IBM DB2 Alphablox

Relational Reporting 開発者用ガイド

バージョン 8.4

SD88-6493-02
(英文原典：SC18-9437-02)

IBM DB2 Alphablox

Relational Reporting 開発者用ガイド

バージョン 8.4

SD88-6493-02
(英文原典：SC18-9437-02)

お願い

本書および本書で紹介する製品をご使用になる前に、 157 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。

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

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

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

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

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

(URL は、変更になる場合があります)

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

原　典： SC18–9437–02
IBM DB2 Alphablox
Relational Reporting 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 Relational Reporting の概説

この章では、IBM DB2 Alphablox for Linux, UNIX and Windows の Relational Reporting 機能の特徴、機能性、コンポーネント、およびブラウザー・サポートについて概説します。

Relational Reporting

Relational Reporting は、リレーショナル・データ・ソースから動的に生成する、対話式レポートをサポートします。DB2 Alphablox に定義されているものであれば、どのようなリレーショナル・データ・ソースからでもデータを抽出して結果セットを取得でき、次いでデータのフォーマット設定、計算、およびレポート編集タスクを実行できます。レポートは Dynamic HTML でレンダリングでき、PDF フォーマットで保管できます。

Relational Reporting の主要な機能は Report Editor ユーザー・インターフェースです。これは、ユーザーがデータを分析することや、必要に応じてオンザフライでレポートを作成することを可能にするものです。対話モードでレポートをレンダリングするときには、一連の対話的なコンテキスト・メニューがオンになります。ユーザーがレポートの「ホット・スポット」の上でマウスを動かすと、これらのメニューがポップアップ表示し、ユーザーはポイント・アンド・クリックで、列を隠すこと、列の再配列、データのソート、ブレーク・グループの追加、グループ・ヘッダーやフッター・テキストの指定、レポートの外観の変更ができるようになります。

一連の Relational Reporting Blox により、以下のことが可能です。

- DB2 Alphablox ホーム・ページを通して定義されたデータ・ソースに接続する
- SQL 照会を使用して、データ・ソースからデータを抽出する
- 計算に基づいてメンバーを追加する
- データのソートとフィルター操作を行う
- メンバーの包含と除外を行う
- 列のレイアウトを調整する
- ブレーク・グループを追加する
- さまざまなデータ・タイプおよび欠落した値のフォーマットを指定する
- バンディングを追加する
- フッターおよびヘッダーのテキストと、集約タイプ (sum、average、min、max、count、および none) を指定する
- レポート内のさまざまなエレメントのカラーとフォントを指定する
- ユーザー用の対話式コンテキスト・メニューを使用可能にする

- 後で検索できるようにレポートにブックマークを付ける
- レポートを PDF に送信する
- レポートを PDF または HTML で保管する

ローカリゼーション

Relational Reporting はローカリゼーションをサポートしています。レポート内の対話式メニューは、クライアントのロケールを基にして、自動的に正しい言語で表示されます。しかし、Blox コンポーネントのスコープはセッションであるため、Blox がすでにページにロードされた後で、ブラウザーの言語を変更する場合、新しい言語は同じセッション内の対話式メニューに反映されません。

ヒント: 英語以外の言語でレポートを正しく表示するには、JSP の `page` ディレクティブの `contentType` 属性を使って、UTF-8 文字セットを使用するように指定する必要があります。詳しくは、25 ページの『一般的な開発ヒント』を参照してください。

Relational Reporting のコンポーネント

Relational Reporting の心臓部に当たるのは ReportBlox です。他の一連の Blox (SortBlox、FilterBlox、MembersBlox、OrderBlox、GroupBlox、CalculateBlox、StyleBlox、FormatBlox、SQLDataBlox、DataSourceConnectionBlox、ErrorBlox、および PdfBlox) は ReportBlox と連動して、離散データの抽出、操作、レポートのフォーマット設定、およびレポートのレンダリング機能を処理します。

Relational Reporting Blox はすべて Java™ Beanです。各 Blox は、その名前が示すとおり、はっきり限定された一連の機能を備えています。ある Blox はリレーショナル・データ・ソースへの JDBC 接続を処理します。別の Blox はデータのソートだけを行います。また別の Blox は計算だけを行います。このように機能性がはっきり分かれているため、特定のタスクを実行する適切な Blox を、必要なときにだけ柔軟に追加することができます。これにより、望ましい結果が得られるよう、より優れた制御を行うことも可能になります。

これらの Blox を使ってリレーショナル・レポートを作成するには、提供されているカスタム・タグを JSP ページの中で使用します。これらの Blox をどのような順番で配置するかによって、これらの Blox 同士がどのように接続されるかや、データ結果セットがどのようにトランスフォームされるかが決まります。例えば、次の図では、算出メンバーが計算に基づいて追加され、次いでフィルター操作が実行されます。このフィルター操作は、算出メンバーの値に基づいて行われます。

以下は、リレーショナル・レポートの作成をサポートする全 Blox のリストです。

Blox	説明
ReportBlox	SQLDataBlox によって作成されたデータ結果セットを基にして、レポートを生成します。レポートは、静的な HTML 表の形か、対話式のレポート編集機能性を備えた Dynamic HTML (DHTML) の形でレンダリングすることができます。
SQLDataBlox	データ・ソースに対して SQL 照会を実行し、結果を結果セットに保管します。
DataSourceConnectionBlox	DB2 Alphablox に定義されたリレーショナル・データ・ソースへの接続を表現します。
RDBResultSetDataBlox	DataBlox から派生した RDBResultSet オブジェクトを表現します。
CalculateBlox	算出メンバーを、指定された計算式に基づいて列ディメンションに追加できます。
SortBlox	指定されたメンバーに基づいてデータをソートできます。
FilterBlox	指定する基準に基づいて、数値データをフィルター操作できるようにします。
GroupBlox	データのグループ化をサポートします。ReportBlox に GroupBlox を追加すると、レポートにブレーク・グループを作成し、各ブレーク・グループの末尾に集約タイプ (sum、average、count、max、min、または none) を指定できます。
MembersBlox	包含または除外するメンバーを指定できます。
OrderBlox	結果セットにメンバーを戻す順序 (左から右) を指定できます。
FormatBlox	数値データ・タイプと日付データ・タイプの表示フォーマットを指定できます。欠落データを表示する場合のテキストを定義することもできます。
StyleBlox	レポートのさまざまなエレメントのフォント、カラ

	ー、および配置などのスタイルを設定できます。欠落している値または負の値のスタイルを設定することや、個々の列のスタイルを設定することができます。
TextBlox	グループ・ヘッダー、フッター、合計、または列ヘッダーに表示するテキストを指定できます。
PdfBlox	指定したレイアウトで、リレーショナル・レポートを PDF に送信できます。
PersistenceBlox	後で検索できるようにレポートの状態を保管できます。
ErrorBlox	スローされた例外をキャッチして詳細を HTML 表に出力します。これにより、ネストされた例外の処理と表示が改良されます。

HTML 表形式のレポート

リレーショナル・レポートは Dynamic HTML 表の形式でレンダリングされます。表の各エレメント (列、行、ブレーク・グループ・ヘッダー、およびブレーク・グループ・フッター) には、Cascading Style Sheet の特定のクラスが関連付けられます。各エレメントのフォントのカラー、フォント・ファミリー、フォント・サイズ、フォント幅、テキストの配置、および背景色のカスタマイズは、通常 HTML と Cascading Style Sheet (CSS) で行うのと同様に行えます。

Relational Reporting には、ユーザーがレポートを編集できるよう、対話式 Report Editor ユーザー・インターフェースが備わっています。ReportBlox に関連して `interactive` プロパティーがあり、これはレポートを静的な HTML としてレンダリングするか、動的な HTML としてレンダリングするかを決定します。`<bloxreport:report>` タグの `interactive` 属性を `true` に設定すると、ユーザーがレポートの「ホット・スポット」の上でマウスを動かしたときにコンテキスト・メニューがアクティブになり、ユーザーはこれを使って変更を加えることができます。この Report Editor については次に説明します。

Report Editor ユーザー・インターフェース

リレーショナル・レポートを対話モードでレンダリングするよう設定した場合、ユーザーがレポートの特定の領域でマウスオーバーするとコンテキスト・メニューがポップアップ表示し、ユーザーが動的かつ対話式にレポートを編集できるようになります。レポート内の「ホット・スポット」は、以下のとおりです。

- 列ヘッダー
- ブレーク・グループ・ヘッダー
- ブレーク・グループ合計

上記のホット・スポットに応じてポップアップ表示するコンテキスト・メニューは、列ヘッダー・コンテキスト・メニュー、グループ・ヘッダー・コンテキスト・メニュー、およびグループ合計コンテキスト・メニューです。 Relational Reporting 用の Blox Sampler には「Simplest Interactive Report」という例が付属しています。これはサンプルのデータを使ったもので、簡単にレポートを実行したり、対話式 Report Editor ユーザー・インターフェースを試してみたりすることのできるもので

す。DB2 Alphablox ホーム・ページの「アセンブリー」タブの下の「Blox Sampler - Relational Reporting」を参照してください。

ユーザーはこれらのメニューを使って動的なレポート編集機能を利用できます。そのような機能としては、ソート、ブレーク・グループの追加、列の非表示/表示、ブレーク・グループ集約タイプの指定、列名およびブレーク・グループ合計テキストの編集があります。「**スタイル...**」メニュー・オプションを選択すると、「レポート・スタイル」ダイアログ・ボックスが表示されます。ユーザーはこれを使って、レポートのさまざまなエレメントのテキストのフォント書体、サイズ、スタイル、カラー、および配置を設定できます。これらのコンテキスト・メニューと「レポート・スタイル」ダイアログ・ボックスはすべて DHTML でインプリメントされています。定義されている各スタイル・クラスのスタイルを指定したスタイル・シートを編集したり、独自のスタイル・シートを準備することにより、これらのメニューおよびダイアログ・ボックスの外観のカスタマイズを簡単に行うことができます。

列ヘッダー・コンテキスト・メニュー

列ヘッダー・コンテキスト・メニューには以下のメニュー項目があります。

- ソート
- 隠す
- すべて表示
- 名前変更
- グループ
- グループのクリア
- スタイル...

ユーザーはこれらのメニュー項目を使って、列のソート、列の隠蔽、列の全表示 (「**隠す**」メニュー項目で隠した列を再表示)、列ヘッダーの変更、列のデータ値に基づくレポートのグループ化、ブレーク・グループを全クリアすること、およびレポートのスタイル設定を動的に実行できます。

グループ・ヘッダー・コンテキスト・メニュー

グループ・ヘッダー・コンテキスト・メニューには以下のメニュー項目があります。

- グループのクリア
- スタイル...

「グループのクリア」オプションでは、ユーザーはすべてのブレーク・グループではなく、特定のブレーク・グループだけをクリアできます。

グループ合計コンテキスト・メニュー

グループ合計コンテキスト・メニューには以下のメニュー項目があります。

- 合計
- カウント
- 平均
- 最小
- 最大
- なし
- テキスト編集
- スタイル...

数値データの列のデフォルト集約タイプは sum です。ストリングの列のデフォルト集約タイプは count です。ユーザーが列ヘッダー・コンテキスト・メニューを使って動的にブレーク・グループを追加する場合、JSP ファイルで別に指定しない限り、これらのデフォルトの集約タイプが適用されます。ブレーク・グループの追加と集約タイプの指定の詳細については、71 ページの『ブレーク・グループ・ヘッダー、フッター、および合計の指定とスタイル』で説明します。

レポートを PDF にレンダリングする場合

PdfBlox を使って、レポートを PDF にレンダリングしたり、編集済みのレポートをユーザーが PDF に保管できるようにしたりできます。PDF のレンダリングにヘッダーまたはフッターを含めるかどうかを指定したり、余白と方向を設定したりすることができます。詳しくは、83 ページの『PDF 形式での保管』のトピックを参照してください。

ブラウザー・サポート

リレーショナル・レポートは、非対話式の HTML 表、または対話式の DHTML 表にレンダリングできます。サポートされているブラウザーは以下のとおりです。

- IE 5.5 以上
- FireFox 1.0.4 以上
- Mozilla 1.7

非対話式レポートでは、Netscape 7.1 もサポートされています。ユーザーが Netscape ブラウザーで対話式レポートにアクセスしようとすると、アラート・ウィンドウがポップアップ表示し、対話式レポートでは Netscape ブラウザーがサポートされていないことを知らせます。

アクセシビリティ・サポート

ReportBlox を使用して作成された対話式レポート内をナビゲートする際に使用するキー・ストロークは通常、標準の Windows キーボード・ショートカットと一致しています。一般的なタスクのショートカットについては、このインフォメーション・センターおよび付属のユーザー・ヘルプの『*リリース概要*』に説明されています。キーボード・アクセスがサポートされるのは Internet Explorer v6 の場合のみです。

目の不自由なユーザーは、スクリーン・リーダー・プログラムを使用して、レンダリングされたレポートのデータを読むことができます。そのためには、アプリケーション開発者は ReportBlox readerMode タグ属性を true に設定する必要があります。これにより対話式メニューの「**スタイル...**」メニュー・オプションがオフになります。スクリーン・リーダーのフォーカスをこのダイアログに設定してテキストを読み取ることができないためです。また、目の不自由なユーザーにとって、「**レポート・スタイル**」ダイアログで提供されるスタイル設定のオプションは便利なものではありません。このタイプの個人情報設定は、カスタム・ユーザー・プロパティーを介するなど、ユーザー・プロファイルを使用して行えます。例えば、DB2 Alphablox 管理ページでカスタム・ユーザー・プロパティーを指定して、スクリーン・リーダーのサポートを必要とするユーザー向けに、そのプロパティーに特定の

値を割り当てることができます。その後、RepositoryBlox `getUserProperty()` メソッドを使用してその値にアクセスし、プログラマチックに `readerMode` タグ属性を `true` に設定できます。

第 2 章 Relational Reporting の概念

この節では、リレーショナル・レポートの全体的な設計を理解したり、効率的な開発を行ったりする上で不可欠な、Relational Reporting の主要な概念を説明します。

Relational Reporting の概念

以下の節では、リレーショナル・レポートの全体的な設計を理解したり、効率的な開発を行ったりする上で不可欠の、主要な概念を説明します。

標準的なテクノロジーに基づく、コンポーネント化された Blox

Relational Reporting 機能をサポートする Blox は、それぞれ機能が分かれています。それぞれが、データ形式変更、アクセス、またはプレゼンテーション・タスクの独自のセットを実行します。それぞれ、自分より前の Blox が作成した結果セットから別の結果セットを作成し、これをプレゼンテーションのために ReportBlox に送信したり、他の Blox に渡してさらにデータ形式変更を加えたりすることができます。これらの Blox はコンポーネント化されていて、それぞれ機能が分かれているため、これを柔軟に接続して、目的に応じた結果を得ることができます。JSP ページに Relational Reporting Blox を追加するときの順序によって、結果セットのトランスフォーム方法が決まります。

これらの Relational Reporting Blox は、JavaBeans™、JavaScript™ 1.2、および Cascading Style Sheet 2 などの標準的な Web テクノロジーに従っています。したがって、JSP ファイルの中で Relational Reporting Blox とともに標準的な Web 開発テクノロジーを使って、目的に応じた結果を作成したり、機能を拡張したりすることができます。

注: 同一の JSP ページに PresentBlox と ReportBlox の両方を含めることができますが、PresentBlox と他のすべてのサポート・ユーザー・インターフェース Blox (GridBlox、ChartBlox、PageBlox、DataLayoutBlox、および ToolbarBlox) は同じ方法で結果セットを処理して渡すわけではないため、Relational Reporting Blox をユーザー・インターフェース Blox に組み込んだり、その逆を行ったりすることはできません。

レポートのレンダリング

リレーショナル・レポートは HTML 表の形式でレンダリングされます。対話モードを true に設定すると、レポートは DHTML でレンダリングされます。レポートを対話モードでレンダリングするか、非対話モードでレンダリングするかを指定できます。これは、<bloxreport:report> タグの interactive 属性の設定で行います。interactive 属性が true に設定されている場合、ユーザーは用途と目的に応じてレポートのフォーマット設定を行えます。

これらの 2 つのレンダリング・モードに加え、別のオプションでは、リアルタイム・データを反映するレポートを PDF 形式でユーザーに提示できるかどうかを決めることができます。PdfBlox を使うと、レポートをオンザフライ (直接処理) で生成して、これを直接 PDF に送信できます。

注: PresentBlox およびサポートする他のすべてのユーザー・インターフェース Blox の場合とは異なり、リレーショナル・レポートの可視性を false に設定することや、ReportBlox を参照して <blox:display> タグを使用することはできません。

レポートのパイプライン

これらの Relational Reporting Blox をどのような順番で配置するかによって、これらの Blox 同士がどのように接続されるかや、このパイプラインの中でデータ結果セットがどのようにトランスフォームされるかが決まります。Relational Reporting の Blox の多くは、データ・コンシューマーとデータ・プロデューサーの両方の役割を兼ねます。前の Blox から戻された結果を消費し、これをトランスフォームし、後続の Blox のためにデータを作成します。最終使用先は ReportBlox です。

各 Blox ははっきり限定されたタスクを処理し、それぞれを接続する順番によって差異が生じるため、目的に応じたレポートが得られるように Blox をさまざまな方法で操作することができます。しかし、注意しないと誤ったデータになってしまうこともあります。例えば、グループ化する前にデータをソートするか、グループ化した後でソートするかによって、得られる結果は異なります。データをグループ化する前に計算列を追加するか、グループ化した後で追加するかによっても、得られるデータは異なります。こうした点については、特定のタスクを扱う章で説明します。

Relational Reporting のすべての Blox がデータ・トランスフォーマーというわけではありません。一部の Blox はデータ・フォーマット、レポートのスタイル設定、表示テキストの指定、PDF レンダリング、またはエラー処理を扱います。このような Blox は、どのような順番で指定しても、データ形式変更プロセスに影響はありません。こうした Blox は別個に処理されるか (ErrorBlox など)、データの処理が完了した後で処理されます。トランスフォーマーと非トランスフォーマーは、以下の表のとおりです。

データ・トランスフォーマー	非データ・トランスフォーマー
• CalculateBlox	• ErrorBlox
• FilterBlox	• FormatBlox
• GroupBlox	• StyleBlox
• MembersBlox	• TextBlox
• OrderBlox	• PersistenceBlox
• SortBlox	• PdfBlox

注: DataSourceConnectionBlox、JDBCConnectionBlox、RDBResultSetDataBlox、および SQLDataBlox はトランスフォーマーではありませんが、データ・トランスフォーマー用のデータを作成します。

個々の Blox へのアクセス

Relational Reporting Blox と、PresentBlox および他のユーザー・インターフェース Blox との間の主要な相違点は、ReportBlox の中にネストされている Blox には id を指定できるということです。PresentBlox、GridBlox、および ChartBlox の場合は、最も外側の Blox にしか id タグ属性を指定できず、ネストされた Blox に id を指定することはできません。Relational Reporting Blox の場合は、<bloxreport:report> タグが最も外側のタグになりますが、ネストされたタグとして追加された Blox にも固有の id を指定することができ、これを直接にスクリプト指定することができます。以下に例を示します。

```
<bloxreport:report id="myReport">
   <bloxreport:sqlData id="mySQLData" ...>
      <bloxreport:dataSourceConnection ... />
   </bloxreport:sqlData>
...
</bloxreport:report>
```

これで mySQLData に対して直接にスクリプト指定を行い、動的に照会を変更することができるようになります。

列とメンバー

Relational Reporting では、リレーショナル・データ・ソースからデータを抽出し、これをトランスフォームしてレポートにできます。レポートは通常、行と列からなります。しかし、さまざまなデータ形式変更操作で、しばしば「列」ではなく「メンバー」を指定する必要があることに気付くことでしょう。「メンバー」は、結果セットの中のデータ・フィールドまたはデータ・メンバーを指して使用されます。「列」はレンダリングされたレポートの中の表列を指して使用されます。一般に、レポートのスタイル設定は「列」を相手にして行い、データ形式変更タスクは「メンバー」を相手にして実行します。ユーザーが Report Editor ユーザー・インターフェースで列の移動または再配列をした場合、メンバーの列位置は変わることがあります。メンバーがブレーク・グループになると、そのメンバーは列上になくなることがあります。

リレーショナル・レポートのスタイル設定

レンダリングされたレポートのさまざまなエレメントには、関連するスタイル・クラスがあります。クラスごとに使用するスタイルを指定することにより、用途に合わせてレポートの外観をカスタマイズできます。

report.css というすぐに使用可能なスタイル・シートが用意されています。このスタイル・シートの保存場所を以下に示します。

```
<alphablox_dir>/repository/theme/
```

<alphablox_dir> は、DB2 Alphablox のインストール先のディレクトリーです。

report.css ファイルは上記のディレクトリーにあります。このスタイル・シートは、すぐに使用できるように提供されています。JSP ページで、以下の内容を <head> セクションに追加してご使用ください。

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

データは HTML 表の形式でレンダリングされるので、外部スタイル・シートまたは JSP ページのインライン・スタイルでスタイルを定義していない場合でも、レポートは表示可能です。これは単なるプレーンな HTML 表として表示されます。対話モードでレポートをレンダリングするする場合は、スタイル・シートがないと、コンテキスト・メニューと「レポート・スタイル」ダイアログ・ボックスが正しく表示および機能しません。

スタイル・クラス

生成されたレポートの中で使用されるクラスには、次のものがあります。

レポート全体用:

- .report
- .loadingmessage

ブレーク・グループ用:

- .groupheader1
- .groupheader2
- .groupheader*N*
- .groupfooter1
- .groupfooter2
- .groupfooter*N*
- .grouptotal1
- .grouptotal2
- .grouptotal*N*

列およびデータ・スタイル用:

- .column
- .columnhover
- .selected
- .data
- .banding

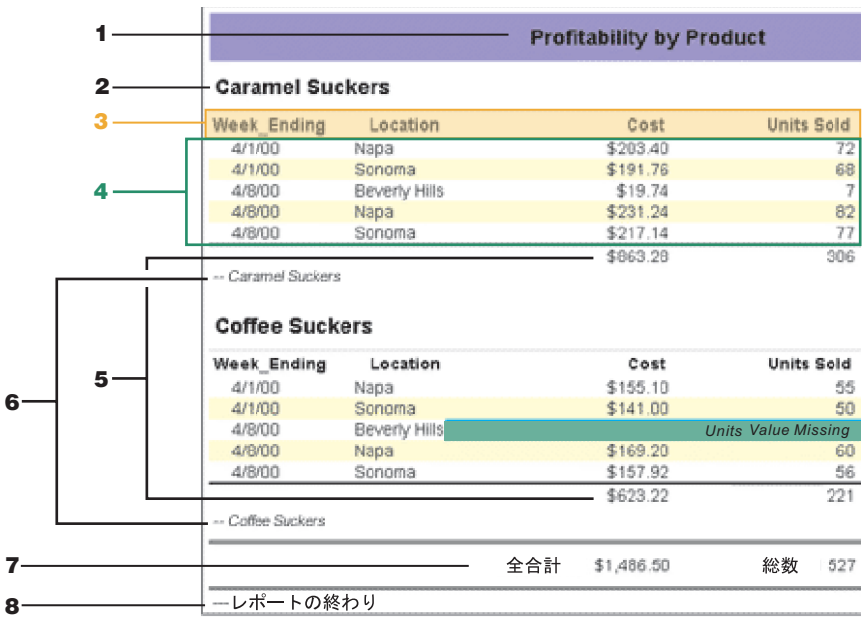
対話式コンテキスト・メニュー用:

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

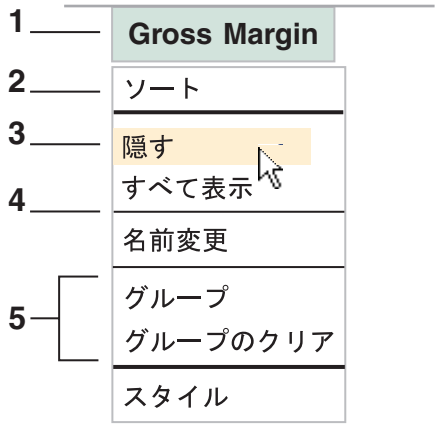
「レポート・スタイル」ダイアログ・ボックス用:
- .dialog
- .dialogtitle
- .dialogbody
- .dialoggroup
- .dialoggrouptitle
- .dialogradiogroup
- .dialogbutton
- .dialogbuttongroup

レポート内のスタイル・クラス

以下の 2 つのイメージは、各レポート・エレメントが使用するスタイル・クラスを示しています。



1. .groupheader1
2. .groupheader2
3. .column
4. .data
5. .grouptotal2
6. .groupfooter2
7. .grouptotal1
8. .groupfooter1



1. .selected
2. .choice
3. .choicehover
4. .separator
5. .menu

ブレーク・グループが追加されるたびに、クラス groupheader*N*、groupfooter*N*、および grouptotal*N* の *N* が 1 ずつ増加します。

各ブレーク・グループの末尾の集約データは grouptotal*N* スタイル・クラスを使用してレンダリングされますが、各ブレーク・グループの末尾に groupfooter*N* スタイル・クラスを使用するフッター・テキストを追加することができます。

各クラスの外見を定義した独自のスタイル・シートを作成し、そのスタイル・シートの場所を JSP ファイルに指定することができます。例えば、以下に挙げるサンプル・スタイル・シートは、各クラスに使用するフォントとカラーをどのように定義できるかを示しています。

```
.report {
   background-color: white;
   color: black;
   font-family :Arial, sans-serif;
   border: solid 1 black;
   padding: 5;
   margin : 5;
   width: 0;
}

/* for display of "Report Loading..." message */
.loadingmessage {
   white-space:nowrap;
}

/* break groups */
.groupheader1 {
   background-color: #99CCFF;
   color: black;
   font-size: 135%;
   text-align: center;
   border: solid white 1;
}

.grouptotal1 {
```

```
   background-color: white;
   color: black;
   font-size: 120%;
   text-align: right;
   padding : 5;
   border-top: double 3 black;
   border-bottom: double 3 black;
}

.groupfooter1 {
   display : none;
}

.groupheader2 {
   background-color: #6699CC;
   color: black;
   font-size: 120%;
   text-align: left;
   padding : 2 5;
   border-bottom: solid lightgrey 1;
}

.grouptotal2 {
   background-color: white;
   color: black;
   font-size: 100%;
   text-align: right;
   padding 2;
   border-top: solid black thin;
}

.groupfooter2 {
   display : none;
}

.column {
   color: black;
   font-size: 90%;
   font-weight: bold;
   padding : 2 3;
   text-align: left;
   border: solid white 1;
}

.columndragged {
   font-size: 90%;
   font-weight: bold;
   padding : 2 3;
   color: white;
   text-align: left;
   border: solid white 1;
   position: absolute;
   display: none;
   background-color: darkgray;
   cursor: hand;
}

.data {
   font-size: 90%;
   text-align: right;
   padding-left:20;
}

.banding {
   background-color: #CCCCFF;
}
```

```
/* Context Menus */
.menu {
   position: absolute;
   background-color: #E3E3E3;
   color: black;
   font-size: 90%;
   font-family: sans-serif;
   text-align: left;
   padding: 1;
   margin: 1;
   cursor: default;
   border: solid white 1;
}

.choice {
   padding : 1 5;
   white-space: nowrap;
   width: 100%;
}

.choicehover {
   background-color: #336699;
   color: white;
   padding : 1 5;
   white-space: nowrap;
}

.separator {
   padding: 1;
   font-size: 0;
   border-top : solid 1 black;
}

.selected {
   cursor: hand;
   border-color: darkgray;
   color: black;
   background-color: lightgrey;
}
```

「レポート・スタイル」ダイアログ・ボックス内のスタイル・クラス

「レポート・スタイル」ダイアログ・ボックスは DHTML でレンダリングされます。ブラウザーから HTML ソースを表示すると、ダイアログ・ウィンドウがどのように <DIV> および <SPAN> タグとスタイル・クラスを使用してレンダリングされているかが分かるでしょう。ダイアログ・ボックスと関連するスタイルはすべて「dialog」という語で始まっています。

ErrorBlox 用のスタイル・クラス

ErrorBlox は、キャッチされていないエラーを、HTML 表を使用して、縮小表示可能なリストに表示します。エラーを拡張すると、詳細なエラー・リスト (スタック・トレース) が表示されます。表と縮小表示可能なリストを正しく表示するには、ErrorBlox を入れたエラー処理ページで、付属の report.css スタイル・シートも使用しなければなりません。

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

<html>
   <head>
      <title>Error</title>
```

```
      <link rel="stylesheet"
            href="/AlphabloxServer/theme/report.css"
            type="text/css" />
   </head>

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

</html>
```

エラー表示用のスタイル・クラスの名前はすべて「error」という語で始まっています。

- .errorbox
- .errorimage (情報アイコン用)
- .errortitlebar
- .errormessage (情報アイコンの右隣のメッセージ用)
- .errornode (スタック・トレースのグループの拡張/縮小可能なノード用)
- td.errorstacktrace (各スタック・トレース用)
- .errordetail (各スタック・トレースの詳細記述用)
- .errorbutton (「詳細の表示/非表示」ボタン用)

ヒント: この拡張と縮小の動作は、Netscape ブラウザーではサポートされていません。

Relational Reporting カスタム・タグ

これらの Relational Reporting Blox を接続するためのカスタム JSP タグが使用可能です。これらのタグは bloxreport.tld ファイルにパッケージされています。これらのタグを使用するには、次のようにして Blox Report タグ・ライブラリーをインポートしなければなりません。

```
<%@ taglib uri="bloxreporttld" prefix="bloxreport" %>
```

各 Relational Reporting Blox には対応する JSP タグがあり、JSP タグには一連の属性があります。<bloxreport:report> タグは「ラッパー」タグであり、以下の例に示すように他のタグをネストします。リレーショナル・レポートを作成するにあたっては、リレーショナル・データ・ソースからデータ結果セットを生成し、次いでこの結果セットに対して計算、ソート、フィルター操作、再配列、データ・フォーマット設定、およびレポート・スタイル設定を実行して、目的に応じたフォーマットにトランスフォームします。

```
<bloxreport:report>
   <%--Generating a data result set from the data source
   and the SQL query specified--%>
   <bloxreport:sqlData ...>
      <bloxreport:dataSourceConnection ... />
   </bloxreport:sqlData>

   <%--Performing data transformation--%>
   <bloxreport:calculate .../>
   <bloxreport:sort .../>
   <bloxreport:filter .../>
   <bloxreport:group .../>
   <bloxreport:order .../>
```

```
   <%--Specifying the text to display for group headers, group
       footers, group totals, column headers, and data cells--%>
   <bloxreport:text>
      <bloxreport:data .../>
      <bloxreport:columnHeader .../>
      <bloxreport:groupHeader .../>
      <bloxreport:groupFooter .../>
      <bloxreport:groupTotal .../>
   </bloxreport:text>

   <%--Formatting the data--%>
   <bloxreport:format>
      <bloxreport:numeric .../>
      <bloxreport:date .../>
      <bloxreport:missing .../>
      <bloxreport:html .../>
      <bloxreport:aggregation .../>
   </bloxreport:format>

   <%--Styling the report using CSS style strings--%>
   <bloxreport:style>
      <bloxreport:text ... />
      <bloxreport:numeric ... />
      <bloxreport:date ... />
      <bloxreport:banding ... />
      <bloxreport:missing ... />
      <bloxreport:negative ... />
      <bloxreport:column ... />
      <bloxreport:data .../>
      <bloxreport:columnHeader .../>
      <bloxreport:groupHeader .../>
      <bloxreport:groupFooter .../>
      <bloxreport:groupTotal .../>
   </bloxreport:style>
</bloxreport:report>
```

<bloxreport:report> タグの外側にラップできる唯一のタグは <bloxreport:pdf> タグです。生きたデータを含んだレポートをユーザーが直接 PDF で表示できるようにする場合に、これを使用します。

```
<bloxreport:pdf>
   <bloxreport:report>
      <bloxreport:sqlData>
         <bloxreport:dataSourceConnection .../>
      </bloxreport:sqlData>
      <bloxreport:calculate .../>
      <bloxreport:sort .../>
      <bloxreport:filter .../>
      <bloxreport:group .../>
      <bloxreport:order .../>
      <bloxreport:format />
      <bloxreport:style />
   </bloxreport:report>
</bloxreport:pdf>
```

これらのすべての Blox タグには、その Blox の特定のインスタンスを一意的に識別するための id を割り当てることができます。多くの場合これは必要ではありませんが、特に、ReportBlox のすべてのインスタンスには、固有の id を指定しておくと良いでしょう。そうすれば、後でこのインスタンスを参照して、その属性値を動的に変更することができます。1 ページに複数の ReportBlox がある場合でも、DB2 Alphablox が ReportBlox の正しいインスタンスの状態を正しく識別できるようになります。

一部の Blox には、複数のエレメントを指定するためのネストされたタグがあります。ネストされたタグの概念については次に説明します。

ネストされたタグ

ネストされたタグを持つ Relational Reporting Blox は、FormatBlox、GroupBlox、SortBlox、StyleBlox、および TextBlox です。例えば、FormatBlox には、数値、日付、および欠落データのフォーマットを定義するためのネストされたタグがあります。

```
  <bloxreport:format>
    <bloxreport:numeric format="####.00;(####.00)" />
    <bloxreport:numeric format="$#,###.##;$(#,###.##)"
       member="Sales" />
    <bloxreport:date format="yyyy.MM.dd G 'at' hh:mm:ss z" />
    <bloxreport:missing format="Value Missing" />
  </bloxreport:format>
```

GroupBlox には、各列メンバーの集約タイプを指定するためのネストされたタグがあります。

```
<bloxreport:group members = "Product, Area">
   <bloxreport:aggregation member = "Sales" type = "sum" />
   <bloxreport:aggregation member = "Cost" type = "average" />
   <bloxreport:aggregation member = "Store" type = "count" />
   <bloxreport:aggregation member = "Units" type = "max" />
</bloxreport:group>
```

SortBlox には、各ソート規則を定義するためのネストされたタグが 1 つずつあります。

```
<bloxreport:sort>
   <bloxreport:rule member="Product" ascending="true" />
   <bloxreport:rule member="Week_Ending" ascending="false" />
</bloxreport:sort>
```

StyleBlox には、欠落した値、負のデータ、およびデータ列のスタイルを定義するためのネストされたタグがあります。

```
<bloxreport:style>
   <bloxreport:missing style="background-color: aqua;" />
   <bloxreport:negative style="color: red;" />
   <bloxreport:column style="background-color: #CCCCCC; color:
white;" columnName="Area" />
   <bloxreport:column style="text-align:center;" columnName="Code" />
</bloxreport:style>
```

これには、グループ・ヘッダー、グループ・フッター、グループ合計、列ヘッダー、およびデータ・セルのスタイルを定義するためのネストされたタグもあります。

```
<bloxreport:style>
   <bloxreport:groupHeader
      level="1" style="font-size:120%;" />
   <bloxreport:groupFooter
      level="1" style="font-size:120%;" />
   <bloxreport:groupTotal
      columnName="Sales" style="font-size:90%;" />
   <bloxreport:columnHeader
      columnName="Cost" style="font-size:90%;" />
   <bloxreport:data
      columnName="Product" style="color: #FFFFCC;" />
</bloxreport:text>
```

TextBlox には、グループ・ヘッダー、グループ・フッター、グループ合計、列ヘッダー、およびデータ・セルの表示テキストを定義するためのネストされたタグがあります。

```
<bloxreport:text>
   <bloxreport:groupHeader
      level="1" text="<b><i>My Group Header Text Here</i></b>" />
   <bloxreport:groupFooter
      level="1" text="My Group Footer Text Here" />
   <bloxreport:groupTotal
      columnName="Sales" text="Total: <value/>" />
   <bloxreport:columnHeader
      columnName="Cost" text="Unit Cost" />
   <bloxreport:data
      columnName="Product" text="<b><value/></b>" />
</bloxreport:text>
```

独立型タグ

Relational Reporting には、単独で JSP ページに追加できる、あるいは単独で JSP ページに追加しなければならない Blox が 3 つあります。すなわち、ErrorBlox、PdfBlox、および PersistenceBlox です。ErrorBlox は単独でエラー処理ページに追加されます。これについては 28 ページの『ErrorBlox を使用してエラー・レポートを改良する』で説明します。PdfBlox を <bloxreport:report> タグの外側に追加すると、レポートを直接 PDF にレンダリングできます。さらに、PdfBlox をJSP ページに単体で追加して、主にHTTP 要求オブジェクトを介して渡されるReportBlox ID を取得して、ReportBlox をPDF に送信することができます。これについては 85 ページの『レポートを PDF に直接レンダリングする』および83 ページの『PDF ファイル形式でのレポートの保管』で説明します。PersistenceBlox も、渡される ReportBlox を取得し、その ReportBlox の状態をリポジトリーに保管します。詳細については、80 ページの『レポートにブックマークを付けて状態を保管する』を参照してください。

構文評価の順序

レポート開発の必要に合わせてフォーマット設定やデータ操作を行うため、<bloxreport:report> タグの中に他のあらゆる Relational Reporting Blox を柔軟に追加することができます。開発に必要な場合は、同じタグの複数のインスタンスが必要になることがあります。例としては、<bloxreport:calculate> (算出メンバーが複数の場合) や、<bloxreport:filter> (フィルター操作が複数の場合) があります。こうしたタグは、先頭から末尾に向かって、宣言された順序で評価されます。したがって、FilterBlox を使用して一部のデータをフィルターに掛けて除外した場合、後続の操作にそれらのデータは含まれなくなります。それらのデータは結果セットの中に存在しなくなるからです。同じタグの複数インスタンスがあって、同一の属性に対して異なる値を設定している場合、最後に設定した値が使用されます。

同じタグの複数インスタンスがある場合の例

例 1:

