OLAP for WPF/Silverlight

2018.02.20 更新

グレープシティ株式会社

表紙情報

ComponentOne, a division of GrapeCity
201 South Highland Avenue, Third Floor
Pittsburgh, PA 15206 USA

Web サイト: http://www.componentone.com
窓口: sales@componentone.com
電話番号: 1.800.858.2739 または 1.412.681.4343（米国事務所、ペンシルベニア州ピッツバーグ）

商標

保証

複製と配布

目次

OLAP for WPF/Silverlight の概要

OLAP for WPF/Silverlightにより、詳細なビジネスインテリジェンス（BI）機能を提供します。表データおよびキューブデータをスライスおよびダイスカットして、リアルタイム情報、見通し、および結果を示すピボットグリッドとチャートを数秒で作成できます。コントロールは使いやすく、また、Microsoft Excel® のピボットテーブルをモデルとして作成されているため、すべてのユーザーにとって強力でなじみ深いものです。

メモ:

- ライセンスなしというメッセージが出ない状態で実行するためには、**OLAP for WPF/Silverlight** のコントロール（**C1OlapPage**、**C1OlapPanel**、**C1OlapGrid** および **C1OlapChart**）に、別売の **OLAP for** WPF/Silverlight ライセンスが必要です。
- 説明内に含まれるクラスおよびメンバーに対するリファレンスへのリンクは、原則として**WPF**版のリファレンスページを参照します。**Silverlight**版については、目次から同名のメンバーを参照してください。

C1Olap とは

OLAP for WPF/Silverlight は、Microsoft Excel のピボットテーブルおよびピボットチャートと同じような分析処理機能を提供する Silverlight コントロールのスイート製品です。

例:同期処理の場合は、負荷が大きな更新を実行すると、更新が完了するまで、アプリケーション全体がユーザーアクションに対する応答を停止します。非同期処理をサポートする C1Olap の場合は、負荷が大きな更新(**C1OlapPage** の行または列ボックスに複数のフィールドを追加するなど)が進行中であっても、アプリケーションはすべてのユーザーアクションに応答します。

OLAP for WPF/Silverlight は、生データを任意の形式で取得し、また、使いやすいインタフェースを備えているので、ユーザーはさまざまな方法でデータを表示するサマリーをすばやく直観的に作成できます。そのため、対話的に傾向を明らかにして貴重な見通しを得ることができます。ユーザーがデータの表示方法を変更すると、**OLAP for WPF/Silverlight** は、直ちにピボットグリッドおよびピボットチャート(その後レポート)を提供します。これは、保存、エクスポート、または印刷することができます。

はじめに

OLAP とは、"オンライン分析処理（online analytical processing）" を意味します。OLAP は、データの動的な視覚化および分析を可能にする技術です。

一般的な OLAP ツールには、"OLAP キューブ" や、Microsoft Excel などに用意されているピボットテーブルがあります。これらのツールは、大規模なデータのセットを取得し、一連の基準に基づいてレコードをグループ化して、データを集計します。たとえば、OLAP キューブは、売上データを集計し、製品、地域、および期間別にグループ化します。この場合、各グリッドセルには、特定の製品、特定の地域、および特定の期間の売上の合計が表示されます。通常、このセルは、元のデータソースのいくつかのレコードからのデータを表します。

ユーザーは、OLAP ツールを使用して、このようなグループ化の基準を動的に（オンラインで）再定義することで、データに関してアドホックな分析を実行することや、隠れたパターンを発見することが容易になります。

たとえば、次の表を見てください。

日付	製品	地域	売上
Oct 2007	製品 A	North	12
Oct 2007	製品 B	North	15
Oct 2007	製品 C	South	4
Oct 2007	製品 A	South	3
Nov 2007	製品 A	South	6
Nov 2007	製品 C	North	8
Nov 2007	製品 A	North	10
Nov 2007	製品 B	North	3

ここで、このデータを分析して次のような質問の回答を得るように求められたと仮定します。

- 売上は上がっているか、下がっているか。
- 会社にとってどの製品が最も重要であるか。
- 各地域で最も人気があるのはどの製品か。

これらの単純な質問に答えるには、これらのデータを集計し、次のような表を取得する必要があります。

日付別および製品別の売上

日付	製品 A	製品 B	製品 C	合計
Oct 2007	15	15	4	34
Nov 2007	16	3	8	27
合計	**31**	**18**	**12**	**61**

製品別および地域別の売上

製品	North	South	合計
製品 A	22	9	31
製品 B	18		18
製品 C	8	4	12

合計	**48**	**13**	**61**

サマリーテーブルの各セルは、元のデータソースのいくつかのレコードを表します。1つまたは複数の値フィールドが集計され（ここでは、売上の合計）、他のフィールド（日付、製品、または地域）の値に基づいて分類されます。

これは、スプレッドシートでは簡単に行うことができますが、作業は単調で、繰り返しが多く、ミスが多くなります。データを集計するためのカスタムアプリケーションを記述しても、新しいビューを追加するようにアプリケーションを管理するためには多くの時間が必要であり、ユーザーの分析は実装されているビューに限定されます。

OLAP ツールでは、ユーザーは、アドホック形式で、対話的に必要なビューを定義できます。定義済みのビューを使用することも、新しいビューを作成して保存することもできます。基底のデータの変更はビューに自動的に反映されます。ユーザーはこれらのビューを表示するレポートを作成して共有できます。つまり OLAP とは、柔軟に、また効率的にデータを分析できるツールです。

主な特長

以下に、**OLAP for WPF/Silverlight** の便利な機能をいくつか示します。

- **OLAP for WPF/Silverlight は OLAP アプリケーション構築のための究極の柔軟性を備えている**
  1つの **C1OlapPage** コントロールをフォームにドロップし、データソースを設定するだけで、グリッドまたはチャートにデータを表示できます。とても簡単です。しかし、複数のチャートまたはグリッドを表示する場合はどうでしょうか。問題ありません。**OLAP for WPF および Silverlight** には、**C1OlapPanel**、**C1OlapChart**、および **C1OlapGrid** コントロールもあり、これらによって必要な柔軟性が得られます。各コントロールの概要については、「C1Olap アーキテクチャ」トピックを参照してください。
- **5つのチャートタイプと 22 のパレットオプションの中から選択してチャートを引き立てる**
  C1OlapChart は、棒グラフ、横棒グラフ、面グラフ、折れ線グラフ、散布図など、情報を表示するための最も一般的なチャートタイプを提供します。22 のパレットオプションの中から選択して、チャートおよび凡例項目の色を定義できます。すべてのチャートタイプおよびパレットについては、「[チャート]メニューの使用」を参照してください。
- **データを印刷、プレビュー、または PDF にエクスポートする**
  データ、グリッド、またはチャートを含むレポートを作成およびプレビューしてから、印刷したり、PDF にエクスポートすることができます。詳細については、「レポートの作成」および「OLAP for WPF/Silverlight のタスクベースのヘルプ」を参照してください。
- **グリッドまたはチャートビューからフィールドまたはフィールド内のデータを削除する**
  フィールドを簡単にフィルタ処理して、グリッドまたはチャートビューに表示されないようにすることができます。これには、フィールドを **C1OlapPanel** の **[フィルタ]** 領域にドラッグするだけです。詳細については、「データビューからフィールドを削除」を参照してください。フィールド内のデータをフィルタ処理する場合、たとえば、姓が "Sim" で始まるすべての従業員を検索する場合は、**[フィールドの設定]** ダイアログボックスを使用できます。詳細な手順については、「フィールド内のデータをフィルタ処理」を参照してください。
- **グリッドまたはチャートビューで情報を表示する**
  OLAP for WPF/Silverlight は、データを表示するための **C1OlapGrid** および **C1OlapChart** コントロールを備えています。これらのコントロールは、**C1OlapPage** コントロールに組み込まれていますが、個別のコントロールとして使用して OLAP アプリケーションをカスタマイズすることもできます。各コントロールの概要については、「C1Olap アーキテクチャ」トピックを参照してください。
- **実行時に情報の表示方法を決定する**
  C1OlapPanel を使用して、データを表示するためにデータソースのどのフィールドをどのように使用するかを決定します。**C1OlapPanel** の下の領域にフィールドをドラッグして、フィルタ、列ヘッダー、行ヘッダーを作成するか、列または行の値の合計を取得します。詳細については、「C1OlapPanel」トピックを参照してください。
- **OLAP for WPF はキューブサポートを提供する** Olap（C1Olap）では、Microsoft® SQL Server® Analysis Services（SSAS）から OLAP データソースに接続できます。数行のコードを記述するだけで、データベースに対して、OLAP を使用した完全なフロントエンドまたはダッシュボードを構築できます。**C1Olap** を使用して、OLAP キューブに提供される次元、メジャー、および KPI（主要業績評価指標）をスライシングおよびダイシングする多次元ピボットテーブルを構築できます。キューブサポートの詳細については、「OLAP キューブ」を参照してください。
- **非同期処理**
  複数の処理が同時的および独立的に実行できます。

