablox.blox.form.FormEvent,
                 com.alphablox.blox.ServerBloxException, fastforward.*,
                 com.alphablox.blox.form.MemberSelectFormBlox,
                 com.alphablox.blox.data.mdb.Member"%>

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

```
<blox:header />

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

<formblox:memberSelect id="pieSliceParent"
   visible="false"
   dataBloxRef="dataBlox"
   dimensionName="All Products" />

<%
    TemplateHelper.applySavedParameters(pageContext);
    TemplateHelper.renderControls(pageContext);
    Template template=(Template)request.getAttribute("template");
    template.setValidator(new Validator());
%>

<%!
    public class Validator implements ReportValidator {
      public void validate(ReportData data) throws ServerBloxException {
          String pieSliceParent=data.getParameterValue("pieSliceParent");
          if (pieSliceParent == null){
             data.addError("Please select a member from the products.");
             return;
          }
          // There is a FormBlox associated with pieSliceParent
          // You can use this FormBlox to get the selected member
          // object and validate it,

         MemberSelectFormBlox select =
            (MemberSelectFormBlox)data.getFormBlox("pieSliceParent");
         Member members[] = select.getSelectedMembers();
         // there is only one selected member -- it should not be a leaf
         if (members[0].isLeaf() == true) {
            data.addError("The selected member must have some children");
         }
      }
    }
%>
```

編集ページは、作業テンプレートを持つために作成する必要がある最後のページです。この簡単なテンプレートの完成後、サンプル・レポート・テンプレートで編集ページを使用して、さらに複雑なテンプレートを作成し始める場合の助けとすることができます。とはいえ、より大掛かりなレポート・テンプレートに取りかかる前に、今作成したレポート・テンプレートをテストする必要があります。

オプションのテンプレート・ページの作成

前述のように、テンプレートには、ヘルプ・ページ、印刷ページ、および Excel ページを組み込むことができます。これらは特定のアプリケーション要件に合うようにカスタマイズできます。

作成されるレポート・テンプレートごとに、ユーザーが使用可能なレポートの使用法に結びついたヘルプ・ページを組み込むことをお勧めします。サンプル・レポート・テンプレートには、レポート・ヘルプ・ページ例を指し示す「レポート・ヘルプ」というリンクが含まれています。ヘルプ・ページのファイル名は、`template.xml` ファイルの `help-page` パラメーターで指定されます。

FastForward アプリケーションでは、アプリケーション・ツールバーに置かれている「印刷プレビュー」ボタンを押すと、render URL 属性が printer に設定されて現行のレポートがレンダリングされます。プリンター形式の詳細については、179 ページの『プリンター・フォーマット (render=printer)』を参照してください。カスタム印刷ページの作成についてのその他の詳細は、182 ページの『HTML ベースの印刷による印刷』にあります。

アプリケーション・ツールバー上には、FastForward レポートのための「Excel にエクスポート」オプションもあります。このオプションでは、render URL 属性が xls に設定されて現行のレポートがレンダリングされます。 Excel 形式の詳細については、180 ページの『「Excel にエクスポート」フォーマット (render=xls)』を参照してください。

FastForward アプリケーションのローカライズ

DB2 Alphablox は、サーバー・レベルでのグローバルなローカライズをサポートします。結果として、Blox は、着信ユーザー要求を基にしてではなく、サーバーのロケールを基にしてローカライズされます。 JSP ファイル内または Java クラス内のすべてのストリングはメッセージ・バンドル・ファイル、FastForwardBundle_<lang>.properties から抽出されます。このファイルには、すべての国別サブコード (例えば、en-GB または fr) を含む <lang> 言語コードが含まれています。 FastForward アプリケーションに組み込まれているすべての JSP 実例ファイルは、Apache プロジェクトから入手できる標準国際化対応 (i18n) タグ・ライブラリーを使用します。『テンプレート・パラメーター・ファイル』セクションですでに説明したように、オプションの lang 属性を display-name、description、および param-label エレメントに適用することができます。複数言語をサポートするために、これらのエレメントの複数インスタンスを使用できます。

レポート・テンプレートのテスト

レポート・ページ (report.jsp)、パラメーター定義ページ (template.xml)、および編集ページ (edit.jsp) が正しく作成されたら、FastForward アプリケーションの管理ページで使用できる簡単なレポート・テンプレートができたことになります。テンプレートをテストするには、このテンプレート全体を FastForward テスト・アプリケーションの templates ディレクトリーに置きます。次に、Microsoft Internet Explorer でアプリケーションを開いた後、「管理タスク」ボタンをクリックします。続いて新規レポートを作成し、選択可能なレポート・テンプレートのリストから新規テンプレートを選択して、うまくいくか観察します。編集ページで「プレビュー」ボタンをクリックしてレポートをプレビューしたり、「アプリケーションに戻る」をクリックしてエンド・ユーザーとしてテストすることができます。