```
<bloxreport:format>
  <bloxreport:numeric format="####.00;(####.00)" />
  <bloxreport:numeric format="$#,###.##;$(#,###.##)"
     member="Sales" />
</bloxreport:format>
```

例 2:

```
<bloxreport:format>
  <bloxreport:numeric format="####.00;(####.00)" />
  <bloxreport:numeric format="$#,###.##;$(#,###.##)"
     member="Sales" />
  <bloxreport:numeric format="$#,###;$(#,###)"
     member="Sales" />
</bloxreport:format>
```

最初の例では、すべての数値データは "####.00;(####.00)" フォーマットで表示されます。唯一の例外は「Sales」列のデータで、これは "$#,###.##;$(#,###.##)" フォーマットで表示されます。2 番目の例では、3 番目の <bloxreport:numeric> タグによって 2 番目のタグが無効になります。どちらもメンバー「Sales」に適用されるためです。結果として Sales データは "$#,###;$(#,###)" フォーマットで表示されます。

異なるタグの例

以下の 2 つの例の結果は異なります。

例 1:

```
<bloxreport:sort
  member="Units"
/>
<bloxreport:members
   excluded = "Units"
/>
```

例 2:

```
<bloxreport:members
   excluded = "Units"
/>
<bloxreport:sort
  member="Units"
/>
```

最初の例では、まず結果セットが「Units」メンバーを基にしてソートされ、次いで「Units」メンバーが除去されます。2 番目の例では、まず結果セットから「Units」メンバーが除去されるため、後続のソート操作は失敗します。

式の構文

操作または評価のための式を指定する場合の一般的な規則は次のとおりです。

- 式全体を二重引用符で囲まなければなりません。

  ```
  <bloxreport:calculate
     expression = "Sales = Unit_Cost * Units_Sold" />

  <bloxreport:filter
     expression = "Units > 500" />
  ```

- メンバー名に a-z、A-Z、0-9、および _ 以外の文字が含まれている場合、名前を大括弧 ([ ]) で囲まなければなりません。メンバー名にスペースまたは特殊文字が含まれている場合、常に [ ] で囲まなければなりません。メンバー名を数と取り違える恐れがある場合も、混乱を避けるために [ ] を使用してください。

```
<%--The following calculated member has "%" in its name--%>
<bloxreport:calculate
   expression = "[Profits%] = Sales/ [Gross Margin]"
/>

<%--Enclose member names with spaces in []--%>
<bloxreport:calculate
   id = "myCalc"
   expression = "CurrentQuarter = [April 01] + [May 01] +
      [June 01]"
/>
```

- メンバー名にすでに [ ] が含まれている場合、右側に 1 つ「]」を追加して、メンバー名の終わりであることを示してください。例えば、式にメンバー名 West[CA] を指定する場合は、[West[CA]]] とします。
- サポートされる演算子:
  - 計算式の場合: +、-、*、および /
  - フィルター式の場合: =、<、>、および !=
- 計算の演算子の区切り記号としてサポートされているのは () です。例えば、次のようになります。

```
<bloxreport:calculate
   expression = "[Profit%] = Sales/(Unit_Cost * Units_Sold)"
/>
```

- 複合式は使用できません。つまり、2 つの式のストリングを AND、OR、&、または | などの接続記号で接続することはできません。複数のフィルター操作を行う場合は、代わりに 2 つの FilterBlox を使用しなければなりません。

```
<bloxreport:filter
   expression = "Code > 240" />
<bloxreport:filter
   expression = "Code < 400" />
```

- サポートされている演算子は数値データにのみ作用します。これには整数、浮動小数点、および通貨が含まれます。ストリング、日付、時間、またはブール・データ型に対して計算、ソート、またはフィルター操作を実行することはできません。

メンバー ID と表示名

ソートの規則、データのフィルター操作の式、除外するメンバー、算出メンバーを追加するための計算式、およびメンバーの正確な順序を指定するときには、データ操作を伴う操作を使用することになります。その際、メンバー名の先頭が文字でない、あるいはメンバー名に特殊文字 (a-z、A-Z、0-9、または下線以外の文字) が含まれているなら、メンバー名を大括弧で囲む必要があります。そうすることで、DB2 Alphablox Relational Reporting エンジンに対して、式の中でこれらが変数であるということを示します。大括弧がないと、レポート・システムがメンバーを正確に識別できない可能性があります。例えば、「2002」という名前のメンバーを数と取り違えたり、「Region-East」という名前のメンバーを数式と取り違えたりする恐れがあります。有効な ID は、名前の先頭が文字で、その後に a-z、A-Z、下線、および 0-9 の任意の組み合わせが続くものでなければなりません。この要件にかなっていない名前はすべて大括弧で囲む必要があります。

以下に、要件にかなっていないメンバー名を大括弧で囲む例を示します。

```
<bloxreport:calculate
   expression="[Profit%] = [Gross Margin]/Sales" />

<bloxreport:filter expression = "[Q1 Sales] <10000" />

<bloxreport:sort member="[Profit%]" />

<bloxreport:order included = "Product, [Profit%], Sales, Cost" />
```

しかし、レンダリングされたレポートに表示されるテキストに大括弧は必要ありません。というのは、テキスト・ストリングがメンバー名か式かについて混同する恐れはないからです。例えば、StyleBlox を使用して列とテキストの表示方法を指定する場合、あるいは TextBlox で <bloxreport:columnHeader> タグを使用して列ヘッダーのテキストを指定する場合は、レンダリングされたレポート内の表示名を操作していることになります。このような場合、メンバー名を大括弧で囲む必要はありません。

```
<bloxreport:report id="salesreport1">
...
   <bloxreport:text>
      <bloxreport:columnHeader
         columnName="Profit%">
         text="Profit Pct" />
      <bloxreport:groupTotal
         columnName="Profit%"
         text="Avg.: <value/>" />
   </bloxreport:text>

   <bloxreport:style>
      <bloxreport:columnHeader style="color: blue;"
         columnName="Profit%" />
   </bloxreport:style>
...
</bloxreport:report>
```

注: メンバー名は、常にデータ・ソースで要求されているのと同じ大/小文字で指定するのが鉄則です。 メンバー「Cost」を「COST」または「cost」として参照すると、エラーになる可能性があります。

第 3 章 リレーショナル・レポートの開発

この節では、リレーショナル・レポートの全体的な開発ステップについて取り上げ、一般的な開発タスクを説明します。設計上の考慮事項やトラブルシューティングに関するヒントも取り上げます。

始める前に

開発を始める前に、「*開発者用ガイド*」で開発準備に関する一般的なヒントを確認してください。また、以下の質問を考慮して、Relational Reporting が本当に必要なのか、ユーザーの必要を満たすにはレポートをどのように設計すべきかを見極めてください。

- レポート作成元がリレーショナル・データベースかどうか。マルチディメンション・データベースの場合は、代わりに GridBlox を使用します。
- ユーザーがデータをチャートにするかどうか。チャートにする場合は、代わりに PresentBlox を使用します。
- ユーザーが回答または傾向を検索するためにデータを分析するかどうか。分析する場合、DB2 Cube Views と DB2 Alphablox キューブ・エンジンの機能を使用して、データのマルチディメンション表現を作成します。これにより、PresentBlox ユーザー・インターフェースを介して、データウェアハウスまたはデータマート・データベースに保管されているリレーショナル・データにすばやく多面的にアクセスすることができます。
- ユーザーが表示したいデータはどれか。たいてい、ユーザーはすべてのデータを必要としているわけではありません。データをすべて表示するページは、ロードに時間がかかり、画面の表示可能な領域に収まらなくなる可能性があります。

一般的な開発ヒント

JSP タグの使用とテストに関して注意を払う必要のある一般規則は、以下のとおりです。

- DB2 Alphablox タグ・ライブラリーをインポートします。Relational Reporting では、Blox Report タグ・ライブラリーを以下のようにインポートする必要があります。

  ```
  <%@ taglib uri="bloxreporttld" prefix="bloxreport" %>
  ```

- Blox タグと属性名の最初の語はすべて小文字です。後続の語の最初の文字は大文字で始まります。例えば、以下のとおりです。
  - `<bloxreport:sqlData />`
  - `<bloxreport:dataSourceConnection />`
  - `<bloxreport:groupHeader />`
  - `<bloxreport:groupFooter />`

- <bloxreport:groupTotal />
- <bloxreport:columnHeader />

- 属性に値を指定する場合は、値を二重または単一引用符で囲んでください。値が数値のようであっても、JSP タグでは属性のすべての値を引用符で囲む必要があります。例えば、以下の例のようになります。

```
<bloxreport:calculate
   expression = "[Profit%] = Sales/GrossMargin"
   index = "4"
/>
```

  算出メンバー Profit% は、index 属性が 4 に設定されており、5 番目の列として追加されます。4 は数字ですが、引用符で囲む必要があります。
- Blox タグを使用した JSP ページをブラウザーでテストする場合は、コンテンツを何もキャッシュに入れないようブラウザーを構成して、要求が出されるたびにサーバーにアクセスしてコンテンツを取り出すようにすることができます。
- 開発中、Blox タグの属性値に変更を加えて、これをテストすることがあります。そのセッションではオブジェクトがサーバー上にすでに存在しているため、そのオブジェクトが再利用され、結果として、加えた変更は反映されません。ブラウザーの新規インスタンスを開いて新規セッションを開始する必要があります。Netscape ブラウザーでは、新規ウィンドウを開く前に、すべてのブラウザー・ウィンドウを閉じなければなりません。
- 照会または特定の Blox 属性を動的に変更する必要がある場合には、ReportBlox の動的な bloxName を使用してください。詳細については、93 ページの『セッションの有効範囲の管理』、および 100 ページの『照会の動的な変更』を確認してください。
- Relational Reporting Blox の属性またはプロパティーの値を動的に設定する場合、Blox のインスタンスを作成するときに id を指定する必要があります。ユーザー・インターフェース Blox (グリッドとチャートでマルチディメンション・データを表示するのに使用する) の場合、ネストされた Blox に id を割り当てることはできません。Relational Reporting では、それが可能です。これにより、ReportBlox をサポートする個々の Blox を十分に制御することができます。
- ページ上に ReportBlox が 2 つあり、そのうちの 1 つ (または両方) を対話モードでレンダリングする場合は、固有の id も指定し、DB2 Alphablox が ReportBlox の各インスタンスの状態を正しく識別できるようにします。通常、以下のいずれかを実行したい場合には ReportBlox に id が必要です。
  - 属性値を動的に設定する
  - 1 ページに 2 つの ReportBlox を入れて、1 つまたは両方を対話モードでレンダリングする
  - ReportBlox の現在の状態を別の JSP ページに渡す (例えば ReportBlox を PDF または Excel に送信する場合)
- リレーショナル・レポートは、ブラウザーのロケールに基づいて表示されます。ただし、英語以外の言語でレポートを正しく表示するには、page ディレクティブの contentType 属性を使って、UTF-8 文字セットを使用するように指定する必要があります。contentType 属性を使用すると、MIME タイプと文字セットを定義できます。

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

- 対話式レポートは DHTML でレンダリングされるので、レポートに数千のデータ・セルが含まれる場合には、ブラウザーがページをレンダリングするのに数分かかる場合もあります。これはブラウザーの制限によるものです。最善のパフォーマンスと結果を得るため、データの作成はご使用のリレーショナル・データ・ソースのネイティブ環境で行い、必要なデータだけを ReportBlox に抽出するようにしてください。これにはデータの非正規化、またはサマリー・データを収容する表の作成が伴うこともあります。

注: 本書では、リレーショナル・レポート・タグの使用を中心に取り上げます。リレーショナル・レポート Blox に関連した Java メソッドについては、Relational Reporting Javadoc を参照してください。Javadoc にはサーバー・サイド API 用と ReportBlox API 用の両方が用意されており、以下のディレクトリーにあります。

```
<alphablox_dir>/system/documentation/javadoc
```

ここで、 `<alphablox_dir>` は、DB2 Alphablox のインストール先のディレクトリーです。

一般的なレポート開発ステップ

ReportBlox とそのサポートする Blox を使用してリレーショナル・レポートを開発するためには、一般的な 5 つのステップがあります。以下にこれらのステップを説明します。

アプリケーションおよびデータ・ソースを定義する

Alphablox アプリケーション開発の場合と同様、DB2 Alphablox Admin Page を使用して、アプリケーションとそのデータ・ソースを DB2 Alphablox に定義する必要があります。アプリケーションおよびデータ・ソース定義の詳細については、 「*管理者用ガイド*」または Admin Pages のオンライン・ヘルプを参照してください。

アプリケーションを定義すると、Admin Page に指定したコンテキスト名のアプリケーション・フォルダーが `webapps/` フォルダーの下に自動的に作成されます。これには、デプロイメント記述子 `web.xml` が入った `WEB-INF` フォルダーと、DB2 Alphablox タグ・ライブラリー記述子ファイルが入った `tlds/` フォルダーがあります。これらのファイルはそのまま手をつけずに置いてください。

リレーショナル・データベースに即時アクセスしない場合は、サンプルのリレーショナル・データが用意されているので、これを使って作業に取り掛かることができます。これについては、 29 ページの『最も簡単なレポート』の節で取り上げます。

Blox Report タグ・ライブラリーをインクルードする

Relational Reporting でカスタム・タグを使用するには、Blox Report タグ・ライブラリーをインクルードする必要があります。JSP ページの先頭に以下の行を組み込んでください。

```
<%@ taglib uri="bloxreporttld" prefix="bloxreport" %>
```

DB2 Alphablox タグ・ライブラリー の使用に関する一般的な情報は、「*開発者用ガイド*」を参照してください。

スタイル・シートを使用する

レンダリングされるレポートはスタイル・クラスと関連付けられているため、スタイルを定義するには外部 Cascading Style Sheet を使用してください。これは、レポートを対話モードでレンダリングする場合に特に重要です。コンテキスト・メニューと「レポート・スタイル」ダイアログ・ボックスの表示に関連する各クラスに定義されたスタイルがないと、Report Editor ユーザー・インターフェースは正しく動作しません。

スタイル・シートはカスケードするので、すぐに使用可能な付属のスタイル・シートを使用するようにし、変更したいクラスのスタイルだけを修正する独自のスタイル・シートを追加するのが最善です。

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

yourStyleSheet.css で、上書きしたいクラスに独自のスタイルを指定します。例えば、データ・セルの背景色を黄色にしたい場合は以下のようになります。

```
.data {
   color: yellow
}
```

別の手法は、カスタム・スタイル・シートからメイン・スタイル・シートをインポートする方法です。

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

重要: DB2 Alphablox の新規バージョンへアップグレードしたり、これをインストールしたりする場合、`<alphablox_dir>/repository/theme/` にあるデフォルトのテーマ・ファイル `report.css` が上書きされます。したがって、直接 `report.css` を変更するのではなく、スタイル・シートのコピーを作成して作業することをお勧めします。

ErrorBlox を使用してエラー・レポートを改良する

エラー・レポートを改良するには、作成する JSP ページのそれぞれにエラー処理ページを指定します。Relational Reporting では、スローされたもののキャッチされていない例外を ErrorBlox がキャッチし、Cascading Style Sheet を使用して詳細を HTML 表に出力します。ErrorBlox を使用するには、以下の行を使用してエラー・レポート JSP ページを作成できます。

```
<%@ taglib uri="bloxreporttld" prefix="bloxreport" %>
<%@ page isErrorPage="true" %>

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

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

最初の行では、 <bloxreport:error> タグを使用するためにロードするタグ・ライブラリー記述子 (TLD) ファイルを指定します。2 行目では、このページ自体をエラー・レポート・ページとして識別します。<head> タグでは、エラーを読みやすい表に表示するため、DB2 Alphablox に付属のスタイル・シートを指定します。次いで ErrorBlox を <body> タグの内部に追加します。基本的なカスタム・エラー処理ページを作成するために必要なことは、これですべてです。

その後、ReportBlox が含まれる JSP ページに以下の行を追加して、上で作成したエラー・レポート・ページをエラー処理ページとして指定します。

```
<%@ page errorPage="yourError.jsp" %>
```

エラーを表示するためのスタイルをカスタマイズできます。DB2 Alphablox の新規バージョンへアップグレードしたり、これをインストールしたりする場合、<alphablox_dir>/repository/theme/ にあるデフォルトのテーマ・ファイル report.css が上書きされることに注意してください。したがって、直接 report.css を変更するのではなく、スタイル・シートのコピーを作成して作業することをお勧めします。

エラー処理および ErrorBlox の使用の詳細については、110 ページの『ErrorBlox を使用したエラー処理』を参照してください。ErrorBlox スタイル・クラスについて詳しくは、16 ページの『ErrorBlox 用のスタイル・クラス』を参照してください。

Blox タグを追加する

Relational Reporting Blox を JSP ページに追加するには、提供されている例、または 113 ページの『第 12 章 Relational Reporting Blox のタグのリファレンス』から構文をコピーできます。先頭に <bloxreport:report> タグを指定してください。このタグは、データ検索、操作、およびフォーマット用の他のすべての Blox の外側に置かれて、これらすべてをラップするからです。Blox の接続方法および式構文の一般規則について詳しくは、 17 ページの『Relational Reporting カスタム・タグ』を参照してください。

最初のリレーショナル・レポートの作成

DB2 Alphablox にアプリケーションおよびリレーショナル・データ・ソースを定義したら、最初のリレーショナル・レポートの作成に進むことができます。

最も簡単なレポート

作成する各リレーショナル・レポートには、次の 3 つの重要な Blox が必要です。

- ReportBlox
- SQLDataBlox
- DataSourceConnectionBlox

DataSourceConnectionBlox を使用すると、リレーショナル・データ・ソースに接続できるようになります。SQLDataBlox は、SQL コマンドを指定して必要なデータをデータ・ソースから抽出するのに使用します。ReportBlox を使用すると、出力を HTML 表に生成できます。これらの Blox により、最初のリレーショナル・レポートを作成できます。以下に必要なすべての JSP コードを示します。これらはまた、作成するすべてのレポートで必要な、基本的なコードでもあります。

```
<%@ taglib uri="bloxreporttld" prefix="bloxreport" %>

<html>
<head>
</head>
<body>

<bloxreport:report id = "myReport">
   <bloxreport:sqlData query = "SELECT... FROM... WHERE...">
      <bloxreport:dataSourceConnection
         dataSourceName = "yourRDBdataSource">
      </bloxreport:dataSourceConnection>
   </bloxreport:sqlData>
</bloxreport:report>

</body>
</html>
```

Relational Reporting では、メンバー名に大/小文字の区別があることに注意してください。SQL 照会ステートメントの SELECT リストで列名を変更する際、大/小文字を保持する場合には列名を二重引用符で囲んでください。例えば、以下のようにします。

```
SELECT FROM myTable total_sq_ft AS "Sq_Ft", sq_ft_pct AS "Pct"
```

<bloxreport:sqlData> タグでは、円マーク (¥) で引用符をエスケープする必要があります。

```
<bloxreport:sqlData
query = "SELECT FROM myTable total_sq_ft AS ¥"Sq_Ft"¥, sq_ft_pct AS ¥"Pct¥"">
```

タスク: 最も簡単なレポートの作成

1. JSP 開発環境で新規ファイルを開き、前述のコードをコピーしてファイルに貼り付けます。
2. SQL 照会とリレーショナル・データ・ソース名を入力します。

   注: リレーショナル・データベースに即時アクセスしない場合は、サンプルのリレーショナル・データが使用可能です。このあらかじめ準備されたデータは実際には Java クラスで、SQL を理解しません。これを使用するには <bloxreport:sqlData> および <bloxreport:dataSourceConnection> の代わりに、<bloxreport:cannedData/> タグをコードに追加します。

   ```
   <%@ taglib uri="bloxreporttld" prefix="bloxreport" %>
   <html>
   <head>
   </head>
   <body>

   <bloxreport:report id="myReport">
      <bloxreport:cannedData/>
   </bloxreport:report>

   </body>
   </html>
   ```

3. ファイルに .jsp という拡張子を付けて、アプリケーション・フォルダーに保管します。これで、ブラウザー・ウィンドウを開いてファイルをテストできるようになりました。

最も簡単な対話式レポート

最初の対話式レポートを作成するには、以下に挙げる 2 つの追加ステップを実行することが必要です。

1. <bloxreport:report> タグに interactive 属性を追加して、値を true に設定します。

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

   デフォルトでは interactive 属性は false に設定されます。

2. <head> セクションに付属のスタイル・シートへの参照を追加します。

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

タスク: 最も簡単な対話式レポートの作成

1. 30 ページの『タスク: 最も簡単なレポートの作成』で作成した最も簡単なレポートを開きます。
2. <bloxreport:report> タグ内に interactive="true" を挿入します。
3. <head> セクションに付属のスタイル・シートへの参照を挿入します。
4. ファイルを保管します。
5. 新規ブラウザー・ウィンドウを開き、保管した JSP ページをテストします。

重要: 開発中に Blox タグの属性値を変更する場合、サーバーは既にオブジェクトのインスタンスを生成しており、そのオブジェクトはセッションの有効範囲の期間中ずっと存在しているので、変更を確認するにはブラウザー・ウィンドウを閉じるか、ブラウザーの別のインスタンスを使用して変更した JSP ページをテストする必要があります。

これで最初のレポートが作成されました。引き続き別の Blox を追加して、結果セットをさらにトランスフォームし、レポートをフォーマットできます。

学習資料

この「*Relational Reporting 開発者用ガイド*」の続く章は、タスク別に編成されています。関心のあるタスクを取り上げた章に進むことに加えて、105 ページの『第 11 章 開発およびトラブルシューティングのヒント』で重要なヒントを確認し、レポート開発に役立ててください。Application Studio の Blox Sampler - Relational Reporting の例のセットには、本書で詳述されている多くのタスクを明示した実例が備えられています。

第 4 章 データのアクセスと検索

リレーショナル・レポートを作成するための最初のステップは、ReportBlox にデータを入れることです。この章では、ReportBlox にデータを供給するさまざまな方法について説明します。

SQLDataBlox および DataSourceConnectionBlox の使用

DataSourceConnectionBlox はデータ接続を処理し、DB2 Alphablox Admin Page によって DB2 Alphablox に定義されたリレーショナル・データ・ソースへのアクセスを提供します。これは、`dataSourceName`、`username`、および `password` などの接続プロパティーを扱います。SQLDataBlox は指定した SQL 照会に基づいてデータを抽出し、Relational Reporting のデータ・パイプラインにデータを供給します。

29 ページの『最も簡単なレポート』で示したように、JSP の概略は以下のようになります。

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

<bloxreport:report id = "myReport">
   <bloxreport:sqlData query = "SELECT... FROM... WHERE...">
      <bloxreport:dataSourceConnection
         dataSourceName = "yourRDBdataSource">
      </bloxreport:dataSourceConnection>
   </bloxreport:sqlData>
</bloxreport:report>

</body>
</html>
```

DataSourceConnectionBlox およびそのタグ属性の詳細については、117 ページの DataSourceConnectionBloxを参照してください。SQLDataBlox およびそのタグ属性の詳細については、143 ページの SQLDataBloxを参照してください。サポートされるリレーショナル・データ・ソースのリストについては、「*インストール・ガイド*」の『System Requirements』の節を参照してください。

照会の動的な設定

照会を動的に設定または変更する必要がある場合のために、SQLDataBlox には `setQuery(queryString)` メソッドと `execute()` メソッドがあります。新規照会を設定してから、`execute()` メソッドを呼び出してください。完全な例が必要な場合は、100 ページの『照会の動的な変更』、および Application Studio の Blox Sampler - Relational Reporting の例のセットを参照してください。

RDBResultSetDataBlox を使用して、DataBlox が提供する RDBResultSet にアクセスする

データを ReportBlox に入れる別の方法は、RDBResultSetDataBlox を介する方法です。RDBResultSetDataBlox を使用すると、既存の DataBlox から戻される RDBResultSet を使用してリレーショナル・レポートを作成できます。DataBlox は、PresentBlox、GridBlox、および ChartBlox などのユーザー・インターフェース Blox にデータを供給します (これらの Blox のタグは Blox タグ・ライブラリーにあり、`<%@ taglib uri="bloxtld" prefix="blox" %>` taglib ディレクティブを使うことによって使用可能です)。DataBlox は、ReportBlox に直接データを供給することができません。しかし、DataBlox から戻された RDBResultSet は、RDBResultSetDataBlox を介することによって、Relational Reporting のデータ・ソースとして使用できます。これは、マルチディメンション・データ・ソースに由来する GridBlox セル内のデータのリレーショナルな詳細を、見栄えのする読みやすいレイアウトで提示できるアプリケーションに便利です。

例えば、DB2 Alphablox は GridBlox および DataBlox 内の Microsoft® Analysis Services データ・ソースのためにドリルスルー・サポートを備えていますが、このサポートでは、DataBlox の RDBResultSet を取得する RDBResultSetDataBlox を使用して、ReportBlox のリレーショナル・レポートを生成します。必要なのは GridBlox の `drillThroughEnable` プロパティーを `true` に設定することだけで、カスタム・コードは不要です。ユーザーがデータ・セルからドリルスルーすることにした場合、ReportBlox を使用した事前フォーマットされたレポートに、セルのリレーショナルな詳細が表示されます。

カスタム・レポートの場合、独自の JSP を用意し、そこに好みの形式にフォーマットした ReportBlox を含めることができます。MSAS バージョンの Blox Sampler の『Retrieving Data』セクションに実際の例があります。

DataBlox の RDBResultSet を ReportBlox に供給するには、GridBlox (または PresentBlox) が入った呼び出し側の JSP の中で、ReportBlox の入った JSP を呼び出し、以下の 3 つの情報を渡さなければなりません。

- DataBlox の `id` (ReportBlox は、この DataBlox の RDBResultSet をデータ・プロデューサーとして使用する)。
- リレーショナルな詳細が要求されるセルの座標 (`colIndex` および `rowIndex`)。

ReportBlox が含まれる JSP は、以下のようになります。

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

<bloxreport:report id="drillThroughFromDataBlox" ...>

   <bloxreport:rdbResultSetData
      bloxRef="myDataBlox"
      columnCoordinate="<%= request.getParameter(¥"colIndex¥") %>"
      rowCoordinate="<%= request.getParameter(¥"rowIndex¥") %>"
   />

   <%--further data manipulation and report formatting--%>
```

```
   ...

</bloxreport:report>
</body>
</html>
```

bloxRef 属性の値は、呼び出し側の JSP で定義される DataBlox の idでなければなりません。colIndex および rowIndex パラメーターは、呼び出し側の JSP の DHTML クライアントから、スクリプトレットまたは Java クラスによって渡されます。

例えば、呼び出し側の JSP に次のように DataBlox が含まれているとします。

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

DHTML クライアント用に、myDataBlox を使用する PresentBlox を以下のように使用している可能性があります。

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri='bloxuitld' prefix='bloxui'%>
...
<body>
<blox:present id="myPresentBlox"
   width="700"
   height="580">
   <blox:data bloxRef="myDataBlox"/>
   <blox:grid drillThroughEnabled="true" />
   <bloxui:actionFilter
      className="<%= myCustomDrillThrough.class.getName() %>"
          componentName="dataAdvancedDrillThrough" />
</blox:present>
...
```

ユーザーが右クリック・メニューから「詳細...」>「ドリルスルー」オプションを選択すると (componentName = "dataAdvancedDrillThrough")、myCustomDrillThrough クラスが呼び出されます。このクラスは、以下のようになります。

```
import com.alphablox.blox.uimodel.core.grid.GridCell;
import com.alphablox.blox.uimodel.tags.IActionFilter;
...

public class myCustomDrillThrough implements IActionFilter
{
   public void actionFilter( DataViewBlox blox, Component component )
throws Exception {
      GridBrixModel grid =
((PresentBloxModel)blox.getBloxModel()).getGrid();
      GridCell[] cells = grid.getSelectedCells();

      // Make sure that a single data cell is selected
      if ( cells.length != 1 || cells[0].isRowHeader() ||
cells[0].isColumnHeader() || !(cells[0] instanceof GridBrixCellModel ))
      {
         MessageBox.message( component, "Error", "You must select a
single data cell to drill through" );
        return;
      }

      GridBrixCellModel cell = (GridBrixCellModel)cells[0];
      int rowIndex = cell.getNativeRow();
      int colIndex = cell.getNativeColumn();
      String bloxName = blox.getBloxName();
```

```
      String urlStr = "myDrillThrough.jsp?bloxRef="+bloxName;
      urlStr += "&colIndex=";
      urlStr += colIndex;
      urlStr += "&rowIndex=";
      urlStr += rowIndex;
      String timestamp = String.valueOf(System.currentTimeMillis());
      urlStr += "&reportName=";
      urlStr =  urlStr + "reportBlox"+timestamp;

      ClientLink link =  new ClientLink( urlStr,
"reportBlox"+timestamp);
      component.getDispatcher().showBrowserWindow( link );
   }
}
```

完全な例は Blox Sampler の『Retrieving Data』セクションにありますので、参照してください。

Java クライアントを使用する場合、eventTriggerDrillThrough JavaScript コールバック・プロパティーを使用できます。

```
<blox:data
<blox:present id="myPresentBlox"
   mayscriptEnabled="true"
   width="700"
   height="580">
   <blox:data
      bloxRef="myDataBlox"/>
   <blox:grid
      drillThroughEnabled="true"
      eventTriggerDrillThrough="multipleDTWindows"/>
</blox:present>
```

ユーザーがドリルスルーを選択すると、同じ呼び出し側の JSP の中の multipleDTWindow という JavaScript 関数がトリガーされます。

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

<script language="JavaScript">
function multipleDTWindows(var1,var2,var3) {
   var bloxName;
   var colIndex;
   var rowIndex;
   var eventObj = var1;

   bloxName = "myDataBlox";
   colIndex = eventObj.getEventProperty(eventObj, "column");
   rowIndex = eventObj.getEventProperty(eventObj, "row");

   var urlStr = "someReportBlox.jsp?bloxRef="+bloxName;
   urlStr += "&colIndex=";
   urlStr += colIndex;
   urlStr += "&rowIndex=";
   urlStr += rowIndex;
   var currentDate = new Date();
   var timestamp = currentDate.getTime();
   urlStr += "&reportName=";
   urlStr =  urlStr + "reportBlox"+timestamp;
   window.open(urlStr,"reportBlox"+timestamp);
   return false;
}
</script>
</head>
```

完全な例は Blox Sampler の『Retrieving Data』セクションにありますので、参照してください。

SQLDataBlox に対するエラー処理

データ照会でデータが生成されない場合、レポートに表示されるデフォルトのメッセージは「データがありません」というものです。ReportBlox の `noDataMessage` タグ属性を使用して、このメッセージをカスタマイズできます。照会が無効な場合、またはデータ・ソースがその時点で使用不可になっている場合、ユーザーに JSP エラーのリストが表示されます。この場合、ReportBlox の `errors` プロパティーを `false` に設定し、スローされた例外をインターセプトしないようにすることもできます。それから try/catch ブロックを追加して問題を識別し、より分かりやすいエラー・メッセージを表示できます。詳細およびコード例については、110 ページの『ErrorBlox を使用したエラー処理』を参照してください。 `noDataMessage` および `noDataDueToErrorMessage` タグ属性の詳細については、138 ページの ReportBloxのタグのリファレンスを参照してください。

第 5 章 データの処理と取り扱い

データを ReportBlox で取得したら、別の Blox を追加してデータを処理し操作できます。この章では、データのソートおよびフィルター操作、列の非表示、除去、および再配列、さらには算出列の追加など、様々な一般的なデータの処理と操作タスクについて取り上げます。これには、SortBlox、FilterBlox、CalculateBlox、GroupBlox、MembersBlox、および OrderBlox などのデータ「トランスフォーマー」Blox が関係しています。

Relational Reporting の Blox Sampler の『Sorting, Filtering, Hiding, and Calculating Data』の例は、この章で説明されている多くの Blox を示しています。

- 39 ページの『データのソート』
- 40 ページの『データのフィルター操作』
- 41 ページの『データのグループ化』
- 42 ページの『算出列の追加』
- 44 ページの『メンバーの除去』
- 45 ページの『メンバーの非表示と表示』
- 46 ページの『欠落データの非表示と表示』

データのソート

データをソートするには、SortBlox を使用します。簡単な単一レベルのグループ化の場合 (「Product」によるレポートのグループ化など)、以下のように指定できます。

```
<bloxreport:sort member="Product" />
```

デフォルトでは、レポートは昇順でソートされます。ascending 属性を使用してソート順を指定できます。

```
<bloxreport:sort member="Product"
   ascending="false" />
```

デフォルトでは、欠落データは最後に表示されます。欠落データを最初に表示するように指定することもできます。

```
<bloxreport:sort member="Product"
   ascending="false"
   missingLast="false" />
```

SortBlox の member 属性が単数形であることに注意してください。つまり、この構文では 1 つのメンバーしかソートできないということです。SortBlox は、複数のソート規則を使用した複合ソートをサポートしています。各規則は SortBlox 内のネストされたタグであり、各規則にソートするメンバーを指定する必要があります。

```
<bloxreport:sort>
   <bloxreport:rule
      member="Type"
```

```
      missingLast="true"
      ascending="true" />
   <bloxreport:rule member="Product" />
</bloxreport:sort>
```

規則を指定する順序によって違いが生じることに注意してください。最初の規則が 1 次ソートとなり、2 番目が 2 次ソートになります。

重要: 複数のソート規則を指定することと、複数の SortBlox を使用することとは異なります。複数の SortBlox を追加すると、後の SortBlox は先に実行されたソート操作をオーバーライドします。

SortBlox はソートを扱う唯一の Blox です。商品タイプごとにレポートをグループ化し、アルファベット順にブレーク・グループを表示したい場合には、グループ化する前にまずデータをソートしなければなりません。GroupBlox はデータのグループ化のみを処理し、グループ化する前にデータをソートすることはないからです。例えば、レポートを最初に「Type」、次いで「Product」を基準にグループ化し、ブレーク・グループをアルファベット順にリストする場合、ブレーク・グループを追加する前に、まず「Type」、次に「Product」、最後に「Sales」の順でソートする必要があります。

```
<bloxreport:sort>
   <bloxreport:rule
      member="Type"
      missingLast="true"
      ascending="true" />
   <bloxreport:rule member="Product" />
   <bloxreport:rule member="Sales" />
</bloxreport:sort>
<bloxreport:group members="Type, Product" />
```

以下の点に注意してください。

- この SortBlox には 3 つのソート規則があります。各ソート規則には、1 つのメンバーしか含めることができません。1 つの SortBlox を使用してソート規則をネストすると、複合ソート操作が可能です。複数の SortBlox を使用した場合、後の SortBlox は前のものをオーバーライドし、以前のソート結果を保持しません。
- SortBlox は GroupBlox の前に追加されるので、レポートは異なるレベルのブレーク・グループでグループ化されて、アルファベット順で表示されます。

ブレーク・グループの追加について詳しくは、69 ページの『ブレーク・グループおよびブレーク・グループ・レベルの概説』を参照してください。SortBlox および SortBlox タグ属性の詳細については、141 ページの SortBloxを参照してください。GroupBlox および GroupBlox タグ属性の詳細については、125 ページの GroupBlox を参照してください。

データのフィルター操作

特定の基準に基づいてデータをフィルター操作するには、FilterBlox を使用します。各 FilterBlox には比較式が 1 つあります。複数の FilterBlox (それぞれ 1 つずつ独自の比較式を持つ) を追加することができます。例えば、販売数が 200 から 401 の範囲のデータのみに興味がある場合、以下のように 2 つの FilterBlox を使用します。

```
<bloxreport:report id="myReport">
   ...
   <bloxreport:filter expression="Sales > 200" />
   <bloxreport:filter expression="Sales < 401" />
   ...
</bloxreport:report>
```

2 番目の FilterBlox は最初の FilterBlox の結果に基づいてデータをフィルター操作するので、例えば売上データが 400 を超えるものか、200 未満のものだけを保持するなどの操作を実行する必要がある場合は、そうしたデータをデータベース環境で準備してから ReportBlox に取り出さなければなりません。

FilterBlox は数値データしか処理しないことに注意してください。これは比較用の 4 つの演算子 (>、<、=、および !=) をサポートしています。フィルターで欠落データを除外する (または欠落データのみを保持する) ために `isMissing()` 関数が提供されています。