  > 注意:非同期処理はWPFの場合のみ使用可能になります。

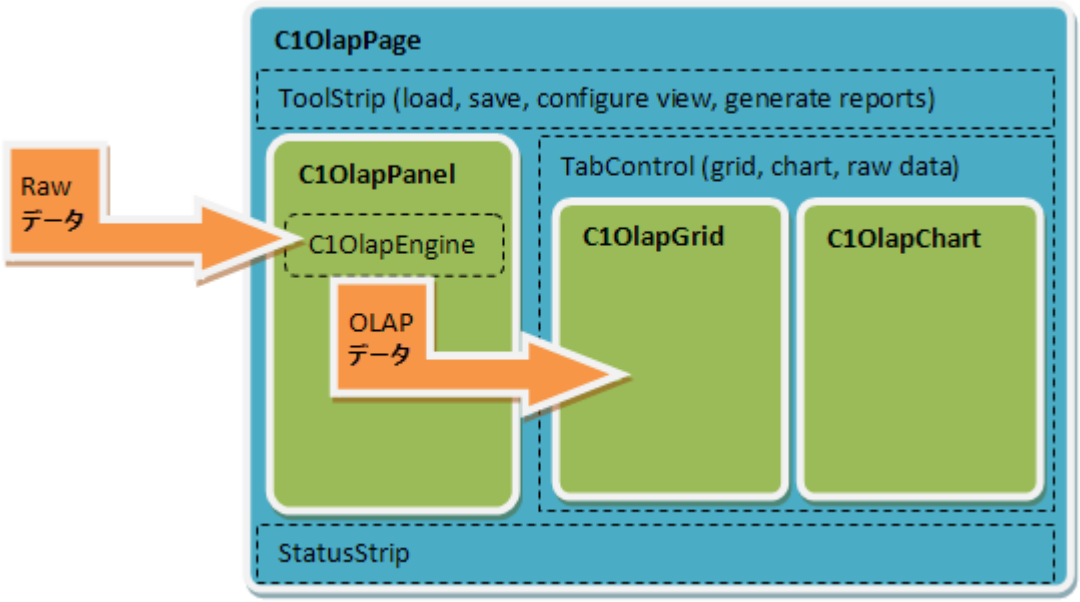
C1Olapのアーキテクチャ

OLAP には、以下のコントロールが含まれます。

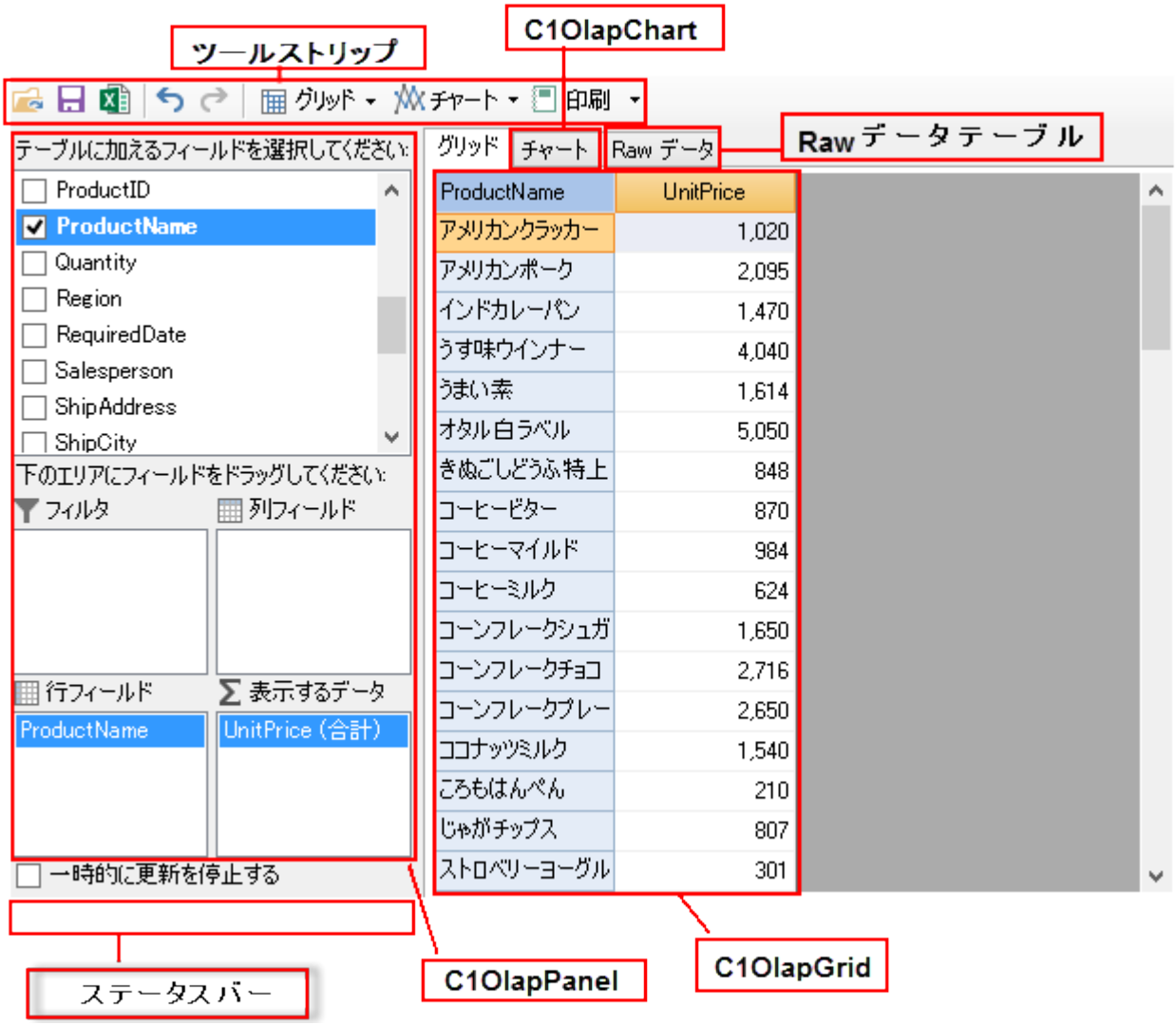
C1OlapPage

C1OlapPage コントロールは、OLAP アプリケーションを迅速かつ簡単に開発する最も簡単な方法です。これにより、**C1OlapGrid** 内の他のコントロールを使用して完全な OLAP ユーザーインタフェースを構築できます。**C1OlapPage** オブジェクトモデルは内部コントロールを全面的に公開しているため、インタフェース要素を追加または削除することによって簡単にカスタマイズできます。ソースコードが含まれているため、さらに広範にカスタマイズを行う場合にはそれを独自の実装の基盤として使用できます。

次の図は、**C1OlapPage** を構成する方法を示しています。



Visual Studio では、コントロールは次のようになります。



C1OlapPanel

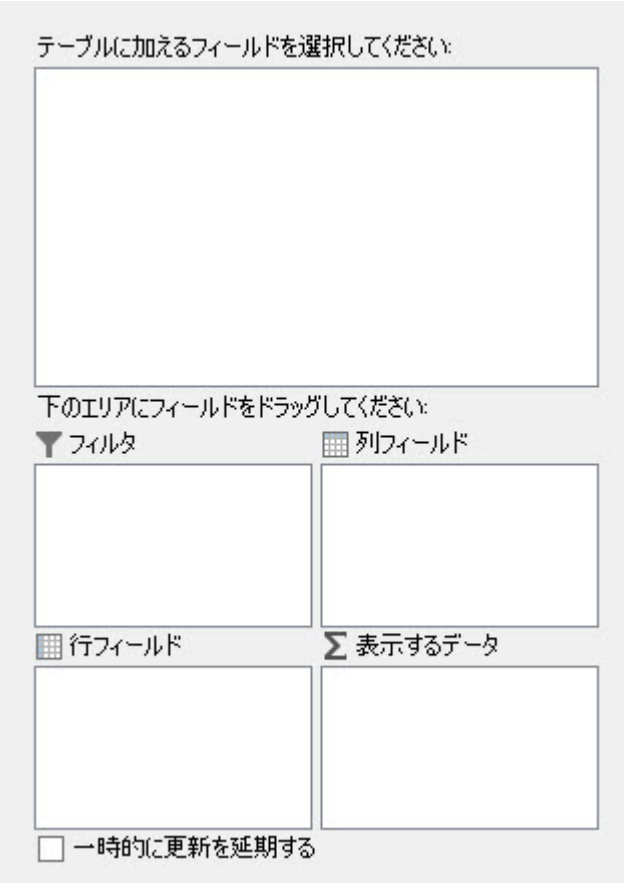
C1OlapPanel コントロールは、**C1OlapGrid** 製品の中核となるものです。入力として生データを受け取る **DataSource** プロパティと、ユーザーが提供する基準に従ってデータを要約するカスタムビューを提供する **PivotTable** プロパティがあります。**PivotTable** は、すべての通常のコントロールに対してデータソースとして使用できる通常の **DataTable** オブジェクトです。

また、**C1OlapPanel** は、一般的な Excel 形式のドラッグアンドドロップインタフェースを備えています。このインタフェースを使用して、データのカスタムビューを定義できます。コントロールには、データソース内のすべてのフィールドを含むリストが表示されます。ユーザーはこれらのフィールドを、出力テーブルの行および列のディメンション、出力データセル内で要約された値、データのフィルタ処理に使用されるフィールドを表すリストにドラッグアンドドロップできます。

C1OlapPanel コントロールの中心には、ユーザーが選択した条件に従って生データを要約する **C1OlapEngine** オブジェクトがあります。これらの条件は、ソースデータ内の指定された列への接続、フィルタ条件、書式設定、および要約オプションを含む **C1OlapField** オブジェクトによって表されます。ユーザーは、ソースの **Fields** リストから **RowFields** リスト、**ColumnFields** リスト、**ValueFields** リスト、および **FilterFields** リストの4つの補助リストに **C1OlapField** オブジェクトをドラッグすることにより、カスタムビューを作成します。フィールドは、コンテキストメニューを使用してカスタマイズできます。

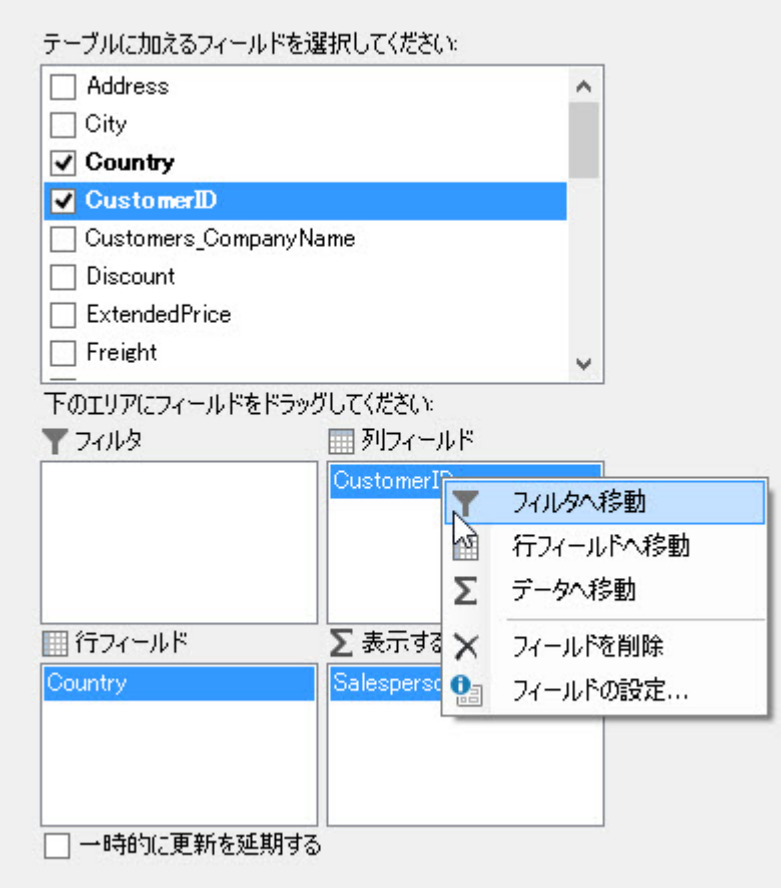
C1OlapGrid アーキテクチャはオープンであることに注意してください。**C1OlapPanel** は、データテーブル、汎用リスト、LINQ 列挙など、すべての通常のコレクションを **DataSource** として扱います。次に、データを要約して、通常の **DataTable** を出力として生成します。**C1OlapGrid** には、OLAP データである **C1OlapGrid** および **C1OlapChart** を表示するために最適化された2つのカスタムコントロールが含まれますが、それ以外のコントロールを使用することもできます。

C1OlapPanel は次のようになります。



C1OlapPanel 領域	説明
フィルタ	フィルタ処理するフィールドを指定します。
行フィールド	指定されたフィールド内の項目は、グリッドの行ヘッダーになります。これらの項目は、チャートの Y 軸に入ります。
列フィールド	指定されたフィールド内の項目は、グリッドの列ヘッダーになります。これらの項目は、チャートで凡例の入力に使用されます。
値	指定されたフィールドの合計を示します。
更新を遅らせる	このチェックボックスをオンにすると、ビュー定義の修正中に発生する自動更新が一時停止されます。

実行時に**[フィルタ]**、**[列フィールド]**、**[行フィールド]**、**[値]**領域内のフィールドを右クリックすると、コンテキストメニューが表示され、フィールドを別の領域に移動できます。フィールドを削除することもできます。また、**[フィールド設定]**をクリックして、フィールドの書式設定やフィルタの適用を行うこともできます。詳細については、「フィールド内のデータをフィルタ処理」を参照してください。



C1OlapGrid

C1OlapGrid コントロールを使用して、OLAP テーブルを表示します。これにより、**C1FlexGrid** コントロールが拡張され、**C1OlapPanel** オブジェクトに対する自動データ連結、行ヘッダーと列ヘッダーのグループ化、さらに列のサイズ変更に関するカスタム動作、クリップボードへのデータのコピー、および指定されたセルに関する詳細の表示が可能になります。