レポート・テンプレートの保管

レポート・テンプレートを使用するには、作成したすべてのテンプレート・ファイル (report.jsp、template.xml、および edit.jsp が含まれる) を、アプリケーションの template ディレクトリーのサブディレクトリーに保管する必要があります。編集ページに表示されるテンプレート名は、template.xml ファイルで定義された display-name エレメントから読み取られます。

レポート・テンプレートの共用

作業テンプレートを持つようになりましたが、自分が作成したものを他のテンプレート開発者と共用する可能性があることを覚えておいてください。テンプレートを交換するならば、他のアプリケーション・ユーザーのニーズを満たす上で役立つことがありますし、他の人に模範となる学習例を提供できるかもしれません。テンプレート・ディレクトリーを ZIP してそれを他の人に渡すならば、その人が ZIP ファイルを FastForward アプリケーションのテンプレート・ディレクトリーにドロップするだけで、アプリケーション管理者はそれを即時に使用し始めることができます。新規ブラウザー・セッションが FastForward アプリケーションを開いた後に、新規レポート・テンプレートが表示されます。

savedState オブジェクトを使用した状態の保管

レポートが管理者により作成され、ユーザーが使用できるようになると、そのレポートは、定義済みパラメーターを持つ簡単な JSP ページである「パブリッシュされた」レポートになります。プライベート・アクセスまたはグループ・アクセスのためにレポートがユーザーにより保管される時、FastForward は、ユーザーの現在の変更と共に後でリストアできるように、レポートの保管を試行します。この方法についての要約を以下に示します。

1. ユーザーが「レポートの保管」ボタンをクリックすると、FastForward は以下の情報を保管します。
   a. 新規に保管されるレポートの基となるテンプレートおよびレポート・テンプレートの名前
   b. そのレポートに関連したパラメーター
   c. ページ上のすべての FormBlox の `FormValues` プロパティー
   d. ページ上のすべてのブックマーク設定可能な Blox (GridBlox、ChartBlox、PresentBlox、および DataBlox) のブックマーク
2. 保管済みレポートがユーザーにより後で再ロードされるとき、レポートは本質的に同じ順序で再構成されます。
   a. ページは、保管済みパラメーターを基にしてロードおよびコンパイルされます
   b. 保管済み `FormValues` プロパティーで `setFormValues()` を呼び出します
   c. 保管済みブックマークで各 Blox 上の `restoreBookmark()` メソッドを呼び出します

これは、ほとんどの状況で満足のいく方法です。しかし時には、この標準リストア手順から離れたいと思うかもしれません。例えば、レポートに表示されるデータを現在のカレンダーの日/四半期/月に基づくように変更したいかもしれませんし、グループ・アクセス・レポートの場合には、どのユーザーがレポートをロードするかを基にして特定の値を設定して個別設定されたレポートをロードしたいかもしれません。

これを容易にするために、FastForward は `savedState` オブジェクトを備えています。これは、レポートがプライベート・アクセスまたはグループ・アクセスからリ

ストアされた場合には必ず JSP ページから使用できます。レポートが「パブリッシュされた」場合には、savedState オブジェクトは使用不可になり、そこへの参照にはすべて null が戻されます。

savedState オブジェクトは以下の機能を備えています。

- デフォルトのリストア動作をオフにする機能
- ページ上の任意の FormBlox またはその他の標準 Blox を取得する機能
- 任意のブックマーク設定可能な Blox に関連したブックマークを取得する機能
- さまざまな Blox の状態を希望する任意の順序でリストアする機能

これらの詳細および savedState オブジェクトのその他の機能の詳細については、「DB2 Alphablox 管理ページ (DB2 Alphablox Admin Pages)」の「ヘルプ」メニューから参照できる、FastForward Javadoc 資料をご覧ください。

新規レポートを作成するときにこのオブジェクトを使用するためには、カスタム・レポート・リストア・ロジックを JSP ファイルの末尾に追加して、影響を受けた Blox がインスタンス化された後、しかし Blox がページにレンダリングされる前にロジックを適用する必要があります。

以下の例では、リストアされるレポート上で GridBlox が変更され、グリッド上で行バンディングが使用不可になります。

```
<blox:present id="myBlox" visible="false"
   width="100%" height="100%"
   dataLayoutAvailable="<%=dataLayoutVisible%>"
   menubarVisible="<%= menubarVisible %>">
   <blox:grid visible="<%=gridVisible%>"
       bandingEnabled="<%=gridBanding%>"/>
   <blox:chart visible="<%=chartVisible%>"
       chartType="<%=chartType%>"
       totalsFilter="0"/>
   <blox:data bloxRef="dataBlox" />
   <blox:toolbar visible="<%=toolbarVisible%>"
       removeButton="Save,Load" />
   combineToolbars(myBlox.getBloxModel()); %>
</blox:present>
<%
   if (savedState != null) { Bookmark bookmark =
   savedState.getBloxBookmark("myBlox");
   BookmarkProperties gridProps =
     bookmark.getBookmarkPropertiesByType(Bookmark.GRID_BLOX_TYPE);
     gridProps.setProperty("bandingEnabled", "false"); }
%>
<blox:display bloxRef="myBlox"/>
```