重要: デフォルトでは FilterBlox は欠落データをフィルターに掛けて除外しません (除外するように指定した場合は除く)。

```
<bloxreport:filter expression="not isMissiing(Sales)" />
```

FilterBlox と SortBlox の動作およびタグ構文の違いを正しく認識してください。

FilterBlox	SortBlox
複合のフィルター操作のために複数の FilterBlox をチェーニングできます。	1 つの SortBlox しか使用できません。最後に宣言したものが、以前のものをオーバーライドします。複合のソートのためには、複数のソート規則を使用します。
各 FilterBlox は、フィルター式を 1 つしか使用しません。	SortBlox は複数のソート規則を使用できます。それぞれの規則は、ネストされた `rule` タグを使用して指定します。
数値データのみをフィルター操作できます。	数値データ、ストリング、日付、および時間をソートできます。

FilterBlox の使用法の詳細については 120 ページの FilterBloxを、メンバー名にスペースか特殊文字が含まれる場合の指定法について詳しくは 21 ページの『式の構文』をそれぞれ参照してください。

データのグループ化

レポートにグループを追加するには、GroupBlox を使用します。

```
<bloxreport:group members="Area, Location" />
```

GroupBlox の `members` 属性を使用すると、グループ化するメンバーを指定できます。前述の例では、「Area」によって、その後「Location」によって、レポートがグループ化されます。レポートは、メンバーを指定した順序に基づいてグループ化されます。GroupBlox を追加すると、以下のことが可能です。

- ブレーク・グループ内の各列の集約値を取得する
- ランキング、累計値、合計のパーセント、現行のカウントを表示するためのグループ・ベースのサマリー列の計算

- ブレーク・グループ・レベルに基づいてグループ合計のテキストを指定する
- ブレーク・グループ・レベルに基づいてブレーク・グループ・ヘッダー・テキストおよびブレーク・グループ・フッター・テキストを指定する

こうした事柄には、密接に関連する複数のタスクが関係しているので、69 ページの『ブレーク・グループおよびブレーク・グループ・レベルの概説』で取り上げられています。GroupBlox タグ構文について詳しくは、125 ページの GroupBloxを参照してください。

算出列の追加

CalculateBlox を使用して、算出列を追加できます。新規列名および計算式を指定するには、expression タグ属性を使用します。

```
<bloxreport:calculate
   expression = "Profit = Sales - Cost"
/>
```

これにより「Profit」という算出列が作成されます。その値は「Sales」から「Cost」を減算することで得られます。メンバー名に特殊文字かスペースが含まれる場合、名前を大括弧で囲んでください。

```
<bloxreport:calculate
   expression = "[Profit%] = Sales/ [Gross Margin]"
/>
```

デフォルトでは、新規メンバーは列ディメンションの最後に追加されます。新規メンバーを追加する正確な列位置を指定するには、index 属性を使用してください。列カウントは 0 から始まるので、以下の例では新規メンバー「Profit%」は最初の列として追加されます。

```
<bloxreport:calculate
   expression = "[Profit%] = Sales/ [Gross Margin]"
   index = "0"
/>
```

CalculateBlox を使用する際には、以下の点に注意してください。

- 計算に使用するメンバーに欠落値または NULL 値が含まれる場合、計算結果は欠落データになります。詳しくは、43 ページの『欠落データを含む計算』を参照してください。
- メンバー名にすでに [ ] が含まれている場合、右側に 1 つ「]」を追加して、メンバー名の終わりであることを示してください。例えば、式にメンバー名 West[CA] を指定する場合は、[West[CA]] とします。
- 計算式でサポートされている演算子は、+、-、*、および / です。
- 計算の演算子の区切り記号としてサポートされているのは () です。例えば、以下のようにします。

  ```
  <bloxreport:calculate
     expression = "[Profit%] = Sales/(Unit_Cost * Units_Sold)"
  />
  ```

- サポートされている演算子は数値データにのみ作用します。これには整数、浮動小数点、および通貨が含まれます。ストリング、日付、時間、またはブール・データ型に対して計算を実行することはできません。

- 計算式では、rank()、percentOfTotal()、runningTotal()、および runningCount() の 4 つの関数がサポートされています。これらの関数はレポートがどのようにグループ化されるかに関連があり、詳しくは 74 ページの『グループ・ベースのサマリー列の計算』で説明されています。

欠落データを含む計算

関係するオペランドのいずれかに欠落値が含まれる場合、結果は欠落または非数値 (NaN) と見なされます。この点、欠落データが無視される列方向の集約とは異なります。以下の表は、グループ合計集約の場合と、CalculateBlox を使った算出列の場合とで、計算の中での欠落データの扱いがどのように異なるかを示しています。

	商品 A	商品 B	A + B
ストア A	5	10	10
ストア B	4	10	14
ストア C	3	7	10
ストア D		3	
ストア E	4	5	7
	合計: 16 カウント: 4 平均: 4	合計: 35 カウント: 5 合計: 7	合計: 41 カウント: 4 平均: 10.25

上の表では、5 つのストアでの 2 つの商品のデータが示されています。ストア D の商品 A のデータは欠落しています。ストア D の算出メンバー A+B の値は欠落と見なされます。各列の集約値では、欠落データは無視されます。デフォルトでは、値が欠落している場合に空ストリングが表示されます。データが欠落している場合に表示されるテキストを指定するには、54 ページの『データのフォーマット設定』を参照してください。

グループ化する前に行う算出メンバーの追加

基礎となるデータ・パイプライン・モデルのせいで、CalculateBlox から見てどのような位置に GroupBlox を追加するかに応じて、算出メンバーの集約値が変わってくることがあります。以下の例は、その違いについて示しています。

GroupBlox が CalculateBlox より先の場合

```
<bloxreport:group members="Product">
<bloxreport:calculate expression = "[Z] = X*Y" />
```

X	Y	Z
2	5	10
3	5	15
5	10	**50**

CalculateBlox が GroupBlox より先の場合

```
<bloxreport:calculate expression = "[Z] = X*Y" />
<bloxreport:group members="Product">
```

X	Y	Z
2	5	10
3	5	15
5	10	**25**

GroupBlox を先に追加する場合、X と Y の集約値が算出メンバー Z の集約値を計算するのに使用されます。GroupBlox を後で追加する場合、Z の集約値は列内のデータに基づいて計算されます。

拡張計算関数 `rank()`、`percentOfTotal()`、`runningTotal()`、および `runningCount()` を使用すると、ブレーク・グループに関連してランクの計算方法を指定できます (合計に対するパーセンテージ、累計値、およびカウント値)。これらの拡張計算関数を使用する場合には、計算関数を指定する前にまずブレーク・グループを追加する必要があります。これらの関数の使用法はブレーク・グループと密接に関係しているため、詳細については 69 ページの『第 7 章 データのグループ化』の中の、74 ページの『グループ・ベースのサマリー列の計算』で取り上げられています。

メンバーの除去

結果セットに包含するメンバー、または結果セットから除外するメンバーを指定できます。除外されたメンバーは、後続の操作で使用できなくなります。例えば、データ・パイプラインから「Cost」と「Units_Sold」を除去するには以下のようにします。

```
<bloxreport:members excluded="Cost, Units_Sold" />
```

OrderBlox で除外されたメンバーとは異なり、MembersBlox で除外されたメンバーは結果セットには含まれなくなります。結果セットにもう含まれていないので、ユーザーが対話式の列ヘッダー・コンテキスト・メニューから「すべて表示」を選択した場合、こうしたメンバーが戻されることはありません。

`<bloxreport:report>` タグに複数の `<bloxreport:members>` タグを追加することができます。各 MembersBlox で使用できるのは、`excluded` または `included` 属性のどちらかだけです。1 つの `<bloxreport:members>` タグに両方の属性を指定すると、後の属性が受け入れられて、前の属性は無視されます。

MembersBlox はメンバーの順序付けには対応しません。 メンバー A、B、C、D、および E が、この順序で結果セット内にあるとします。ここで次のようにするとします。

```
<bloxreport:members included="E, D, C" />
```

結果セット内のメンバーと順序は C、D、および E となります。メンバーの順序を指定するには、OrderBlox を使用します。データ・パイプラインからメンバーを除去しないで非表示にするには、45 ページの『メンバーの非表示と表示』の詳細情報を参照してください。MembersBlox タグ属性について詳しくは、127 ページの MembersBlox を参照してください。

メンバーの非表示と表示

OrderBlox を使用すると、結果セット内のメンバーをユーザーから隠すことができます。OrderBlox は、結果セットで一時的に包含あるいは除外するメンバーと、その順序を指定できます。<bloxreport:order> タグには、 excluded と included の 2 つの属性があります。excluded を使用すると、非表示にするメンバーを指定できます。included を使用すると、表示するメンバーを指定できます。メンバーを指定する順序が、表示順になります。

例えば、結果セット内に以下の 6 つのメンバーがあるとします。

Product	Location	Sales	Cost	Gross_Margin	Units_Sold

ユーザーから「Cost」と「Units_Sold」を隠すには、以下のようにします。

```
<bloxreport:order excluded="Cost, Units_Sold" />
```

OrderBlox を使用すると、結果セット内の列を再配列することもできます。レポートに含める列とその順序を指定するには、included 属性の値を、包含したいメンバーのコンマ区切りリストに設定します。メンバーは、指定する順序に従って、レポートで左から右に表示されます。以下に例を示します。

```
<bloxreport:order included="Product, Gross_Margin, Sales" />
```

上記のようにすると、Product、Gross_Margin、および Sales が結果セット内の最初の 3 列になります。他のすべてのメンバーは非表示になります。

<bloxreport:report> タグに複数の <bloxreport:order> タグを追加することができます。各 OrderBlox で使用できるのは、excluded または included 属性のどちらかだけです。1 つの <bloxreport:order> タグに両方の属性を指定すると、後の属性が受け入れられて、前の属性は無視されます。

除外されたメンバーは引き続き結果セットに含まれており、後続の操作で使用できます。<bloxreport:report> タグの interactive 属性を true に設定すると、ユーザーは、対話式の列ヘッダー・コンテキスト・メニューから「**すべて表示**」 を選択することにより、除外されたメンバーを再び表示することができます。

レポートから永久にメンバーを除去するには、44 ページの『メンバーの除去』を参照してください。欠落データを隠すことについては、46 ページの『欠落データの非表示と表示』を参照してください。OrderBlox タグ属性およびその使用法について詳しくは、 129 ページの OrderBloxを参照してください。以下の表には、MembersBlox と OrderBlox の類似点と相違点が要約されています。

MembersBlox	**OrderBlox**
結果セットからメンバーを除去できます。除外されたメンバーはデータ・パイプラインからなくなってしまうので、後続のデータ形式変更では使用できません。	ユーザーからメンバーを隠すことができます。除外されたメンバーは依然としてデータ・パイプラインにあるため、後続のデータ形式変更タスクで使用できます。
included 属性を使用しても、メンバーの順序は設定されません。	included 属性を使用すると、メンバーの順序が設定されます。

MembersBlox	OrderBlox
各 MembersBlox タグには、included か excluded 属性のどちらかしか含めることができません。	MembersBlox 同様、各 OrderBlox タグには、included か excluded 属性のどちらかしか含めることができません。
included タグ属性にないメンバーは除外されます (データ・パイプラインから永久に除外されます)。	included タグ属性にないメンバーは除外されます (ユーザーから一時的に隠されます)。

欠落データの非表示と表示

欠落データをフィルターに掛けて除外したい場合、FilterBlox にはこれを可能にする isMissing(*memberName*) 関数があります。以下の例では、Type (商品タイプ) および Gross Margin データが欠落していない行だけが戻されます。

```
<bloxreport:filter expression = "not isMissing(Type)"/>
<bloxreport:filter expression = "not isMissing([Gross Margin])"/>
```

同様に、売上データが欠落している行のみを表示する場合には、以下のように指定できます。

```
<bloxreport:filter expression="isMissiing(Sales)" />
```

FilterBlox は一度に 1 つの式しか持てません。データのフィルター操作について詳しくは、40 ページの『データのフィルター操作』を参照してください。FilterBlox 構文および使用法については、120 ページの FilterBloxを参照してください。

FormatBlox を使用すると、データが欠落している場合に必要なテキストを表示できます。<bloxreport:format> 内で <bloxreport:missing> タグをネストして使用すると、データが欠落している場合に指定のメンバーに対して表示するストリングを指定できます。

```
<bloxreport:format>
   <bloxreport:missing format = "Sales value missing" member = "Sales" />
   <bloxreport:missing format = "Units value missing" member = "Units" />
</bloxreport:format>
```

format 属性にはストリングしか指定できません。計算式や変数を値として割り当てることはできません。欠落データに表示するテキストを指定するには、StyleBlox を使用できます。StyleBlox を使用すると、特に欠落データまたは負の値のデータに使用するスタイルを指定できます。以下の例では、欠落データの表示用のフォント・スタイルおよびカラーを設定します。

```
<bloxreport:style>
   <bloxreport:missing style="font-style: italic;
      color: white;background-color: gray;"/>
</bloxreport:style>
```

FormatBlox を使用したデータ表示形式の設定について詳しくは、54 ページの『データのフォーマット設定』を参照してください。StyleBlox を使用したスタイルの設定については、145 ページの StyleBlox で構文および使用法を参照してください。

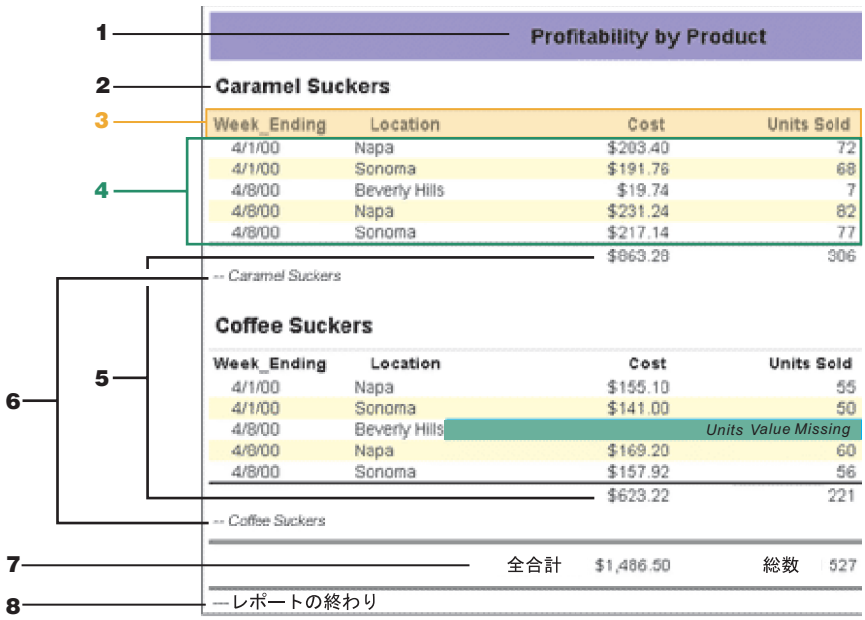
第 6 章 レポートとデータのフォーマット設定

この節では、列幅、フォント・カラーおよびスタイル、テキスト背景色、背景イメージ、または表示域のサイズなど、レポートの一般的なレイアウトのフォーマット方法を説明します。主に、データ・フォーマットの指定には FormatBlox、フォント・サイズ、テキスト/背景色、およびテキスト配置には StyleBlox、レポートの表示テキストの指定には TextBlox を使用します。これらの 3 つの Blox はデータ・トランスフォーマーではありません。データ形式変更タスクを実行した後の、レポート JSP の最後の部分に追加するのが一般的です。

Relational Reporting の Blox Sampler の『Formatting the Report and Data』の例は、この章で説明されている多くのタスクを示しています。

レンダリングされるレポートの表示域

リレーショナル・レポートは、いくつかの一般的な領域にレンダリングされます。どのレポートでも使用可能な表示域として、列ヘッダーとデータという 2 つの表示域があります。レポートがグループ化されると、グループ・ヘッダー、グループ・フッター、およびグループ合計域が使用可能になります。



上記の例の場合、レポートは Product ごとにグループ化されています。以下の領域が作成されます。

- 領域 1: レポート全体のレベル 1 のグループ・ヘッダー域
- 領域 2: 各 Product カテゴリーのレベル 2 のグループ・ヘッダー
- 領域 3: 列ヘッダー域 (グループ化しないレポートでも使用可能)
- 領域 4: データ域 (グループ化しないレポートでも使用可能)
- 領域 5: 各 Product カテゴリー内の数値データを集約した、レベル 2 のグループ合計域
- 領域 6: 各 Product カテゴリーのレベル 2 のグループ・フッター域
- 領域 7: レポート全体の数値データを集約した、レベル 1 のグループ合計域
- 領域 8: レポート全体のレベル 1 のグループ・フッター域

レポートを Product、次に Week_Ending の順でグループ化すると、レベル 3 が追加されます。これには、各週のデータのグループ・ヘッダー域、グループ・フッター域、およびグループ合計域が伴います。

レポートの各表示域では、TextBlox を使用して、表示するテキストの指定、列ヘッダーの名前変更、グループ・ヘッダー/フッター/合計テキストの設定を行ったり、さらには表示データやブレーク・グループ・メンバー名の変更や、HTML コードでこれをラップすることさえ可能です。さらに、StyleBlox を使用すると、各レポート域のスタイルも指定できます。

レポート・レイアウトのフォーマット設定とサマリー表のスタイル設定

以下の表に、レンダリングされたレポートの各領域のカスタマイズ方法の要約を示します。大括弧（ [ ] ）内の属性はオプションです。オプション属性を指定しない場合には、すべての列またはレベルにそのテキストあるいはスタイルが適用されます。

TextBlox を使用してテキストを設定する場合	**StyleBlox を使用してスタイルを設定する場合**	**CSS スタイル・クラスを使用してスタイルを設定する場合**
列ヘッダー		
columnHeader サブタグを使用する <bloxreport:text> <bloxreport:columnHeader text="*new column header*" columnName="*columnName*" /> </bloxreport:text>	columnHeader サブタグを使用する <bloxreport:style> <bloxreport:columnHeader style="*CSS style string*" [columnName="*columnName*"] /> </bloxreport:style>	.column
データ・セル		
data サブタグを使用する <bloxreport:text> <bloxreport:data text="*new data text*" [columnName="*columnName*"] /> </bloxreport:text>	data サブタグを使用する <bloxreport:style> <bloxreport:data style="*CSS style string*" [columnName="*columnName]*" /> </bloxreport:style>	.data
グループ・フッター		
groupFooter サブタグを使用する <bloxreport:text> <bloxreport:groupFooter text="group footer text" [level="*N*"] /> </bloxreport:text>	groupFooter サブタグを使用する <bloxreport:style> <bloxreport:groupFooter style="*CSS style string*" level="*N*" /> </bloxreport:style>	.groupfooter1, .groupfooter2, ...groupfooter*N*
グループ・ヘッダー		
groupHeader サブタグを使用する <bloxreport:text> <bloxreport:groupHeader text="group header text" [level="*N*"] /> </bloxreport:text>	groupHeader サブタグを使用する <bloxreport:style> <bloxreport:groupHeader style="*CSS style string*" level="*N*" /> </bloxreport:style>	.groupheader1, .groupheader2, ...groupheader*N*
グループ合計		
groupTotal サブタグを使用する <bloxreport:text> <bloxreport:groupTotal text="group total text" [level="*N*"] [columnName="*columnName*"] /> </bloxreport:text>	groupTotal サブタグを使用する <bloxreport:style> <bloxreport:groupTotal style="*CSS style string*" level="*N*" [columnName="*columnName*"] /> </bloxreport:style>	.grouptotal1, .grouptotal2, ...grouptotal*N*

この章ではさらに詳細なタスクを説明します。グループ・ヘッダー、フッター、および合計を使用できるかどうかは、レポートをグループ化するかどうか、およびその方法に依存しているので、69 ページの『第 7 章 データのグループ化』でもさらに詳しい内容が説明されています。

スタイル設定、フォーマット設定、テキスト設定の比較

しばしば「スタイル設定」と「フォーマット設定」という語は、交換可能な語として、タイトルの追加、列幅の指定、フォント・サイズ、カラー、配置の設定や、さらには数値データを何らかの方法でフォーマット設定することまで、あらゆる事柄に言及するのに使用されます。Relational Reporting は Cascading Style Sheet のクラスを使用してレポートを DHTML でレンダリングするので、スタイル関連のタスク (CSS の原理に基づいて行う) とデータ・フォーマット設定のタスク (CSS とは何の関係もない) を区別するのは重要なことです。

Relational Reporting の StyleBlox は、CSS スタイル・ストリングを使用して、レポート内のデータのスタイル設定を、データ・タイプまたはメンバーに基づいて行うことができます。例えば、「font-size: 85%; color: white; background-color: blue;」などの CSS スタイル・ストリングを使用して、数値データ、テキスト・データ、負のデータ値のスタイルを設定できます。

一方 FormatBlox を使用すると、Java のフォーマット・マスクに従って、日付および数値データのデータ表示フォーマットを指定できます。以下の表では、StyleBlox と FormatBlox の相違点を比較します。

StyleBlox	FormatBlox
さまざまなデータ・タイプおよびレポート域の表示スタイル (文字サイズ、フォント、カラー、配置、および背景色など) を設定する。	数値、日付、および欠落データの表示フォーマットを設定する。
スタイル指定は、CSS の原理に従う。	フォーマット仕様は Java フォーマット・マスクに従う。

StyleBlox	FormatBlox
ネストされたタグ: • numeric • date • missing • text • banding • negative • data (データのみ) • column (指定された列のデータとヘッダーの両方) • columnHeader (列ヘッダーのみ) • groupHeader • groupFooter • groupTotal 上記すべてのタグには、CSS スタイル・ストリングを指定する style 属性があります。	ネストされたタグ: • numeric: 数値フォーマットを指定する。例: $#,###.00 • date: 日付フォーマットを指定する。例: yyyy.MM.dd • missing: データが欠落している場合に表示するテキストを指定する。 • aggregation: 集約値 (グループ合計) のフォーマットを指定する。 上記すべてのタグには、フォーマット・ストリングを指定する format 属性があります。

TextBlox を使用すると、レンダリングされたレポートの 5 つの領域 (グループ・ヘッダー、グループ・フッター、グループ合計、列ヘッダー、およびデータ) の表示テキストを設定したり、現行のテキストを HTML コードでラップしたりできます。これには、StyleBlox と同じ 5 つのサブタグがあります。すなわち、columnHeader、data、groupHeader、groupFooter、および groupTotal です。こうしたタグを比較したサマリー表については、49 ページの『レポート・レイアウトのフォーマット設定とサマリー表のスタイル設定』を参照してください。

FormatBlox、StyleBlox、および TextBlox はデータ・トランスフォーマーではないので、これらを JSP に追加する順序がデータ・パイプラインに影響することはありません。メンバーをソート、フィルター操作、計算、非表示/除去、またはグループ化することによってデータのトランスフォームが済んだ後、これらの Blox のタグが処理されて、最終的なレポートがレンダリングされます。

StyleBlox、FormatBlox、および TextBlox の処理順序

データをパイプラインでトランスフォームした後、DB2 Alphablox Relational Reporting エンジン は以下の順序でレポートをレンダリングします。

1. まずフォーマット・マスクが値に適用されます (FormatBlox)。
2. フォーマット済みの値がテキストでラップされます (TextBlox)。
3. その後、スタイル設定されたセルが出力されます (StyleBlox)。

以下のようなコードがあるとします。

```
<bloxreport:format>
   <bloxreport:numeric format = "$#,###.00" />
</bloxreport:format>
```

```
<bloxreport:text>
  <bloxreport:groupTotal
    text = "Total: <value/>" />
</bloxreport:text>

<bloxreport:style>
  <bloxreport:groupTotal
    style = "color: green" />
</bloxreport:style>
```

結果として生成されるデータ・セルの HTML コードは次のようになります。

```
<td style="color: green">Total: $1,000.00</td>
```

こうしたタグは必ず前述の順序で処理されるので、タグを追加する順序が影響することはありません。TextBlox を使用するのはテキストを設定するときにだけにし、スタイル設定は StyleBlox に任せるのが重要です。TextBlox ではテキスト・セットに HTML コードを追加することができるので、スタイル関連以外のコードだけを追加するようにしなければなりません。というのは、スタイルは StyleBlox またはスタイル・シートによってオーバーライドされる可能性が高いからです。

レポートのスタイル設定について詳しくは、57 ページの『レポートに表示されるデータのスタイル設定』、および 59 ページの『列ヘッダーの指定とスタイル設定』を参照してください。StyleBlox の使用法およびタグ構文の詳細は、145 ページの StyleBloxで説明されています。

データ・フォーマットの詳細については、54 ページの『データのフォーマット設定』を参照してください。FormatBlox の使用法およびタグ構文の詳細は、122 ページの FormatBloxで説明されています。

テキストの設定について詳しくは、59 ページの『列ヘッダーの指定とスタイル設定』、および 71 ページの『ブレーク・グループ・ヘッダー、フッター、および合計の指定とスタイル』を参照してください。TextBlox の使用法およびタグ構文については、148 ページの TextBloxで説明されています。

StyleBlox と CSS スタイルの比較

リレーショナル・レポートは、`<alphablox_dir>/repository/theme/` 内の `report.css` で定義された一連のスタイルを使用してレンダリングされます。完全なリストが、12 ページの『スタイル・クラス』にあります。StyleBlox ではデータ・タイプまたはメンバーに基づいてデータおよび列ヘッダーをスタイル設定できますが、CSS スタイルはレポート全体を包含します。それには、レポート全体、ブレーク・グループ関連領域、すべての対話式メニュー、および「レポート・スタイル」ダイアログ・ボックスが含まれます。

レポート全体

リレーショナル・レポートにおいて最も外側に配置されるラップ HTML タグは `DIV` です。 `.report` クラスを使用すると、レポートの背景色、枠、埋め込み、および余白など、レポート全体の表示属性を定義できます。`.report` のデフォルト・スタイルは、背景色が白で、枠が 1 ピクセルの黒の実線です。

ブレーク・グループ

レポートが GroupBlox によってグループに編成されると、グループ・ヘッダー、フッター、および合計が使用可能になります。こうした領域にはそれぞれ対応するスタイル・クラスがあります。グループのレベルに応じて、以下のスタイル・クラスがあります。

- `.groupheader1`, `.groupheader2`, ..., `.groupheader`*N*
- `.groupfooter1`, `.groupfooter2`, ..., `.groupfooter`*N*
- `.grouptotal1`, `.grouptotal2`, ..., `.grouptotal`*N*

こうした領域のビジュアル表示については、47 ページの『レンダリングされるレポートの表示域』の例を参照してください。ブレーク・グループによるレポートのグループ化およびスタイル設定の詳細については、69 ページの『第 7 章 データのグループ化』を参照してください。

対話式コンテキスト・メニュー

すべての対話式コンテキスト・メニューはスタイル・クラスを使用してレンダリングされ、レポート全体の外観に合わせてカスタマイズすることができます。クラスには、 `.menu`、`.choice`、`.choicehover`、`.separator`、および `selected` が含まれます。

「レポート・スタイル」ダイアログ・ボックス

「レポート・スタイル」ダイアログは一連のスタイル・クラスを使用してレンダリングされ、アプリケーション全体の外観に合わせてカスタマイズすることができます。詳しくは、 89 ページの『第 9 章 Report Editor ユーザー・インターフェースのスタイル設定』を参照してください。

列およびデータ

レポート内の実データと列ヘッダーのためのクラスがあります。ここが、StyleBlox とスタイル・クラスがいくらか重複する点です。この 2 つが衝突する場合は、StyleBlox が優先します。以下に挙げるクラスはそれぞれ、列ヘッダーのスタイル、マウスを上で動かした場合の列ヘッダーのスタイル、選択されたときの列ヘッダーのスタイル、データのスタイル、およびバンディングのスタイルを定義します。

- `.column`: 列ヘッダー用
- `.columnhover`: マウスが上を移動するときの列ヘッダー用
- `.selected`: 選択時の列ヘッダー
- `.data`: データ行用
- `.banding`: 交互データ行用

StyleBlox とスタイル・クラスが重複する箇所では、`.column`、`.data`、および `.banding` スタイルを使用してください。以下の表は、重複とその動作の違いを示しています。

StyleBlox 内のネストされたタグ	CSS スタイル・クラス
`column`: スタイルは、指定されたメンバーのヘッダーおよびデータの両方に適用されます。	[なし]

StyleBlox 内のネストされたタグ	CSS スタイル・クラス
columnHeader: メンバーが指定されていない場合、スタイルはすべての列ヘッダーに適用されます。メンバーが指定されている場合、スタイルはその列ヘッダーにのみ適用されます。	.column: スタイルは、すべての列ヘッダーに適用されます。
data: メンバーが指定されていない場合、スタイルはすべてのデータに適用されます。メンバーが指定されている場合、スタイルはその列のデータにのみ適用されます。	.data: スタイルは、すべてのデータに適用されます。
banding: スタイルは、すべての交互データ行に適用されます。	.banding: スタイルは、交互データ行に適用されます。

注: これらのスタイルが両方の場所で指定されている場合には、StyleBlox で指定されたスタイルが適用されます。

データのフォーマット設定

データ・タイプまたはメンバーごとのデフォルト・データ・フォーマットを指定するには、FormatBlox を使用できます。FormatBlox は、以下のデータ・タイプの表示フォーマットの指定をサポートしています。

- 数値
- 日付 (および時間)
- 欠落データ
- 集約 (集約データ用。これはグループ合計とも言い、レポートがグループ化されており、かつ値が常に数値の場合に使用可能です)。

以下の例に示されているように、Java フォーマット・マスクを使用して、データ・フォーマットを指定できます。

```
<bloxreport:format>
   <bloxreport:numeric format="####.00;(####.00)" />
   <bloxreport:numeric format="$#,###.00;$(#,###.00)"
    member="Sales"    />
   <bloxreport:aggregation format="$#,###;$(#,###)"
    member="Sales"    />
   <bloxreport:date format="yyyy.MM.dd G 'at' hh:mm:ss z" />
   <bloxreport:date format="EEE, MMM d, ''yy" member="Date" />
   <bloxreport:missing format="Units Value Missing"
    member="Units" />
   <bloxreport:missing format="Sales Value Missing"
    member="Sales" />
</bloxreport:format>
```

このコード例は以下の設定を行います。

- 正の数値データのデフォルト・フォーマットを「####.00」に設定し、負の数値データのデフォルト・フォーマットを「(####.00)」に設定します。例えば、1234.5 は 1234.50 となり、-1234.5 は (1234.50) になります。
- メンバー「Sales」の数値データを「$#,###.00;$(#,###.00)」に設定します。例えば、1234.5 は $1,234.50 となり、-1234.5 は $(1,234.50) になります。

- 「Sales」のグループ合計を「$#,###;$(#,###)」に設定します。例えば、1234.5 は $1,235 に、-1234.5 は $(1,235) になります。
- 日付のデフォルト・フォーマットを「yyyy.MM.dd G 'at' hh:mm:ss z」に設定します。このフォーマットの一例は、2001.10.01 AD at 09:27:13 PDT です。
- メンバー「Date Member」の日付フォーマットを「EEE, MMM d,''yy」に設定します。このフォーマットの表示例は、Mon, October 1, '01 です。
- メンバー「Units」が欠落データの場合に表示されるテキストを「Units Value Missing」に設定します。
- メンバー「Sales」が欠落データの場合に表示されるテキストを「Sales Value Missing」に設定します。

フォーマット・マスクについて詳しくは、
http://java.sun.com/j2se/1.4.2/docs/api/java/text/DecimalFormat.html を参照してください。

<bloxreport:report> タグに追加できる <bloxreport:format> タグは 1 つだけです。ただし、<bloxreport:format> タグの中では、複数の <bloxreport:numeric>、<bloxreport:date>、および <bloxreport:missing> タグを使用できます。これにより、さまざまなメンバーにデータ・フォーマットを設定できます。

スタイルの追加、またはデータ値の周りを HTML コードでラップすることについては、次の節の 『データ値の周りを HTML コードでラップする』、および 57 ページの『レポートに表示されるデータのスタイル設定』を参照してください。

データ値の周りを HTML コードでラップする

レポート内のデータ値の周りに HTML コードを追加して、追加のテキスト、イメージ、スタイル、または他の関連情報へのリンクを追加することができます。TextBlox を使用すると、レポート内の 5 つの別個の領域 (そのうちの 1 つはデータ域) のテキストを設定できます。TextBlox は、2 つの特別な置換変数 <member/> および <value/> 以外は、テキスト・ストリング全体をブラウザーにそのまま送ります。

レンダリングされたレポートの表示域については、47 ページの『レンダリングされるレポートの表示域』で説明されています。TextBlox を使用すると、5 つの表示域 (列ヘッダー、グループ・ヘッダー、グループ・フッター、グループ合計、およびデータ) を名前変更したり、その周りを HTML コードでラップしたりすることができます。以下の例では、TextBlox のネストされた data タグを使用して、列「Location」内のデータ値を「Not to be disclosed」というストリングと置換する方法を示します。

```
<bloxreport:text>
  <bloxreport:data
    columnName = "Location"
    text = "Not to be disclosed"
  />
</bloxreport:text>
```

ただし多くの場合、値を置換するのではなく、値の周りを HTML コードでラップすることができます。以下の例では、列「Location」内のデータ値を、それぞれの場所のより詳細な情報を表示するハイパー・リンクにする方法を示します。

```
<bloxreport:report id="myReport">
  ...
  <bloxreport:text>
    <bloxreport:data
      columnName="Location"
      text="<a href=¥"javascript:getURL('<value/>')¥"><value/></a>" />
  </bloxreport:text>
  ...
</bloxreport:report>
```

<value/> は、TextBlox がテキスト・ストリングをブラウザーに送る際に実データ値に置換する置換変数 (JSP タグではない) です。TextBlox タグ内では、複数の data タグを使用できます。変更したいデータ値のある各列に、ネストされた data タグを追加します。

getURL() JavaScript 関数は、以下のようになります。

```
function getURL(location) {
   shortName = location.substr(0,product.indexOf(" "));
   url="http://myserver/sites/" + shortName + ".html";
   open(url);
}
```

ユーザーが「Beverly Hills」をクリックすると、JavaScript getURL("Beverly Hills") が呼び出され、url は http://myserver/sites/Beverly.html という値を取得します。

TextBlox、そのネストされたタグ、および置換変数について詳しくは、148 ページの TextBlox、および 62 ページの『メンバー名および値を表示するための特殊な置換変数』を参照してください。

照会から戻されたデータへの HTML コードの追加

照会から戻されたデータに HTML コードを追加する別の方法は、FormatBlox の <bloxreport:html> というネストされたタグを使用して、HTML をエンコードしないよう FormatBlox に指示することです。

```
<%
   // This example always starts with a new ReportBlox. So remove the
   // old one if it exists.
 BloxContext context = BloxContextFactory.getBloxContext(pageContext.getRequest(),
     pageContext.getResponse());
 Blox blox = context.getBlox("htmlLinkReport");
 if(blox != null) {
  context.deleteBlox(blox);
 }

   Query = "Select Week_Ending, Area, '<a href=info.jsp?location='
   + Loc + '>' + Loc + '</a>' as 'Location', Product" +
   "FROM qcc";
%>

<bloxreport:report id="htmlLinkReport" interactive="true">
   <bloxreport:sqlData
      query="<%= Query %>">
   <bloxreport:dataSourceConnection
   dataSourceName="myRDB" />
</bloxreport:sqlData>

<bloxreport:format>
   <bloxreport:html member="Location" />
</bloxreport:format>
```

Location 列に戻されるデータは、HTML コードを維持します。ユーザーが「Sonoma」リンクをクリックすると、ページの要求 moreInfo.jsp?location=Sonoma が発行されます。照会の中で、メンバーに別名 「Location」が与えられていることに注意してください。DB2 Alphablox Relational Reporting エンジンは「HTML を放っておき」、これをエンコードしないように指示されているので、<bloxreport:html> タグでメンバーを参照できるよう、メンバーの表示名が必要です。

レポートに表示されるデータのスタイル設定

フォント書体、サイズ、スタイル、カラー、テキスト配置、または背景色などのデータ表示スタイルを指定するには、StyleBlox または Cascading Style Sheet を使用します。

リレーショナル・レポートは、Relational Reporting が使用する各スタイル・クラスに指定したスタイルに基づいて表示されます。データ表示に関連するスタイル・クラスは、以下のとおりです。

- .data
- .banding

以下の表は、こうしたクラスの使用法を示しています。

例	効果
.data { font-size: 11; text-align: right; padding-left: 20; color: black; background-color: white; }	すべてのデータ・セルが指定されたスタイルで表示されます。
.banding { background-color: #CCCCFF; }	交互データ行が、指定されたスタイルで表示されます。

レポートのスタイルを設定するのに使用可能なクラスの完全なリストについては、12 ページの『スタイル・クラス』を参照してください。

欠落データ、負のデータ値、または指定されたデータ・タイプやデータ列の表示スタイルを設定するには、StyleBlox を使用できます。StyleBlox を使用すると、以下のデータ・タイプの表示スタイルを指定できます。