C1OlapGrid コントロールは、**C1FlexGrid** コントロール、汎用グリッドコントロールを拡張します。これは、**C1OlapGrid** ユーザーは **C1FlexGrid** オブジェクトモデル全体も使用できることを意味しています。たとえば、グリッドコンテンツを Excel にエクスポートすることや、スタイルとオーナー描画セルを使用してグリッドの外観をカスタマイズすることができます。

C1OlapGrid に入力するには、データソースに連結される **C1OlapPanel** に連結します。これを実行する方法については、「**C1OlapPanel** 」への **C1OlapGrid** の連結」を参照してください。

C1FlexGrid コントロールの詳細については、FlexGrid for WPF/Silverlight のマニュアルを参照してください。

C1OlapChart

C1OlapChart コントロールを使用して、OLAP チャートを表示します。これにより、**C1Chart** コントロールが拡張され、C1OlapPanel オブジェクトへの自動データ連結、自動ツールチップ、チャートタイプ、およびパレットの選択が実現されます。

C1OlapChart コントロールは、**C1Chart** コントロール、汎用チャート化コントロールを拡張します。これは、**C1OlapGrid** ユーザーは **C1Chart** オブジェクトモデル全体も使用できることを意味しています。たとえば、PNG や JPG など、別のファイル形式でチャートをエクスポートできます。また、チャートのスタイルと対話的操作をカスタマイズできます。

C1OlapChart に入力するには、データソースに連結される C1OlapPanel に連結します。これを実行する方法については、「C1OlapPanel への C1OlapChart の連結」を参照してください。

C1Chart コントロールの詳細については、Chart for WPF/Silverlight のマニュアルを参照してください。

C1OlapPrintDocument

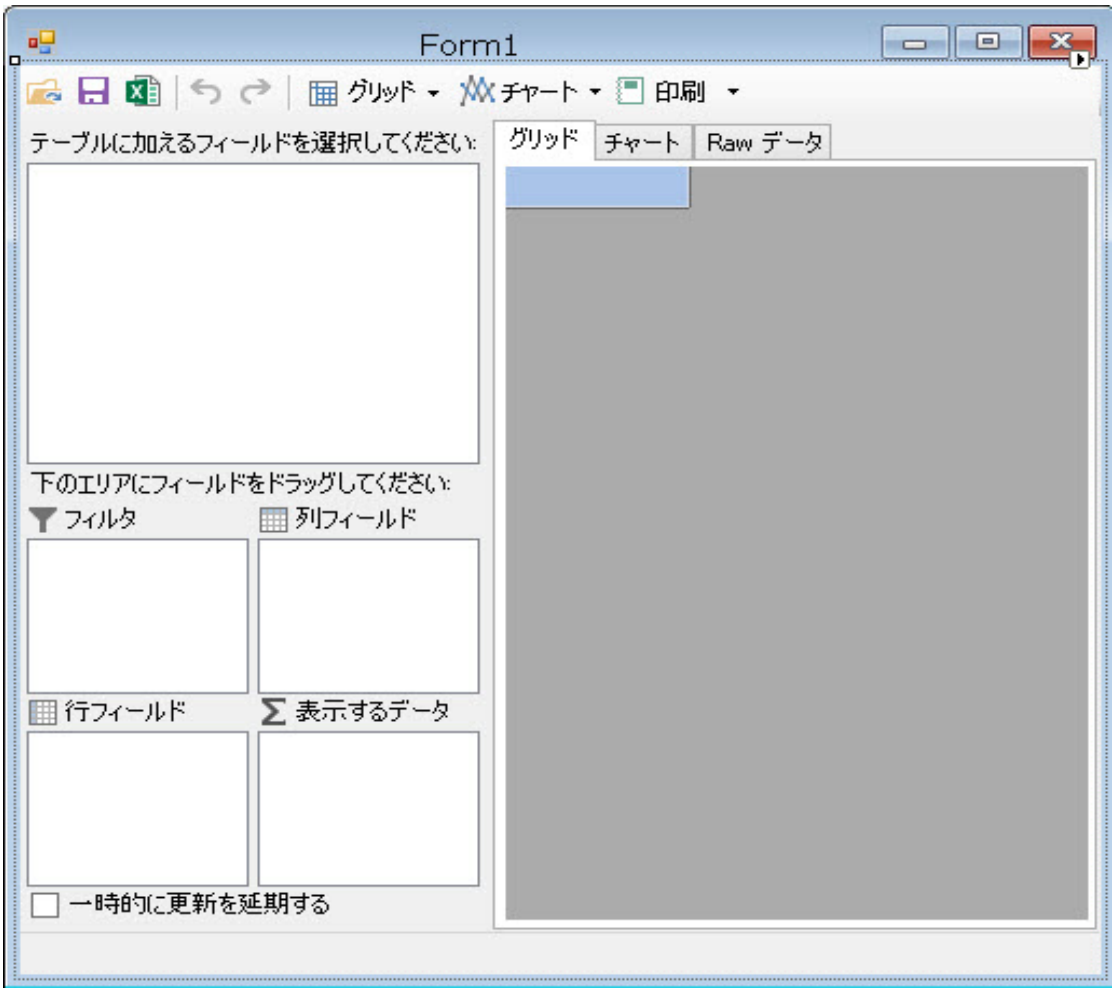
C1OlapPrintDocument コンポーネントを使用して、OLAP ビューに基づいてレポートを作成します。これにより、**PrintDocument** クラスが拡張され、コンテンツを指定するためのプロパティ、およびレポートの作成で使用する OLAP グリッド、チャート、および生データを表示する書式設定を指定するためのプロパティが提供されます。

C1Olap クイックスタート

このセクションでは、コードのウォークスルーを行います。単純な WPF または Silverlight アプリケーションから開始し、広く使用される機能に進みます。

単純な OLAP アプリケーション

最も単純な **C1OLAP** アプリケーションを作成するには、新しい WPF アプリケーションまたは Silverlight アプリケーションを作成して、**C1OlapPage** コントロールをページにドラッグすることから開始します。すべてのマージンおよび配置設定を削除して、**C1OlapPage** コントロールがページ全体を満たすことができるようにします。



ここで、アプリケーションのデータソースを設定します。

このサンプルでは、Northwind 製品データを XML データスキーマファイルからロードします。**ComponentOne Data** を使用します。これは、データを読み込むために、よく知られた DataSet オブジェクトと DataTable オブジェクトを提供します。また、**C1Zip** をクライアント上で使用して、zip 形式で圧縮された XML ファイルをアンパックします。

C#

```
// 埋め込まれた zip リソースからデータを読み込みます
var ds = new DataSet();
var asm = Assembly.GetExecutingAssembly();
using (var s = asm.GetManifestResourceStream("OlapQuickStart.nwind.zip"))
        var zip = new C1ZipFile(s);
```

```
        using (var zr = zip.Entries[0].OpenReader())
         {
             // データを読み込みます
             ds.ReadXml(zr);
         }
    }
```

次に、**C1OlapPage** コントロールに対して、単純に **DataSource** プロパティを設定します。このコントロールでは、任意のデータ連結方法を使用できます。

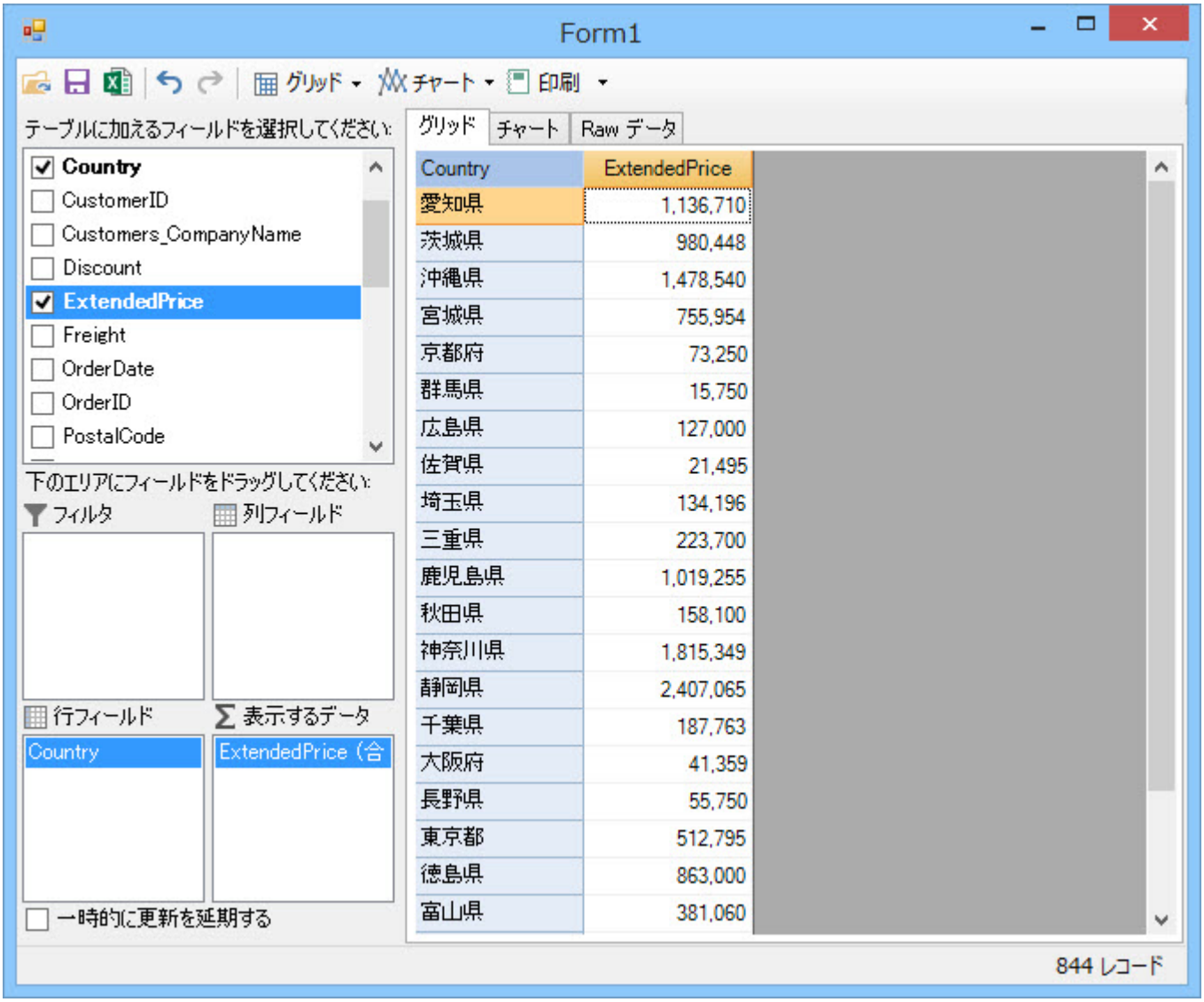
C#

```
// olap ページをデータに連結します
_c1OlapPage.DataSource = ds.Tables[0].DefaultView;
```

これで、アプリケーションを使用する準備が整います。以降のセクションでは、データソースの設定のほかにはコードを記述することなく、デフォルトで提供される機能について説明します。