次のステップ

いくつかの簡単なレポート・テンプレートをマスターしたら、さらに難しいレポート・テンプレートに挑戦できます。 Alphablox FastForward サンプル・アプリケーションに含まれているサンプル・レポート・テンプレートは、独自のテンプレートを作成するために使用できるアイデアとコードの豊かな源です。

また、「*開発者用リファレンス*」、本書、*FastForward API Javadoc* 資料、および「*Blox API Javadoc*」資料を含む、すべての使用可能な開発者リソースを使用してください。これらは、 DB2 Alphablox 管理ページの「ヘルプ」メニューからも入手できます。

第 26 章 DHTML Client DOM API

開発者は、以下のセクションで説明されている、公表された DHTML DB2 Client DOM API を使用しない限り、DHTML により生成される DOM を操作あるいは全探索するようなクライアント・サイドのコードを書くべきではありません。なお、インプリメンテーションは今後変更される可能性があります。

GridBlox クライアント API

以下のトピックでは、JSP ページで表示される際に、GridBlox コンポーネントで使用可能なクライアント API について説明します。

Blox 定義

```
<blox:grid id="myBlox"
  width="80%"
  height="30%"
  bandingEnabled="true"
  visible="false">
  <blox:data bloxRef="dataBlox" />
</blox:grid>
```

DHTML Client は文書ネーム・スペース内の JavaScript オブジェクトを戻します。

Blox 用の JavaScript オブジェクトへの参照を取得するには、次のようにします。

```
var gridBlox = document.myBlox;
```

または

```
var gridBlox = myBlox;
```

グリッド

GridBlox は、ゼロ・ベースのグリッド配列を使用して、GridBlox 内にあるグリッドへのアクセスを提供します。

以前に定義された GridBlox に含まれるグリッドへの参照を取得するには、次のようにします。

```
var myGrid = myBlox.grids[n];
```

ここで n は、ゼロをベースとするグリッドの番号です。

これは、グリッドを表すエレメントを戻します。加えて、グリッドには行と列の合計数のための 2 つの属性があります。

グリッド内のスクロール可能な行または列の数を戻すには、次のようにします。

```
var numRows=myGrid.getRowCount(); var numCols=myGrid.getColumnCount();
```

これらの値はスクロール可能なエレメントの総数を表しており、ヘッダーとデータ行やデータ列を区別しません (それらがスクロール可能な場合)。スクロール可能エレメントの数が、使用可能な領域に入れられない場合、スクロールは必要とされていません。グリッド領域のサイズが変化すると、スクロール・バーは必要に応じて増減されます。

指示された行と列にグリッドをスクロールするには、次のように scrollTo メソッドを使用します。

```
myGrid.scrollTo(row,column)
```

scrollTo メソッドは、指示された行と列にグリッドをスクロールします。

グリッドにスクロールが使用可能かどうかを判別するには、次のようにします。

```
var enabled = myGrid.isScrollingEnabled()
```

グリッドにスクロールが使用可能であれば、isScrollingEnabled メソッドは true を戻します。

選択

グリッドはエンド・ユーザーに、1 つ以上のセルを選択する能力を提供します。それから、選択したセルでアクションを実行できます。 DHTML Client は、以下のメソッドを使用して、選択したセルへのプログラマチック・アクセスを提供します。

選択オブジェクト

グリッド内で現在選択されているセルを表す選択オブジェクトにアクセスするには、次のようにします。

```
var myGrid=myBlox.grids[0]; var select = myGrid.selection;
```

可視の選択済みセル ID 値の取得

選択オブジェクトは、ストリングのゼロ・ベースの配列を取得するためのメソッドを提供します。各ストリングは、関連したグリッドで現在選択されているセルの ID です。

```
var selectedCellIds = select.getCellIds();
```

この配列には、選択され、現在可視であるセルの ID だけが含まれます。選択済みセルの完全セットが必要な場合は、代わりにモデルにアクセスしてください。

あるセルが選択されているかどうかを判別するには、次のようにします。

```
var selected = selection.isSelected( cellID)
```

セルが選択されていれば、true を戻します。選択済みセルはクライアントで可視の場合も不可視の場合もありますが、その選択された状態はクライアントにより保持されます。

セルの選択状態を制御するには、次のようにします。

```
selection.selectCell(cellID,selected)
```

cellID は、有効なグリッド・セル ID でなければなりません。セルを選択するには selected を true に設定し、セルを選択解除するには false に設定します。

選択済みセルをすべてクリアするには、次のようにします。

```
selection.clearSelection()
```

すべてのセル選択がクリアされます。

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。

DB2	DB2 OLAP Server	DB2 Universal Database™
IBM	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。

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]

[ヤ行]

[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

プログラム番号: 5724-L14

Printed in Japan

SD88-6490-02

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