データ・タイプ	<bloxreport:style> 内のネストされたタグ	効果
日付	<bloxreport:date style="yourStyle"/>	日付またはタイム・スタンプの列がすべて、指定されたスタイルで表示されます。
数値	<bloxreport:numeric style="yourStyle" />	数値データの列がすべて、指定されたスタイルで表示されます。
テキスト	<bloxreport:text style="yourStyle" />	テキストの列がすべて、指定されたスタイルで表示されます。

レンダリングされたレポートにおけるデフォルトのテキストの配置を、データ・タイプごとに以下に示します。

データ・タイプ	テキストの配置
日付	左
数値	右
テキスト	左

以下の例に示されているように、StyleBlox を使用してデフォルトを上書きできます。

```
<bloxreport:style>
   <bloxreport:text style="text-align: center" />
</bloxreport:style>
```

さらに、StyleBlox によって以下のレポート・データおよびエレメントの表示スタイルを設定できます。

- セル・バンディング
- 欠落データ
- 負のデータ値
- 指定された列

対応するタグとその効果について、以下の表で説明します。

レポート・データまたはエレメント	<bloxreport:style> 内のネストされたタグ	効果
セル・バンディング	<bloxreport:banding style="yourStyle"/>	交互データ行が、指定されたスタイルで表示されます。 **注:** 交互データ行のスタイルは、.banding スタイル・クラスを使用して指定することもできます。ただし、StyleBlox を使用したスタイル設定が、スタイル・クラスによるスタイル設定より優先されます。
欠落データ	<bloxreport:missing style="yourStyle"/>	すべての欠落データ (または NULL データ) が、指定されたスタイルで表示されます。
負のデータ値	<bloxreport:negative style="yourStyle"/>	レポート内のすべての負のデータ値が、指定のスタイルで表示されます。
指定された列	<bloxreport:column style="yourStyle" member="memberName"/>	指定された列のデータおよび列ヘッダーの両方が、指定されたスタイルで表示されます。

例えば、以下の例のようになります。

```
<bloxreport:style>
   <bloxreport:banding style="background-color: #FFCCFF" />
   <bloxreport:missing style="background-color: aqua" />
```

```
   <bloxreport:negative style="background-color: red" />
   <bloxreport:column style="color: yellow" member="Country"/>
   <bloxreport:column style="color: blue" member="State"/>
</bloxreport:style>
```

- 交互データ行の背景色は #FFCCFF です。
- 欠落データのセルの背景色は水色に設定されます。
- 負のデータのセルの背景色は赤に設定されます。
- 「Country」メンバーのテキスト・カラーは黄色に設定されます。
- 「State」メンバーのテキスト・カラーは青に設定されます。

注: レポートに含めることのできる StyleBlox は 1 つだけです。複数の StyleBlox を含めた場合、レポートに対して効果があるのは最後の StyleBlox だけです。しかし、タグをネストする方法で、複数の `column`、`missing` 、`negative`、または `banding` タグを指定でき、そのようにネストされたタグを使用して設定されるスタイルはすべてが有効になります。

列ヘッダーの指定とスタイル設定

データベースのフィールド名とは異なる列ヘッダーを指定するには、「`SELECT FROM myTable column1 AS newColumnName1, column2 AS newColumnName2...`」などの、データを抽出する SQL ステートメントを使用することができます。この場合、ReportBlox に対し、メンバーは newColumnName1 および newColumnName2 として認識されます。メンバー名には大/小文字の区別があるので、こうしたメンバーを後で参照する場合には、大/小文字の区別をする必要があります。

注: Oracle は、新規名をすべて大文字で戻します。これを避けるには、以下のようにしてメンバー名を引用符で囲みます。

```
"SELECT FROM myTable total_sq_ft AS ¥"Sq_Ft"¥, sq_ft_pct AS ¥"Pct¥""
```

別の方法は、TextBlox を使用することです。TextBlox を使用すると、レンダリングされたレポートの 5 つの領域 (列ヘッダー、データ、およびレポートがグループ化されている場合にはグループ・ヘッダー、グループ・フッター、およびグループ合計) の表示テキストを指定できます (レポート上のこうした領域については、47 ページの『レンダリングされるレポートの表示域』を参照してください)。
`<bloxreport:text>` タグには、5 つの各領域でのテキストを指定するための、または値/メンバーの周囲に HTML コードを追加するためのネストされたタグがあります。

列ヘッダーを指定するには、TextBlox の `columnHeader` というネストされたタグを使用します。このタグには、2 つの属性があります。

- `columnName`— 必須。この列のメンバー名。
- `text`— 必須。列ヘッダーに表示するテキスト。ここで指定するテキストはすべて、ブラウザーに送られて処理されます (追加する HTML コードも含む)。

以下の例では、メンバー「Sales」の列ヘッダーを「Product Sales」に、メンバー「Units」の列ヘッダーを「Units Sold」に設定します。

```
<%@ taglib uri="bloxreporttld" prefix="bloxreport" %>
<bloxreport:report id = "MyReport">
...
   <bloxreport:text>
```

```
      <bloxreport:columnHeader
         columnName="Sales">
         text="Product Sales" />
      <bloxreport:columnHeader
         columnName="Units"
         text="Units Sold" />
   </bloxreport:text>
...
</bloxreport:report>
```

フォーマット、リンク、またはイメージを追加するため、列ヘッダーの周りを HTML コードでラップすることができます。以下の例では、次の事柄が実行されます。

- メンバー Cost の列ヘッダーを Unit Cost に変更します。
- ヘッダーから myApp アプリケーション内の別の URL へ飛ぶリンクを追加し、コストの追加情報を表示するようにします。
- このレポート JSP と同じディレクトリーにあるイメージ info.gif を追加します。

```
<bloxreport:text>
  <bloxreport:columnHeader
    columnName="Cost"
    text="<a href=¥"/myApp/products/CostList.html¥">
       <img src="info.gif">Unit Cost</a>" />
</bloxreport:text>
```

注: text 属性のテキスト・ストリングは、改行を含めずに 1 行に収めなければなりません。複数行に分割されているのは、印刷ページの幅に限りがあるためです。オンライン・バージョンからこのコードをコピーして貼り付ける場合には、改行を削除してください。

実際の例については、Blox Sampler-Relational Reporting の『Saving and Exporting Data with Dynamic Query』の例を参照してください。

TextBlox は列ヘッダーの HTML コードを保持するので、ヘッダーをフォーマットするために標準の HTML を使用できます。URL およびイメージ・パスは相対パスでも絶対パスでもかまいません。

- 絶対 URL の場合、ストリングの先頭を「http://」にします。
- 相対 URL の場合
  - スラッシュ (/) で始まるストリングは、URL がサーバーのルートに対して相対的であることを示します。 アプリケーション・コンテキストは、URL に含める必要があります。
  - スラッシュなしで始まるストリングは、URL が現行の文書に対して相対的であることを示します。

メンバー名を変更しないでその周囲に HTML コードを配置するには、以下の例に示すように、<member/> 置換変数を使用します。

```
<bloxreport:text>
   <bloxreport:columnHeader
      columnName="Cost"
      text="<a href=¥"/myApp/products/CostList.html¥">
         <img src="info.gif"><member/></a>"/>
</bloxreport:text>
```

<member/> は置換変数で、リレーショナル・レポート HTML をブラウザーに送る際にメンバー名に置換されます。これは JSP タグではありません。

重要: レポートを対話モードでレンダリングした場合、ユーザーが列ヘッダー・コンテキスト・メニューを使用して列ヘッダーを変更しようとすると、すべての HTML コードが表示されます。これは、望ましいことではない可能性があります。さらに、ユーザーは HTML コードとフォーマット設定を簡単に上書きできます。HTML コードで列ヘッダーやフッターの周りをラップした場合には、ReportBlox の interactive 属性を false に設定することをお勧めします。

重要: text 属性のテキスト・ストリングは、改行を含めずに 1 行に収めなければなりません。複数行に分割されているのは、印刷ページの幅に限りがあるためです。オンライン・バージョンからこのコードをコピーして貼り付ける場合には、改行を削除してください。

ヒント: TextBlox を使用してストリングのスタイル設定を追加しないようにしてください。スタイルは、データのフォーマット設定と、テキストへの埋め込みが済んでから、DB2 Alphablox Relational Reporting エンジンによって最後に出力されます。TextBlox のネストされたタグの text 属性を使用してスタイルを設定するのは、スタイル・シートおよび StyleBlox で設定したスタイルによってオーバーライドされることを考えると、効果的ではなく、混乱を招く恐れがあります。

ヒント: 対話式レポートで列ヘッダー・テキスト (またはグループ・ヘッダー、フッター、および合計) の外側をアンカー・タグでラップすると、列ヘッダーの上にマウスが来ても、コンテキスト・メニューがポップアップ表示しません。セル内の別の場所 (リンク以外の場所) でマウスを動かさないと、メニューはポップアップ表示しません。これはブラウザーの動作によるものです。レポートを設計する際は、このことを念頭に置いてください。

列ヘッダーのスタイル設定

列ヘッダーをスタイル指定するには、StyleBlox の columnHeader というネストされたタグを使用します。このタグには、2 つの属性があります。

- columnName— オプション。この列のメンバー名。
- style— 必須。列ヘッダーに適用されるスタイル。

以下の例では、「Sales」の列ヘッダー・スタイルを中央揃えにします。

```
<%@ taglib uri="bloxreporttld" prefix="bloxreport" %>
<bloxreport:report id = "MyReport">
...
   <bloxreport:style>
      <bloxreport:columnHeader
         columnName="Sales">
         style="text-align: center;" />
   </bloxreport:style>
...
</bloxreport:report>
```

TextBlox のネストされたタグ columnHeader では、列ヘッダー・テキストの周りを HTML コードでラップできることを思い出してください。スタイルは、データのフ

ォーマット設定と、テキストへの埋め込みが済んでから、DB2 Alphablox Relational Reporting エンジンによって最後に出力されるので、TextBlox を使用してストリングのスタイルを追加しないようにしてください。TextBlox のネストされたタグの text 属性を使用してスタイルを設定するのは、スタイル・シートおよび StyleBlox で設定したスタイルによってオーバーライドされることを考えると、効果的ではなく、混乱を招く恐れがあります。

列幅、カラーおよびスタイルの指定

StyleBlox の <bloxreport:column> タグを使用すると、名前を指定した列のデータおよび列ヘッダーの両方について、幅、背景色、およびフォント・スタイルを指定できます。通常 HTML 表のセルにするのと同様に、各スタイル属性と値の組をセミコロン (;) で区切って、スタイルを指定します。

```
<bloxreport:style>
   <bloxreport:column style="font-weight: bold; color: white;
      background-color: gray" member="Type" />
   <bloxreport:column style="width:20px; background-color:
      #99ffcc; font-weight: bold" member="Product" />
</bloxreport:style>
```

表のセルに指定できる標準的な属性には、以下が含まれます。

- 背景色
- フォント書体、サイズ、スタイル、太さ、およびカラー
- テキストの配置: 左、右、中央
- 枠のカラー

グラフィカル・ユーザー・インターフェースのあるスタイル・シート・エディターまたは HTML エディターを使用すると、好みのスタイルにしやすくなります。

列ヘッダーのスタイルを指定するには、以下のようにします。

- .column クラスのスタイルを指定します。これは、すべての列ヘッダーに適用されます。
- 独自のスタイル・クラスを指定して、ネストされた <bloxreport:columnHeader> タグを使用して指定の列ヘッダーに適用します。

詳細およびコード例については、59 ページの『列ヘッダーの指定とスタイル設定』を参照してください。

メンバー名および値を表示するための特殊な置換変数

列上のメンバーの名前、ブレーク・グループ・メンバー名、またはメンバーの値を取得するには、2 つの特別な置換変数を使用できます。列ヘッダーの周囲に HTML コードを追加する場合、ブレーク・グループ・メンバー名を動的に表示する場合、指定のメンバーの値を取得する場合、または別の列のデータを表示する場合に、こうした置換変数が役立ちます。以下の節で、この 2 つの変数について説明します。

<member/> 置換変数

<member/> 変数は、現行のメンバー名を現在場所で置換するのに使用します。TextBlox のネストされたタグすべての内側で有効です。オプションの属性が 1 つあ

ります。これは level という属性で、現在レベルまたは上位レベルのブレーク・グループ・メンバーを参照するためのものです (*level* <= *currentLevel*)。

例:

- TextBlox のネストされたタグ groupHeader の内側で使用して、現在レベルまたは上位レベルのブレーク・グループ・メンバー名を抽出する場合。

```
<bloxreport:report ...>
  <bloxreport:text>
     <bloxreport:groupHeader level="1"
       text="Sales Report by <member/>" />
     <bloxreport:groupHeader level="2"
       text="<i><member/></i>" />
  </bloxreport:text>
</bloxreport:report>
```

- TextBlox のネストされたタグ columnHeader の内側で使用して、列ヘッダーの名前を抽出する場合。

```
<bloxreport:report ...>
  <bloxreport:text>
    <bloxreport:columnHeader columnName="Product"
       text="<span class="myStyleclass"><member/></span> />
  </bloxreport:text>
</bloxreport:report>
```

- TextBlox のネストされたタグ groupFooter の内側で使用して、現在レベルまたは上位レベルのブレーク・グループ・メンバー名を抽出する場合。

```
<bloxreport:report ...>
  <bloxreport:text>
    <bloxreport:groupFooter level="2"
       text="Group footer for Level 2: by <member level="¥1¥"/>,
<member/>" />
  </bloxreport:text>
</bloxreport:report>
```

現在レベルまたは上位レベルのブレーク・グループ・メンバーしか参照できません。下位レベルを指定すると、タグ・ストリング全体がテキストとして扱われ、変数置換は行われません。レベルを指定しないと、現行レベルが暗黙のうちに適用されます。例えば、レベル 2 のブレーク・グループ・フッターはレベル 2 およびレベル 1 のブレーク・グループ・メンバーのみを参照し、レベル 3 は参照しません。

注: テキスト・ストリング内に二重引用符を含める場合はエスケープする必要があります (″¥″)。

<value/> 置換変数

<value/> 変数は、現行のメンバーまたは指定されたメンバーの値を置換するのに使用します。data、groupTotal、groupFooter、および groupHeader タグの内側で有効です。別の列の値を現行列のデータに抽出したり、別のメンバーのグループ集約データ (グループ合計) を現行列のグループ集約データに抽出したりするのにも使用できます。

例:

- TextBlox のネストされたタグ groupTotal の内側で使用して、ブレーク・グループの集約値を抽出する場合。

```
<bloxreport:report ...>
  <bloxreport:text>
    <bloxreport:groupTotal level="1"
      columnName="products"
      text="<i>Total:</i> <value />" />
  </bloxreport:text>
</bloxreport:report>
```

- TextBlox のネストされたタグ groupHeader の内側で使用して、指定するメンバーのグループ合計を抽出する場合。

```
<bloxreport:report ...>
  <bloxreport:text>
    <bloxreport:groupHeader level="1"
      columnName="products"
      text="<div align=¥"left¥"><member/></div>
      <div align=¥"right¥">Rank:<value member=¥"RankbyProduct¥" /
></div>" />
  </bloxreport:text>
</bloxreport:report>
```

- TextBlox のネストされたタグ data の内側で使用して、列のデータ値を抽出する場合。

```
<bloxreport:report ...>
  <bloxreport:text>
    <bloxreport:data
      columnName ="products"
      text="<a href=¥"javascript:getURL();¥"><value/></a>" />
  </bloxreport:text>
</bloxreport:report>
```

別のメンバーの値を取得するには、メンバー名を指定します。例えば、以下のようにします。

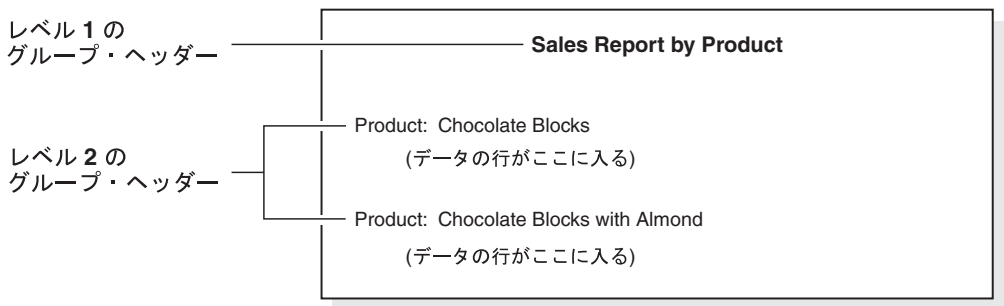
```
<value member="Sales" />
```

メンバー名が認識されないまたは存在しないと、タグ・ストリング全体がテキストとして扱われ、変数置換は行われません。

注: テキスト・ストリング内に二重引用符を含める場合はエスケープする必要があります (″¥″)。

<member/> および <value/> 変数の使用

以下は、レベル 1 のブレーク・グループ・メンバーを参照するレベル 2 のグループ・ヘッダーの例です。



以下のコードで実行されます。

```
<bloxreport:report ...>
  <bloxreport:text>
     <bloxreport:groupHeader level="1"
       text="Sales Report by <member/>" />
     <bloxreport:groupHeader level="2"
       text="<member level="1"/>: <member/>" />
  </bloxreport:text>
</bloxreport:report>
```

<member/> 変数には、level 属性があります。level を指定しないと、groupHeader、groupFooter または groupTotal タグで指定された現行のグループ化レベルが暗黙のうちに使用されます。現行のグループ化レベルまたは上位レベルのメンバーのみを参照できます。例えば、レベル 2 グループ・ヘッダーはレベル 2 とレベル 1 グループ・メンバーのみを参照でき、レベル 3 は参照できません。レベルを 3 に設定すると、タグ・ストリング全体はテキストとして扱われ、変数置換は行われません。

<value/> 変数には、member 属性があります。member を指定しないと、列の現行メンバーが暗黙のうちに適用されます。groupHeader タグの場合、抽出するグループ化レベルの集約値を持つ member を指定してください。メンバーの指定がないと、DB2 Alphablox Relational Reporting エンジンは使用する集約値を判別できないので、<value/> 置換変数は無視されます。

グループ・ヘッダー、フッター、および合計はレポートがグループ化されている場合にのみ使用可能なので、詳細については 69 ページの『第 7 章 データのグループ化』、および 71 ページの『ブレーク・グループ・ヘッダー、フッター、および合計の指定とスタイル』という節で説明されています。

これらの変数の有用性を示す実例については、Relational Reporting Blox Sampler の『Formatting the Report and Data』および『Grouping and Adding Group-based Summary Columns』セクションの例を参照してください。

セル・バンディングの設定または停止

データ表示に関連するスタイル・クラスは、.data および .banding です。デフォルト・スタイル・シートには、異なる背景色を設定する 2 つのクラスがあるので、データ行は互い違いのカラーで表示されます。バンディング・カラーを設定するには、.banding スタイル・クラスのスタイルを指定するか、StyleBlox のネストされたタグ banding を使用します。

```
<style>
   .banding { background-color: #ffffcc; }
</style>
```

または

```
<bloxreport:style>
   <bloxreport:banding style="background-color: #ffffcc;" />
</bloxreport:style>
```

セル・バンディングを無効にするには、両方のクラスに同じスタイルを設定します。例えば、以下のようにします。

```
.data, .banding {
   background-color: white;
}
```

以下の点に注意してください。

- StyleBlox を使用したスタイル設定が、スタイル・クラスによるスタイル設定より優先されます。
- 対話式レポートでは、ユーザーは (コンテキスト・メニューの「スタイル...」オプションから)「レポート・スタイル」ダイアログ使用して、列内のデータ・セルまたはレポート内のすべてのデータ・セルのスタイルを設定できます。この設定は、JSP で指定したセル・バンディング設定をオーバーライドします。「レポート・スタイル」ダイアログはレンダリングされたレポートの既存のスタイルを認識しないため、特定の列 (すべてのデータ・セルに適用されるスタイル設定になっている場合はすべての列) のデータの背景色をダイアログで指定すると、既存のセル・バンディングはその背景色で置換されます。

レポート表示域の設定

リレーショナル・レポートは Cascading Style Sheet を使用して表示されるので、.report クラスを使用してページ上の表示域を指定できます。場合によっては、レポートをブラウザー・ウィンドウ全体に表示するのではなく、特定ページの指定位置の一部に表示したいこともあります。JSP ページが参照するスタイル・シートでそうすることもできますし、JSP ページの中でさえ、そのようにインライン指定できます。例えば、次のようにします。

```
<head>
...
   <style>
      .report { height: 400; width: 500 }
   </style>
</head>
```

このようにすると、レポートは、ブラウザー・ウィンドウの内側のスクロール可能な 400 X 500 のボックスとして表示されます。これにより、ページ・レイアウトをより強力に制御できます。例えば、同一ページ上に、ReportBlox と、それに続く GridBlox とテキストを含めることができます。

重要: すべてのブラウザーが、Cascading Style Sheet を完全にサポートしているわけではありません。DHTML 関連の一部の技法は Netscape ブラウザーには作用しない場合があります。Web には、CSS を複数のブラウザーで使用するヒントを記載した資料が多数あります。

背景イメージの追加

背景イメージを追加するには、イメージをスタイル・シート内の .report クラスに追加します。

```
.report { height: 300; width: 450;
 background: white url(background.gif) no-repeat
 fixed center
}
```

こうすると、レポートの表示域から見て中央の位置にイメージが追加されます。

重要: すべてのブラウザーが、Cascading Style Sheet を完全にサポートしているわけではありません。DHTML 関連の一部の技法は Netscape ブラウザーには作用しない場合があります。Web には、CSS を複数のブラウザーで使用するヒントを記載した資料が多数あります。

第 7 章 データのグループ化

この節では、データのグループ化に密接に関連したタスクについて説明します。こうしたタスクには、ブレーク・グループの作成、ブレーク・グループごとの集約データの追加、グループ・ベースのサマリー・データを提供する算出列の追加、および各ブレーク・グループ・レベルのブレーク・グループ・ヘッダー、フッター、および合計に対するレポートのフォーマットなどが含まれます。主に、ブレーク・グループの追加には GroupBlox、レポートにおける様々な表示域のテキストの指定には TextBlox、さらにグループ・ベースのサマリー・データ列の追加には CalculateBlox の使用がそれぞれ関係しています。

Relational Reporting の Blox Sampler の『Grouping and Adding Group-based Summary Columns』セクションには、この章で取り上げられるほとんどのタスクが示されている例があります。

- 69 ページの『ブレーク・グループおよびブレーク・グループ・レベルの概説』
- 71 ページの『ブレーク・グループ・ヘッダー、フッター、および合計の指定とスタイル』
- 74 ページの『グループ・ベースのサマリー列の計算』
- 76 ページの『レポート・タイトルおよび列サマリー (集約) の追加』
- 76 ページの『GroupBlox とともに MembersBlox を使用する場合』

ブレーク・グループおよびブレーク・グループ・レベルの概説

しばしばレポートは、読みやすくするため、また簡単にデータを解釈したり、比較したりするためにグループ化されます。レポートをグループ化すると、次のことが可能になります。

- グループ・ヘッダー、フッター、および合計/集約域が使用可能になります。こうした各領域にテキスト・スタイルと表示スタイルを指定できます。
- 指定されたブレーク・グループ・レベルに基づいてグループ・ベースの計算 (ランキング、合計のパーセント、累計値、およびカウント値) を追加できます。

GroupBlox を使用すると、メンバーを指定することによって、レポートをグループ化できます。そのメンバーの値がブレーク・グループとして使用されます。例えば、以下のコードは「Product」ごとにデータをグループ化します。

```
<bloxreport:group members="Product" />
```

ブレーク・グループを追加すると、レベルが自動的に追加されます。レベル 1 はすべてのデータを含む全体レベルで、レベル 2 は個々のブレーク・グループのためのレベルです。前述の例では、レポートのタイトルはレベル 1 のグループ・ヘッダーに、また各「Product」グループはレベル 2 のグループ・ヘッダーになります。レポートのこうした領域については、47 ページの『レンダリングされるレポートの表示域』を参照してください。

グループ化されていないレポートには、列ヘッダーとデータ領域しかありません。何らかの方法でレポートをグループ化すると、グループ・ヘッダーおよびグルー

プ・フッターが使用可能になります。さらに、ブレーク・グループごとに、各数値列の集約値、またはグループ合計が使用可能です。デフォルトの集約タイプは、sum です。ネストされた <bloxreport:aggregation> タグを使用して、集約値を指定できます。以下の例では、「Product」でグループ化したレポートを示しています。各製品の販売場所の数と販売個数の平均を示します。

```
<bloxreport:group members="Product">
   <bloxreport:aggregation member="Location" type="count" />
   <bloxreport:aggregation member="Units" type="average" />
</bloxreport:group>
```

ヒント: グループ化しないで、各列の集約値を持つレポートを作成したり、レポートの「タイトル」を作成したりするには、グループ・メンバーを空ストリングに設定して GroupBlox を追加します。

```
<bloxreport:group members="" />
```

以下の点に注意することが重要です。

- グループ化メンバーを指定する順序は重要です。members 属性にメンバーを指定するたびに、グループ化のレベルが 1 つ追加されます。
- *GroupBlox はグループ化のみを扱い、ソートの指定は含まれません。*前述の例の場合、「Product」上のデータを最初にソートしない限り、「Product」ブレーク・グループがアルファベット順で表示されることはありません。ソートについては、39 ページの『データのソート』を参照してください。

ブレーク・グループ集約

グループ化されたレポートでは、数値データの集約値が自動的に使用可能になります。集約値はレポートのグループ合計域に表示されます。以下の集約タイプがサポートされます。

- average
- count
- max
- min
- none
- sum

別の指定を行わない限り、デフォルトでは、数値データの集約タイプは sum です。集約したくない場合には、タイプを none に設定します。非数値列の場合、デフォルトの集約値は欠落データとして扱われます。非数値データで唯一有効な集約タイプは count です。

集約を計算するときに欠落データは無視されます。以下の例は、所定のデータに対する集約値を示しています。

列のデータ	各集約値
1	• sum: 15 • average: 3 • count: 5 • max: 5 • min: 1
2	
3	
4	
5	
欠落	

以下の表は、グループ合計に関連した種々のタスクで使用する Blox とタグを示しています。

タスク	ソリューション
スタイルの設定	CSS スタイル・クラス: grouptotal1, .grouptotal2 , ..., .grouptotal*N*

	StyleBlox の groupTotal サブタグ:
	`<bloxreport:style>`
	`  <bloxreport:groupTotal level="N"`
	`  style="color: red" />`
	`</bloxreport:style>`

データ・フォーマットの指定	FormatBlox の集約サブタグ:
	`<bloxreport:format>`
	`  <bloxreport:aggregation format="$#,###" />`
	`</bloxreport:format>`

表示するグループ合計の変更	TextBlox のネストされたタグ groupTotal :
	`<bloxreport:text>`
	`  <bloxreport:groupTotal`
	`  columnName="Cost"`
	`  **text="Total: <value/>" />`**
	`</bloxreport:text>`

詳しくは、71 ページの『ブレーク・グループ・ヘッダー、フッター、および合計の指定とスタイル』、および 54 ページの『データのフォーマット設定』を参照してください。

ブレーク・グループ・ヘッダー、フッター、および合計の指定とスタイル

GroupBlox を使用してレポートをグループ化すると、グループ・ヘッダー、グループ・フッター、およびグループ合計の 3 つのレポート表示域が使用可能になります。グループ化のレベルを追加するたびに、新たなレベルのグループ・ヘッダー、フッター、および合計の各領域が使用可能になります。それぞれの領域を示した図については、69 ページの『ブレーク・グループおよびブレーク・グループ・レベルの概説』を参照してください。

例えば、「Product」でグループ化したレポートで、レポート・タイトルを「Profitability by Product」とするとします。各製品グループには、グループ・ヘッダー、グループ合計 (各数値データ列につき 1 つずつ)、およびグループ・フッターがあります。グループ・ヘッダーおよびフッターは、TextBlox のネストされたタグ groupHeader および groupFooter タグを使用して指定します。

```
<bloxreport:report id="anotherSampleReport" ...>
  ...
  <bloxreport:text>
    <bloxreport:groupHeader level="1"
      text="Profitability by <member/>" />

    <bloxreport:groupFooter level="1"
      text="--End of Report" />

    <bloxreport:groupTotal columnName="Cost"
      text="Grand Total Cost: <value/>" />

  </bloxreport:text>
</bloxreport:report>
```

レベル 1 は最も外側のレベルです。グループ化のレベルを追加するたびに、レベルが 1 つ追加されます。例えば、レポートを「Product」、次に「Country」でグループ化する場合、レベル 1 はレポート全体を表現し、レベル 2 は各「Product」ブレーク・グループを表現し、レベル 3 は各「Country」ブレーク・グループを表現します。この同じ規則は、groupHeader、groupFooter、および groupTotal タグにも適用されます。

TextBlox は、指定されたテキスト・ストリング全体をブラウザーにそのまま送ります。ただし、2 つの特殊な置換変数 <member/> および <value/> は例外です。これらの変数は、実際のメンバー名またはデータ値で置換されます。

以下の点に注意してください。

- テキスト・ストリングに二重引用符が含まれる場合は、これをエスケープする必要があります (″¥″)。
- TextBlox を使用してストリングのスタイル設定を追加しないようにしてください。TextBlox のネストされたタグの text 属性を使用してスタイルを設定するのは、スタイル・シートおよび StyleBlox で設定したスタイルによってオーバーライドされることを考えると、効果的ではなく、混乱を招く恐れがあります。
- 付属のスタイル・シートでは、デフォルトで、レベル 1 から 3 までのグループ・フッターが使用不可になっています。これは次のことを意味します。
  - レベル 1 から 3 までのグループ・フッターを表示したい場合には、ページまたはスタイル・シートのスタイル・クラスを上書きすることが必要です。.groupfooterN { display: inline; }
  - ブレーク・グループのレベルが 4 つ以上ある場合、レベル 4 以上のグループ・フッターは、レベル 1 から 3 までのフッターが表示されない場合でも表示されます。レベル 4 以上のフッターを非表示にするには、 display を none に設定します。.groupfooterN { display: none; }

知っておくべき重要な点として、グループ化の追加、グループ・ヘッダー/フッター/合計の指定、それらの表示スタイルの設定には、さまざまな Blox、複数のタグ、および複数のスタイル・シート・クラスが関係します。これらの点を以下の表に説明します。

グループの追加	

GroupBlox を使用します。
 	
`<bloxreport:group`
`  members="Product,Week_Ending" />`
	「Product」が最初のレベルのグループで、「Week_Ending」が 2 番目のレベルです。

グループ・ヘッダー、フッター、および合計のテキストの指定

TextBlox を使用します。

```
<bloxreport:text>
  <bloxreport:groupHeader
    level="1"
    text="Profitability by <member/>" />
  <bloxreport:groupHeader
    level="2"
    text="<member level="1" />: <member/>" />
  <bloxreport:groupHeader
    level="2"
    text="<member/> (Ranking:<value member=¥"SalesRank¥"/>"
  />
  <bloxreport:groupFooter
    level="2"
    text="--End of <member/>" />
  <bloxreport:groupFooter
    level="1"
    text="--End of Report" />
  <bloxreport:groupTotal
    level="1"
    columnName="Cost">
    text="Grand Total: <value/>" />
  <bloxreport:groupTotal
    level="2"
    columnName="Cost">
    text="subtotal: <value/>" />
</bloxreport:text>
```

groupHeader、groupFooter、および groupTotal は、<bloxreport:text> タグ内のネストされたタグです。ただし、これらのタグはレポートがグループ化されている場合にのみ有効です。level を指定しないと、テキストはすべてのレベルのグループ・ヘッダー、フッター、および合計に適用されます。

<member/> 変数は、指定されたグループ化レベルのメンバー名によって置換されます。level を指定しないと、現行のグループ化レベルが暗黙のうちに使用されます。参照できるのは、現行レベルか上位のレベルのブレーク・グループ・メンバーだけです。

<value/> 変数は、指定されたグループ化レベルの指定されたメンバーの集約値によって置換されます。level を指定しないと、現行のグループ化レベルが暗黙のうちに使用されます。参照できるのは、現行レベルか上位のレベルのブレーク・グループ・メンバーだけです。

<member/> および <value/> 変数の詳細については、 62 ページの『メンバー名および値を表示するための特殊な置換変数』を参照してください。

グループ・ヘッダー、フッター、および合計の表示スタイルの指定

CSS スタイル・シートを使用します。

```
.groupheader1 {
   font-size: 110%;
   background-color: #9999FF;
   font-weight: bold; }

.groupfooter1 {
   font-size: 80%;
   background-color: #9999FF;  }
.grouptotal1 {
   font-size: 100%;
   text-align: right; }
```

スタイル・クラス groupheader*N*、groupfooter*N*、および grouptotal*N* を使用して、表示スタイルを指定します。

StyleBlox を使用します。

```
<bloxreport:style>
  <bloxreport:groupHeader
    level="1"
    style="font-size: 110%;" />
  <bloxreport:groupHeader
    level="2"
    style="font-size: 90%;" />
  <bloxreport:groupFooter
    level="1"
    style="font-weight: bold;" />
  <bloxreport:groupTotals
    style="font-size:90%;font-style:italic;" />
</bloxreport:style>
```

groupHeader、groupFooter、および groupTotal は、<bloxreport:style> タグ内のネストされたタグです。これらのタグには、CSS スタイル・ストリングを指定するための style 属性があります。これらのタグはレポートがグループ化されている場合にのみ有効です。level を指定しないと、スタイルはすべてのレベルのグループ・ヘッダー、フッター、または合計に適用されます。

グループ合計の集約タイプの指定

GroupBlox を使用します。

```
<bloxreport:group
  members = "Product, Week_Ending">
  <bloxreport:aggregation
    member="Units"
    type="sum"/>
</bloxreport:group>
```

<bloxreport:group> タグ内でネストされたタグ <bloxreport:aggregation> を使用して、各数値データ列の集約タイプを指定します。数値データのデフォルト集約タイプは、sum です。有効値は、sum、average、count、min、max、および none です。

この場合も、付属のスタイル・シートでは、デフォルトで、レベル 1 から 3 までのグループ・フッターが使用不可になっています。

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

特定のレベルのグループ・フッターを使用可能にするには、以下の CSS をスタイル・シートに追加します。

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

または

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

注: レポートを対話モードでレンダリングする場合には、ユーザーが対話式コンテキスト・メニューを使用して、列名、グループ合計テキスト、およびレポート内の各エレメントの表示スタイルを変更できるということに注意してください。ですから、ヘッダーおよびフッター・テキストを動的に設定するには、<member/> および <value/> 置換変数を使用してください。

注: 対話式レポートで、列ヘッダー、グループ・ヘッダー、グループ・フッター、またはグループ合計をリンクにするため、これの周りをアンカー・タグでラップした場合、その上にマウスが来ても、コンテキスト・メニューがポップアップ表示しません。セル内の別の場所 (リンク以外の場所) でマウスを動かさないと、メニューはポップアップ表示しません。これはブラウザーの動作によるものです。レポートを設計する際は、このことを念頭に置いてください。

グループ・ベースのサマリー列の計算

CalculateBlox を使用すると、計算式に基づいて算出列を追加できます。また CalculateBlox にはグループ・ベースのサマリー列を追加するための 4 つの計算関数があり、データのランク付け、累計値、合計のパーセント、および各ブレーク・グループでのカウント値の計算を行います。これらはグループ・ベースの計算なので、既にグループ化されているレポートに基づいて動作します。つまり、まず GroupBlox を使用してグループ化してから、CalculateBlox をパイプラインに追加します。

実際の例については、Blox Sampler-Relational Reporting の『Saving and Exporting Data with Dynamic Queries』の例を参照してください。この例では、「Area」、「Location」、そして「Week_Ending」でグループ化され、「Units」の値に基づいて追加された列を持つレポートを示しています。

計算ごとにグループ化レベルを指定できます。デフォルトでは、最下位のグループ・レベルに設定されます。例えば、レポートを以下のようにグループ化するとします。

- Area
  - Location
    - Week_Ending

この場合、デフォルトでは、計算はレベル 4 (各週) のグループ化 (使用可能な最低レベル) に基づいたものとなります。オプションとして、レベル 3 (各場所)、2 (各地域)、または 1 (レポート内の全域、グループ化なし) に基づいてサマリー列を計算するよう指定することもできます。<value/> 置換変数を使用すると、各グループ化レベルで集約値を取得できます。例でグループ化のレベルを設定し、どのようにサマリー・データが計算されるかを試してみてください。