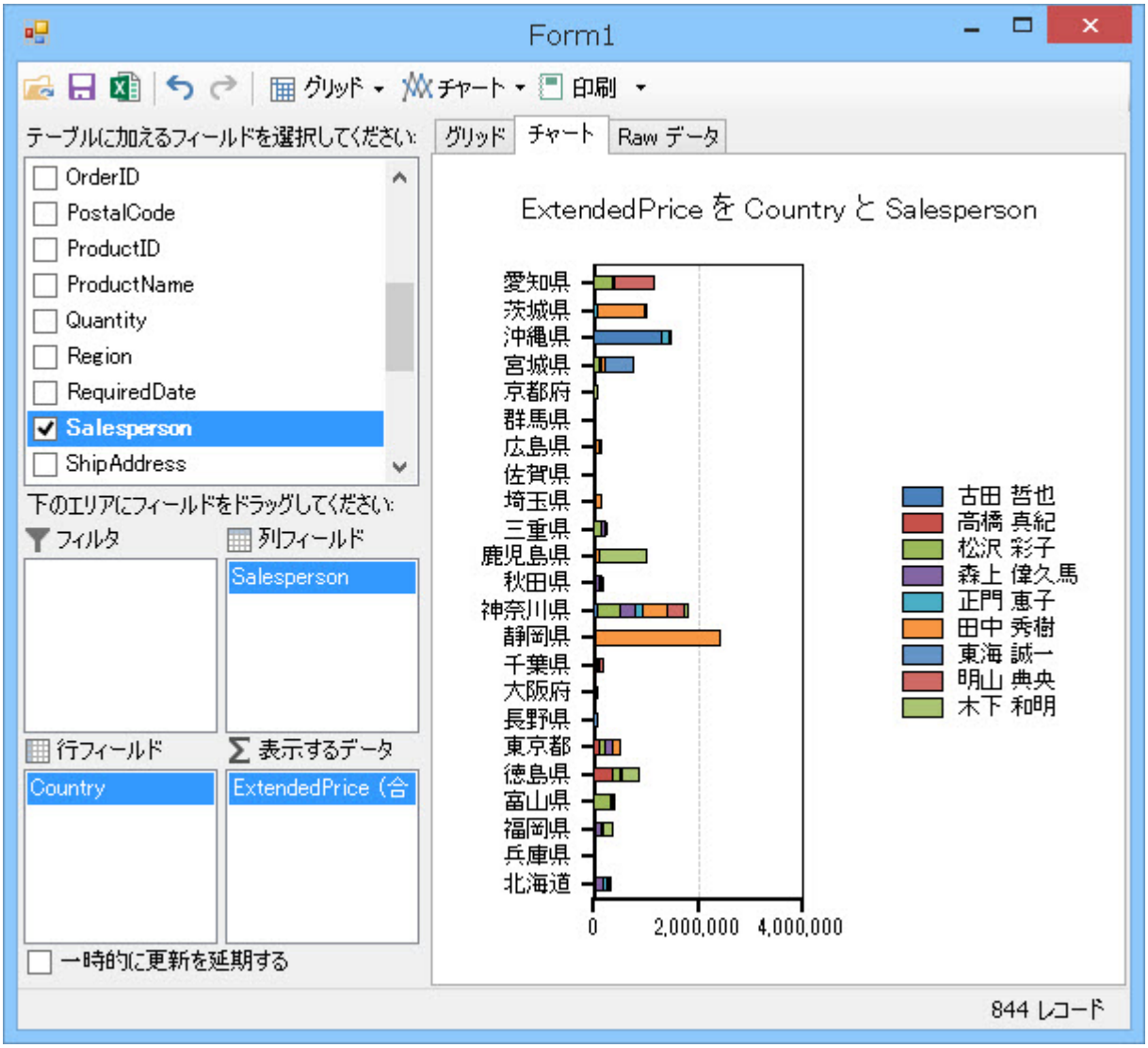
OLAP ビューの作成

アプリケーションを実行すると、Microsoft Excel と同じようなインタフェースが表示されます。[Country]フィールドを[行フィールド]リストにドラッグし、[ExtendedPrice]を[値フィールド]リストにドラッグすると、次の図に示すように、各国で課金される売上の概要が表示されます。



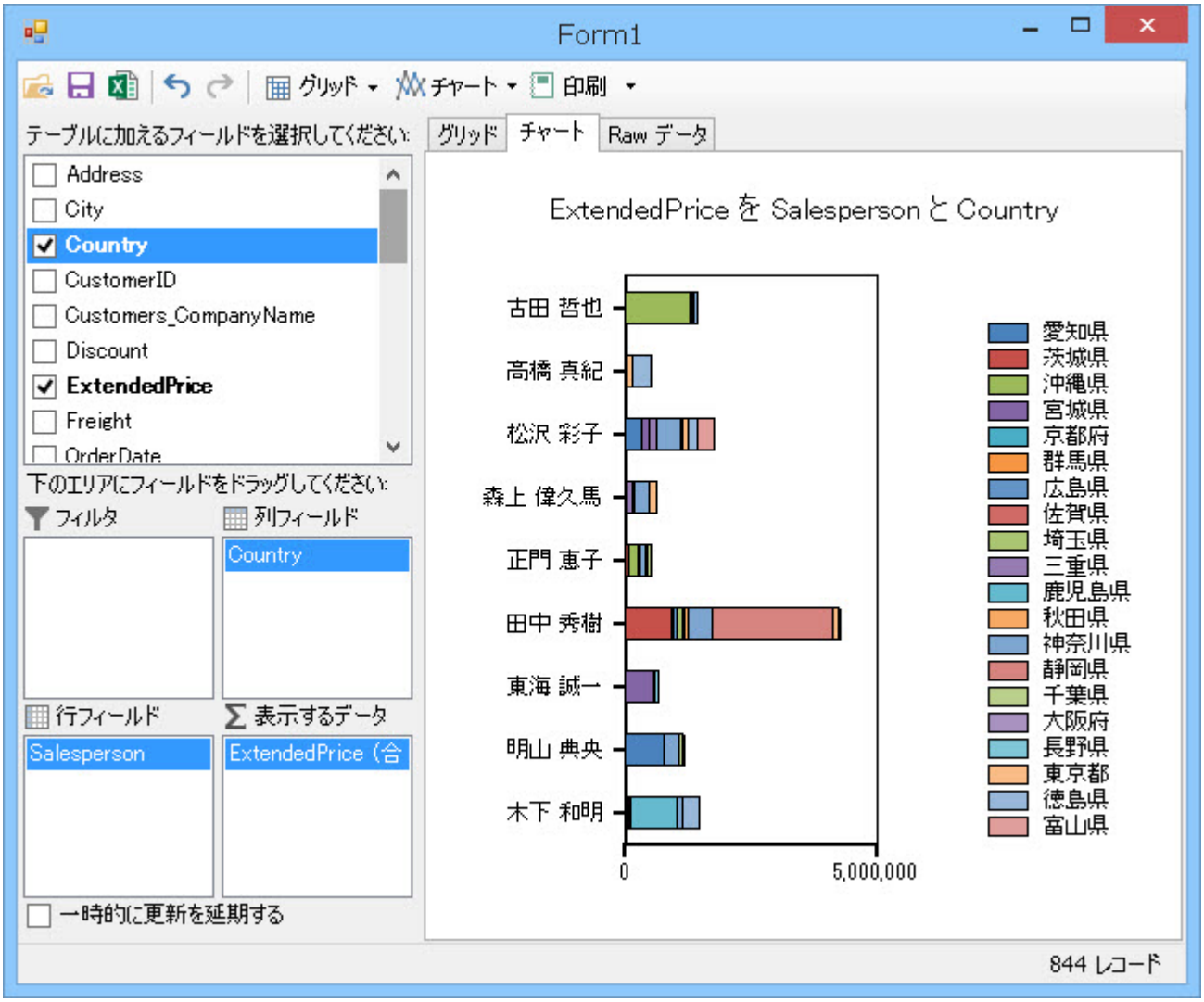
[OLAP チャート]タブをクリックすると、同じデータがチャート形式で表示され、米国、ドイツ、およびオーストリアが主要な顧客であることがわかります。

次に、[Salesperson]フィールドを[列フィールド]リストにドラッグすると、今度は国とカテゴリ別による売上の新しいサマリーが表示されます。引き続き[チャート]タブを選択してあると、先ほどと同じようなチャートですが、今回は横棒が分割され、営業担当者ごとの売上高が表示されます。



チャートの上にマウスを移動し、チャート要素の上にマウスポインタを置くと、営業担当者の名前および売上高を示すツールチップが表示されます。

次に、[Salesperson]フィールドと[Country]フィールドを向かい側のリストにドラッグして交換することにより、新しいビューを作成します。これで、営業担当者ではなく、カテゴリを強調した新しいチャートが作成されます。



ビューに変更を加えると、**C1OlapPanel** コントロールによって記録されます。**C1OlapPanel** メニューの[元に戻す]ボタンをクリックすることで、作成した前のビューに戻ることができます。

データの要約

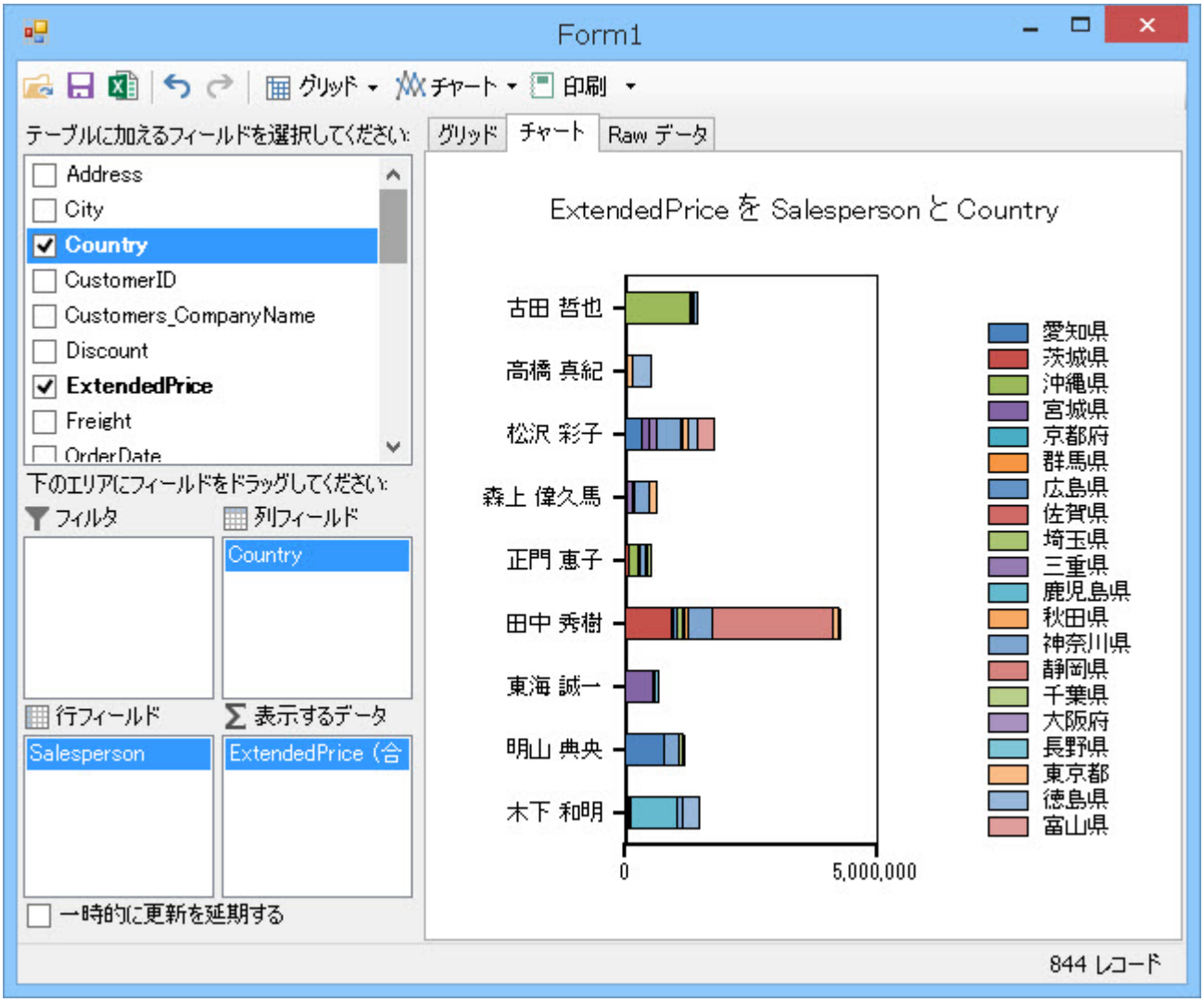
次の例に移る前に、データをさまざまな方法でどのように簡単に集計できるかを示すために、新しいビューを作成します。

まず、**[Country]**フィールドの横にあるチェックボックスをオフにして、ビューから国を削除します。

今回は、[Salesperson]フィールドを[行フィールド]リストにドラッグし、[OrderDate]フィールドを[列フィールド]リストにドラッグします。結果として得られるビューには、注文日ごとに1つの列が含まれます。列が多すぎて傾向がはっきりと示されないため、これはあまり役立つ情報とはいえません。そこで、データを月別または年別に集計しようと思います。

1つめの方法としては、SQL で新しいクエリーを作成するか、LINQ を使用して、ソースデータを変更します。この2つのテクニックについては、この後のセクションで説明します。もう1つの方法は、[OrderDate]フィールドのパラメータを変更するというものです。これには、[OrderDate]フィールドを右クリックし、**[フィールドの設定]**をクリックします。次に、ダイアログボックスの[書式]タブを選択し、[カスタム]書式を選択し、「yyyy」と入力し、**[OK]**をクリックします。

日付が書式設定され、年ごとに集計されて、OLAP チャートは次のようになります。

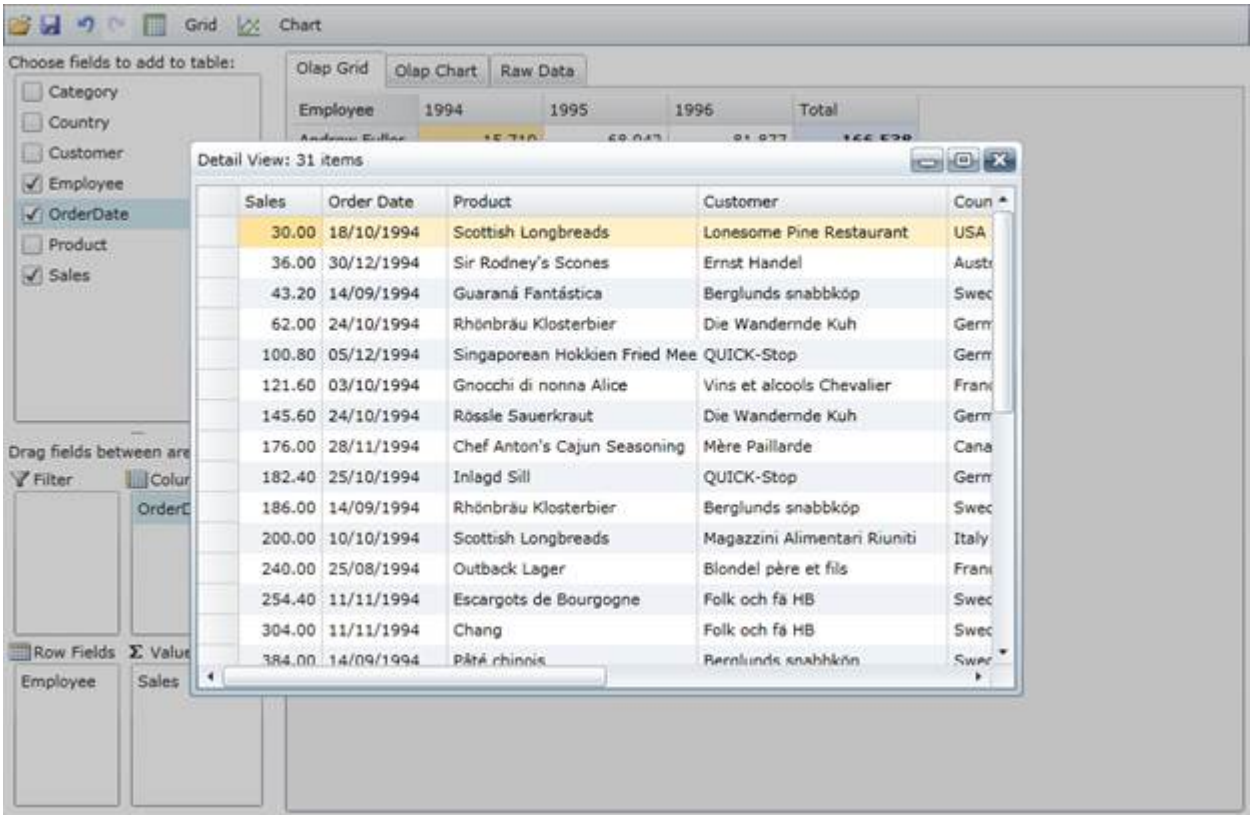


月別または平日別に売上方法を確認する場合は、書式を "MMMM" または "dddd" に変更します。

データのドリルダウン

前に説明したように、OLAP グリッド内の各セルは、データソースのいくつかのレコードのサマリーを表します。OLAP グリッド内の各セルの基底のレコードは、セルをマウスでダブルクリックすることで表示できます。

これを確認するには、[OLAP グリッド]タブをクリックし、グリッドの最初のセル、つまり Andrew Fullers の 1994 年の売上を表すセルをダブルクリックします。もう1つのグリッドに、OLAP グリッドに表示される合計の計算に使用された 31 のレコードが表示されます。



C1OlapPage のカスタマイズ

前の例では、**C1OlapPage** コントロールと最小限のコードだけを使用して、完全な OLAP アプリケーションを作成する方法を示しました。これは簡便な方法ですが、ほとんどの場合は、アプリケーションおよびユーザーインタフェースを多少カスタマイズすることになります。

コードによるフィールドの設定

OLAP アプリケーションの主な長所の1つは、対話的操作です。ユーザーは、ビューを簡単に作成および変更して、直ちにその結果を確認できる必要があります。**OLAP for WPF/Silverlight** は、Excel 形式のユーザーインタフェースと、ユーザーフレンドリで簡潔なダイアログによってこれを可能にしています。

ただし、場合によっては、コードを使用してビューを構成したいことがあります。**OLAP for WPF/Silverlight** は、その簡潔でありながら強力なオブジェクトモデル、特に **Field** クラスと **Filter** クラスによってこれを可能にしています。

以下の例は、**OLAP for WPF/Silverlight** でロード時にビューを構成する方法を示しています。

C#

```
// 売上高を顧客とカテゴリ別に表示します
var olap = _c1OlapPage.OlapPanel.OlapEngine;
olap.DataSource = ds.Tables[0].DefaultView;
olap.BeginUpdate();
olap.RowFields.Add("Country");
olap.ColumnFields.Add("Category");
olap.ValueFields.Add("Sales");
olap.Fields["Sales"].Format = "n0";
olap.EndUpdate();
```

このコードは、出力テーブルへの自動的な更新を一時停止する BeginUpdate メソッドを呼び出します。行、列、および値フィールドのコレクション用にフィールドを追加するので、ユーザーがこのアクションを行う必要はありません。そのため、アプリケーションの **C1OlapPanel** の部分を非表示にできます。また、このコードは、[Sales]フィールドに数値書式を適用して、最後に EndUpdate メソッドを呼び出します。

ここでサンプルを実行すると、最初の例に類似した OLAP ビューが表示されます。

次に、**OLAP for WPF/Silverlight** オブジェクトモデルを使用して、注文日付と合計価格の表示に使用する書式を変更します。

C#

```
// 注文日付の書式を設定します
var field = olap.Fields["OrderDate"];
field.Format = "yyyy";
// 合計価格の書式を設定し、小計の型を変更します
// (合計ではなく)平均合計価格を表示します
field = olap.Fields["Sales"];
field.Format = "c";
field.Subtotal = C1.Olap.Subtotal.Average;
```

このコードは、データソースで指定されたすべてのフィールドを含む **Fields** コレクションから個々のフィールドを取得します。次に、目的の値を **Format** プロパティと **Subtotal** プロパティに割り当てます。**Format** は通常の .NET 書式文字列であり、**Subtotal** は OLAP ビューで表示するために値を集計する方法を決定します。デフォルトでは値が追加されますが、平均、最大、最小、標準偏差、分散など、その他のさまざまな集計値を使用できます。

ここで、データのサブセット、たとえばいくつかの製品と1年という期間だけに注目すると想定します。それらのフィールドを右クリックして、フィルタを適用します。次に示すように、まったく同じことをコードで実行できます。

C#

```
// 注文日付と合計価格の書式を設定します
// 変更しません...
// 値フィルタを適用して、少数の製品だけを表示します
C1.Olap.C1OlapFilter filter = olap.Fields["Product"].Filter;
filter.Clear();
filter.ShowValues = "Chai,Chang,Geitost,Ikura".Split(',');
// 条件フィルタを適用して、いくつかの日付だけを表示します
filter = olap.Fields["OrderDate"].Filter;
filter.Clear();
filter.Condition1.Operator =
         C1.Olap.ConditionOperator.GreaterThanOrEqualTo;
filter.Condition1.Parameter = new DateTime(1996, 1, 1);
filter.Condition2.Operator =
         C1.Olap.ConditionOperator.LessThanOrEqualTo;
filter.Condition2.Parameter = new DateTime(1996, 12, 31);
filter.AndConditions = true;
```