以下の表に、各関数の構文を示します。すべての関数名には、大/小文字の区別があります。

関数	構文
rank()	rank(memberName [, ASC|DESC] [, *level*]) level が指定されていない場合、あるいは指定されたレベルが存在しない場合は、デフォルトで最下位のレベルのグループが使用されます。デフォルトのソート方向は DESC です。方向には、大/小文字の区別があります。例: <bloxreport:calculate expression = "Rank = rank(Units)" /> <bloxreport:calculate expression = "[Rank of Sales] = rank(Sales, 2)" /> <bloxreport:calculate expression="Rank = rank(Cost), ASC" /> <bloxreport:calculate expression="Rank = rank(Cost, ASC, 1)" />
percentOfTotal()	percentOfTotal(memberName [, *level*]) level が指定されていない場合、あるいは指定されたレベルが存在しない場合は、デフォルトで最下位のレベルのグループが使用されます。 <bloxreport:calculate expression = "[% Total] = percentOfTotal(Units)" /> <bloxreport:calculate expression = "[% Total] = percentOfTotal(Units, 2)" /> 計算値は、小数です (例えば、0.245)。データを 24.5% のようにフォーマットするには、 122 ページのFormatBlox を使用します。
runningCount()	runningCount(memberName [, *level*]) level が指定されていない場合、あるいは指定されたレベルが存在しない場合は、デフォルトで最下位のレベルのグループが使用されます。例: <bloxreport:calculate expression = "RunningCount = runningCount(Units)" /> <bloxreport:calculate expression = "RunningCount = runningCount(Units, 3)" />
runningTotal()	runningTotal(memberName [, *level*]) level が指定されていない場合、あるいは指定されたレベルが存在しない場合は、デフォルトで最下位のレベルのグループが使用されます。例: <bloxreport:calculate expression = "RunningTotal = runningTotal(Units)" />

*こうしたサマリー列はグループに基づいて計算されるということに注意してください。サマリー・データが意味のある正確なものになるためには、CalculateBlox の前に GroupBlox を指定してください。*GroupBlox が CalculateBlox の後に追加される場合 (JSP 内でそうする場合、あるいはユーザーが対話式ユーザー・インターフェースを使用してブレーク・グループを変更する場合)、算出列は通常の数値データ列

として扱われ、グループ集約値もそのようなものとして計算されます。結果として、レポートには意味のないデータが含まれることになります。

レポート・タイトルおよび列サマリー (集約) の追加

レポートのタイトルを追加するには、ReportBlox の外側の HTML を使用できます。レポートがグループ化されている場合には、レベル 1 のグループ・ヘッダーがレポートのタイトルとして表示されます。 71 ページの『ブレーク・グループ・ヘッダー、フッター、および合計の指定とスタイル』を参照してください。

ブレーク・グループを追加せずに、レポートの最後にサマリー・データを示す場合、またはタイトルを追加する場合は、<bloxreport:group>タグを使用し、members 属性の値を空ストリング (members = "") に設定します。<bloxreport:group> タグを追加しないと、サマリー・データ (グループ合計) は提供できません。

以下の例は、ブレーク・グループなしで集約データを追加する方法を示しています。

```
<bloxreport:group members = "" >
   <bloxreport:aggregation member = "Sales" type = "sum" />
   <bloxreport:aggregation member = "Units" type = "average" />
</bloxreport:group>
```

GroupBlox とともに MembersBlox を使用する場合

GroupBlox とともに MembersBlox を使用する場合、GroupBlox の前に MembersBlox を使用してください。 これは、対話式レポートでユーザーが「**グループのクリア**」を選択すると、問題が発生する可能性があるためです。例えば、結果セットにメンバー A、B、C、D、および E があり、メンバー A をブレーク・グループ・メンバーとする GroupBlox があるとします。

```
<bloxreport:group
   members="A" />
```

これにより、メンバー A は異なる「ディメンション」に移動します。これは、マルチディメンション・データ・ソースを処理するときの PresentBlox のページ・ディメンションに類似しています。

GroupBlox の後に、MembersBlox を使用して「Column」ディメンションにメンバー B と C だけを含めることにするとします。

```
<bloxreport:members
   included="B, C" />
```

これによって、結果セットの「Column」ディメンションにはメンバー B と C だけが保持されます。レポートを対話モードでレンダリングし、ユーザーが列ヘッダー・コンテキスト・メニューから「**グループのクリア**」を選択すると、エラーが発生します。MembersBlox で指定したとおり、メンバー A はもう「Column」ディメンションにはないからです。この問題が起こらないようにするには、GroupBlox の前に MembersBlox を使用してください。

第 8 章 データの保管およびエクスポート

この節では、リレーショナル・レポートを共有、印刷、またはオフラインで参照するために保管したりエクスポートしたりする様々な方法を取り上げます。オプションとしては、静的 HTML または PDF ファイルとしての保管や、Excel や他のアプリケーションへの送信、またレポートのブックマークの作成があります。

Blox Sampler - Relational Reporting の『Saving and Exporting Data with Dynamic Queries』の例では、この節で取り上げられるさまざまなオプションが示されています。

対話式レポートをブラウザーから直接保管する際の問題

リレーショナル・レポートは HTML 表としてレンダリングされるので、ユーザーはブラウザーの「ファイル」->「名前を付けて保存」オプションを使用して、ローカル・システム上にコピーを保存することができます。しかし、望ましい結果が得られない場合があります。Netscape ブラウザーを使用してレポートを保管する場合、HTML のみが保管されます。イメージ、スタイル・シート、および参照される他の外部リソースはダウンロードされません。その結果、保管されるレポートの外観は、元のフォーマットおよびレイアウトと異なります。Internet Explorer では、完全な Web ページ、Web アーカイブ・ファイル、または HTML のみとしてページを保管するよう選択することができます。選択するオプションにより、結果は異なります。以下の表では、ブラウザーでのリレーショナル・レポートの保管の試行結果について要約します。

ブラウザー	ファイル保管オプション	「ファイル」-->「名前を付けて保存...」の結果
非対話式レポート		
Internet Explorer	完全な Web ページとして保管	Web ページは保管できないというエラー・メッセージ。
	HTML のみ保管	レポートはスタイル・シートなしで保管されるので、外観が異なる場合がある。
	Web アーカイブ (.mht) として保管	レポートは、すべてのイメージおよびスタイルとともに 1 つの .mht ファイルに正しく保管される。Internet Explorer で表示可能。
Mozilla、Firefox	完全な Web ページとして保管	レポートはスタイル・シートとともに保管される。
Netscape		レポートはスタイル・シートなしで保管されるので、外観が異なる場合がある。
対話式レポート		

ブラウザー	ファイル保管オプション	「ファイル」-->「名前を付けて保存...」の結果
Internet Explorer	完全な Web ページとして保管	Web ページを保管できないというエラー・メッセージが出る。または、保存はされても、ロードしたときに「レポートのロード中...」というメッセージしかページに表示されない。
	HTML のみ保管	レポートは保管されるが、データが含まれない。
	Web アーカイブ (.mht) として保管	レポートは、すべてのイメージおよびスタイルとともに 1 つの .mht ファイルに正しく保管される。Report Editor インターフェースに関連する DHTML も保管されるが、ページをロードしても機能しない。
Mozilla、Firefox	完全な Web ページとして保管	レポートはスタイル・シートとともに保管される。Report Editor インターフェースに関連する DHTML も保管されるが、ページをロードしても機能しない。
Netscape		サポートされていない。 レポートは静的 HTML 表としてレンダリングされ、レイアウトとフォーマットは不正確な場合がある。

Internet Explorer の「Web アーカイブ、単一ファイル」ファイル保管機能を使用すると、Web ページは単一ドキュメント (.mht ファイル拡張子を持つ) として保管され、グラフィックスとスタイルも組み込まれます (別のフォルダーに保管されるのではない)。MHT ファイルは、Internet Explorer で直接表示できます。外部リソースへのリンクや参照は必要ありません。リレーショナル・レポートが MHT ファイルとして保管されると、レポート・フォーマットおよびレイアウトが正確に保持されます。 ただし対話式レポートの場合、DHTML のコンテキスト・メニューも同様に保管されます。ユーザーが保管した MHT ファイルをローカル・マシン上で表示すると、Report Editor ユーザー・インターフェースは表示されますが、実際には全く機能しません。

ユーザーがレポートを正しいレイアウトで確実に保管できるようにするため、「ブックマーク」機能を備えることができます。これはレポートの状態を DB2 Alphablox リポジトリーに保管するものです。ユーザーがローカル・システムでレポートを印刷したり保管したりできるようにするため、レポートを PDF にレンダリングすることもできます。または Report Editor ユーザー・インターフェースが機能せずに生じる混乱を避けるため、対話式レポートを Report Editor インターフェースなしで静的 HTML に保管することもできます。こうしたオプションについては、次に説明します。

静的 HTML としてファイル・システムへ保管する場合

後で参照できるよう、対話式レポートを静的な HTML ページの形でローカル・システムに保管するためのオプションをユーザーに提供する上でかぎとなるのは、ReportBlox の interactive プロパティーを false に設定することです。これにより、保管するレポートから、機能しないコンテキスト・メニューが省かれます。そ

うしないと、コンテキスト・メニューに関連する DHTML がレポート一緒に保管されますが、このようなコンテキスト・メニューはオフラインでは機能しないため、混乱の起こる可能性があります。

以下に、サンプルのレポート・セーバー JSP ファイルを示します。これは、ユーザーが Report Editor インターフェースを使ってレポートに加えた変更をすべて保ったまま、レポートを指定の位置に保管する方法を示したものです。

ReportBlox が入っている JSP ファイルに、レポート･セーバー JSP ページを呼び出して、レポートの bloxName を渡すための、次のようなコードがあるとします。

```
<head>
<%
  String reportName = "myReport";
%>
<script>
  var REPORT_NAME = "<%= reportName %>";
  function toFile() {
    window.open( "file.jsp?reportname=" + REPORT_NAME, REPORT_NAME +
      "_file", "height=400, width=600, scrollbars=yes, resizable=yes");
  }
</script>
</head>

<body>
...
<a href="javascript:toFile()" >Save File</a>

<bloxreport:report id="report" bloxName="<%=reportName%>"
  interactive="true" errors="true" >
  ...
</bloxreport:report>
```

レポート・セーバー JSP ページ file.jsp は、以下のようになります。

```
<!--Importing the associated java classes in order to use
the APIs -->
<%@ page import="com.alphablox.blox.*" %>
<%@ page errorPage="error.jsp" %>
<%@ page import="java.io.*" %>

<!---Getting the reportName via the URL-->
<% String reportName = (String)request.getParameter( "reportname" );%>

<html>
<head>
  <title><%= reportName %> Saving Report As File</title>
</head>
<body>
<%
  // set the report to non-interactive mode so the menu items do not get
  // rendered
  if( reportName != null ){
    BloxContext context = BloxContextFactory.getBloxContext(request, response);
    ReportBlox reportBlox = (ReportBlox)context.getBlox( reportName );
    boolean isInteractive  = reportBlox.isInteractive();
    if( reportBlox != null ){
      reportBlox.setInteractive(false);
      try {
        /* Specify a location to save the file. Modify this
        for your application. */
        String filePath =  "C:¥¥temp¥¥" + reportName + ".html";
        FileWriter file = new FileWriter( filePath );
        file.write("<html>");
        file.write("<head>");
```

```
        file.write("<title>" + reportName + " Report</title>" );
        file.write( "<style>" );

        // Write the contents of the style sheet into the page
        String appPath = application.getRealPath("/");
        String styleSheetPath = appPath + "¥¥style¥¥reportstyles.css";
        // Create input stream object.
        FileInputStream fis = new FileInputStream( styleSheetPath );
        // Set variable for looping through bytes.
        int c;
        while( (c = fis.read() ) != -1) {
          file.write(c); // Loop to read and write bytes.
        }
        fis.close(); // Close output and input resources.
        // Done writing out style sheet

        file.write( "</style>" );
        file.write( "</head>" );
        file.write( "<body>" );
        reportBlox.writeUpdate( file ); // Generate the report table.
        file.write( "</body>" );
        file.write( "</html>" );
        file.close( );

        out.println( "Your file was successfully saved at " + filePath +"." );
      }
      catch( Exception e ) {
        out.println( "The page not saved :¥n" + e.getMessage() );
      }
      finally {
        // Set back to initial state
        reportBlox.setInteractive( isInteractive );       }
      } else {
        out.println( "The report " + reportName + " does not exist." );
      }
    } else {
    out.println( "This page was called without a report name." );
  }
%>
</body>
</html>
```

ブラウザー・ウィンドウからの印刷に一層適したページを提供する場合も、同様の方法を使用できます。

レポートにブックマークを付けて状態を保管する

レポートにブックマークを付けることにより、ユーザーは DB2 Alphablox リポジトリーにレポートの「状態」を保管できます。PersistenceBlox を使用すると、レポートのフォーマットおよびデータ・レイアウトを、リポジトリーの reportingpersistence/ ディレクトリーに保管できます。ブックマークを付けたレポートをロードすると、データ・ソースへの接続がインスタンス生成され、保存されているフォーマットとレイアウトによって最新のデータが表現されます。ブックマークをリポジトリーからロードするときには生きたデータが取り出されるため、ロード時には基礎となるデータ・ソースが使用可能になっている必要があります。

リレーショナル・レポートのブックマークを保管するには、<bloxreport:persistence> タグに以下のものを指定する必要があります。

- リポジトリーの中のブックマークの保管場所
- ブックマークの名前

- ブックマークを付ける ReportBlox の ID
- 「保管」操作

レポートが対話モードでレンダリングされている場合は、レポートにブックマークを付ける前にこれを非対話式設定しなければなりません。非対話式にしないと、ブックマークのロード時にレポートが正しく表示されません。つまり、ブックマークを付けたレポートは非対話式レポートとしてロードすることしかできないということです。

リポジトリーにブックマークを保管した後、ユーザーが画面上でそのレポートと対話を続けるには、対話モードを true にリセットする必要があります。以下の例は、HTML フォームによりブックマーク保管機能を提供する、report.jsp というファイルです。

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

<html>
<head>
   <title>Bookmark Example</title>
   <link rel="stylesheet" href="/AlphabloxServer/theme/report.css">
</head>
<body>
   <h1>Bookmark Example</h1>
   <form method="GET" action="save.jsp" target="_blank">
     <input type="text" name="bookmark" value="bookmark name"/>
     <input type="submit" value="Save this bookmark"/>
   </form>

   <bloxreport:report id="MySalesReport1" interactive="true">
      <bloxreport:cannedData/>
   </bloxreport:report>

</body>
</html>
```

ユーザーがブックマーク名を入力して「Save this bookmark (このブックマークを保管)」ボタンをクリックすると、save.jsp が呼び出され、別のブラウザー・ウィンドウに表示されます。以下のコードの抜粋は、レポートを非対話式モードに設定し、そのレポートのブックマークを保管し、さらに対話式をtrue にリセットする一連の手順を示しています。

```
//save.jsp
<%@ taglib uri="bloxreporttld" prefix="bloxreport" %>
<%@ page errorPage="error.jsp" %>
<%@ page import="com.alphablox.blox.*" %>

<html>
<head></head>
<body>
<h1>Report Bookmark Saved</h1>
<%
   String bookmark = request.getParameter("bookmark");

   // "MySalesReport1" is the ID of a ReportBlox
   // page that calls this SaveBookmark.jsp page.
   BloxContext context = BloxContextFactory.getBloxContext(request, response);
   ReportBlox report = (ReportBlox) context.getBlox("MySalesReport1" );

   // Set the interactive mode to false
   report.setInteractive( false );
%>
```

```
<%--Add a PersistenceBlox, providing the ReportBlox ID
    (targetBloxID), location, bookmark name (persistedName),
    and operation to perform--%>
<bloxreport:persistence id="p1"
   targetBloxId="MySalesReport1"
   location="sales/east"
   persistedName="<%= bookmark %>"
   operation="save" />

// Set the report back to interactive mode
<%
   report.setInteractive( true );
%>

<p>The bookmark <b><%= bookmark %></b> has been saved.</p>
</body>
</html>
```

DB2 Alphablox リポジトリーがファイル・ベースの場合、上記の例では「SavesApr02」という名前のブックマークが以下のロケーションに保管されます。

```
<alphablox_dir>/repository/reportingpersistence/sales/east/
```

しかし、DB2 Alphablox リポジトリーがデータベース・ベースの場合、上記の例ではブックマークをリポジトリーに保管するにあたり、"reportingpersistence/sales/east" が CONTEXT 列に、 "SavesApr02" が NAME 列に保管されます。

ブックマークのロード

リレーショナル・レポートのブックマークを取得するには、ロケーション、ブックマーク名、および 「load」操作を指定します。ブックマークを正常にロードするには、基礎となるデータ・ソースが使用可能で、生きたデータが取得できるようでなければなりません。以下の例は、リポジトリーに保管されているブックマークのリストを生成し、各ブックマークをリンクにする方法を示しています。

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

<html>
<head>
   <title>List Bookmarks</title>
</head>
<body>
   <p>Your bookmarks:</p>
   <bloxreport:persistence id="repository" />
   <table>
   <%
      String[] files = repository.list("sales/east",
         repository.LIST_TYPE_STATES );
      for( int i = 0; i < files.length; i++){
      %>
        <tr><td><a href="load.jsp?bookmark=<%= files[i] %>">
        <%= files[i] %></td></tr>
      <%
      }
   %>
   </table>
</body>
</html>
```

ユーザーがブックマーク・リンクをクリックすると、渡されたブックマーク名を指定して load.jsp が呼び出されます。

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

<html>
<head>
   <title>Loading Bookmark</title>
   <link rel="stylesheet" href="/AlphabloxServer/theme/report.css">
</head>

<%
   String bookmark = request.getParameter( "bookmark" );
%>
<h1>Bookmark: <%= bookmark %></h1>
<%
   if( bookmark != null ) {
      %>
      <bloxreport:persistence
         location="sales/east"
         persistedName="<%= bookmark %>"
         operation="load" />
      <%
   }
%>
</html>
```

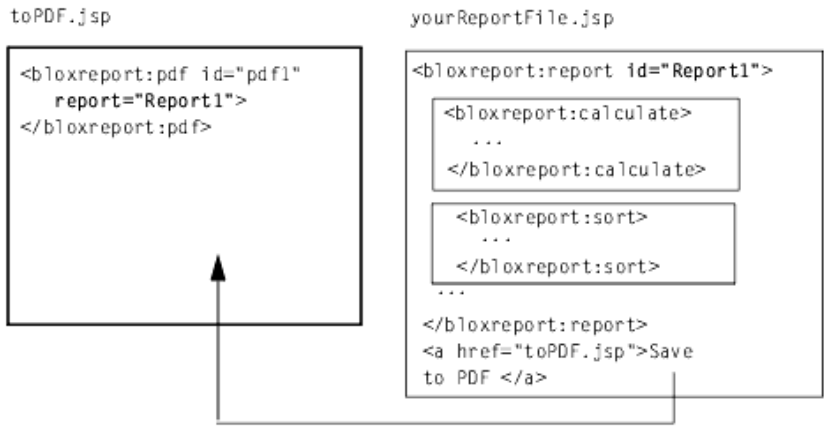
ブックマーク・レポートがロードされると PersistenceBlox によってサーバー上に ReportBlox オブジェクトのインスタンスが作成されることにご注意ください。インスタンスが既にセッションに存在しているのであれば、PersistenceBlox は古いインスタンスを上書きします。Blox Sampler - Relational Reporting の『Saving and Exporting Data with Dynamic Queries』の例には、bloxName を使用してこの問題を扱う方法が示されています。

PDF 形式での保管

リレーショナル・レポートは、PdfBlox を使用して PDF フォーマットにレンダリングできます。この節では、リレーショナル・レポートの PDF 版を提供する 2 つのシナリオを取り上げます。最初のシナリオの方がより一般的なシナリオであり、レポートを PDF でレンダリングするためのリンクまたはボタンをページ上に設けます。説明するもう 1 つのシナリオは、ユーザーに対して PDF のレポートしか表示しない場合です。つまり、レポートを DHTML (対話モード) または静的 HTML (非対話モード) で表示するのではなく、直接 PDF で表示できるというものです。

PDF ファイル形式でのレポートの保管

PDF ファイルとしてレポートを保管するというオプションをユーザーに提供するには、これをクリックするとレポートをPDFファイルとして保管できるような、ページ上のボタンまたはリンクを用意します。これには、以下の図に示されているように、別個の JSP ファイルが伴います。



yourReport.jsp ページには、Report1 という id の ReportBlox があります。ユーザーが「Save to PDF (PDF に保管)」ボタンをクリックすると、2 番目のページ toPDF.jsp が呼び出されます。このページには、Report1 ReportBlox を取得して PDF ファイルとして保管する PdfBlox があります。

どのようなレポートも toPDF.jsp ファイルの 1 つのコピーで動的にレンダリングできるようにするには、次のコードを使用して、要求と一緒にパラメーターを渡すようにします。

```
<%@ taglib uri="bloxreporttld" prefix="bloxreport" %>
<bloxreport:pdf id="mypdf"
   report='<%= request.getParameter("report") %>' />
```

toPDF.jsp に必要なのはこれだけです。最初の行では必要なタグ・ライブラリーをインクルードします。2 行目では PdfBlox を追加し、その report 属性を、要求を介して渡された report パラメーターの値に設定します。<bloxreport:pdf> タグには固有の id が必要なことに注意してください。レポートを含む JSP ファイルで、report パラメーターを介してレポートの id を渡すようにリンクを変更してください。

```
<a href="toPDF.jsp?report=reportId">Send to PDF</a>
```

PdfBlox が PDF を生成するレポートの状態を正しく反映するには、ReportBlox に id を指定する必要があります。つまり、PDF レンダリング・エンジンは、ユーザーが対話式コンテキスト・メニューを使用して加えた変更を保持します。

以下の点に注意してください。

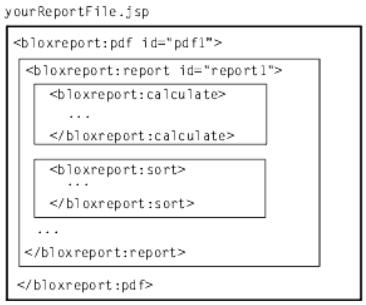
- 1 ページには PdfBlox を 1 つしか入れられません。複数の場合には、最初の PdfBlox がブラウザーに表示されます。
- PdfBlox は、スタイル・シート、対話式コンテキスト・メニュー、または JSP ファイル内の StyleBlox によって設定されたスタイル・セットを保持します。
- レンダリングされた PDF には、JSP 内の他の HTML テキストは含まれません。PdfBlox は JSP ページ内の ReportBlox のみをレンダリングし、ページ上の他のすべての HTML テキストは無視します。
- ヘッダーの左上隅には、指定のロゴが組み込まれます。headerText タグ属性を使用して指定したヘッダーのテキストは常に中央に表示されます。上部枠の幅とヘッダーの高さを指定できます。また、標準 XHTML を使用してヘッダーのテキス

トにスタイルを設定できます。例: `headerText="<span style='color:blue'>Product Sales</span>"`

- フッターには、常にその左下隅にタイム・スタンプがあり、右下隅には、合計ページ数の内の現行ページ番号が表示されます。`footerText` タグ属性を使用して指定したフッターのテキストは常に中央に表示されます。下部枠の幅とフッターの高さを指定できます。また、標準 XHTML を使用してフッターのテキストにスタイルを設定できます。

レポートを PDF に直接レンダリングする

生きたデータを反映したレポートを直接 PDF で表示するには、以下の図に示されているように、`<bloxreport:pdf>` タグを使用して、`<bloxreport:report>` タグの周囲をラップします。



注:

- `<bloxreport:pdf>` タグには固有の `id` が必要です。
- 1 ページに含めることのできる PdfBlox は 1 つだけです。複数の PdfBlox を追加し、それぞれに固有の `id` がある場合、各オブジェクトが同じサーバー上に作成されることになりますが、ブラウザーには最初の PdfBlox しか表示されません。
- 各 PdfBlox は 1 つの ReportBlox しか使用しません。`<bloxreport:pdf>` タグに複数の ReportBlox がある場合は、最後に宣言した ReportBlox が使用されます。

Excel または他のアプリケーションへの保管

この節では、ユーザーが Excel または他のアプリケーションでデータを参照および操作できるようにするための 2 つのシナリオを説明します。最初のシナリオの方がより一般的なシナリオであり、現在表示されているレポートを Excel に送信するためのリンクまたはボタンをページ上に設けます。説明するもう 1 つのシナリオは、ユーザーに対してデータを直接に Excel で表示する場合です。つまり、レポートを DHTML (対話モード) または静的 HTML (非対話モード) で表示するのではなく、直接 Excel で表示するというものです。

Excel へのエクスポート

JSP の `page` ディレクティブには、要求されたページで期待されているコンテンツ・タイプをブラウザーに知らせる `contentType` 属性があります。この属性を設定

すると、ブラウザーに対して、ページを処理するために呼び出すアプリケーションを指示できます。現在表示されているレポートを Excel に送信するには、以下のことが必要です。

- contentType を設定します。

```
<%@ page contentType="application/vnd.ms-excel;
    charset=UTF-8" %>
```

- ReportBlox の interactive プロパティーを false に設定します。レポートを非対話式に設定すると、結果ファイルにコンテキスト・メニューのメニュー項目が含まれなくなります。これは、78 ページの『静的 HTML としてファイル・システムへ保管する場合』で述べられているのと同様の方法で行います。

ReportBlox が入っている JSP ファイルに、Excel にレポートをエクスポートするための別の JSP ページを呼び出して、レポートの bloxName を渡すための、次のようなコードがあるとします。

```
<head>
<%
  String reportName = "myReport";
%>
<script>
  var REPORT_NAME = "<%= reportName %>";
  function toExcel() {
    window.open( "excel.jsp?reportname=" + REPORT_NAME, REPORT_NAME +
      "_excel", "height=400, width=600, scrollbars=yes, resizable=yes");
  }
</script>
</head>

<body>
...
<a href="javascript:toExcel()" >Export to Excel</a>

<bloxreport:report id="report" bloxName="<%=reportName%>"
  interactive="true" errors="true" >
  ...
</bloxreport:report>
```

excel.jsp は、以下のようになります。

```
<!--Using the taglibs -->
<%@ taglib uri="bloxreporttld" prefix="bloxreport" %>
<!--Importing the associated java classes in order to use the APIs -->
<%@ page import="com.alphablox.blox.*" %>
<!-- Setting the contentType -->
<%@ page contentType="application/vnd.ms-excel; charset=UTF-8" %>

<%
  //Set report to non-interactive mode so the menus don't get rendered
   String reportName = (String)request.getParameter( "reportname" );
   if( reportName != null ) {
    BloxContext context = BloxContextFactory.getBloxContext(request, response);
    ReportBlox reportBlox = (ReportBlox)context.getBlox( reportName );
    if( reportBlox != null ) {
      reportBlox.setInteractive(false);
    } else {
      out.println( "The report " + reportName + " does not exist." );
    }
  } else {
    out.println( "This page was called without a report name." );
  }
%>
<html>
```

```
<head>
  <title><%= reportName %> Excel Report</title>
  <%--In line the styles so they'll be saved if the page is saved and
      you are not asked to be authenticated--%>
  <style>
    <jsp:include page="style/reportstyles.css" flush="true"/>
  </style>
</head>
<body>

<%--The reportblox tag that follows causes the html of the report to be
rendered --%>
<bloxreport:report id="reportBlox" bloxName="<%=reportName%>" />
</body>
</html>
```

インライン・スタイル・シートをインクルードするために、JSP の include ステートメントを使用していることに注意してください。Excel およびご使用のオペレーティング・システムのバージョンによっては、インポートされたスタイル・シートを使用すると、Excel がスタイル・シートにアクセスするときにユーザーの認証が必要になる可能性があります。JSP のインクルード技法を使用してスタイル・シートをインライン化する場合、以下の点に注意してください。

- Excel はインポートを解決できないので、このスタイル・シートは他のスタイル・シートをインポートすべきではありません。
- /AlphabloxServer/theme/ で提供されている report.css スタイル・シートを使用したい場合、このスタイル・シートは実際には 3 つの他のスタイル・シート (レポート用の styles.css、コンテキスト・メニュー用の dialog.css、および ErrorBlox 用の errors.css) をインポートするため、styles.css を使用するように JSP の include ステートメントを変更する必要があります。

  ```
  <head>
    <style>
      <jsp:include page="/AlphabloxServer/theme/styles.css" flush="true"/>
    </style>
  </head>
  ```

  これにより、レポートのフォーマットとレイアウトは Excel で保持され、追加の認証は必要なくなります。さらにスタイル・シートのインポート・ステートメントを解決できずに Excel がハングすることもなくなります。

Excel へのレポートの直接送信

JSP の page ディレクティブには、要求されたページで期待されているコンテンツ・タイプをブラウザーに知らせる contentType 属性があります。以下のコードを JSP ページの最初で使用して、適切な contentType を設定すると、レポートを直接 Excel に送信できます。

```
<%@ page contentType="application/vnd.ms-excel;
charset=UTF-8" %>
```

ブラウザーは、サーバーから戻された応答を受け取ると、この特定のコンテンツ・タイプのファイルを開くために指定されたアプリケーションを起動し、戻されたページを表示します。

レポートをコンマ区切り形式またはタブ区切り形式に送信する

多くの場合、Excel または他のアプリケーションで使用するために、データをコンマ区切り形式またはタブ区切り形式でエクスポートする必要が生じます。これには、Relational Reporting API を使用して、結果セットをウォークスルーし、データ列の間に区切り文字を、データ行の間に改行文字を挿入することが必要です。この API に関する説明については、94 ページの『Relational Reporting API』のトピックを参照してください。詳細な例については、98 ページの『結果セットのナビゲート: レポートをコンマ区切り形式またはタブ区切り形式にレンダリングする』のトピックを参照してください。

第 9 章 Report Editor ユーザー・インターフェースのスタイル設定

Report Editor ユーザー・インターフェースの外観を企業のカラー・スキームや、アプリケーション全体の外観に合わせてカスタマイズすることができます。この節では、Report Editor で使用されるスタイル・クラスを説明し、それらのクラスに対して独自のスタイルを指定できるようにします。

- 89 ページの『Report Editor ユーザー・インターフェースのスタイル・クラス』
- 92 ページの『Report Editor の使用のためのユーザー・ヘルプ』

Report Editor ユーザー・インターフェースのスタイル・クラス

Report Editor ユーザー・インターフェースには、3 つのコンテキスト・メニュー、および「レポート・スタイル」ダイアログ・ボックスが含まれます。表示されるフォントおよびカラーはすべて一連のスタイル・クラスに基づいています。

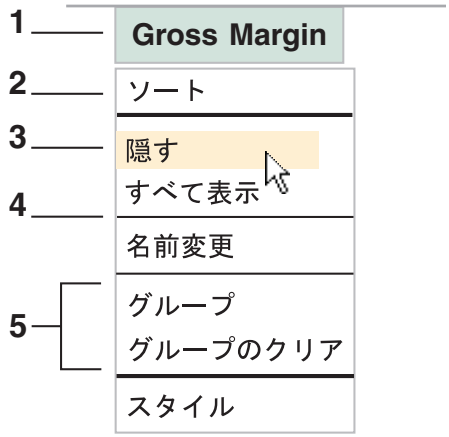
コンテキスト・メニューのスタイル・クラスは、次のとおりです。

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

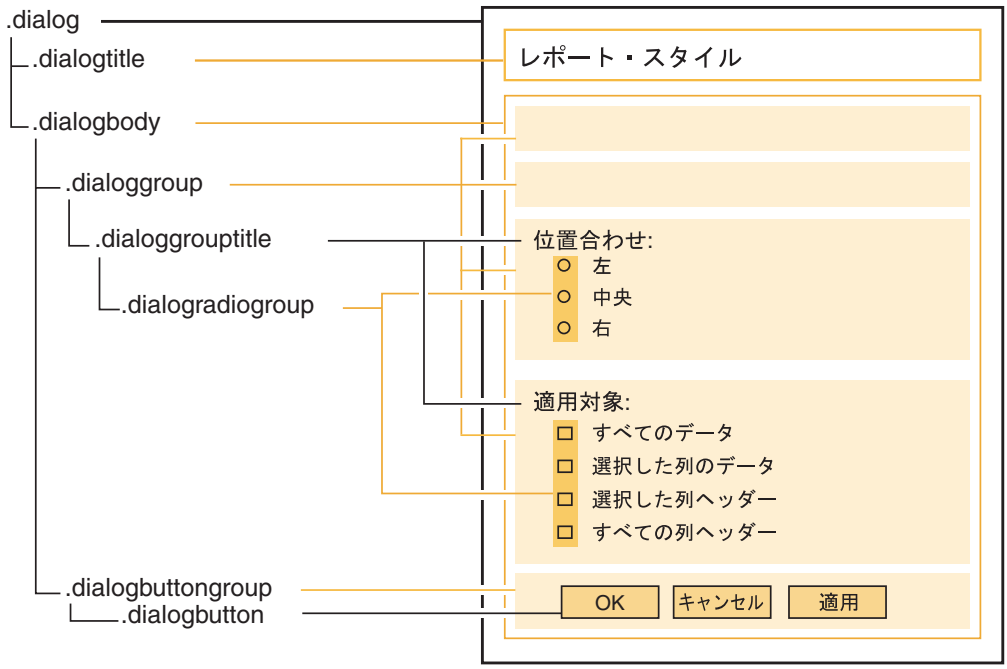
「レポート・スタイル」ダイアログ・ボックスに関連したスタイル・クラスは、次のとおりです。

- `.dialog`
- `.dialogtitle`
- `.dialogbody`
- `.dialoggroup`
- `.dialoggrouptitle`
- `.dialogradiogroup`
- `.dialogbutton`
- `.dialogbuttongroup`

以下のイメージは、コンテキスト・メニューのスタイルを示しています。



以下の図は、「レポート・スタイル」ダイアログ・ボックスのクラスを示しています。



以下は、dialog.css で提供されるデフォルトのスタイルです。これは、「レポート・スタイル」ダイアログ・ウィンドウのフォント、カラー、およびスタイルを定義する方法を例示しています。

```
//for overall background color, fonts, position, borders...
.dialog {
   background-color: #E3E3E3;
   font-size: 80%;
   font-family: sans-serif;
   position: absolute;
   cursor: default;
   border: solid 2 black;
   border-top-width: 1;
   border-left-width: 1;
}

//for the dialog window title area
.dialogtitle {
```

```
   background-color: #CCCCCC;
   text-align: left;
   font-weight: bold;
   padding : 5 10 5 10;
   margin-bottom: 5;
   cursor: hand;
}

//for the body of the dialog window
.dialogbody {
   padding : 10;
   vertical-align: middle;
}

/* for each group of selections; there are four groups in the
dialog window, each with a thin border (1px) in color
#CCCCCC */
.dialoggroup {
   text-align: left;
   padding : 5;
   border : solid 1 #CCCCCC;
   margin: 5;
}

//for the title in each group
.dialoggrouptitle {
   margin-bottom: 5;
}

//for each button, either radio or check box
.dialogradiogroup {
   padding-left: 10;
}

//for each of the three action buttons--OK, Cancel, and
//Apply
.dialogbutton {
   margin-right: 5;
   font-size: 80%;
   padding: 2 0 1 0;
   width: 60;
}

//for the group of three action buttons--OK, Cancel, and
//Apply
.dialogbuttongroup {
   text-align: right;
   padding-top :20;
}
```

注: スタイル・シートにフォント書体、スタイル、およびサイズを設定する際には、特にフォント書体とサイズの整合性に注意を払ってください。通常、次のクラスには同じフォント書体およびサイズを使用する必要があります。そうすれば、ユーザーがホット・スポットの上でカーソルを動かしても、列の幅が変わってレポートが揺れて見えたりしません。

- .column
- .choicehover
- .selected

スタイル・クラスのオーバーライド

外部スタイル・シートを使用すれば、JSP ページがよりきれいになりますし、アプリケーション全体で同じスタイル・シートを再利用できるようになります。独自の外部スタイル・シートを作成し、DB2 Alphablox に付属のスタイル・シートをインポートし、カスタマイズしたいスタイル・クラスをオーバーライドするのが良い方法です。そうすれば、定義しなければならないクラスが欠けたりしませんし、外部スタイル・シートへのリンクを 1 つ追加するだけで済みます。

例えば、ReportBlox を含んだ JSP ページが独自のスタイル・シートを参照するとします。

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

myreportbuilder.css の先頭で、DB2 Alphablox に付属のスタイル・シートをインポートします。

```
@import url( /AlphabloxServer/theme/report.css );