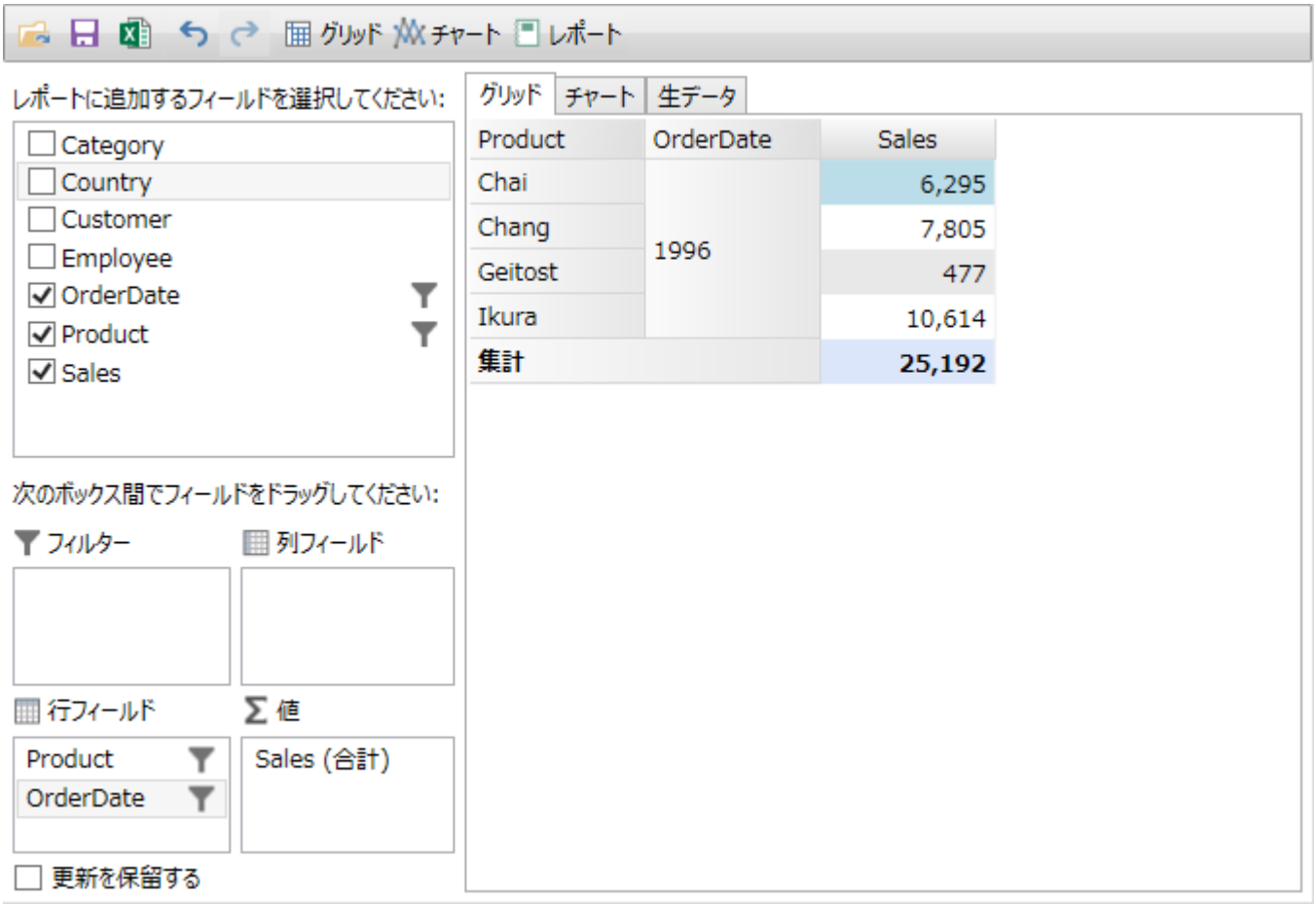
コードでは、最初に、[Product]フィールドに関連付けられた **C1OlapFilter** オブジェクトを取得します。次に、フィルタをクリアして、その **ShowValues** プロパティを設定します。このプロパティは、フィルタで表示される値の配列を取ります。**OLAP for WPF/Silverlight** では、これを「値フィルタ」と呼びます。

次に、コードは、[OrderDate]フィールドに関連付けられたフィルタを取得します。ここでは、特定の年度の値を表示します。ただし、対象年度のすべての日を列挙するわけではありません。代わりに、2つの条件によって定義される「条件フィルタ」を使用します。

最初の条件は、[OrderDate]が 1996 年1月1日以降でなければならないことを指定します。2番目の条件は、[OrderDate]

が 1996 年 12 月 31 日以前でなければならないことを指定します。**AndConditions** プロパティは、最初の条件と2番目の条件を適用する方法（AND 演算または OR 演算）を指定します。この場合、両方の条件が true になることが求められているので、**AndConditions** に true を設定します。

もう一度プロジェクトを実行すると、次のように表示されます。



ローカルストレージ内の永続的な OLAP ビュー

デフォルトビューがロードされることは問題ありませんが、ユーザーは、アプリケーションを実行するたびにビューを変更しなければならないことにうんざりしているかもしれません。Silverlight では、分離ストレージに単純なデータを保存することにより、ビューを保存してセッションをまたいで利用することができます。最初に、セッションをまたいで維持されるデフォルトビューを分離ストレージ内に作成します。**IsolatedStorageSettings.ApplicationSettings** クラスを使用することで、アプリケーション設定を極めて簡単に保存およびロードすることができます。デフォルトでは、分離ストレージは1 MB に制限されますが、OLAP ビューのサイズはこれに影響されません。

この例では、現在のアプリケーションの Exit イベントで現在のビューを保存します。ユーザーによって行われたカスタマイズは、アプリケーションを閉じると自動的に保存され、次回のアプリケーションの実行時に復元することができます。

C#

```
// アプリケーションが終了するときに現在のビューをストレージに保存します
void Current_Exit(object sender, EventArgs e)
{
var userSettings = IsolatedStorageSettings.ApplicationSettings;
userSettings[VIEWDEF_KEY] = _c1OlapPage.ViewDefinition;
userSettings.Save();
}
```

ここでは、一意のキーインデックスを使用してアプリケーション設定にアクセスしていることに注目してください。ViewDefinition プロパティから、このデータセットのビューを定義する XML 形式の文字列であるデータを保存します。アプリケーションのどの時点でも、コードの2行目と逆の動作を行うことで、OLAP ビューを復元できます。次に、分離ストレージからビューをロードします。

const string VIEWDEF_KEY = "C1OlapViewDefinition";

アプリケーションを通じて保存されたデータビューに、同じ一意のキーを使用して簡単にアクセスできるように、VIEWDEF_KEY 定数を宣言するこのコード行を追加します。

Application.Current.Exit += Current_Exit;

上のコード行は、アプリケーションが終了する前に発生する Exit イベントをアタッチします。次に、保存で使用したコードと逆の動作を行うことで、分離ストレージからビューをロードします。

C#

```
// olap ビューを初期化します
var userSettings = IsolatedStorageSettings.ApplicationSettings;
if (userSettings.Contains(VIEWDEF_KEY))
{
// 最後に使用された OLAP ビューを分離ストレージからロードします
_c1OlapPage.ViewDefinition = userSettings[VIEWDEF_KEY] as string;
}
```

ここでプロジェクトを実行すると、コードで作成されたデフォルトビューでプロジェクトが開始されます。ビューに任意の変更を加えてからアプリケーションを終了し、再起動して、変更が復元されていることを確認します。

定義済みビューの作成

現在のビューを XML 文字列として取得または設定する **ViewDefinition** プロパティに加えて、**C1OlapPage** コントロールは、ビューをファイルおよびストリームに維持できる **ReadXml** および **WriteXml** メソッドも公開しています。これらのメソッドは、組み込みメニューの[ロード]および[保存]ボタンをクリックすると、**C1OlapPage** によって自動的に呼び出されます。

これらのメソッドを使用することで、事前定義されたビューをとても簡単に実装することができます。この場合は、最初にいくつかビューを作成し、[保存]ボタンを押してそれぞれのビューを保存します。この例では、以下の分類で売上を示す5つのビューを作成します。

1. 製品と国
2. 従業員と国
3. 従業員と月
4. 従業員と平日
5. 従業員と年

すべてのビューを作成して保存したら、"OlapViews" ノードを含む "DefaultViews.xml" という名前の新しい XML ファイルを作成し、すべてのデフォルトビューをコピーしてこのドキュメントに貼り付けます。次に、各ビューに "id" タグを追加し、それぞれに一意の名前を割り当てます。この名前は、ユーザーインタフェースに表示されます（**C1OlapGrid** では必要ありません）。XML ファイルは次のようになります。

XAML

```
<OlapViews>
<C1OlapPage id="Product vs Country">
<!-- view definition omitted... -->
C1OlapPage id="Employee vs Country">
<!-- view definition omitted... -->
<C1OlapPage id="Employee vs Month">
```

```
<!-- view definition omitted... -->
<C1OlapPage id="Employee vs Weekday">>
<!-- view definition omitted... -->
<C1OlapPage id="Employee vs Year">
<!-- view definition omitted... -->
</OlapViews>
```

このファイルをプロジェクトにリソースとして追加します。それには、プロジェクトに新しいフォルダを追加し、"Resources" という名前を付けます。次に、ソリューションエクスプローラーで、[Resources]フォルダを右クリックし、[既存のファイルの追加...]オプションを選択します。XML ファイルを選択し、[OK]をクリックします。

ビュー定義の準備ができたので、ユーザーが選択できるようにメニューに公開する必要があります。それには、次のコードをプロジェクトにコピーします。

C#

```
public MainPage()
{
InitializeComponent();
//ここは変更しません
//…
// 事前定義されたビューを XML リソースから取得します
var views = new Dictionary<string, string>();
using (var s =
asm.GetManifestResourceStream("OlapQuickStart.Resources.OlapViews.xml"))
using (var reader = XmlReader.Create(s))
{
// 事前定義されたビュー定義を読み込みます
while (reader.Read())
{
if (reader.NodeType == XmlNodeType.Element && reader.Name == "C1OlapPage")
{
var id = reader.GetAttribute("id");
var def = reader.ReadOuterXml();
views[id] = def;
}
}
}
// 事前定義されたビューで新しいメニューを構築します
var menuViews = new C1MenuItem();
menuViews.Header = "View";
menuViews.Icon = GetImage("Resources/views.png");
menuViews.VerticalAlignment = VerticalAlignment.Center;
ToolTipService.SetToolTip(menuViews, "Select a predefined Olap view.");
foreach (var id in views.Keys)
{
var mi = new C1MenuItem();
mi.Header = id;
mi.Tag = views[id];
mi.Click += mi_Click;
menuViews.Items.Add(mi);
}
// ページのメインメニューに新しいメニューを追加します
_c1OlapPage.MainMenu.Items.Insert(6, menuViews);
```

```
}
```

このコードは、OLAP 定義を含む XML ドキュメントをロードし、C1Menu を使用して新しいドロップダウンメニュー項目を作成し、検出されたビューをドロップダウンに挿入します。各メニュー項目の **Header** プロパティにはビュー名、**Tag** プロパティには実際の XML ノードが含まれます。このノードは、後でユーザーが選択したビューを適用するために使用されます。

ドロップダウンの準備ができたら、このコードは、**MainMenu** プロパティを使用して **C1OlapPage** にドロップダウンを追加します。最初のいくつかのボタンの後に、新しいボタンが追加されます。

上のコードでは、新しいメニューボタンの画像をロードするために、単純なメソッド GetImage が呼び出されています。1つの画像をロードするだけであれば、このような作業は必要ありませんが、複数の画像をロードする場合いは、何度も使用できる一般的なメソッドが役立ちます。

C#

```
// URI から画像をロードするためのユーティリティ
static Image GetImage(string name)
{
var uri = new Uri(name, UriKind.Relative);
var img = new Image();
img.Source = new BitmapImage(uri);
img.Stretch = Stretch.None;
img.VerticalAlignment = VerticalAlignment.Center;
img.HorizontalAlignment = HorizontalAlignment.Center;
return img;
}
```

ここまでで不足しているコードは、ユーザーがメニュー項目をクリックして選択する際に、**C1OlapPage** にビューを適用するコードです。これは、次のコードを使用して行います。

C#

```
// 事前定義されたビューを適用します
void mi_Click(object sender, SourcedEventArgs e)
{
var mi = sender as C1MenuItem;
var viewDef = mi.Tag as string;
_c1OlapPage.ViewDefinition = viewDef;
}
```

このコードは、メニューの **Tag** プロパティを読み込むことで、XML 文字列として OLAP 定義を取得し、それを **C1OlapPage.ViewDefinition** プロパティに割り当てます。

さらにカスタマイズが必要な場合は、**C1OlapPage** をまったく使用せずに、下位レベルの **C1OlapPanel**、**C1OlapGrid**、および **C1OlapChart** コントロールを使用してユーザーインタフェースを構築することもできます。**C1OlapPage** コントロールのソースコードがパッケージに含まれており、作業の開始点として使用できます。「カスタムユーザーインタフェースの作成」セクションの例は、この方法を示しています。

OLAP ビューの更新

C1OlapPage または **C1O** に対して特定の時点で更新を適用することにより、分析を再生成したい場合があります。C1OlapEngine の Update メソッドを呼び出すと実現できます。この機能を UI に追加するには、1つのボタンを追加し、そのクリックイベントに次のコードを追加します。

C#

```
// OLAP ビューを再生成します
void Button_Click(object sender, RoutedEventArgs e)
{
    _c1OlapPage.OlapPanel.OlapEngine.Update();
}
```

条件付き書式設定

C1OlapGrid コントロールは **C1FlexGrid** コントロールから派生したものなので、グリッドのカスタムセル機能を使用すると、そのコンテンツに基づいてセルにスタイルを適用できます。このサンプルは、100 よりも大きい値が薄緑の背景で表示されるグリッドを示しています。

この C1OlapGrid コントロールには、グリッドに表示されるすべてのセルを作成する **CellFactory** クラスが含まれます。カスタムセルを作成するには、**ICellFactory** インタフェースを実装するクラスを作成し、このクラスをグリッドの **CellFactory** プロパティに割り当てる必要があります。カスタム列と同様に、カスタム **ICellFactory** クラスは、極めて特殊なアプリケーション固有のクラスである場合も、一般的で再利用可能な構成可能なクラスである場合も考えられます。通常、カスタム **ICellFactory** クラスは直接セルを操作するため、カスタム列よりも格段に簡潔です。

以下に、100 を超える値のセルに緑のカスタム背景を適用する ConditionalCellFactory クラスを実装するコードを示します。

C#

```
public class ConditionalCellFactory : C1.Silverlight.FlexGrid.CellFactory
{
    public override FrameworkElement CreateCell(C1FlexGrid grid, CellType cellType,
CellRange range)
    {
       // 大部分の作業は基本クラスで処理します
        var cell = base.CreateCell(grid, cellType, range);
       // 必要に応じて緑の背景を適用します
        if (cellType == CellType.Cell)
        {
            var cellValue = grid[range.Row, range.Column];
            if (cellValue is double && (double)cellValue > 100)

            {
                var border = cell as Border;
                border.Background = _greenBrush;
            }
        }
        // 終了
        return cell;
    }
    static Brush _greenBrush = new SolidColorBrush(Color.FromArgb(0xff, 88, 183,
112));
}
```

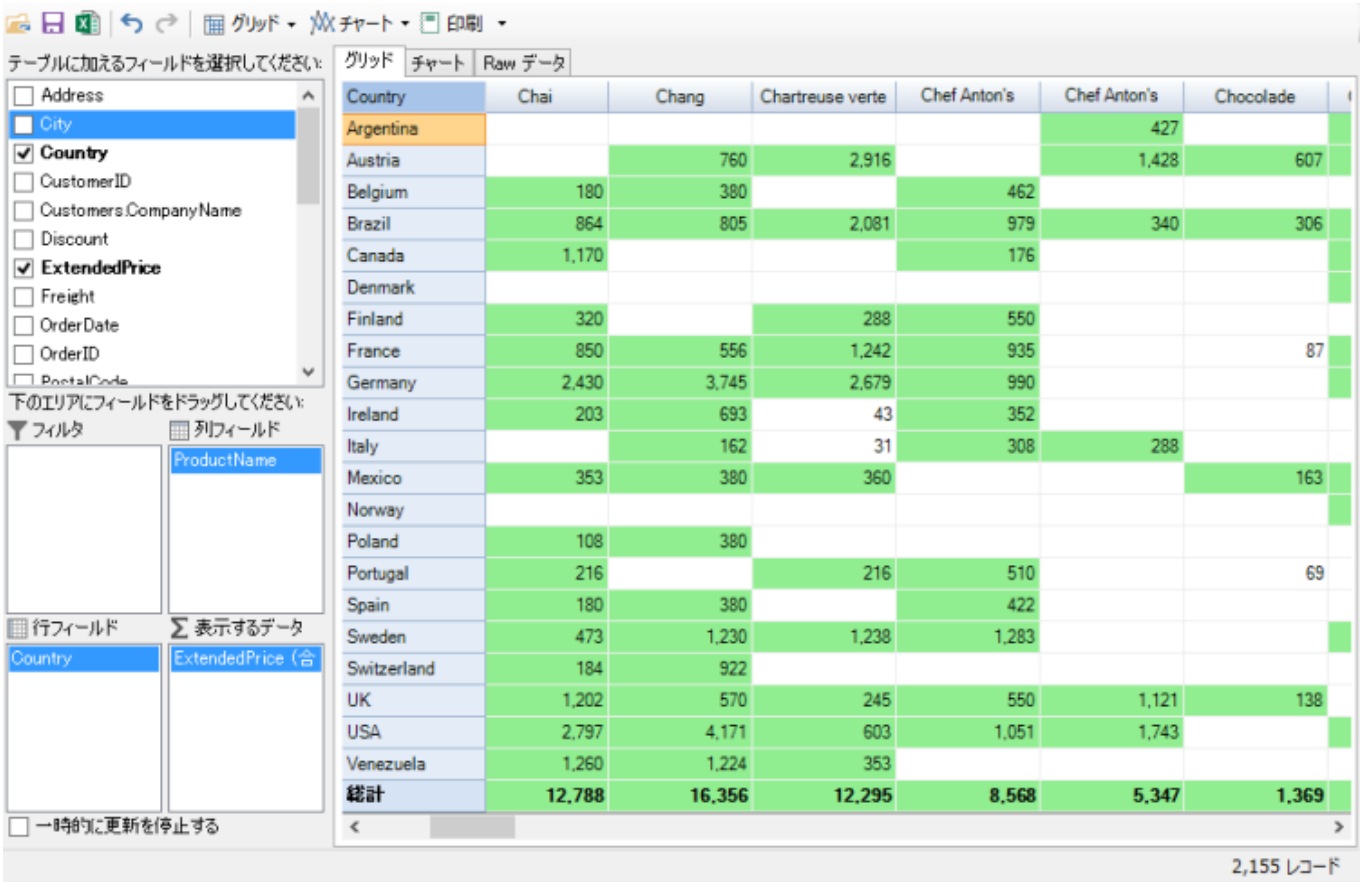
C1OlapGrid でこれを使用するために必要なコードを以下に示します。

C#

```
// グリッドセルに条件付き書式設定を適用します
```

```
_c1OlapPage.OlapGrid.CellFactory = new ConditionalCellFactory();
```

前の例にこのコードを追加すると、実行時にどのように表示されるかを確認できます。



大規模なデータソース

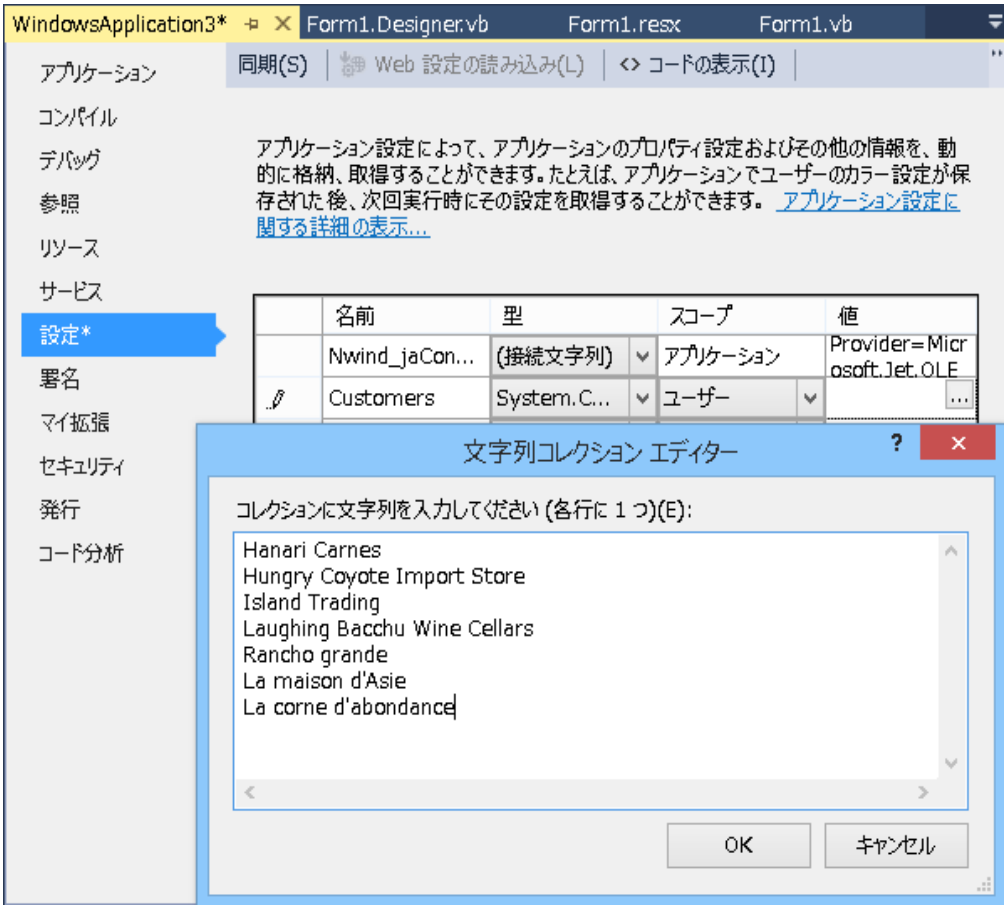
データが多すぎて一度にメモリにロードできない場合があります。たとえば、行数が百万以上あるテーブルを考えてみます。すべてのデータをメモリにロードできたとしても、このプロセスには長い時間がかかります。

これらのシナリオに対処するには、多くの方法があります。サーバーでデータを集計およびキャッシュするクエリーを作成し、Web サービスを使用して、データを Silverlight クライアントに配信する方法があります。これでも、**C1OlapPage** で使用可能なテーブルが作成されます。

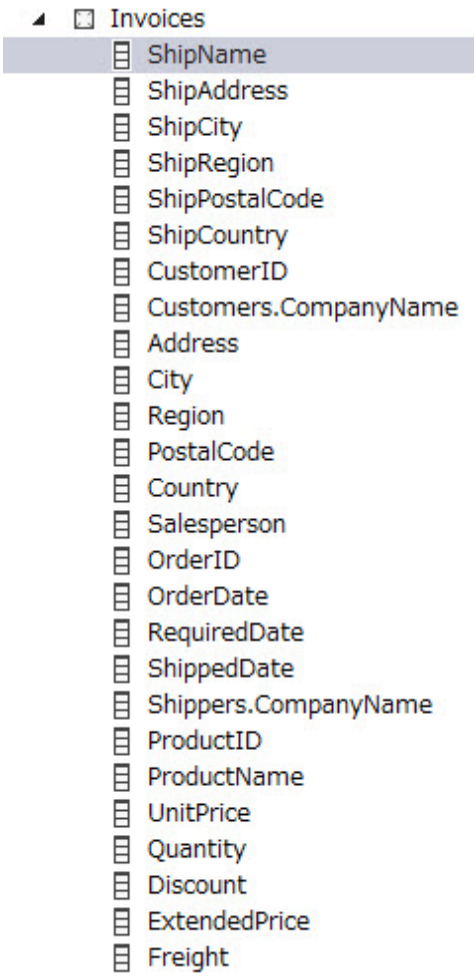
この例では、WCF サービスを使用して、SQL データベースに格納されている Northwind データにアクセスします。この例の注目すべき点は、必ずしもすべてのデータが一度にメモリにロードされるわけではないことです。**C1OlapPage** は、フィルタに現在含まれる顧客のデータだけを要求します。