.report {
   background-color: white;
   color: black;
   font-family: Trebuchet MS,Verdana, Arial, Helvetica, sans-serif;
   border: solid 1 black;
   padding: 5;
   margin : 5;
   width: 0;
}
```

こうすると、付属のスタイル・シートのスタイルが、独自のスタイルでオーバーライドされます。リポジトリーの theme ディレクトリーにあるスタイル・シートは絶対に変更してはならないことに注意してください。 DB2 Alphablox のアップグレード時に置換されます。

Report Editor の使用のためのユーザー・ヘルプ

レポートが対話式の場合には、ユーザー・インターフェースがどのように機能するかを説明するユーザー・ヘルプを提供できます。Report Editor ユーザー・インターフェースの中からエンド・ユーザー・ヘルプ・ファイルへのリンクはありません。それで独自のリンクを提供する必要があります。ヘルプ・ファイルは HTML ファイルとイメージの集合に過ぎないので、それらを自分のアプリケーション用に簡単にカスタマイズできます。ファイルの場所およびそれらへリンクする方法についての詳細は、106 ページの『ユーザー・ヘルプの提供』を参照してください。

第 10 章 上級トピック

この節では、リレーショナル・レポート開発に関連した上級トピックを解説します。特に、セッションの有効範囲の管理、Alphablox Relational Reporting API の使用、およびレンダリングされたレポート内のデータ行およびセル値へのアクセスに関するトピックを解説します。

セッションの有効範囲の管理

Blox Report タグを使用して Relational Reporting Blox を JSP ページに追加する場合、デフォルトで、これらの Blox にはセッションの有効範囲があります。ユーザーが指定する基準に基づいてリレーショナル・レポートを動的に作成する、レポート・ビルダー・アプリケーションがあるとします。ユーザーは、データのグループ化またはソートの仕方を選択してから「レポートの生成」ボタンをクリックして、レポートを作成します。生成されたレポートは、対話モードでレンダリングするように設定されているので、ユーザーは Report Editor ユーザー・インターフェースを使用してレポートを編集できます。

新しい基準のセットが指定されて、新しいレポートが要求されるたびに、既存のサーバー・オブジェクトを再利用する必要があります。そうしなければ、新しいオブジェクトのインスタンスが生成されないからです。これは、bloxName 属性を使用することによって実行できます。オプションの bloxName タグ属性を使うと、値を動的に割り当てることができるため、オブジェクトの再利用が可能になります。必須の id がページ上の Blox を一意的に識別する一方で、bloxName は変数を使用して動的に割り当てることができます。id の値は、JSP ではスクリプト変数名として使用されます。bloxName の値は、サーバーがオブジェクトを識別するための名前となります。bloxName を指定しない場合は、id がサーバー・オブジェクト名およびスクリプト変数名の両方として使用されます。bloxName を指定した場合は、それを JSP 内で動的に変更できます。

```
<% String bloxName="report1"; %>

<bloxreport:report id="myReportBlox"
   bloxName="<%= bloxName %>" >
...
```

bloxName の値を変更するのは便利な開発技法です。これにより、JSP に変更を加えるたびにブラウザー・ウィンドウを閉じて再始動する必要がなくなります。さらに重要なこととして、これによりユーザーの選択に基づいて Blox プロパティーを動的に設定できます。Blox Sampler - Relational Reporting の『Saving and Exporting Data with Dynamic Queries』の例も、bloxName を使用して、別個のブラウザー・ウィンドウに異なるフォーマットでレポートをレンダリングする例を示しています。

有効範囲に関連した別の現象として、ユーザーがレポートに加えた変更が、別のセッションでレポートにアクセスしたときには保たれないということがあります。レポートを PDF または Excel に送るなどのレポート保管オプションを提供して、ユーザーがローカル・システムまたはサーバー上のどこかにレポートを保管できるようにすることができます。レポート保管機能の提供については、78 ページの『静的 HTML としてファイル・システムへ保管する場合』、83 ページの『PDF ファイル形式でのレポートの保管』、および 85 ページの『Excel へのエクスポート』を参照してください。

Relational Reporting API

すでに 10 ページの『レポートのパイプライン』で説明したとおり、リレーショナル・レポートは、Relational Reporting Blox を、追加した順序で処理することによって作成されます。根本的なデータ・プロデューサーは SQLDataBlox および RDBResultSetDataBlox です。最終使用先は ReportBlox です。中間にある CalculateBlox、FilterBlox、GroupBlox、MembersBlox、OrderBlox、および SortBlox は、データ・コンシューマーともデータ・プロデューサーともなる「トランスフォーマー」です。これらは前の Blox からデータを取り出し、それを何らかの仕方でトランスフォームし、それを次の Blox に渡します。

これらすべての「トランスフォーマー」の継承元は *IConsumer* および *IProducer* インターフェースです。*IConsumer* には、Blox の入力データを設定する `setInput()` メソッドがあります。IProducer には `getData()` メソッドがあります。これは、*IDataSet* インターフェースへのアクセス、さらには *IDimension* および *IMember* へのアクセスを提供します。たとえば、SQLDataBlox `getData()` メソッドを呼び出すことによって、*IDataSet* にアクセスして、結果セットをウォークスルーすることができます。

以下の例は、API を使用してリレーショナル・レポートを作成する方法を示しています。

```
<%@ page import="com.alphablox.blox.*" %>
<html>
<head>
   <link rel="stylesheet" href="/AlphabloxServer/theme/report.css" />
</head>
<body>
<%
  String query="SELECT location, product_name, sales, units, cost
FROM qcc WHERE week_ending = '2000-04-08'";

  try {
    ReportBlox rBlox = new ReportBlox();
    rBlox.setErrors(true);
    rBlox.setId("myRBlox");

    DataSourceConnectionBlox dConn = new DataSourceConnectionBlox();
    dConn.setDataSourceName("qcc-rdb");
    dConn.connect();

    SQLDataBlox dBlox = new SQLDataBlox();
    dBlox.setInput(dConn);
    dBlox.setQuery(query);
    dBlox.execute();

    // Create the grouping
    GroupBlox myGroup = new GroupBlox();
```

```
    myGroup.setMembers( new String [] {"location"});
    myGroup.setAggregationType("units", "none");

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

    // Finally call ReportBlox's write() method to write it out
    rBlox.write(out);

  }
  catch ( Exception e ) {
    ErrorBlox eBlox = new ErrorBlox();
    Throwable msg = eBlox.getRootCause(e);
    out.println("<br><b> This Exception was captured</b><br> "+
msg.getMessage());
  }
%>
</body>
</html>
```

この例が非対話式レポートを作成することに注意してください。それがデフォルトの動作だからです。

API を使用して対話式レポートを作成する

API を使用して対話式レポートを作成する場合には、3 つの追加事項を指定する必要があります。

1. setInteractive メソッドを使用して、ReportBlox を対話式に設定します。

   ```
   rBlox.setInteractive(true);
   ```

2. URL 接頭部を設定します。対話を処理するサーブレットがこの情報を必要とします。Blox Report タグ・ライブラリーを使用して ReportBlox を作成する場合、これは普通、自動的に処理されます。API を使用する場合は、次のようにして URL 接頭部を明示的に指定しなければなりません。

   ```
   BloxContext context = BloxContextFactory.getBloxContext(request, response);
   rBlox.setUrlPrefix(context.getContextPath() + "/" +
      URLFactory.ALPHABLOX_SERVER_PREFIX);
   ```

3. ReportBlox を BloxContext の中に入れます。Blox Report タグ・ライブラリーを使用する場合は、これも自動的に処理されます。API を使用する場合は、ReportBlox を BloxContext に明示的に追加して、init() メソッドを呼び出さなければなりません。

   ```
    String scriptId = rBlox.getBloxName();
    if(scriptId == null)
    {
       scriptId = rBlox.getId();
    }
    rBlox.init(context,scriptId);
   ```

   コードのこの部分では、オプションの bloxName が指定されているなら、これを使用して ReportBlox を BloxContext に登録します。指定されていないなら、Blox id を使用します。

完全な例は、以下のようになります。

```
<%@ page import="com.alphablox.blox.*,
                 com.alphablox.net.URLFactory"%>
<html>
<head>
```

```
   <link rel="stylesheet" href="/AlphabloxServer/theme/report.css" />
</head>
<body>
<%
  String query="SELECT Week_Ending, location, product_name, sales, units, cost
     FROM qcc WHERE Week_Ending_Text = '04-08-2000'";

  try {
    ReportBlox rBlox = new ReportBlox();
    rBlox.setErrors(true);
    rBlox.setId("myRBlox");

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

    DataSourceConnectionBlox dConn = new DataSourceConnectionBlox();
    dConn.setDataSourceName("qcc2003-rdb");
    dConn.connect();

    SQLDataBlox dBlox = new SQLDataBlox();
    dBlox.setInput(dConn);
    dBlox.setQuery(query);
    dBlox.execute();

    // Create the grouping
    GroupBlox myGroup = new GroupBlox();
    myGroup.setMembers( new String [] {"location"});
    myGroup.setAggregationType("units", "none");

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

    // 2. Set the URL prefix
    BloxContext context = BloxContextFactory.getBloxContext(request, response);
    rBlox.setUrlPrefix(context.getContextPath() + "/" +
       URLFactory.ALPHABLOX_SERVER_PREFIX);


    // 3. Add the ReportBlox to the Context
    String scriptId = rBlox.getBloxName();
    if(scriptId == null)
    {
       scriptId = rBlox.getId();
    }
    rBlox.init(context,scriptId);

    // Finally call ReportBlox's write() method to write it out
    rBlox.write(out);

  }
  catch ( Exception e ) {
    ErrorBlox eBlox = new ErrorBlox();
    Throwable msg = eBlox.getRootCause(e);
    out.println("<br><b> This Exception was captured</b><br> "+
msg.getMessage());
  }
%>
</body>
</html>
```

リレーショナル・レポートを作成するのにタグを使用すると、たいへん便利なことにお気付きになったことでしょう。必要な場合にはいつでも、オブジェクトの id を使用して、オブジェクトをスクリプト指定できます。

パイプラインのウォークスルー: ソート順序のクリア

この例では、ReportBlox パイプラインをウォークスルーして、Blox のインスタンスを検出する方法を示します。この例では、レポートでソートが行われていることが検出され、そのソートをクリアします。

1. IConsumer および IProducer パッケージをインポートします。

```
<%@ page import="com.alphablox.blox.*,
                 com.alphablox.pipeline.IConsumer,
                 com.alphablox.pipeline.IProducer" %>
```

2. 照会は、SQLDataBlox `setQuery()` および `execute()` メソッドを使用して動的に設定されます。詳しくは、100 ページの『照会の動的な変更』のトピックを参照してください。

3. WHILE ループを使用して、各 IConsumer について、その入力が SortBlox であるかどうかを確認します。入力が SortBlox である場合、ルールをヌルに設定します。IConsumer および IProducer について詳しくは、94 ページの『Relational Reporting API』を参照してください。

完全な例は、以下のようになります。

```
<%@ taglib uri="bloxreporttld" prefix="bloxreport" %>
<%@ page import="com.alphablox.blox.*,
                 com.alphablox.pipeline.IConsumer,
                 com.alphablox.pipeline.IProducer" %>
<html>
<head>
   <link rel="stylesheet" href="/AlphabloxServer/theme/report.css" />
</head>
<body>
<%
   String DATASOURCE = "qcc-rdb";
   String query = "select * from qcc where sales > 8000";

   // This block of code is placed before the ReportBlox in case the report
   // is set to non-interactive mode
   BloxContext context = BloxContextFactory.getBloxContext(request, response);
   ReportBlox reportBlox = (ReportBlox)context.getBlox("salesReport");
   if( reportBlox != null ) {
      Object input = reportBlox.getInput();

      while( input instanceof IConsumer )
      {
         if( input instanceof SortBlox )
         {
            ((SortBlox)input).setRules( null );
            out.println( "Sort rules cleared" );
         }

         input = ((IConsumer)input).getInput();
      }
      // You could check if input is SQLDataBlox to be safe.
      // Dynamicaly execute the query each time the page is loaded.
      ((SQLDataBlox)input).setQuery( query );
      ((SQLDataBlox)input).execute();
   }
%>

<bloxreport:report id = "assetReport" interactive="true">

   <bloxreport:sqlData id="assetReportSql"
      query = "<%= query %>" >
      <bloxreport:dataSourceConnection
         dataSourceName = "<%=DATASOURCE%>" >
```

```
      </bloxreport:dataSourceConnection>
   </bloxreport:sqlData>
   <bloxreport:style>
      <bloxreport:banding style="background-color: white;" />
      <bloxreport:numeric style="text-align: center" />
   </bloxreport:style>
</bloxreport:report>
</body>
</html>
```

結果セットのナビゲート: レポートをコンマ区切り形式またはタブ区切り形式にレンダリングする

この例は、SQLDataBlox 結果セットをウォークスルーすることによって、タブ区切り形式のデータを Excel または他のアプリケーションにエクスポートする方法を示す上級者向けの内容です。さらに、BloxContext からのリレーショナル・レポートのために ReportBlox または SQLDataBlox にアクセスする方法も示します。

以下に示すのは、BloxContext から ReportBlox が作成され、取得される場合の JSP ファイルです。

- ユーザーがデータをエクスポートするためにリンクが用意されています。リンクをクリックすると、別の JSP ファイルが呼び出され (toExcelWithTab.jsp)、レポート名が渡されます。
- レポート名は、BloxContext からこのレポートの ReportBlox に関する SQLDataBlox を識別するために使用されます。

```
<%@ taglib uri="bloxreporttld" prefix="bloxreport" %>
<%@ page import="com.alphablox.blox.*"  %>
<%@ page errorPage="error.jsp" %>
<%
  String reportName = "qcc";
  String reportDataName = reportName + "_data" ;
  String dataSourceName = "qcc-rdb";
  String query = "SELECT * from QCC where sales > 5000";

  //Get the ReportBlox and SQLDataBlox for this report from BloxContext
  BloxContext context = BloxContextFactory.getBloxContext(request, response);
  ReportBlox reportBlox = (ReportBlox)context.getBlox(reportName);
  SQLDataBlox report_dataBlox = (SQLDataBlox)context.getBlox(reportDataName);

  if( report_dataBlox != null )
  {
    report_dataBlox.setQuery( query );
    report_dataBlox.execute();
  }
%>

<html>
<head>
  <link href="/AlphabloxServer/theme/coleman.css" rel="stylesheet">
</head>
<body>
<a href="toExcelWithTab.jsp?reportname=<%= reportName %>"
target="<%= reportName %>_csv" >Send to CSV </a>
<hr>
<bloxreport:report id="report" bloxName="<%=reportName%>" interactive="false" errors="true" >
  <bloxreport:sqlData id="data" bloxName="<%= reportDataName %>"
    query="<%=query%>" >
    <bloxreport:dataSourceConnection
      dataSourceName="<%= dataSourceName %>" />
```

```
  </bloxreport:sqlData>
</bloxreport:report>
</body>
</html>
```

その後、toExcelWithTab.jsp ファイルは以下のアクションを実行します。

- MIME タイプは application/vnd.ms-excel に設定されます。これにより、ブラウザーが起動され、このページを Excel ワークシートとしてロードします。
- このレポートの SQLDataBlox からデータ結果セットを取得します。
- 結果セットをウォークスルーして、タブを列の区切り文字として、改行を行の区切り文字として追加します。この場合、Relational Reporting API 用の com.alphablox.pipeline.* パッケージをインポートする必要があります。

MIME タイプを application/vnd.ms-excel に設定します。

```
<%@ taglib uri="bloxreporttld" prefix="bloxreport" %>
<%@ page import="com.alphablox.blox.*,
                 com.alphablox.pipeline.*"  %>
<%@ page errorPage="error.jsp" %>
<%@ page import="java.io.*" %>
<%@ page contentType="application/vnd.ms-excel; charset=UTF-8" %

<%
String reportName = request.getParameter( "reportname" );
String delimiterName = "tab";
String reportDataName = reportName + "_data" ;
String title = "Render to tab-delimited format";
String lineDelimiter = "¥n";
String columnDelimiter = "¥t";

// Output to strings used to indicate the beginning and end of the report data
String dataSetBegin = "Data for " + reportName;
String dataSetEnd  = "¥nEnd " + reportName + " data";

%>

<bloxreport:sqlData id="data"
 bloxName="<%= reportDataName %>"

<%
// Get the results from SQLDataBlox using getData().
// IDataSet represents the relational data in the pipeline, with a column
// dimension and a row dimension-- IDimension.COLUMN_DIMENSION_NAME and
// IDimension.ROW_DIMENSION_NAME.
IDataSet dataSet = data.getData();
IDimension columnDimension = dataSet.getDimension(IDimension.COLUMN_DIMENSION_NAME);
int rowIdx=dataSet.getDimensionIndex(IDimension.ROW_DIMENSION_NAME);
int colIdx=dataSet.getDimensionIndex(IDimension.COLUMN_DIMENSION_NAME);
int rowCnt=dataSet.getDimension(IDimension.ROW_DIMENSION_NAME).getMemberCount();
int colCnt=dataSet.getDimension(IDimension.COLUMN_DIMENSION_NAME).getMemberCount();
int[] coordinates = new int[2];

out.println( "There are " + rowCnt + " rows and " + colCnt + " columns." ) ;

if( dataSetBegin != null )
  out.println( dataSetBegin );

// itterate over the column headers
for( int column = 0; column < colCnt; column++ )
{
  if( column != 0 )
    out.print( columnDelimiter );
    out.print( columnDimension.getMember( column ).getName() );
}
```

```
out.print( lineDelimiter );

// itterate over the actual data
for( int row = 0; row < rowCnt; row++ )
{
  if( lineDelimiter != null && row != 0 )
    out.print( lineDelimiter );

  for( int column = 0; column < colCnt; column++ )
  {
    if( column != 0 )
      out.print( columnDelimiter );
      coordinates[rowIdx]    = row;
      coordinates[colIdx] = column;
      out.print( dataSet.getValue( coordinates ) );
  } // end columns

} //end rows

if( dataSetBegin != null )
  out.print( dataSetEnd );
%>
```

照会の動的な変更

すべての Relational Reporting Blox は、Bean として BloxContext 内でインスタンス生成されます。 JSP ページの中で、これらの Bean に個々にアクセスしたり、それらのメソッドを呼び出したりことができます。これらの Blox に関連したメソッドについては、以下にある ReportBlox Javadoc を参照してください。

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

タグ属性値に加えた変更がページの再ロード時に再評価されることはないため、レポートを作成およびレンダリングした後で照会を動的に変更する場合には、サーバー上のオブジェクトを再利用する必要があります。オブジェクトを再利用するには以下のようにします。

- Blox に `bloxName` 属性を設定してください。
- Blox context から Blox を取得します。

  ```
  <%
    BlxoContext context = BloxContextFactory.getBloxContext(request, response);
    SQLDataBlox dataBlox  = (SQLDataBlox)context.getBlox( "myReportData" );
    dataBlox.setQuery( query );
    dataBlox.execute();
  %>
  ```

  ここで、″myReportData″ は SQLDataBlox に割り当てられる `bloxName` です。

以下の例は、セッションからオブジェクトを削除せず、より効果的にタスクを実行する、さまざまな方法を示しています。1 つの技法は、変更する必要のある Blox に、セッション属性から直接アクセスする方法です (例えば、新規照会を設定および実行するために SQLDataBlox にアクセスする)。

例 1: SQLDataBlox に直接アクセスし、その照会をリセットする

- この例では、ユーザーの選択する 2 つのラジオ・ボタンがあります。1 つは February 用、もう 1 つは March 用です。

- com.alphablox.blox.* の import ステートメントは ReportBlox API を使用するのに必要です。
- ReportBlox 内の SQLDataBlox には sqlDataBlox という id があります。初期ページ・ロード時に、その query が query2 に設定されます。これは February のデータを示しています。
- monthlyData は型キャストされて、タイプ SQLDataBlox になります。
- ユーザーが月を選択すると、ページが再ロードされ、要求された月が検査されます。setQuery() および execute() メソッドを使用して、 既存の SQLDataBlox の照会がリセットされ、実行されます。

```
<%@ taglib uri="bloxreporttld" prefix="bloxreport"%>
<%@ page errorPage="error.jsp" %>
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ page import="com.alphablox.blox.*" %>
<%
   String query2 = "SELECT month, sales FROM qcc WHERE year=2000 AND
month=2";
   String query3 = "SELECT month, sales FROM qcc WHERE year=2000 AND
month=3";
   String query  = query2; //defaults to February
   String monthNum = null;
   String thisURL  = "http://" + request.getServerName() +
      (request.getServerPort()==80?"":":" +
      request.getServerPort()) + request.getRequestURI();
   SQLDataBlox monthlyData = null;

   // if monthlyData already exists, there will be a sesson
   // object of same name. This will never happen in the first
   // pass through this page

   BloxContext context = BloxContactFactory.getBloxContext(request, response);
   monthlyData = (SQLDataBlox) context.getBlox( "sqlDataBlox" );
   monthNum = request.getParameter( "monthNum" );

   // User did request a monthnum
   if( (monthNum != null) && ! ("".equals( monthNum )) )     {
      if( "3".equals( monthNum ) )
      {
         query = query3;
      }
      if( monthlyData != null )
      { // if monthlyData exists, the query is executed through
        // a method on the Blox
         monthlyData.setQuery( query );
         monthlyData.execute();
      }
   }
%>
<html>
<head>
   <title>ReportBlox with Changing Query</title>
   <!--Uses the default style sheet; required for interactive
       mode-->
   <link rel="stylesheet"
   href="/AlphabloxServer/theme/report.css" />
</head>

<body bgcolor="#FFFFFF">
<form name="monthForm" action="<%=thisURL%>" method=POST  >
   <input type="radio" name="monthNum" value="2"
      OnClick="document.monthForm.submit()"
      <%=(! "3".equals( monthNum ))?"checked":""%> >February
   <input type="radio" name="monthNum" value="3"
```

```
      OnClick="document.monthForm.submit()"
      <%=("3".equals( monthNum ) )?"checked":""%> >March
</form>

<b>The current Query is:</b>
<pre><%=query%></pre>
<bloxreport:report id="profitReport" interactive="false">
   <bloxreport:sqlData id="sqlDataBlox"
      query="<%=query%>" >
      <bloxreport:dataSourceConnection
         dataSourceName="qcc-rdb" />
   </bloxreport:sqlData>
</bloxreport:report>
```

Blox Sampler - Relational Reporting の例にある『Sales Report with HTML Links』の例は、同様の技法を使用して、照会を動的に設定します。

例 2: グローバルな refreshReport() JavaScript メソッドを使用して、ページ全体をリフレッシュせずに照会を動的に設定する

この例は、サーバー上で照会を設定してから現行ページの ReportBlox をリフレッシュする別の JSP を呼び出すことにより、ページを再ロードせずに照会を動的に設定する方法を例示しています。

- ユーザーがフォームの選択などによって別の照会を要求する場合 (新規ページのロード要求や、ページ全体の最新表示の要求ではない)、この例では iframe をフォーム通知アクションのターゲットとして使用します。これにより、現行ページを再ロードせずにサーバー・サイドのコードを実行できます。
- 呼び出された JSP ページは、基礎となる SQLDataBlox の照会をリセットします。
- 基礎となる照会に対する変更を反映するよう、現行ページ上の ReportBlox だけをリフレッシュするには、以下のグローバルな `refreshReport(`*ReportBloxName*`)` JavaScript メソッドを使用します。

  ```
  <script>
     function refresh( reportName ) {
        refreshReport( reportName );
     }
  </script>
  ```

  注: `refreshReport()` JavaScript メソッドは、対話式レポートでのみ作動します。
- この例はまた、`bloxName` の使用も例示しています。動的には設定できない `id` タグ属性とは異なり、オプションの `bloxName` を使用すれば、タグを使って Blox 名を動的に作成できます。Blox に `bloxName` の値を指定した場合、次のようになります。
  - この `bloxName` は DB2 Alphablox サーバーがこのオブジェクトを認識する際の Blox の名前となります。
  - この `bloxName` は、レンダリングされた JavaScript オブジェクトの名前となります (グローバルな JavaScript 関数 `refreshReport()` が ReportBlox を参照するときに使用する)。
  - `id` は、ここでは JSP ページ内で使用する Java スクリプト変数としてのみ使用されます。

完全な例については、Blox Sampler-Relational Reporting の例のセットの『Saving and Exporting Data with Dynamic Queries』の例を参照してください。

ユーザーがレポートの新しい基準を指定する場合、呼び出されたページは BloxContext にある ReportBlox および SQLDataBlox オブジェクトを再利用し、新規照会を設定します。

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

<%
   // Some code omitted here that dynamically sets the new query.
   // The variables reportName and reportDataName are set to the values of bloxName
   // for ReportBlox and SQLDataBlox.


   BloxContext context = BloxContextFactory.getBloxContext(request, response);
   ReportBlox reportBlox = (ReportBlox)context.getBlox(reportName);
   SQLDataBlox dataBlox = (SQLDataBlox)context.getBlox(reportDataName);
   dataBlox.setQuery( query );
   dataBlox.execute();
%>

<html>
<head>
</head>

<body>
<script>
   <% if( isError )
      {//pop the error message in an alert dialog if there was one
         out.println( "alert( ¥"" + errorMessage + "¥")" );
      }
      else
      {//there is a js function in the parent page called refresh( reportName )
         out.println("parent.refresh( ¥"" + reportName + "¥");");
      }
       %>
</script>
</body>
</html>
```

レンダリングされたレポート内のデータ行およびセル値へのアクセス

レポート内の各データ行は HTML の <TR> タグでレンダリングされ、各データ・セルは <TD> タグでレンダリングされます。JavaScript を使用すると、データ行またはそのセル値にアクセスできます。例えば、TextBlox の data タグを使用すると、データ・セルに対し、クリックされたときに JavaScript 関数を呼び出してセルの値を取得するような HTML コードを追加できます。データ・セルから、その親であるデータ行へ進むこともできます。以下の例は、ユーザーが行上の情報アイコンをクリックしたときに、行全体のセル値を取得する方法を示しています。

この例では、TextBlox の data タグを使用して、情報アイコンを列に追加します。この列の列ヘッダーは空のスペースに置換され、データは情報アイコンに置換されます。

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

ユーザーが i.gif アイコンをクリックすると getRowValues() 関数が呼び出され、現行のアンカー・オブジェクトが渡されます。すると getRowValues() 関数は、クリックされたオブジェクトの親オブジェクト (行オブジェクト) を取得し、表の行にある各セルに対して繰り返し処理を行い、セル値を取得します。

```
function getRowValues( anchorObj )         {
  var currObj = anchorObj;
  while( (currObj.tagName != "TR") && (currObj.tagName != null) ){
    currObj = currObj.parentElement;
  }

  var rowObj    = currObj;
  var cellCount = rowObj.cells.length;
  var tdObj     = rowObj.firstChild;
  var colValues = new Array( cellCount );

  for( i = 0; i < cellCount; i++ ) {
    currObj = rowObj.cells[i];
    while( (currObj != null) && (currObj.innerText == null) ) {
      currObj = currObj.firstChild;
        }

    if( currObj != null )
      colValues[i] = currObj.innerText;
  }

  alert( "ColValues is: " + colValues );
}
```

Blox Sampler - Relational Reporting の『Saving and Exporting Data with Dynamic Queries』の例は、この技法や他の JavaScript の技法を例示しています。

第 11 章 開発およびトラブルシューティングのヒント

この節では、リレーショナル・レポートの開発作業に役立つ、一般的な設計上の考慮事項およびトラブルシューティングのヒントを解説します。

一般的なヒントおよび開発ステップ

リレーショナル・レポートを作成するための必須のステップについては、必ず 27 ページの『一般的なレポート開発ステップ』を確認してください。また、リレーショナル・レポートの開発を成功させるための重要な注記および考察を記載した 25 ページの『一般的な開発ヒント』に進んでください。

設計上の考慮事項

以下は、リレーショナル・レポートを開発する際に考慮すべき、いくつかの設計上のヒントです。

- レポートに表示されるデータを制限するために OrderBlox を使用できますが、レポートを対話モードでレンダリングした場合 (`<bloxreport:report>` タグの `interactive` 属性を `true` に設定した場合)、OrderBlox を追加した後にさらになおデータ操作 (データのフィルター操作やソート) がないなら、ユーザーは「**すべて表示**」または「**グループのクリア**」を選択して、すべてのデータを回復できることに注意してください。
- 対話式 Report Editor をオンにして、ユーザーがレポート内のレイアウトおよびブレーク・グループを動的に変更できるようにする場合には、JSP ファイル内で指定したブレーク・グループをユーザーがクリアして独自のブレーク・グループを作成できることに留意してください。この場合、`<bloxreport:groupHeader>` タグを使用してグループ・ヘッダーを設計する際には、必ず `<member/>` 置換変数を使用してブレーク・グループの名前を抽出するようにしてください。また、`<bloxreport:groupTotal>` タグの中で `<value/>` 変数を使用して、ブレーク・グループの指定された列メンバーの集約値を抽出するようにしてください。
- 一般に、対話式レポートはブラウザーがレンダリングするのにより多くの時間を要します。レポートに何百ものデータ行が含まれる場合には、差異がはっきりします。
- Netscape ブラウザーを使用するユーザーがいる場合には、以下の事柄に留意してください。
  - 対話式レポートの表示または動作は正しく行われません。

  - StyleBlox を使用すると、レンダリング速度が低下することがあります。(107 ページの『スタイル設定がパフォーマンスに与える影響』を参照)
- レポートが対話式の場合には、ユーザー・インターフェースがどのように機能するかを説明するユーザー・ヘルプを提供できます。ユーザーにヘルプ・ファイルを提供したい場合のために、ヘルプ・ファイルが用意されています。ヘルプ・ファイルへの独自のリンクを提供する必要があります。詳細は、次の節である 106 ページの『ユーザー・ヘルプの提供』で説明します。

ユーザー・ヘルプの提供

Report Editor ユーザー・インターフェースが作動する仕方を説明するユーザー・ヘルプ・ファイルは、複数の言語で用意されています。Report Editor からエンド・ユーザー・ヘルプ・ファイルへのリンクはありません。これらのオンライン・ヘルプ・ファイルをユーザーに提供する場合には、ヘルプ・ファイルのコピーを作成し、アプリケーションからヘルプ・ファイルのコピーに飛ぶリンクまたはボタンを追加します。

ヘルプ・ファイルは以下の場所にあります。

```
<db2alphablox_dir¥system¥documentation¥help¥ReportBlox¥<locale>
```

ここで、<db2alphablox_dir> は、DB2 Alphablox のインストール先のディレクトリーです。

エンド・ユーザー・ヘルプは、HTML ページのセット、スタイル・シート、およびイメージのセットで構成され、容易にカスタマイズできます。以下の例では、ヘルプ・ファイルのコピーがアプリケーション・フォルダーの help/ サブディレクトリー下にあり、JavaScript を使用してユーザー・ヘルプが別個のブラウザー・ウィンドウにロードされます。

```
<script>
function openHelp(url) {
   window.open(url,'reportHelp','width=500,height=560,
   scrollbars=yes,toolbar=no,menubar=no,directories=no,
   status=no');
}
</script>
...
<a href="javascript:openHelp('help/index.html');">
<img src="images/help.gif" alt="Instructions on using interactive reports"
border=0></a>
```

注: ヘルプ・ファイルを変更せずに使用する場合は、以下を使用して、documentation ディレクトリーに直接リンクすることもできます。

```
<a href="javascript:openHelp('/AlphabloxServer/documentation/help/
ReportBlox/{<locale>/index.html');">
```

documentation ディレクトリー内のヘルプ・ファイルの元のコピーを使用すれば、DB2 Alphablox のアップグレード時にヘルプ・ファイルが常に最新のものとなる、という利点があります。同じ理由で、ヘルプ・ファイルをカスタマイズする必要がある場合には、必ず自分独自のコピーを変更してください。そうしなければ、アップグレード時に変更が失われることになります。

ヘルプのローカリゼーション

英語以外の言語でレポートを正しく表示するには、page ディレクティブの contentType 属性を使って、UTF-8 文字セットを使用するように指定する必要があります。contentType 属性を使用すると、MIME タイプと文字セットを定義できます。

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

別の言語を使用するユーザーがいる場合には、HTTP request オブジェクトを通してブラウザーのロケールを検出できます。この場合、必要なすべてのバージョンのヘルプ・ファイルをコピーし、request.getLocale() メソッドを使用してロケールを取得した後、正しいバージョンのヘルプへのリンクを動的に設定します。

スタイル設定がパフォーマンスに与える影響

StyleBlox を使用してスタイルを設定すると、レンダリングされたレポートでは、各データ・セルに対してスタイルが指定されます。例えば、数値データのテキスト・カラーを「blue」に設定する以下のコードがあるとします。

```
<bloxreport:style>
   <bloxreport:numeric style="color: blue"/>
</bloxreport:style>
```

数値データのデータ・セルごとに、以下のような HTML コードが生成されます。

```
<td class='data' style='text-align: right;color: blue;'>
145</td>
<td class='data' style='text-align: right;color: blue;'>
12.55</td>
```

「text-align: right」がスタイル・リストの先頭にあることに注意してください。これは ReportBlox には、データ・タイプに基づくデフォルトのテキスト配置があるためです。Netscape ブラウザーでは、レポートが大規模で何千ものデータ・セルが含まれる場合には、レポートをレンダリングするためにかかる速度が著しく低下します。Netscape ブラウザーにおいて大規模なレポートのパフォーマンスを改善するためには、表セルごとに長いスタイル・リストを指定することにならないよう、StyleBlox の使用を制限するようにしてください。また、デフォルトのテキスト配置をなしに設定して、上書きすることもできます。

```
<bloxreport:style>
  <bloxreport:text style=""/>
  <bloxreport:column style="text-align: center" columnName="Area"/>
  <bloxreport:column style="text-align: left" columnName="Product"/>
</bloxreport:style>
```

これは、デフォルトのテキスト配置を削除して、次のようにスタイル・リストを削減します。削減前 ：

```
<td class='data'
    style='text-align: left;text-align: center;'> N. Cal</td>
```

削減後 ：

```
<td class='data' style=';text-align: center;'> N. Cal</td>
```

一般に、スタイル・クラスを使用したほうが、StyleBlox を使用するよりも効率的です。というのは StyleBlox 内のスタイル・セットは、そのスタイルの適用対象であ

るそれぞれのコンポーネント (各データ・セルなど) に追加され、その要素が現れるたびに繰り返される、長い CSS スタイル・ストリングを作成することになるからです。

Internet Explorer では CSS は通常、問題となりません。

Reporting Blox タグの一般的なエラー

以下のリストには、Blox Report タグ・ライブラリーを使って作業するときに最もよく生じるエラーのいくつかが記載されています。

Blox Report タグ・ライブラリーの taglib ディレクティブを組み込むのを忘れる

Blox Report タグ・ライブラリーの taglib ディレクティブを JSP ページの先頭に配置し損なうと、どのタグも認識されないため、レポートがレンダリングされません。Relational Reporting Blox を含む JSP ページすべてに、以下の taglib ディレクティブがあるかを確認してください。

```
<%@ taglib uri="bloxreporttld" prefix="bloxreport" %>
```

Relational Reporting Blox の正しい接頭部を使用するのを忘れる

Blox タグ・ライブラリー (blox.tld) のための taglib ディレクティブがある場合、リレーショナル・レポートに blox 接頭部を使用しようとすると、JSP エラーが生じます。例えば、<blox:report> タグの場合、インポートされたタグ・ライブラリーにそのようなタグはないというエラーを受け取ります。

Blox Report タグ・ライブラリーは Blox タグ・ライブラリーとは別なので、JSP ページの先頭に正しい taglib ディレクティブがあることを確認し、2 つのタグ・ライブラリーに対して必ず別個の接頭部を使用してください。

タグまたはタグ属性の大/小文字の間違い

大/小文字およびスペルの間違いがあると、コンパイル・エラーになります。このような場合、JSP コンパイラーは、無効なタグまたはタグ属性が使用されていることを示す例外をスローします。タグと属性名の最初の語はすべて小文字です。後続の語の最初の文字は大文字で始まります。例えば、<bloxreport:sqlData />、<bloxreport:dataSourceConnection />、<bloxreport:groupHeader />、<bloxreport:groupFooter />、または <bloxreport:groupTotal/> となります。

スタイル・シートをインクルードするのを忘れる

コンテキスト・メニューと「レポート・スタイル」ダイアログ・ボックスの表示に関連する各クラスに定義されたスタイルがないと、Report Editor ユーザー・インターフェースは正しく動作しません。Report Editor ユーザー・インターフェースは雑然と崩れた状態になり、コンテキスト・メニューが間違った場所に、間違ったフォント・サイズでポップアップ表示します。

列ヘッダー、データ、またはブレーク・グループ・フッターの表示に関連する各クラスに定義されたスタイルがないと、レポートは非常に単純な HTML 表としてレンダリングされることになり、魅力に欠けた読みにくいものとなります。

スタイル・シートはカスケードするので、すぐに使用可能な付属のスタイル・シートを使用するようにし、変更したいクラスのスタイルだけを修正する独自のスタイル・シートを追加する (またはインライン・スタイルを使用する) のが最善です。

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

更新されたページがコード変更を反映しない

Blox タグおよびこれらのタグ内の JSP ステートメントが解釈されるのは、ページが初めてロードされるときだけです。そのセッションではオブジェクトがサーバー上にすでに存在しているため、そのオブジェクトが再利用され、結果として、加えた変更は反映されません。ブラウザーの新規インスタンスを開いて新規セッションを開始する必要があります。

間違ったメンバーまたは列名を参照している

ソート、計算、グループ化、フィルター処理などのデータ操作タスクを実行する際、メンバー名の先頭が文字でない、あるいはメンバー名に特殊文字 (a-z、A-Z、0-9、または下線以外の文字) が含まれているなら、メンバー名を大括弧で囲む必要があります。そうすることで、DB2 Alphablox Relational Reporting エンジンに対して、式の中でこれらが変数であるということを示します。大括弧がないと、レポート・システムがメンバーを正確に識別できない可能性があります。グループ・ヘッダーおよびフッターの設定やデータおよび列ヘッダーのスタイル設定など、レポート・レイアウトのフォーマット設定およびスタイル設定タスクを実行する場合には、レンダリングされたレポートを扱っていることになるため、大括弧はもう必要ありません。というのは、テキスト・ストリングがメンバー名か式かについて混同する恐れはないからです。

データ操作とレポート・レイアウトにはこのような差があるので、Relational Reporting Blox には `member` (または `members`) 属性を持つものと、`columnName` 属性を持つものとがあります。例えば、GroupBlox はメンバー属性を持ち、TextBlox および StyleBlox 内部のサブタグは `columnName` 属性を持ちます。詳しくは、11 ページの『列とメンバー』と 22 ページの『メンバー ID と表示名』を参照してください。

トラブルシューティングのヒント

ReportBlox からのメッセージは、DEBUG レベル・メッセージとして DB2 Alphablox サーバー・ログに送られます。ReportBlox およびそのサポートする Blox からのアクティビティーを調べる場合は、メッセージ・レベルを DEBUG に設定できます。ただし、ログ・ファイルはかなり急速に大容量になり得ることにご注意ください。キャッチされていないエラーをキャッチするためには、エラー・キャッチ・ページを作成し、これがエラー・レポートを処理するよう、自分が作成するすべての JSP ページに指定しなければなりません。28 ページの『ErrorBlox を使用してエラー・レポートを改良する』を参照してください。

以下に、一般的なトラブルシューティングのヒントをいくつか示します。

- 参照されるスタイル・シートが実際に存在し、正確な場所にあることを確認してください。ブラウザーが異なれば、スタイル・シートの欠落に対する許容度も異なります。

- <bloxreport:report> タグの interactive 属性を true に設定しているのに Report Editor ユーザー・インターフェース内の対話式コンテキスト・メニューが表示されない、あるいはレポートの先頭にプレーン・テキストの行として表示される場合、その原因は、レポートに関連したスタイル・シートがないこと、あるいは参照されているスタイル・シートが見つからないである場合が少なくありません。自分のスタイル・シート、または付属のスタイル・シートへのリンクを追加してください。

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

- レポートを表示している間にブラウザーがハングしているように見える場合、原因はおそらく、レンダリング中のレポートのサイズです。レポートは HTML 表としてレンダリングされることに留意してください。10,000 行 x 10 列の表は、どのブラウザーでも表示するのに数分かかる可能性があります。
- メンバー名には大/小文字の区別があります。メンバー「Cost」を「COST」または「cost」として参照すると、エラーになる場合
  (CalculateBlox、OrderBlox、FilterBlox、SortBlox、および GroupBlox など、処理する正確なメンバーを識別する必要があるものの場合) と、無視される場合
  (FormatBlox または StyleBlox など) とがあります。

  注: IBM® DB2 Universal Database™、Oracle、または Microsoft SQL Server を使用していて、SELECT リストで列名を変更する際、大/小文字を保持する場合には列名を二重引用符で囲んでください。以下に例を示します。

  ```
  SELECT FROM myTable total_sq_ft AS "Sq_Ft", sq_ft_pct AS "Pct"
  ```

  JSP ページに引用符を含める場合には、必ずそれを円マーク (¥) でエスケープしてください。

  ```
  "SELECT FROM myTable total_sq_ft AS ¥"Sq_Ft"¥, sq_ft_pct AS ¥"Pct¥""
  ```

- page ディレクティブ内で指定したエラー処理ページが、正確な場所に実際に存在することを確認してください。そうでないと、ページが見つからないというエラーが発生します。

ErrorBlox を使用したエラー処理

デフォルトで、照会がデータを戻さないときには、メッセージ「データがありません」が表示されます。データ形式変更中、エラーのためにパイプラインをさらに下って行くデータがなくなると、「データがありません : レポート・データの生成中にエラーが起きました」というデフォルトのメッセージが表示されます。これは例えば、存在しないメンバーに基づいてデータをグループ化またはソートしようとした場合に発生することがあります。

ReportBlox の errors 属性が false (デフォルト) に設定されていると、例外がインターセプトされます。エラー処理ページを提供して、エラーをキャッチする try/catch ブロックを使用しなければなりません。そうしなければ、照会が誤っている場合やデータ・ソースが使用不能な場合に、なじみのないエラー・メッセージがユーザーに表示されることになります。

場合によっては、問題の発生箇所を追跡するために、例外を確認する必要があります。Relational Reporting Blox がスローする例外はネストしており、根本原因を識別するのが難しい場合があります。ネストした例外を繰り返し処理して根本原因に達するには、以下のようにします。

1. ReportBlox の `errors` タグ属性を `true` に設定します。すると例外がインターセプトされなくなり、それらを try/catch ブロックでキャッチできます。
2. ErrorBlox の `getRootCause(Throwable`*exception*`)` メソッドを使用します。これは、指定された例外のネストした例外を繰り返し処理して、根本原因の例外を戻します。ErrorBlox には `getNextException(Throwable` *exception*`)` メソッドもあります。これは、指定された例外の次のネストした例外を戻します。

以下の例は、例外をキャッチして、根本原因を識別する方法を示しています。

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

<html>
<head>
  <title>Exception Test</title>
  <link rel="stylesheet" href="/AlphabloxServer/theme/report.css" />
</head>

<%
  String query="SELECT product_name, area, location From qcc WHERE
week_ending = '2000-04-08'";
%>

<body>
<%
  try { %>
    <bloxreport:report id="testTryCatch" errors="true">
      <bloxreport:sqlData query="<%= query%>" >
         <bloxreport:dataSourceConnection dataSourceName="qcc-rdb" />
      </bloxreport:sqlData>
    </bloxreport:report>
<% }
  catch ( Exception e ) {
    ErrorBlox eblox = new ErrorBlox();
    Throwable msg = eblox.getRootCause(e);
    out.println("<br><b> This Exception was captured</b><br> "+
msg.getMessage());
  }
%>
</body>
</html>
```

第 12 章 Relational Reporting Blox のタグのリファレンス

この節では、それぞれの Relational Reporting Blox に関連したタグをリストします。それぞれの Blox ごとに、その関連タグ、タグ属性、プロパティー、メソッド、メソッド構文、および使用法を記載します。Relational Reporting Blox で使用可能な Java メソッドについては、DB2 Alphablox 管理ページのヘルプのリンク、または DB2 Alphablox ライブラリー・ページ (http://www.ibm.com/software/data/db2/alphablox/library.html) から、Relational Reporting Javadoc を参照してください。

Blox タグの使用

Relational Reporting のカスタム Blox タグは、DB2 Alphablox の他の Blox のタグと同様の仕方で動作します。

- Relational Reporting Blox のカスタム・タグを使用する場合は、以下の行を JSP ファイルの先頭に組み込む必要があります。

  ```
  <%@ taglib uri="bloxreporttld" prefix="bloxreport" %>
  ```

- タグは `<bloxreport:...>` で開始し、対応する終了タグ `</bloxreport:...>` で終了します。タグにネストされたタグが含まれていない場合は、別個の終了タグを使わずに、斜線 (`<bloxreport:calculate ... />`) で終了できます。
- `<bloxreport:report>` タグは `<blox:present>` タグ同様、他のほとんどすべての Relational Reporting Blox に対するラッピング・タグです。ただし、ErrorBlox、PdfBlox、および PersistenceBlox は例外です。

- タグおよび属性名は常に、最初の単語が小文字、後続の単語の最初の文字が大文字です (例えば、dataSourceConnection)。

<bloxreport:report> タグと <blox:present> タグの大きな違いは、<bloxreport:report> タグ内でネストした Blox はスクリプト指定可能な独自の id を持てるのに対し、<blox:present> タグ内でネストした Blox はそれができない、という点です。これは、Relational Reporting 機能をサポートする Blox が機能上明確に分かれているためです。

ReportBlox に関連したすべてのタグのリストについては、153 ページの『コピー・アンド・ペースト用の Relational Reporting タグ』を参照してください。

CalculateBlox

CalculateBlox は算出メンバーをレポートに追加します。算出メンバーには、計算式、およびこの新規メンバーが出力される位置を指定する必要があります。CalculateBlox の JSP タグは <bloxreport:calculate> です。

構文

```
<bloxreport:report id="idName">
   ...
   <bloxreport:calculate id = "idName"
      expression="calculatedMemberName= calculationExpression"
      index = "int">
   </bloxreport:calculate>
   ...
</bloxreport:report>
```

使用法

<bloxreport:report> タグ内では、複数の <bloxreport:calculate> タグを使用できます。ReportBlox は計算を、それらが指定された順序で実行します。式は、算出メンバーの名前と計算式の 2 つの部分で構成されます。下記の例では、

```
"[Total Sales] = [Unit Price] * [Units_Sold]"
```

「Total Sales」が算出メンバーの名前で、その値は [Unit Price] * [Units_Sold] の結果です。式に大括弧 ([ ]) が必要なのは、メンバー名にスペースが含まれているからです。

CalculateBlox を使用する際には、以下の点に注意してください。

- メンバー名にすでに [ ] が含まれている場合、右側に 1 つ「]」を追加して、メンバー名の終わりであることを示してください。例えば、式にメンバー名 West[CA] を指定する場合は、[West[CA]]] とします。
- 計算式でサポートされている演算子は、+、-、*、および / です。
- 計算の演算子の区切り記号としてサポートされているのは () です。以下に例を示します。

  ```
  <bloxreport:calculate
     expression = "[Profit%] = Sales/(Unit_Cost * Units_Sold)"
  />
  ```

- サポートされている計算関数は rank()、percentOfTotal()、runningTotal() 、および runningCount() です。以下に例を示します。

  ```
  <bloxreport:calculate
     expression = "[% Total] = percentOfTotal(Units)"
  />
  ```

  これらの関数はレポートがどのようにグループ化されるかに関連があり、詳しくは 74 ページの『グループ・ベースのサマリー列の計算』で説明されています。
- サポートされている演算子は数値データにのみ作用します。これには整数、浮動小数点、および通貨が含まれます。ストリング、日付、時間、またはブール・データ型に対して計算を実行することはできません。

計算に使用するメンバーに欠落値または NULL 値が含まれる場合、計算結果は欠落データになります。

タグの説明

<bloxreport:calculate> タグ

タグ属性	必須	デフォルト	説明
id	いいえ		CalculateBlox のこのインスタンスの固有 ID。
bloxName	いいえ		サーバー上の CalculateBlox のこのインスタンスの固有 ID。名前は動的に設定できます。93 ページの『セッションの有効範囲の管理』を参照してください。
expression	いいえ	「Total」という名の新規列にあるすべての数値列の合計。	指定された算出メンバーを追加するために CalculateBlox が評価する式。 無効または空の式は、エラーとなります。 expression 属性を指定しないで <bloxreport:calculate> タグを追加すると、CalculateBlox は自動的に「Total」という算出メンバーを追加します。その値は、すべての列メンバーの合計です。 有効な式の構文については、21 ページの『式の構文』を参照してください。
index	いいえ	メンバーのカウント	算出メンバーが出力される、Column ディメンション内の位置。最初のメンバーは 0。 デフォルトで、算出メンバーは末尾に追加されます。

例

```
<bloxreport:calculate
   expression = "[Profit%] = Sales/GrossMargin"
   index = "4"
/>
```

このタグは、Profit% という算出メンバーを 5 番目のメンバーとして追加します。Profit% は Sales を GrossMargin で除算して算出されます。

参照

42 ページの『算出列の追加』、74 ページの『グループ・ベースのサマリー列の計算』。

DataSourceConnectionBlox

DataSourceConnectionBlox は、DB2 Alphablox に定義されたリレーショナル・データ・ソースへの接続を表現します。DataSourceConnectionBlox の JSP タグは <bloxreport:dataSourceConnection> です。DataSourceConnectionBlox のタグは、<bloxreport:sqlData> の下にネストする必要があります。それにもかかわらず、これは Blox であり、コード内で後ほど参照するための ID を割り当てることができます。

構文

```
<bloxreport:report id = "idName">
   <bloxreport:sqlData>
      <bloxreport:dataSourceConnection
         id = "dataSourceIdName"
         dataSourceName = "dataSourceName" >
      </bloxreport:dataSourceConnection>
   </bloxreport:sqlData>
</bloxreport:report>
```

使用法

<bloxreport:sqlData> タグに追加できる <bloxreport:dataSourceConnection> タグは 1 つだけです。<bloxreport:report> タグに追加できる <bloxreport:sqlData> タグは 1 つだけです。

タグの説明

<bloxreport:dataSourceConnection> タグ

タグ属性	必須	デフォルト	説明
id	いいえ		DataSourceConnectionBlox のこのインスタンスの固有 ID。
bloxName	いいえ		サーバー上の DataSourceConnectionBlox のこのインスタンスの固有 ID。名前は動的に設定できます。93 ページの『セッションの有効範囲の管理』を参照してください。
dataSourceName	いいえ		DB2 Alphablox ホーム・ページを介して定義されたリレーショナル・データ・ソースの名前。非リレーショナル・データ・ソースはエラーとなります。
userName	いいえ		指定されたデータ・ソースにアクセスするためのユーザー名。
password	いいえ		データ・ソースにアクセスするために指定するユーザー名のパスワード。 パスワードが空ストリングの場合には、password="" とします。 パスワードがブランクまたはヌルの場合は、パスワードを指定してはなりません。

例

```
<bloxreport:report id="profitReport">
   <bloxreport:sqlData>
      <bloxreport:dataSourceConnection
         id = "myDataSource"
         dataSourceName = "chocoblocks"
         userName = "sa"
         password = "allmighty">
      </bloxreport:dataSourceConnection>
   </bloxreport:sqlData>
</bloxreport:report>
```

上記のコードは、「profitReport」という ReportBlox のインスタンスの SQL データ・ソースとして、「chocoblocks」という定義済みデータ・ソースを指定します。ユーザー名は「sa」、パスワードは「allmighty」です。

ErrorBlox

ErrorBlox は、エラーの詳細を HTML 表に出力します。ErrorBlox の JSP タグは <bloxreport:error> です。

構文

```
<bloxreport:error id = "idName">
</bloxreport:error>
```

使用法

<bloxreport:error> タグは、カスタム JSP エラー・レポート・ページ内で使用してください。JSP ファイルをエラー・レポート・ページとして識別するには、ファイルの先頭に <%@ page isErrorPage="true" %> を追加してください。さらに、通常の JSP ページ内で以下のディレクティブを使用して、使用するカスタム JSP エラー・レポート・ページを指示してください。

```
<%@ page errorPage="yourErrorPage.jsp" %>
```

エラー・レポート JSP の作成方法の詳細は、28 ページの『ErrorBlox を使用してエラー・レポートを改良する』を参照してください。ErrorBlox を使用した例外のキャッチの詳細については、110 ページの『ErrorBlox を使用したエラー処理』を参照してください。

タグの説明

<bloxreport:error> タグ

タグ属性	必須	デフォルト	説明
id	いいえ		ErrorBlox のこのインスタンスの固有 ID。
bloxName	いいえ		サーバー上の ErrorBlox のこのインスタンスの固有 ID。名前は動的に設定できます。93 ページの『セッションの有効範囲の管理』を参照してください。

例

ErrorBlox を使用したカスタム JSP エラー・ページ:

```
<%@ taglib uri="bloxreporttld" prefix="bloxreport" %>
<%@ page isErrorPage="true" %>

<html xmlns="http://www.w3.org/1999/xhtml">
   <head>
      <title>My JSP Error Reporting Page</title>
      <link rel="stylesheet" href="/AlphabloxServer/theme/report.css"
 type="text/css" />
   </head>

   <body>
      <H1>Error Reporting Page</H1>
      <bloxreport:error id="errorBlox" />
   </body>
</html>
```

FilterBlox

FilterBlox は、指定された式を基にして、数値の結果セットをフィルターに掛けます。FilterBlox の JSP タグは <bloxreport:filter> です。フィルタリングにより除去されたデータは、結果セットから削除されます。

構文

```
<bloxreport:report id="idName">
   ...
   <bloxreport:filter id = "filterIdName"
      expression = "filterExpression" >
   </bloxreport:filter>
   ...
</bloxreport:report>
```

使用法

フィルター操作できるのは数値データだけです。<bloxreport:report> タグには、複数の <bloxreport:filter> タグを入れることができます。各 FilterBlox は、フィルター式を 1 つだけ使用します。AND や OR を使用した複合フィルターは指定できません。その代わり、複数の FilterBlox をチェーニングして、望む結果を得ることができます。後続の FilterBlox は直前の FilterBlox の結果に基づいてデータをフィルター操作するので、例えば売上データが 400 を超えるものか、100 未満のものだけを保持するなどの操作を実行する必要がある場合は、そうしたデータをデータベース環境で準備してから ReportBlox に取り出さなければなりません。

デフォルトでは、欠落データはフィルタリングにより除去されません。フィルタリングされた結果から欠落データを除外するには、isMissing() 関数を使用して別個のフィルターを追加します (否定の場合は not isMissing() を指定)。詳細については、下記の例を参照してください。

メンバー名に a-z、A-Z、0-9、および _ 以外の文字が含まれている場合、名前を大括弧 ([ ]) で囲まなければなりません。メンバー名を数と取り違える恐れがある場合も、混乱を避けるために [ ] を使用してください。メンバー名にすでに [または] が含まれている場合、右側に 1 つ「]」を追加して、メンバー名の終わりであることを示してください。例えば、式にメンバー名 West[CA] を指定する場合は、[West[CA]] とします。

タグの説明

<bloxreport:filter> タグ

タグ属性	必須	デフォルト	説明
id	いいえ		FilterBlox のこのインスタンスの固有 ID。
bloxName	いいえ		サーバー上の FilterBlox のこのインスタンスの固有 ID。名前は動的に設定できます。93 ページの『セッションの有効範囲の管理』を参照してください。

タグ属性	必須	デフォルト	説明

expression	はい		フィルタリングの基準。
			
			式に指定された条件に値が適合しない場合には、フィルタリングされます。フィルター式に有効な演算子は =、<、>、および != です。以下に例を示します。
			
			`expression = "Sales > 3000"`
			`expression = "[Product Code] != 200"`
			
			フィルター式の内部に計算演算子を入れることはできず (+、-、/、または * は許可されない)、式内部にメソッドまたは Java スクリプトレットを追加することもできません。無効な式はエラーとなります。
			
			次の 1 つの関数をフィルター式に使用できます。すなわち、`isMissing(`*memberName*`)` です。
			
			式の構文、および特殊文字またはスペースが含まれるメンバー名の指定方法については、21 ページの『式の構文』 を参照してください。

例

```
<bloxreport:report id="salesReport">
   ...
   <bloxreport:filter
      id = "filter1"
      expression = "Sales < 10000"
   />
   <bloxreport:filter
      id = "filter2"
      expression = "Sales > 0"
   />
   ...
</bloxreport:report>
```

上記のコード・ブロックは、Sales が 0 より大きく 10000 より小さいデータ行を保持します。欠落データは戻されることに注意してください。欠落データを除外するには、別のフィルターを追加します。

```
   <bloxreport:filter
      id = "filter3"
      expression = "not isMissing(Sales)"
   />
```

FormatBlox

FormatBlox は数値、日付、および欠落データの表示フォーマットを設定します。データベース内の NULL データは欠落データとして処理されます。フォーマットを指定しないと、次のようになります。

- クライアント・ロケールのデフォルトのフォーマット・タイプが適用されます。
- 欠落または NULL データのデフォルトの表示テキストが、空ストリングとなります。

FormatBlox の JSP タグは <bloxreport:format> です。FormatBlox は複数のデータ・タイプのデータ・フォーマットを処理し、それぞれのデータ・タイプには複数の属性があるので、それぞれのデータ・タイプごとに別個のタグが必要です。さらに、データ照会から戻されたデータの周りに HTML コードを追加できるかどうかを指定できるタグがあります。これらのタグは以下のとおりです。

- <bloxreport:numeric>
- <bloxreport:date>
- <bloxreport:missing>
- <bloxreport:html>
- <bloxreport:aggregation>

<bloxreport:numeric> タグは整数、浮動小数点、通貨を含む、すべての数値データ・タイプに適用されます。<bloxreport:date> タグは、java.util.Date から継承したすべてのデータ・タイプに適用されます。その中には Date、Time、および Timestamp サブクラスが含まれます。すべてのフォーマットは Java のフォーマット・マスクに準拠します。

構文

```
<bloxreport:report id="SalesReport">
   ...
  <bloxreport:format>
    <bloxreport:numeric format = "formatExpression1" />
    <bloxreport:numeric format = "formatExpression2"
       member = "memberName" />
    <bloxreport:date format = "formatExpression1" />
    <bloxreport:date format = "formatExpression2"
       member = "memberName" />
    <bloxreport:missing format = "stringToDisplay" member =
      "memberName" />
    <bloxreport:html member = "memberName1" />
    <bloxreport:html member = "memberName2" />
    <bloxreport:aggregation member = "memberName2"
format="formatExpression" />
  </bloxreport:format>
 ...
</bloxreport:report>
```

使用法

<bloxreport:report> タグに追加できる <bloxreport:format> タグは 1 つだけです。<bloxreport:format> タグの中では、複数の <bloxreport:numeric>、<bloxreport:date>、および <bloxreport:missing> タグを使用できます。以下の点に注意してください。

- フォーマット式には Java フォーマット・マスクを使用してください。
- メンバーを指定しないと、フォーマットはそのデータ・タイプのすべてのデータに適用されます。
- メンバーを指定した場合、指定したフォーマットはそのメンバーにのみ適用されます。
- 同じメンバーに対してフォーマットが 2 回指定されている場合には、最後に指定されたフォーマットが適用されます。

Java 10 進フォーマットについては、http://java.sun.com/j2se/1.4.2/docs/api/java/text/DecimalFormat.html を参照してください。 Java 日時フォーマットについては、http://java.sun.com/j2se/1.3/docs/api/java/text/SimpleDateFormat.html を参照してください。

FormatBlox はレポートのレイアウトのフォーマットを処理しません。レポートのレイアウトのフォーマットに関連したタスクについては、47 ページの『第 6 章 レポートとデータのフォーマット設定』を参照してください。フォント・サイズ、背景色、およびテキスト配置などのスタイルを指定するタグについては、145 ページの StyleBloxを参照してください。

タグの説明

<bloxreport:format> タグ

タグ属性	必須	デフォルト	説明
<bloxreport:format>			
id	いいえ		FormatBlox のこのインスタンスの固有 ID。
bloxName	いいえ		サーバー上の FormatBlox のこのインスタンスの固有 ID。名前は動的に設定できます。93 ページの『セッションの有効範囲の管理』を参照してください。
<bloxreport:numeric>、<bloxreport:date>、<bloxreport:missing>			
format	はい		データ・タイプのフォーマット。
member	いいえ		指定されたフォーマットを適用するメンバー。
<bloxreport:html>			
member	いいえ		メンバーの表示名。指定したメンバーについては、データが SQL 照会から戻されるとき、追加された HTML コードが保持されることになります。デフォルトでは、データの HTML をそのまま残すよう指定したメンバー以外、追加の HTML コードはみなエンコードされます。メンバーの表示名の例、および詳細については、56 ページの『照会から戻されたデータへの HTML コードの追加』を参照してください。 複数の <bloxreport:html> タグを指定できますが、それぞれに対して指定できるメンバーは 1 つだけです。すべてのメンバーに関して、戻されるデータの HTML コードを全部そのまま残すように指定するには、以下のように値を空ストリングに設定します。 member=""

タグ属性	必須	デフォルト	説明
<bloxreport:aggregation>			
member	いいえ		メンバーの表示名。
format	はい		指定されたメンバーの集約値 (グループ合計) に適用されるフォーマット。member を指定しないと、フォーマットはすべての集約値に適用されます。

例

```
<bloxreport:report id="SalesReport">
   ...
  <bloxreport:format>
    <bloxreport:numeric format="####.00;(####.00)" />
    <bloxreport:numeric format="$#,###.00;$(#,###.00)"     member="Sales"    />
    <bloxreport:aggregation member="Sales" format="$#,###;$(#,###)" />
    <bloxreport:date format="yyyy.MM.dd G 'at' hh:mm:ss z" />
    <bloxreport:date format="EEE, MMM d, ''yy" member="Date" />
    <bloxreport:missing format="Units Value Missing"
       member="Units" />
    <bloxreport:missing format="Sales Value Missing"
       member="Sales" />
  </bloxreport:format>
 ...
</bloxreport:report>
```

このコード例は以下の設定を行います。

- 正の数値データのデフォルト・フォーマットを「####.00」に設定し、負の数値データのデフォルト・フォーマットを「(####.00)」に設定します。例えば、1234.5 は 1234.50 となり、-1234.5 は (1234.50) になります。
- メンバー「Sales」の数値データを「$#,###.00;$(#,###.00)」に設定します。例えば、1234.5 は $1,234.50 となります。
- メンバー「Sales」の集約値を「$#,###;$(#,###)」に設定します。例えば、1234.5 は $1,235となります。
- 日付のデフォルト・フォーマットを「yyyy.MM.dd G 'at' hh:mm:ss z」に設定します。このフォーマットの一例は、2001.10.01 AD at 09:27:13 PDT です。
- メンバー「Date Member」の日付フォーマットを「EEE, MMM d,''yy」に設定します。このフォーマットの表示例は、Mon, October 1, '01 です。
- メンバー「Units」が欠落データの場合に表示されるテキストを「Units Value Missing」に設定します。
- メンバー「Sales」が欠落データの場合に表示されるテキストを「Sales Value Missing」に設定します。

GroupBlox

GroupBlox はブレーク・グループを作成します。GroupBlox の JSP タグは <bloxreport:group> です。メンバーごとに集約のサマリーが異なる場合があるため、そのメンバーのメンバー名および集約タイプを設定する、別個の <bloxreport:aggregation> タグが必要です。

構文

```
<bloxreport:report id="idName">
   ...
   <bloxreport:group members="breakmember1, breakmember2, ...">
      <bloxreport:aggregation
         member="memberName"
         type="aggregationType">
      </bloxreport:aggregation>
   </bloxreport:group>
   ...
</bloxreport:report>
```

使用法

<bloxreport:report> タグ内には、複数の <bloxreport:group> タグを入れることができます。それぞれの <bloxreport:group> タグ内には、複数の <bloxreport:aggregation> タグを入れることができます。members 属性にブレーク・グループを指定する順序によって、ブレーク・グループ・レベルが決まります。例えば、次のタグがあるとします。

```
<bloxreport:group
   members="Product, Country">
</bloxreport:group>
```

このタグは、レポートをまず Product でグループ化し、次いで Country でグループ化します。レポート全体 (全製品を表す) がレベル 1、個々の製品グループがレベル 2、個々の国グループがレベル 3 になります。このグループ・レベルは、TextBlox のネストしたタグに指定したレベルに対応します。

```
<bloxreport:text>
   <bloxreport:groupHeader level = "int"
     text="Some group header text here" />
</bloxreport:text>
```

または

```
<bloxreport:text>
   <bloxreport:groupFooter level = "int"
     text="Some group footer text here" />
</bloxreport:text>
```

ブレーク・グループのヘッダーおよびフッターのフォント・サイズ、カラー、および背景色などのスタイルを指定するには、12 ページの『リレーショナル・レポートのスタイル設定』および 57 ページの『レポートに表示されるデータのスタイル設定』を参照してください。

タグの説明

<bloxreport:group> タグ

タグ属性	必須	デフォルト	説明
<bloxreport:group>			
id	いいえ		GroupBlox のこのインスタンスの固有 ID。
bloxName	いいえ		サーバー上の GroupBlox のこのインスタンスの固有 ID。名前は動的に設定できます。93 ページの『セッションの有効範囲の管理』を参照してください。
members	いいえ		グループのメンバー。 列の集約データ (sum、average、count、max、および min) を、ブレーク・グループのないレポートの末尾に追加するには、members の値を空ストリングに設定します (members = "")。
<bloxreport:aggregation>			
member	はい		集約値を提供するためのメンバー。
type	いいえ	sum	集約タイプ。有効な値は以下のとおりです。 • sum • average • count • max • min • none 以下の点に注意してください。 • すべての数値データ列のデフォルトの集約タイプは sum です。 • タイプを指定していない場合、デフォルトは sum です。 • タイプ count はストリングと数値の両方に対して作用します。他のすべては、数値に対してのみ作用します。 • 無効なデータ・タイプに対して集約を指定すると、無視されます。 • 集約値を提供するためのメンバーがない場合には、無視されます。 • 欠落データがある場合、それらは集約から除外されます。つまり、それらはデータ・カウントまたは他のタイプの集約に含まれません。

例

```
<bloxreport:group members = "Product">
   <bloxreport:aggregation member = "Sales" type = "sum" />
   <bloxreport:aggregation member = "Units" type = "average" />
</bloxreport:group>
```

上記のコードは、Product 別のブレーク・グループを作成し、サマリー行の集約値として、States のカウント数と Units の平均を表示します。集約タイプを指定していない他のすべてのメンバーについては、それらの合計が表示されます。

MembersBlox

MembersBlox は、レポートに包含するメンバーと除外するメンバーを指定します。除外されたメンバーは、結果セットから存在しなくなります。MembersBlox の JSP タグは <bloxreport:members> です。

構文

```
<bloxreport:report id = "idName">
   ...
   <bloxreport:members id = "membersIdName"
      excluded = "member1, member2,..."
      included = "member1, member2,..." >
   </bloxreport:members>
   ...
</bloxreport:report>
```

使用法

<bloxreport:report> タグに複数の <bloxreport:members> タグを追加することができます。各 MembersBlox で使用できるのは、excluded または included 属性のどちらかだけです。1 つの <bloxreport:members> タグに両方の属性を指定すると、後の属性が受け入れられて、前の属性は無視されます。

除外されたメンバーは、結果セットから永久的に削除されます。メンバーを一時的に隠すには、 129 ページのOrderBlox を使用してください。

タグの説明

<bloxreport:members> タグ

タグ属性	必須	デフォルト	説明
excluded	いいえ		除外されるメンバー。 除外されたメンバーは、結果セットからなくなります。 メンバー名の先頭が文字でない場合、あるいはメンバー名にスペースまたは特殊文字が含まれている場合は、メンバー名を大括弧で囲む必要があります。詳細については、22 ページの『メンバー ID と表示名』を参照してください。
id	いいえ		MembersBlox のこのインスタンスの固有 ID。
bloxName	いいえ		サーバー上の MembersBlox のこのインスタンスの固有 ID。名前は動的に設定できます。93 ページの『セッションの有効範囲の管理』を参照してください。
included	いいえ		包含されるメンバー。リストにないメンバーは除外されます。 メンバー名の先頭が文字でない場合、あるいはメンバー名にスペースまたは特殊文字が含まれている場合は、メンバー名を大括弧で囲む必要があります。詳細については、22 ページの『メンバー ID と表示名』を参照してください。

例

```
<bloxreport:report id = "ProfitReport">
   <bloxreport:members
      id = "members1"
      excluded = "[Unit Cost], [Unit Price]"
   />
</bloxreport:report>
```

上記の例では、Unit Cost と Unit Price の各メンバーをレポートに含めないよう指定しています。両者は名前にスペースが含まれるので、大括弧で囲んであります。

参照

129 ページの OrderBlox

OrderBlox

OrderBlox は、メンバーを戻す順序 (左から右) を指定します。また、一時的にメンバーを隠すこともできます。OrderBlox の JSP タグは <bloxreport:order> です。

構文

```
<bloxreport:report id = "idName">
   ...
   <bloxreport:order id = "orderIdName"
      excluded = "member1, member2,..."
      included = "member1, member2,..." >
   </bloxreport:order>
   ...
</bloxreport:report>
```

使用法

<bloxreport:report> タグに複数の <bloxreport:order> タグを追加することができます。各 OrderBlox で使用できるのは、excluded または included 属性のどちらかだけです。1 つの <bloxreport:order> タグに両方の属性を指定すると、後の属性が受け入れられて、前の属性は無視されます。

excluded 属性により、メンバーを一時的に隠すことができます。除外されたメンバーは、レンダリングされたレポートには表示されませんが、結果セットには引き続き存在します。対話式レポート (interactive = "true") では、対話式コンテキスト・メニューから「**すべて表示**」を選択すると、隠れたメンバーが表示されます。メンバーを今後再び表示しない場合は、 127 ページのMembersBlox を使用してください。

タグの説明

<bloxreport:order> タグ

タグ属性	必須	デフォルト	説明
excluded	いいえ		順序から除外されるメンバー。 MembersBlox の場合と異なり、除外されたメンバーは依然として結果セットの中にあるため、パイプラインの後続のデータ形式変更アクションで使用できます。 メンバー名の先頭が文字でない場合、あるいはメンバー名にスペースまたは特殊文字が含まれている場合は、メンバー名を大括弧で囲む必要があります。詳細については、22 ページの『メンバー ID と表示名』を参照してください。
id	いいえ		OrderBlox のこのインスタンスの固有 ID。
bloxName	いいえ		サーバー上の OrderBlox のこのインスタンスの固有 ID。名前は動的に設定できます。93 ページの『セッションの有効範囲の管理』を参照してください。

タグ属性	必須	デフォルト	説明
included	いいえ		包含されるメンバー (左から右の順番で指定)。 メンバー名の先頭が文字でない場合、あるいはメンバー名にスペースまたは特殊文字が含まれている場合は、メンバー名を大括弧で囲む必要があります。詳細については、22 ページの『メンバー ID と表示名』を参照してください。

例

```
<bloxreport:report id = "ProfitReport">
   <bloxreport:order
      id = "order1"
      excluded = "[Unit Cost], [Unit Price]"
   />
</bloxreport:report>
```

上記の例では、Unit Cost と Unit Price の各メンバーをレポートに表示しないよう指定しています。両者は名前にスペースが含まれるので、大括弧で囲んでありま す。対話式レポートの場合、ユーザーが列ヘッダー・コンテキスト・メニューから「**すべて表示**」を選択すれば、これらのメンバーが表示されます。

参照

127 ページの MembersBlox

PdfBlox

PdfBlox はリレーショナル・レポートを PDF に送信します。PdfBlox の JSP タグは <bloxreport:pdf> です。

構文

```
<bloxreport:pdf id = "idName"
   [attributeN = "valueN"] >
</bloxreport:pdf>
```

使用法

<bloxreport:pdf> タグは、JSP ページに 1 つだけ入れることができます。他の Blox とは異なり、id が必須です。<bloxreport:pdf> タグを <bloxreport:report> タグの外側に追加すると、生きたデータが入ったレポートを PDF に直接送信できます。また、JSP ページで独立的に使用して、ReportBlox のインスタンスを PDF に送信することもできます。

タグの説明

<bloxreport:pdf> タグ

タグ属性	必須	デフォルト	説明
id	はい		PdfBlox のこのインスタンスの固有 ID。
bloxName	いいえ		サーバー上の PdfBlox のこのインスタンスの固有 ID。名前は動的に設定できます。93 ページの『セッションの有効範囲の管理』を参照してください。
bottom	いいえ	1 in	下部余白を指定します。有効な単位には、ピクセル (px)、ポイント (pt)、インチ (in)、ミリメートル (mm)、およびセンチメートル (cm) が含まれます。指定がない場合は、ピクセル (px) が使用されます。
footerHeight	いいえ	30 px	フッターの高さを指定します。有効な単位には、ピクセル (px)、ポイント (pt)、インチ (in)、ミリメートル (mm)、およびセンチメートル (cm) が含まれます。指定がない場合は、ピクセル (px) が使用されます。
footerText	いいえ		フッターは、テキストとレイアウトを含め、XHTML を使用して定義されます。以下に例を示します。 `headerText="<span style='color:blue'> Company confidential</span>"`
footerVisible	いいえ	true	フッター (ページ番号) を表示するかどうかを指定します。デフォルトは true です。
headerHeight	いいえ	50 px	ヘッダーの高さを指定します。有効な単位には、ピクセル (px)、ポイント (pt)、インチ (in)、ミリメートル (mm)、およびセンチメートル (cm) が含まれます。指定がない場合は、ピクセル (px) が使用されます。
headerText	いいえ		ヘッダーは、テキストとレイアウトも含め、XHTML を使用して定義されます。以下に例を示します。 `headerText="<span style='color:blue'> Annual Report</span>"`

タグ属性	必須	デフォルト	説明
headerVisible	いいえ	true	ヘッダー (ロゴ、時刻、日付) を表示するかどうかを指定します。デフォルトは true です。
height	いいえ	1 1in	ページの高さを指定します。有効な単位には、ピクセル (px)、ポイント (pt)、インチ (in)、ミリメートル (mm)、およびセンチメートル (cm) が含まれます。指定がない場合は、ピクセル (px) が使用されます。
left	いいえ	1.25 in	左余白を指定します。有効な単位には、ピクセル (px)、ポイント (pt)、インチ (in)、ミリメートル (mm)、およびセンチメートル (cm) が含まれます。指定がない場合は、ピクセル (px) が使用されます。
logoSource	いいえ		レポートを PDF にレンダリングするときに使用するロゴを指定します。HTML タグ <img src="..."> を使用して JSP ファイルに追加した画像は、PDF レンダリングには組み込まれません。この属性を使用して、ソース・イメージの URL を明示的に指定しなければなりません。 URL は以下のいずれかの形式にする必要があります。 • ファイル・プロトコルと絶対パスを指定した完全な URL。たとえば、Windows システムの場合: file:///C:¥¥alphablox¥¥webapps¥¥MyApp¥¥background.gif • DB2 Alphablox インストール場所への相対 URL。たとえば、画像 background.gif が *images/* ディレクトリー内の DB2 Alphablox Repository に保管されている場合、logoSource は /repository/images/background.gif に設定する必要があります。
portrait	いいえ	true	ページを縦長にレンダリングするかどうかを指定します。False を指定すると、ページが横長にレンダリングされます。デフォルトは true で、レポートは縦長にレンダリングされます。
report	いいえ		PDF にレンダリングする ReportBlox のインスタンス (ReportBlox id) を指定します。
right	いいえ	1.25 in	右余白を指定します。有効な単位には、ピクセル (px)、ポイント (pt)、インチ (in)、ミリメートル (mm)、およびセンチメートル (cm) が含まれます。指定がない場合は、ピクセル (px) が使用されます。
top	いいえ	1 in	上部マージンを指定します。有効な単位には、ピクセル (px)、ポイント (pt)、インチ (in)、ミリメートル (mm)、およびセンチメートル (cm) が含まれます。指定がない場合は、ピクセル (px) が使用されます。
width	いいえ	8.5 in	ページの幅を指定します。有効な単位には、ピクセル (px)、ポイント (pt)、インチ (in)、ミリメートル (mm)、およびセンチメートル (cm) が含まれます。指定がない場合は、ピクセル (px) が使用されます。

例

```
<!---Content of toPDF.jsp---->
<%@ taglib uri="bloxreporttld" prefix="bloxreport" %>
<bloxreport:pdf id="myPDF"
   report='<%= request.getParameter("report") %>' />
```

上記の例は、簡潔ながらも完全な JSP ページです。以下に示す、ReportBlox が入った別の JSP ページでユーザーがボタンまたはリンクをクリックすると、上の JSP ページが呼び出されます。

```
<!---Content of myReport.jsp---->
<%@ taglib uri="bloxreporttld" prefix="bloxreport" %>
<html>
<body>
<a href="toPDF.jsp?report=myReport1">Send to PDF</a>
<bloxreport:report id="myReport1" interactive="true">
   ...
</bloxreport:report>
</body>
</html>
```

PersistenceBlox

PersistenceBlox は、ReportBlox の状態を Alphablox リポジトリーの中の指定された位置に、指定された名前で保管し、後で検索できるようにします。PersistenceBlox の JSP タグは <bloxreport:persistence> です。

構文

```
<bloxreport:persistence id = "idName"
   targetBloxId="ReportBloxId"
   location="location"
   persistedName="bookmarkName"
   operation="operation"
/>
```

使用法

1 つの JSP ページに複数の <bloxreport:persistence> タグを入れることができます。指定した ReportBlox のブックマークを保管する前に、ReportBlox の interactive 属性を false に設定してください。対話式の ReportBlox のブックマークを保管すると、リポジトリーからブックマークをロードするときにレポートが正しく表示されません。ブックマークを検索するときには、必要なデータ・ソースがアクセス可能でなければなりません。すべてのブックマークが、リポジトリーの下の reportingpersistence/ フォルダーの下に保管されます。

タグの説明

<bloxreport:persistence> タグ

タグ属性	必須	デフォルト	説明
id	いいえ		PersistenceBlox のこのインスタンスの固有 ID。
bloxName	いいえ		サーバー上の PersistenceBlox のこのインスタンスの固有 ID。名前は動的に設定できます。93 ページの『セッションの有効範囲の管理』を参照してください。
location	はい		リポジトリーの reportingpersistence/ フォルダーの下のフォルダー名。“sales/east” などのサブフォルダーを指定できます。この場合、リポジトリー内の reportingpersistence/ フォルダーの下に sales/、その下に east/ フォルダーが作成されます (これらのフォルダーがまだ存在していない場合)。
operation	はい		(persistedName 属性によって) 指定するブックマークに対して実行する操作。有効な値は save と load です。
persistedName	はい		ブックマークの名前。
targetBloxId			ブックマークを付ける ReportBlox の id。この属性は、操作が save のときには必須です。

例

```
<bloxreport:persistence id = "myBookmark"
   targetBloxId="MyReport"
   location="sales/east"
   persistedName="salesApr02"
   operation="save"
/>
```

上記の例は、「MyReport」という id の、以前に定義された ReportBlox の状態をリポジトリーに保管します。

```
<alphablox_dir>/repository/reportingpersistence/sales/east/salesApr02
```

ここで <alphablox_dir> は、Alphablox のインストール・ディレクトリーです。

RDBResultSetDataBlox

RDBResultSetDataBlox は、DataBlox が提供している RDBResultSet オブジェクトを使用し、Relational Reporting Blox を使用してリレーショナル・レポートを生成します。

構文

```
<bloxreport:rdbResultSetData id="id"
   bloxName="bloxName"
   bloxRef= "DataBloxName"
   columnCoordinate="col"
   drillThroughReportName="reportName"
   rowCoordinate="row"
/>
```

使用法

<bloxreport:report> タグに追加できる <bloxreport:rdbResultSetData> タグは 1 つだけです。その bloxRef 属性を使用して、既存の DataBlox を指定する必要があります。列および行の座標を指定しない場合には、参照される DataBlox の getResultSet() メソッドが呼び出されます。列および行の座標を指定する場合には、参照される DataBlox の drillThrough() メソッドが呼び出されます。DB2 OLAP Server または Hyperion Essbase ドリルスルーの場合には、1 つのセルに対して複数のレポートが定義されていることがあるので、drillThroughReportName を使用して希望のレポートを指定してください。

タグの説明

<bloxreport:rdbResultSetData> タグ

タグ属性	必須	デフォルト	説明
id	いいえ		RDBResultSetDataBlox のこのインスタンスの固有 ID。
bloxName	いいえ		サーバー上の RDBResultSetDataBlox のこのインスタンスの固有 ID。名前は動的に設定できます。93 ページの『セッションの有効範囲の管理』を参照してください。
bloxRef	はい		既存の DataBlox の名前。
columnCoordinate	いいえ		指定するセルの列の座標。
drillThroughReportName	いいえ		DB2 OLAP Server または Essbase ドリルスルー・レポートの名前。
rowCoordinate	いいえ		指定するセルの行の座標。

例

以下の例の JSP の中では、「myDataBlox」という DataBlox を含む別の JSP が参照されます。

```
<bloxreport:rdbResultSetData
   bloxRef="myDataBlox"
   columnCoordinate="<%= request.getParameter(¥"colIndex¥") %>"
   rowCoordinate="<%= request.getParameter(¥"rowIndex¥") %>"
/>
```

PresentBlox または GridBlox のレンダリング・モードに応じて、JavaScript 関数または Java スクリプトレットから、`colIndex` および `rowIndex` パラメーターが渡されます。詳細については、34 ページの『RDBResultSetDataBlox を使用して、DataBlox が提供する RDBResultSet にアクセスする』を参照してください。

参照

Blox Sampler のデータの検索についてのセクション (MSAS バージョン) のドリルスルーの例、および「*開発者用ガイド*」のデータの検索についての章。

ReportBlox

ReportBlox はレポートを HTML 表に生成します。ReportBlox の JSP タグは <bloxreport:report> です。

interactive 属性を true に設定すると、Report Editor ユーザー・インターフェースがオンになります。列ヘッダー、ブレーク・グループ・ヘッダー、またはブレーク・グループ・フッター上でマウスを動かすと、コンテキスト・メニューが自動的に表示され、レポートを動的に編集できるようになります。これらのコンテキスト・メニューについては、12 ページの『リレーショナル・レポートのスタイル設定』を参照してください。

構文

```
<bloxreport:report id="reportID">
   ...
</bloxreport:report>
```

使用法

1 つの JSP ファイルで複数の <bloxreport:report> タグを使用できます。それぞれのレポートには、143 ページのSQLDataBlox と 117 ページのDataSourceConnectionBlox をそれぞれ 1 つずつしか入れることができません。あるいは、136 ページのRDBResultSetDataBlox を使用することもできます。これは、DataBlox が提供している RDBResultSet オブジェクトにアクセスすることを可能にします。ReportBlox のそれぞれのインスタンスに id を指定しておくとよいでしょう。

タグの説明

<bloxreport:report> タグ

タグ属性	必須	デフォルト	説明
<bloxreport:report>			
id	いいえ		bloxName を指定しない場合は、ReportBlox のこのインスタンスの固有 ID。 bloxName を指定する場合、id は、ローカル Java スクリプト変数名。詳細については、「*開発者用リファレンス*」の bloxName の項目を参照してください。
bloxName	いいえ		サーバー上の ReportBlox のこのインスタンスの固有 ID。名前は動的に設定できます。93 ページの『セッションの有効範囲の管理』、および「*開発者用リファレンス*」の bloxName の項目を参照してください。

タグ属性	必須	デフォルト	説明
errors	いいえ	false	スローされた例外をインターセプトします。 デフォルトで、データ照会がデータを戻さないときには、「データがありません」というテキストが表示されます。存在しないメンバーのデータに対してグループ化や計算などのデータ形式変更を加えようとした結果として何もデータが生じなかった場合は、「データがありません : レポート・データの生成中にエラーが起きました」というテキストが表示されます。これらのメッセージは、noDataMessage および noDataDueToErrorMessage 属性を使用してカスタマイズできます。 このデフォルトの動作により、例外のままではなく、より親切なメッセージがユーザーに表示されることになります。ただし、問題を追跡するために、例外を調べたい場合もあることでしょう。その場合には、この属性を true に設定してください。例外の根本原因の追跡について詳しくは、110 ページの『ErrorBlox を使用したエラー処理』を参照してください。
interactive	いいえ	false	対話式のレポート編集のためにレポートを DHTML 表の形でレンダリングする場合は true。 デフォルト値は false です。つまり、ユーザーのための対話式コンテキスト・メニューはありません。 interactive を true に設定した場合は、Report Editor ユーザー・インターフェースが正しく動作するよう、対話式メニューおよび列に関連したクラスのスタイルを定義するスタイル・シートを提供する必要があります。スタイルについての詳細は、12 ページの『リレーショナル・レポートのスタイル設定』を参照してください。
noDataMessage	いいえ	データがありません	ReportBlox がデータを戻さない場合に、レンダリングされたレポートに表示されるメッセージ。errors 属性は false (デフォルト) に設定する必要あることに注意してください。そうしないと例外が表示されることになります。
noDataDueToErrorMessage	いいえ	レポート・データの生成中にエラーが起きました	パイプラインのデータ形式変更中にエラーが発生した場合に、レンダリングされたレポートに表示されるメッセージ。 そのような例外が原因で ReportBlox がデータを戻さない場合、noDataMessage および noDataDueToErrorMessage の両方が 1 つのストリングで表示されます。すなわち、「データがありません: レポート・データの生成中にエラーが起きました」のようになります。

タグ属性	必須	デフォルト	説明
noDataDueToErrorMessage	いいえ	レポート・データの生成中にエラーが起きました	パイプラインのデータ形式変更中にエラーが発生した場合に、レンダリングされたレポートに表示されるメッセージ。 そのような例外が原因で ReportBlox がデータを戻さない場合、noDataMessage および noDataDueToErrorMessage の両方が 1 つのストリングで表示されます。すなわち、「データがありません: レポート・データの生成中にエラーが起きました」のようになります。
noDataDueToErrorMessage	いいえ	レポート・データの生成中にエラーが起きました	パイプラインのデータ形式変更中にエラーが発生した場合に、レンダリングされたレポートに表示されるメッセージ。 そのような例外が原因で ReportBlox がデータを戻さない場合、noDataMessage および noDataDueToErrorMessage の両方が 1 つのストリングで表示されます。すなわち、「データがありません: レポート・データの生成中にエラーが起きました」のようになります。
readerMode	いいえ	false	対話式メニューの「**スタイル...**」メニュー・オプションをオフにする場合は true。このリーダー・モードは、スクリーン・リーダー・プログラムでの使用により適しています。スクリーン・リーダーによっては、フォーカスをダイアログに設定してテキストが読み取れない場合があるからです。また、目の不自由なユーザーにとって、「**レポート・スタイル**」ダイアログで提供されるスタイル設定のオプションは便利なものではありません。

SortBlox

SortBlox は、指定したメンバーに基づいて昇順または降順のいずれかでソートします。SortBlox のタグは <bloxreport:sort> です。SortBlox は、ネストされたタグ <bloxreport:rule> を使って各ソート規則を指定することにより、複合のソートをサポートしています。

構文

```
<bloxreport:report id="reportID">
   ...
   <bloxreport:sort member="memberName"
      ascending="boolean"
      missingLast="boolean" />
   ...
</bloxreport:report>
```

複数の規則を使った複合ソートの場合は次のとおりです。

```
<bloxreport:report id="reportID">
   ...
   <bloxreport:sort>
      <bloxreport:rule
         member="sortMemberName1"
         ascending="boolean"
         missingLast="boolean" />
      <bloxreport:rule member="sortMemberName2" />
   </bloxreport:sort>
   ...
</bloxreport:report>
```

使用法

数値データ、ストリング、日付、および時間をソートできます。SortBlox は、規則を指定した順序に従ってソートを実行します。複数の SortBlox を追加した場合、先のソート操作が後の操作に引き継がれることはありません。デフォルトでは、データは昇順でソートされ、欠落データは最後に表示されます。

タグの説明

<bloxreport:sort> タグ

タグ属性	必須	デフォルト	説明
<bloxreport:sort>			
id	いいえ		SortBlox のこのインスタンスの固有 ID。
bloxName	いいえ		サーバー上の SortBlox のこのインスタンスの固有 ID。名前は動的に設定できます。93 ページの『セッションの有効範囲の管理』を参照してください。
<bloxreport:sort>、<bloxreport:rule>			
ascending	いいえ	true	昇順でソートするかどうか。

タグ属性	必須	デフォルト	説明
member	はい		ソートの基準となるメンバーの名前。各規則に指定できるメンバー名は 1 つだけです。 メンバー名の先頭が文字でない場合、あるいはメンバー名にスペースまたは特殊文字が含まれている場合は、メンバー名を大括弧で囲む必要があります。詳細については、22 ページの『メンバー ID と表示名』を参照してください。
missingLast	いいえ	true	欠落値を最後に表示するかどうか。

例

```
<bloxreport:report id="MarketReport">
   ...
   <bloxreport:sort id="sort1">
      <bloxreport:rule
         member="Area"
         missingLast = "false"
         ascending = "false" />
      <bloxreport:rule
         member="Location"
         missingLast = "false"
         ascending = "false" />
   </bloxreport:sort>
   ...
</bloxreport:report>
```

上記の例では、Area、次いで Location に基づいてデータをソートし、以下のような結果を生成します。

Area	Location	Product	Units	Cost	Sales
S. Cal	Beverly Hills	Truffles	#Missing	#Missing	#Missing
S. Cal	Beverly Hills	Brittles	72	120	240
N. Cal	Sonoma	Truffles	#Missing	#Missing	#Missing
N. Cal	Sonoma	Brittles	27	45	90
N. Cal	Napa	Brittles	48	80	160

SQLDataBlox

SQLDataBlox は、DataSourceConnectionBlox に対して実行できる SQL 照会を表現します。

構文

```
<bloxreport:report id="profitReport">
   <bloxreport:sqlData
    id = "idName"
    query = "sqlCommand" >
       <bloxreport:dataSourceConnection >
          ...
       </bloxreport:dataSourceConnection>
   </bloxreport:sqlData>
</bloxreport:report>
```

使用法

<bloxreport:sqlData> タグは 1 つだけ、<bloxreport:report> タグに追加できます。

タグの説明

<bloxreport:sqlData> タグ

タグ属性	必須	デフォルト	説明
id	いいえ		SQLDataBlox のこのインスタンスの固有 ID。
bloxName	いいえ		サーバー上の SQLDataBlox のこのインスタンスの固有 ID。名前は動的に設定できます。93 ページの『セッションの有効範囲の管理』を参照してください。
query	はい		リレーショナル・データ・ソースからデータを抽出する SQL コマンド。 データ・ソースは <bloxreport:dataSourceConnection> タグを使用して指定しなければなりません。このタグは、<bloxreport:sqlData> 内でネストします。

例

```
<bloxreport:report id="profitReport">
   <bloxreport:sqlData
    id = "dataquery1"
    query = "select Product, UnitSold, UnitCost, Sales from
       chocoblocks" >

      <bloxreport:dataSourceConnection
         dataSourceName="chocoblocks" />
   </bloxreport:sqlData>
</bloxreport:report>
```

以下に出力例を示します。

Product	UnitSold	UnitCost	Sales
Brittles	x	x	651,345

Product	UnitSold	UnitCost	Sales
Fudge	x	x	454,450
Truffles	x	x	450,000
Seasonal	x	x	1,004,045

StyleBlox

StyleBlox は、レポートに表示されるデータやさまざまな領域のスタイルを設定します。データ・タイプに基づいてデータのスタイルを設定したり、欠落データまたは負のデータのスタイルや、レポートのさまざまな領域 (列ヘッダー、データ、グループ・ヘッダー、グループ・フッター、グループ合計) のスタイルを設定したりできます。StyleBlox のタグは <bloxreport:style> です。これには次のネストしたタグがあります。

- <bloxreport:text>: スタイルは、すべてのテキスト列のデータに適用されます。
- <bloxreport:numeric>: スタイルは、すべての数値列のデータに適用されます。
- <bloxreport:date>: スタイルは、すべての日付列のデータに適用されます。
- <bloxreport:banding>: スタイルは、交互データ行に適用されます。
- <bloxreport:missing>: スタイルは、欠落データに適用されます。
- <bloxreport:negative>: スタイルは、すべての負の値に適用されます。
- <bloxreport:column>: スタイルは、指定されたメンバーの列ヘッダーとデータの両方に適用されます。メンバーの指定が必要です。
- <bloxreport:data>: スタイルは、列が指定されていない限り、レポート内のすべてのデータに適用されます。
- <bloxreport:columnHeader>: スタイルは、列が指定されていない限り、レポート内のすべての列ヘッダーに適用されます。
- <bloxreport:groupHeader>: スタイルは、レベルが指定されているのでない限り、レポート内のすべてのグループ・ヘッダーに適用されます。
- <bloxreport:groupFooter>: スタイルは、レベルが指定されているのでない限り、レポート内のすべてのグループ・フッターに適用されます。
- <bloxreport:groupTotal>: スタイルは、レベルが指定されているのでない限り、レポート内のすべてのグループ合計に適用されます。

構文

```
<bloxreport:style>
   <bloxreport:text style="yourStyle" />
   <bloxreport:numeric style="yourStyle" />
   <bloxreport:date style="yourStyle" />
   <bloxreport:banding style="yourStyle" />
   <bloxreport:missing style="yourStyle" />
   <bloxreport:negative style="yourStyle" />
   <bloxreport:column style="yourStyle" columnName="member" />
   <bloxreport:data style="yourStyle" columnName="member" />
   <bloxreport:columnHeader style="yourStyle" columnName="member" /
   <bloxreport:groupHeader style="yourStyle" level="level" />
   <bloxreport:groupFooter style="yourStyle" level="level" />
   <bloxreport:groupTotal style="yourStyle" level="level" />
</bloxreport:style>
```

使用法

StyleBlox は 1 つだけ、<bloxreport:report> タグに追加できます。複数の <bloxreport:style> タグがある場合には、最後のものだけが適用されます。同じ <bloxreport:style> タグ内の同じ要素に対して異なるスタイルが指定されている場合には、最後に宣言されたスタイルが適用されます。

StyleBlox を使用したスタイル設定が、スタイル・クラスによるスタイル設定より優先されます。例えば .data スタイル・クラスのフォント・カラーをスタイル・シートで青と指定し、さらに StyleBlox を使用してテキスト列のすべてのデータを黒と設定した場合、データは黒で表示されます。

タグの説明

<bloxreport:style> タグおよびそのサブタグ

タグ属性	必須	デフォルト	説明
<bloxreport:style>			
id	いいえ		StyleBlox のこのインスタンスの固有 ID。
bloxName	いいえ		サーバー上の StyleBlox のこのインスタンスの固有 ID。名前は動的に設定できます。93 ページの『セッションの有効範囲の管理』を参照してください。
ネストされた text、negative、banding タグ			
style	はい		適用されるスタイル。CSS 属性を使用して、属性の形式でスタイルを指定します。すなわち、値の組を「;」で区切り、引用符で囲みます。
ネストされた column、numeric、date、missing タグ			
style	はい		適用されるスタイル。CSS 属性を使用して、属性の形式でスタイルを指定します。すなわち、値の組を「;」で区切り、引用符で囲みます。
columnName	column タグの場合は、はい。date、missing、および numeric の場合は、いいえ。		このスタイルが適用される列の名前。 <bloxreport:column> タグの場合、columnName を指定しなければなりません。 指定したスタイルは、指定した列のデータと列ヘッダーの両方に適用されます。他のタグの場合には、columnName を指定しなければ、指定したスタイルがすべての列に適用されます。
ネストされた columnHeader、data、groupHeader、groupFooter、groupTotal タグ			
style	はい		適用されるスタイル。CSS 属性を使用して、属性の形式でスタイルを指定します。すなわち、値の組を「;」で区切り、引用符で囲みます。
columnName	いいえ		data および columnHeader タグに適用されます。列名を指定しないと、スタイルはすべてのデータとすべての列ヘッダーに適用されます。
level	はい		groupHeader、groupFooter、および groupTotal タグにのみ適用されます。

ヒント: StyleBlox の内側の 5 つのサブタグは、TextBlox 内でも、レポートのそれぞれの領域の表示テキストを設定するために使用できます。TextBlox 内でネストしている場合には、text 属性を使用して表示テキストを設定しなければなりません。StyleBlox 内でネストしている場合には、style 属性を使用して表示スタイルを設定しなければなりません。TextBlox 内で style 属性を使用しても、StyleBlox 内で text 属性を使用しても、それらは無視されます。

例

```
<bloxreport:style>
   <bloxreport:text style="text-align: right;" />
   <bloxreport:numeric style="text-align: right;" />
   <bloxreport:date style="text-align: left;" />
   <bloxreport:banding style="background-color: #CCCCCC;" />
   <bloxreport:missing style="background-color: aqua;" />
   <bloxreport:negative style="background-color: red;" />
   <bloxreport:column style="font-size: 85%; color: white;
      background-color: gray;" member="Type" />
   <bloxreport:column style="color: purple; " member="Country"/>
   <bloxreport:column style="color: blue;" member="State"/>
</bloxreport:style>
```

上記の例は以下の設定を行います。

- すべてのテキスト列は、右寄せ
- すべて数値列は、右寄せ
- すべての日付列は、左寄せ
- 交互データ行の背景色は #CCCCCC (ベール・グレー)
- 欠落データのセルの背景色は水色
- 負のデータのセルの背景色は赤
- Type 列の背景色はグレーで、テキストは白、通常の文字サイズの 85%
- Country 列のテキスト・カラー (列ヘッダーとデータの両方) は紫
- State メンバーのテキスト・カラー (列ヘッダーとデータの両方) は赤

TextBlox

TextBlox は、レンダリングされたレポートの 5 つの領域 (グループ・ヘッダー、グループ・フッター、グループ合計、列ヘッダー、およびデータ) の表示テキストを指定します。これには次のネストしたタグがあります。

- <bloxreport:groupHeader>: テキストは、すべてのグループ・ヘッダー、または指定されたレベルのグループ・ヘッダーに適用されます。
- <bloxreport:groupFooter>: テキストは、すべてのグループ・フッター、または指定されたレベルのグループ・フッターに適用されます。
- <bloxreport:groupTotal>: テキストは、すべてのグループ合計、または指定されたレベルのグループ合計に適用されます。
- <bloxreport:columnHeader>: テキストは、すべての列ヘッダー、または指定された列の列ヘッダーに適用されます。
- <bloxreport:data>: テキストは、すべてのデータ・セル、または指定された列のデータ・セルに適用されます。

それぞれのブレーク・グループ内では、列またはブレーク・グループの名前、および列の集約値を抽出するための 2 つの置換変数を使用できます。

- <member/>: これは、<bloxreport:groupHeader> タグ (ブレーク・グループ・ヘッダーを指定する)、<bloxreport:groupFooter> タグ (ブレーク・グループ・フッターを指定する)、および <bloxreport:columnHeader> タグ (列ヘッダーを指定する) 内で有効です。
- <value/>: これは、ネストされた groupHeader および groupFooter タグ (指定されたメンバーのグループ合計を抽出する)、ネストされた groupTotal タグ (ブレーク・グループ合計を指定する)、ネストされた data タグ (データ値を指定する) 内で有効です。

構文

```
<bloxreport:text>
<bloxreport:groupHeader
  level="level"
  text="text for group header" />
<bloxreport:groupFooter
  level="level"
  text="text for group footer" />
  <bloxreport:columnHeader
  columnName="columnName"
  text="text for the column header" />
<bloxreport:data
  columnName="columnName"
  text="text for data in the column" />
<bloxreport:groupTotal level="level"
  text="text for group totals" />
<bloxreport:text>
```

使用法

TextBlox は 1 つだけ、<bloxreport:report> タグの内部に追加できます。複数の <bloxreport:text> タグがある場合には、最後のものだけが適用されます。同じ <bloxreport:text> タグ内の同じ要素に対して異なるテキストが指定されている場合には、最後に宣言されたテキストが適用されます。

タグの説明

<bloxreport:text> 内部のネストされたタグ

以下の表は、TextBlox のサブタグおよびその属性を説明します。

属性	必須	デフォルト	説明
ネストされた columnHeader および data タグ			
columnName	columnHeader の場合は、はい、data の場合は、いいえ		この列ヘッダー・テキストまたはデータ・テキストが適用される列の名前。
text	はい	columnHeader の場合は <member/>、data の場合は <value/>	指定された列、またはすべての列 (columnName が指定されない場合) に表示されるテキスト。text を指定しないと、columnHeader および data タグ全体が無視されます。 この属性の値全体がブラウザーに送信されるので、カスタム HTML コードを列ヘッダーおよびデータ値に追加できます。DB2 Alphablox Relational Reporting エンジンが処理するのは、2 つの置換変数 (<member/> および <value/>) だけです。62 ページの『メンバー名および値を表示するための特殊な置換変数』を参照してください。
ネストされた groupFooter、groupHeader、および groupTotal タグ			
columnName	いいえ		groupTotal タグにのみ適用されます。 この列ヘッダー・テキストが適用される列の名前。
level	いいえ		このグループ・フッター、ヘッダー、合計テキストまたはスタイルが適用されるレベル。レベル 1 はレポート全体のレベルです (グループ化は GroupBlox によって追加されます)。 グループ化を追加するたびに、レベルが 1 つ追加されます。例えば、レポートを商品別、次いで国別にグループ化した場合、レベル 1 は全商品のレポート全体を表現し、レベル 2 は各商品のブレーク・グループ、レベル 3 は各国のブレーク・グループを表現します。 level を指定しないと、指定したフッター、ヘッダー、または合計テキストおよびスタイルは、すべてのグループ・レベルに適用されます。

属性	必須	デフォルト	説明
text	はい	groupHeader および groupFooter の場合は <member/>、groupTotal の場合は <value/>	グループ・フッター、ヘッダー、または合計に表示されるテキスト。level を指定しないと、指定したテキストは、すべてのグループ・レベルに適用されます。 この属性の値全体がブラウザーに送信されるので、カスタム HTML コードを列ヘッダーおよびデータ値に追加できます。DB2 Alphablox Relational Reporting エンジンが処理するのは、2 つの置換変数 (<member/> および <value/>) だけです。62 ページの『メンバー名および値を表示するための特殊な置換変数』を参照してください。

ヒント: TextBlox の内側の 5 つのサブタグは、StyleBlox 内でも、レポートのそれぞれの領域のスタイルを設定するために使用できます。TextBlox 内でネストしている場合には、text 属性を使用して表示テキストを設定しなければなりません。StyleBlox 内でネストしている場合には、style 属性を使用して表示スタイルを設定しなければなりません。TextBlox 内で style 属性を使用しても、StyleBlox 内で text 属性を使用しても、それらは無視されます。

<member/> および <value/> 置換変数

DB2 Alphablox Relational Reporting エンジンは、<member/> 変数を検出すると、その変数を現行メンバーの名前、または指定されたより高いレベルのグループ・メンバーの名前に置換します。この変数により、レポートのグループ・ヘッダー、グループ・フッター、および列ヘッダーの各領域内のメンバー名を抽出し、そのメンバー名の周りを HTML コードでラップできます。groupHeader および groupFooter タグの内部で使用すると、これはブレーク・グループ・メンバー名によって置換されます。columnHeader タグの内部で使用すると、これは列名で置換されます。

<member/> 変数には、次の 1 つの属性があります。

タグ属性	必須	デフォルト	説明
level	いいえ	groupHeader、groupFooter、または groupTotal タグで指定された、現行グループ・レベル。	現行のグループ化レベルまたは上位レベルのメンバーのみを参照できます。例えば、レベル 2 グループ・ヘッダーはレベル 2 とレベル 1 グループ・メンバーのみを参照でき、レベル 3 は参照できません。レベルを 3 に設定すると、タグ・ストリング全体はテキストとして扱われ、変数置換は行われません。レベルを指定しないと、現行レベルが暗黙のうちに適用されます。

DB2 Alphablox Relational Reporting エンジンが <value/> 変数を検出すると、その変数を現行列または指定された列の値に置換します。この変数が有効なのは、TextBlox の data、groupTotal、groupFooter および groupHeader タグの内部です。その値の周りを HTML コードでラップすることができます。groupTotal タグ

内部で使用されると、それはブレーク・グループの現行列の集約値、またはブレーク・グループの指定された列の集約値で置換されます。groupHeader タグの内部で使用すると、これは同じグループ・レベルの指定された列の集約値によって置換されます。

<value> 変数には、次の 1 つの属性があります。

タグ属性	必須	デフォルト	説明
member	groupTotal および data の場合は、いいえ。groupHeader の場合は、はい。	列の現行メンバー	メンバー名が認識されないまたは存在しないと、タグ・ストリング全体がテキストとして扱われ、変数置換は行われません。

詳細および例については、62 ページの『メンバー名および値を表示するための特殊な置換変数』を参照してください。

例

```
<bloxreport:report id="SummaryReport">
   ...
  <bloxreport:text>
    <bloxreport:groupHeader level="1"
      text="Profitability by <member/>" />
    <bloxreport:groupHeader level="2"
      text="<member level=¥"1¥"/>: <member/>" />
    <bloxreport:columnHeader
      columnName="Units"
      text="Units Sold" />
    <bloxreport:groupTotal level="1"
      columnName="Cost"
      text="Grand Total Cost: <value/>" />
    <bloxreport:groupTotal level="1"
      columnName="Sales"
      text="Grand Total Sales: <value/>" />
    <bloxreport:groupTotal level="1"
      columnName="Units"
      text="Overall Total Units: <value/>" />

        <bloxreport:groupTotal level="2"
      columnName="Cost"
      text="total: <value/>" />
    <bloxreport:groupTotal level="2"
      columnName="Sales"
      text="total: <value/>" />
    <bloxreport:groupTotal level="2"
      columnName="Units"
      text="total: <value/>" />
  </bloxreport:text>
</bloxreport:report>
```

付録. コピー・アンド・ペースト用の Relational Reporting タグ

この節には、ReportBlox に関連するすべてのタグのリストを 2 つ示します。最初のリストでは、<bloxreport:report> タグ内にネストされるすべてのタグを示します。2 番目のリストでは、PdfBlox のタグを示します。Blox を表現するタグ名は太字になっており、Blox のそのインスタンスに id を割り当てられることを示しています。

- 153 ページの『<bloxreport:report> 内にネストされるすべてのタグ』
- 155 ページの『PdfBlox のすべてのタグ属性』

<bloxreport:report> 内にネストされるすべてのタグ

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

   <bloxreport:sqlData
      query = "sqlCommand" >
      <bloxreport:dataSourceConnection
         id="uniqueID"
         dataSourceName = "dataSourceName" >
      </bloxreport:dataSourceConnection>
   </bloxreport:sqlData>

   <bloxreport:calculate
      expression="calculatedMemberName= calculationExpression"
      index = "int">
   </bloxreport:calculate>

   <bloxreport:sort>
      <bloxreport:rule
         member="sortMember1"
         ascending="boolean"
         missingLast="true|false" />
   </bloxreport:sort>

   <bloxreport:filter
      expression = "filterExpression" >
   </bloxreport:filter>

   <bloxreport:members
      excluded = "member1, member2,..."
      included = "member1, member2,..." >
   </bloxreport:members>

   <bloxreport:order
      excluded = "member1, member2,..."
      included = "member1, member2,..." >
   </bloxreport:order>

   <bloxreport:group members="member1, member2, ...">
      <bloxreport:aggregation
         member="memberName"
         type="aggregationType">
```

```
      </bloxreport:aggregation>
   </bloxreport:group>

   <bloxreport:format>
      <bloxreport:numeric format="numericFormat1" />
      <bloxreport:numeric format="numericFormat2"
         member="member"/>
      <bloxreport:date format="dateFormat1" />
      <bloxreport:date format="dateFormat2" member="member"/>
      <bloxreport:missing format="missingDataFormat" />
      <bloxreport:html member = "member"/>
      <bloxreport:aggregation format="numericFormat1"
         member = "member1" />
      <bloxreport:aggregation format="numericFormat2"
         member = "member2" />
   </bloxreport:format>

   <bloxreport:style>
      <bloxreport:text style="textStyle" />
      <bloxreport:numeric style="numericStyle" />
      <bloxreport:date style="dateStyle" />
      <bloxreport:banding style="dateStyle" />
      <bloxreport:missing style="missingDataStyle" />
      <bloxreport:negative style="negativeStyle" />
      <bloxreport:column style="colStyle1" columnName="member1" />
      <bloxreport:column style="colStyle2" columnName="member2" />
      <bloxreport:data style="yourStyle" />
      <bloxreport:data style="yourStyle" columnName="member" />
      <bloxreport:columnHeader style="yourStyle"/>
      <bloxreport:columnHeader style="yourStyle"
columnName="member" />
      <bloxreport:columnHeader style="yourStyle"
columnName="member" level="n" />
      <bloxreport:groupHeader style="yourStyle" />
      <bloxreport:groupHeader style="yourStyle" level="n" />
      <bloxreport:groupFooter style="yourStyle" />
      <bloxreport:groupFooter style="yourStyle" level="n" />
      <bloxreport:groupTotal style="yourStyle" />
      <bloxreport:groupTotal style="yourStyle" level="n" />
   </bloxreport:style>

   <bloxreport:text>
      <bloxreport:data text="yourText" />
      <bloxreport:data text="yourText"" columnName="column" />
      <bloxreport:columnHeader text="yourText" />
      <bloxreport:columnHeader text="yourText" columnName="column"
level="n" />
      <bloxreport:columnHeader text="yourText" columnName="n" />
      <bloxreport:groupHeader text="yourText" />
      <bloxreport:groupHeader text="yourText" level="n" />
      <bloxreport:groupFooter text="yourText" />
      <bloxreport:groupFooter text="yourText" level="n" />
      <bloxreport:groupTotal text="yourText" />
      <bloxreport:groupTotal text="yourText" level="n" />
   </bloxreport:text>

</bloxreport:report>
```

ReportBlox では RDBResultSetDataBlox も使用できます。これを使用すると、DataBlox が提供している RDBResultSet オブジェクトに基づいてリレーショナル・レポートを作成できます。

```
<bloxreport:report id="reportId">
   <bloxreport:rdbResultSetData
      bloxRef= "DataBloxName"
      columnCoordinate="col"
```

```
      drillThroughReportName="reportName"
      rowCoordinate="row"
   </bloxreport>
   ...
</bloxreport:report>
```

PdfBlox のすべてのタグ属性

```
<bloxreport:pdf id = "idName"
   bloxName="bloxName"
   logoSource = "urlToImage"
   portrait = "true"
   bottom = "1in"
   top = "1in"
   height = "11in"
   width = "8.5in"
   left = "1.25in"
   right = "1.25in"
   footerText="text"
   footerHeight="30px"
   footerVisible = "true"
   headerText="text"
   headerHeight="50px"
   headerVisible = "true" >
   report="ReportBlox_id"
</bloxreport:pdf>
```

重要: 示されている属性値はデフォルトです (イタリックのものを除く)。

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。

AIX	IBM	DB2
DB2 OLAP Server	DB2 Universal Database	WebSphere

Alphablox および Blox は、Alphablox Corporation の米国およびその他の国における登録商標です。

Microsoft は、Microsoft Corporation の米国およびその他の国における商標です。

Java およびすべての Java 関連の商標およびロゴは、Sun Microsystems, Inc. の米国およびその他の国における商標または登録商標です。

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]

B

C

D

E

F

G

H

M

O

P

R

S

T

プログラム番号: 5724-L14

Printed in Japan

SD88-6493-02

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