この例では、ASP.Net Web サイト内に Silverlight プロジェクトを作成します。また、LINQ to SQL クラスを使用して、サンプルの Northwind データベースからデータをクエリーします。LINQ to SQL は、Visual Studio（2008 以降）に含まれている ORM（オブジェクト関係マッピング）実装です。これにより、.NET クラスを使用してリレーショナルデータベースをモデル化し、そのデータベースに対して LINQ を使用してクエリーできます。

最初に、Northwind データベースの LINQ to SQL 表現を作成します。Silverlight プロジェクトに関連付けられている Web サイトプロジェクトを右クリックし、[新しい項目の追加]をクリックします。[LINQ to SQL クラス]を選択し、NorthwindDataClasses.dbml という名前を付けます。



次に、サーバーエクスプローラーから項目をドラッグして、[請求書]ビューからのすべてのデータフィールドを取り込みます。



その後、LINQ および作成したばかりの LINQ to SQL クラス（NorthwindDataClasses）を使用して、このデータをクエリーする WCF サービスを作成します。Web サイトプロジェクトノードを右クリックし、[新しい項目の追加]をクリックします。[WCF サービス]を選択し、NorthwindDataService.svc という名前を付けます。

NorthwindDataService.svc のコードを次のコードに置き換えます。

C#

```
using System;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Activation;
using System.Collections.Generic;
using System.Text;
namespace SqlFilter.Web
{
    [ServiceContract(Namespace = "")]
    [AspNetCompatibilityRequirements(RequirementsMode =
AspNetCompatibilityRequirementsMode.Allowed)]
    public class NorthwindDataService
    {
        /// <summary>/// すべての請求書を取得します。/// </summary>
[OperationContract]
        public List<Invoice> GetInvoices()
```

```
        {
            var ctx = new NorthwindDataClassesDataContext();
            var invoices =
                from inv in ctx.Invoices
                select inv;
            return invoices.ToList();
        }
        /// <summary>/// すべての顧客を取得します。/// </summary>
[OperationContract]
        public List<string> GetCustomers()
        {
            var ctx = new NorthwindDataClassesDataContext();
            var customers =
                (from inv in ctx.Invoices
                    select inv.CustomerName).Distinct();
            return customers.ToList();
        }
        /// <summary>/// 特定の顧客セットのすべての請求書を取得します。/// </summary>
[OperationContract]
        public List<Invoice> GetCustomerInvoices(params string[] customers)
        {
            // ハッシュセットを構築します              var hash = new HashSet<string>();
            foreach (string c in customers)
            {
                hash.Add(c);
            }
            string[] customerList = hash.ToArray();
            // リスト内の顧客の請求書を取得します              var ctx = new
NorthwindDataClassesDataContext();
            var invoices =
                from inv in ctx.Invoices
                where customerList.Contains(inv.CustomerName)
                select inv;
            return invoices.ToList();
        }
    }
}
```

ここで、Web サービスに対して3つのメソッドを定義したことに注目してください。最初の2つは単純な Get メソッドであり、LINQ および以前に作成した LINQ to SQL クラスを使用して、項目のリストを返します。GetCustomerInvoices メソッドは、顧客の配列をパラメータとして受け取るという点で特殊です。これは、Silverlight C1OlapGrid プロジェクトのクライアント側で定義されるフィルタです。

Silverlight プロジェクトに移動する前に、Web サイトプロジェクトを構築し、Web サービスへの参照を追加する必要があります。参照を追加するには、ソリューションエクスプローラーで、Silverlight プロジェクトノードを右クリックし、[サービス参照の追加]をクリックします。次に、[探索]をクリックし、NorthwindDataService.svc を選択します。「NorthwindDataServiceReference」に名前を変更し、[OK]をクリックします。

これでデータソースの準備ができたので、これを **C1OlapPage** に接続して以下を確認する必要があります。

1. ユーザーがフィルタ内のすべての顧客を表示できる(現在ロードされている顧客だけでなく)。
2. ユーザーがフィルタを変更すると、新しいデータがロードされ、要求された新しい顧客が表示される。

これらのタスクを完了する前に、UI を設定する必要があります。MainPage.XAML に、1つの **C1OlapPage** コントロールと、ステータスストリップとして使用されるいくつかの TextBlock を追加します。

XAML

```
<Grid x:Name="LayoutRoot">
<Grid.RowDefinitions>
              <RowDefinition />
              <RowDefinition Height="Auto"/>
</Grid.RowDefinitions>
<olap:C1OlapPage x:Name="_c1OlapPage"/>
<TextBlock x:Name="_lblLoading"FontSize="24"Opacity=".5"Text="Loading
data..."HorizontalAlignment="Center"VerticalAlignment="Center"/>
<TextBlock x:Name="_lblStatus"Text="Ready"HorizontalAlignment="Right"Grid.Row="1"/>
</Grid>
```

次に、次のコードをフォームに追加します。

C#

```
ObservableCollection<string> _allCustomers;
ObservableCollection<NorthwindDataServiceReference.Invoice> _invoices;
C1OlapFilter _customerFilter;
```

これらのフィールドには、データベース内のすべての顧客の完全なリスト、現在ユーザーによって選択されている顧客のリスト、および任意の一時点で選択可能な顧客の最大数が含まれます。

顧客の完全なリストは、**C1OlapField.Values** プロパティに割り当てる必要があります。このプロパティには、フィルタで表示される値のリストが含まれます。デフォルトでは、**C1OlapPage** により、このリストには生データから検出された値が挿入されます。この場合、生データに含まれているのはリストの一部分だけなので、代わりに完全なバージョンを提供する必要があります。**_allCustomers** ObservableCollection は、ユーザーが選択できる顧客のコレクション全体を保持します。**C1OlapPage** は、実際には _invoices コレクションとの組み合わせで使用されます。このコレクションは、選択した顧客によってフィルタ処理されたデータセットです。

MainPage() のコードを次のコードに置き換えます。

C#

```
public MainPage()
{
    InitializeComponent();
     // OlapPage データソースを初期化します     _invoices = new
ObservableCollection<SqlFilter.NorthwindDataServiceReference.Invoice>();
    _c1OlapPage.DataSource = _invoices;
    // OlapPage ビューを初期化します     var olap = _c1OlapPage.OlapEngine;
    olap.BeginUpdate();
    olap.ColumnFields.Add("OrderDate");
    olap.RowFields.Add("CustomerName");
    olap.ValueFields.Add("ExtendedPrice");
    olap.RowFields[0].Width = 200;
    olap.Fields["OrderDate"].Format = "yyyy";
    olap.Fields["CustomerName"].Filter.ShowValues = selectedCustomers.ToArray();
    olap.EndUpdate();

    // データベース内のすべての顧客のリストを取得します     var sc = new
SqlFilter.NorthwindDataServiceReference.NorthwindDataServiceClient();
    sc.GetCustomersCompleted += sc_GetCustomersCompleted;
    sc.GetCustomersAsync();
```

```
    // ステータスを表示します     _lblStatus.Text = "Retrieving customer list...";
}
```

ここでは、**C1OlapPage** データソースを初期化し、デフォルトビューを作成し、データベース内のすべての顧客のリストを取得します。ユーザーが必要な顧客を選択できるように、データベース内のすべての顧客の完全なリストを取得する必要があります。このリストは、件数は多いもののサイズは大きくありません。リストには顧客名だけが含まれ、注文や注文詳細などの関連付けられた詳細は含まれません。

データは Web サービスから入手し、非同期に取得されるため、データのロードが終了すると sc_GetCustomersCompleted イベントが発生します。

C#

```
void sc_GetCustomersCompleted(object sender,
SqlFilter.NorthwindDataServiceReference.GetCustomersCompletedEventArgs e)
{
    // "ロード中" メッセージを非表示にします     _lblLoading.Visibility = Visibility.Collapsed;
    // CustomerName フィルタを監視します     _customerFilter =
_c1OlapPage.OlapEngine.Fields["CustomerName"].Filter;
    _customerFilter.PropertyChanged += filter_PropertyChanged;
    // ビュー定義を監視して、[CustomerName]フィールドが常にアクティブであることを確認します
_c1OlapPage.ViewDefinitionChanged += _c1OlapPage_ViewDefinitionChanged;
    // 利用可能な顧客を[CustomerName]フィールドのフィルタに表示します     _allCustomers =
e.Result;
    _customerFilter.Values = _allCustomers;
    // データを取得します     GetData();
}
```

このイベントは、データベース内の顧客の完全なリストを取得します。フィルタで表示するために、このリストを保存します。**C1OlapField.PropertyChanged** イベントを監視する必要があります。このイベントは、ユーザーがフィルタを含む任意のフィールドプロパティを変更すると発生します。このイベントが発生したら、ユーザーによって選択された顧客のリストを取得し、そのリストをデータソースに渡します。

次は、フィルタが変更されたときにデータソースを更新するイベントハンドラです。

C#

```
// [CustomerName]フィールドのフィルタが変更されています。新しいデータを取得します
void filter_PropertyChanged(object sender,
System.ComponentModel.PropertyChangedEventArgs e)
{
    GetData();
}
```

フィールドの **Filter** プロパティは、このフィールドがビューで "アクティブ" である場合に、**C1OlapEngine** によって考慮されます。"アクティブ" とは、フィールドが **RowFields**、**ColumnFields**、**ValueFields**、または **FilterFields** コレクションのメンバであることを意味します。この場合、[CustomerName]フィールドは特殊なフィルタが設定され、常にアクティブである必要があります。このためには、エンジンの **ViewDefinitionChanged** イベントを処理して、[Customers]フィールドを常にアクティブにします。

次は、[CustomerName]フィールドを常にアクティブにするコードです。

C#

```
// Customer フィールドを常にアクティブにします
void _c1OlapPage_ViewDefinitionChanged(object sender, EventArgs e)
```

```
{
    var olap = _c1OlapPage.OlapEngine;
    var field = olap.Fields["CustomerName"];
    if (!field.IsActive)
    {
        olap.FilterFields.Add(field);
    }
}
```

GetData メソッドが呼び出されて、フィルタで選択された顧客の請求書データが取得されます。

C#

```
// 選択された顧客の請求書データを取得します
void GetData()
{
    // 現在のフィルタ設定に基づいて、アクティブな顧客リストを再作成します     var selectedCustomers =
new ObservableCollection<string>();
    foreach (string customer in _allCustomers)
    {
        if (_customerFilter.Apply(customer))
        {
            selectedCustomers.Add(customer);
        }
    }
    _customerFilter.ShowValues = selectedCustomers.ToArray();
    // 選択された顧客の請求書を取得します     var sc = new
SqlFilter.NorthwindDataServiceReference.NorthwindDataServiceClient();
    sc.GetCustomerInvoicesCompleted += sc_GetCustomerInvoicesCompleted;
    sc.GetCustomerInvoicesAsync(selectedCustomers);
    // ステータスを表示します     _lblStatus.Text = string.Format("Retrieving invoices for
{0} customers...", selectedCustomers.Count);
}
```

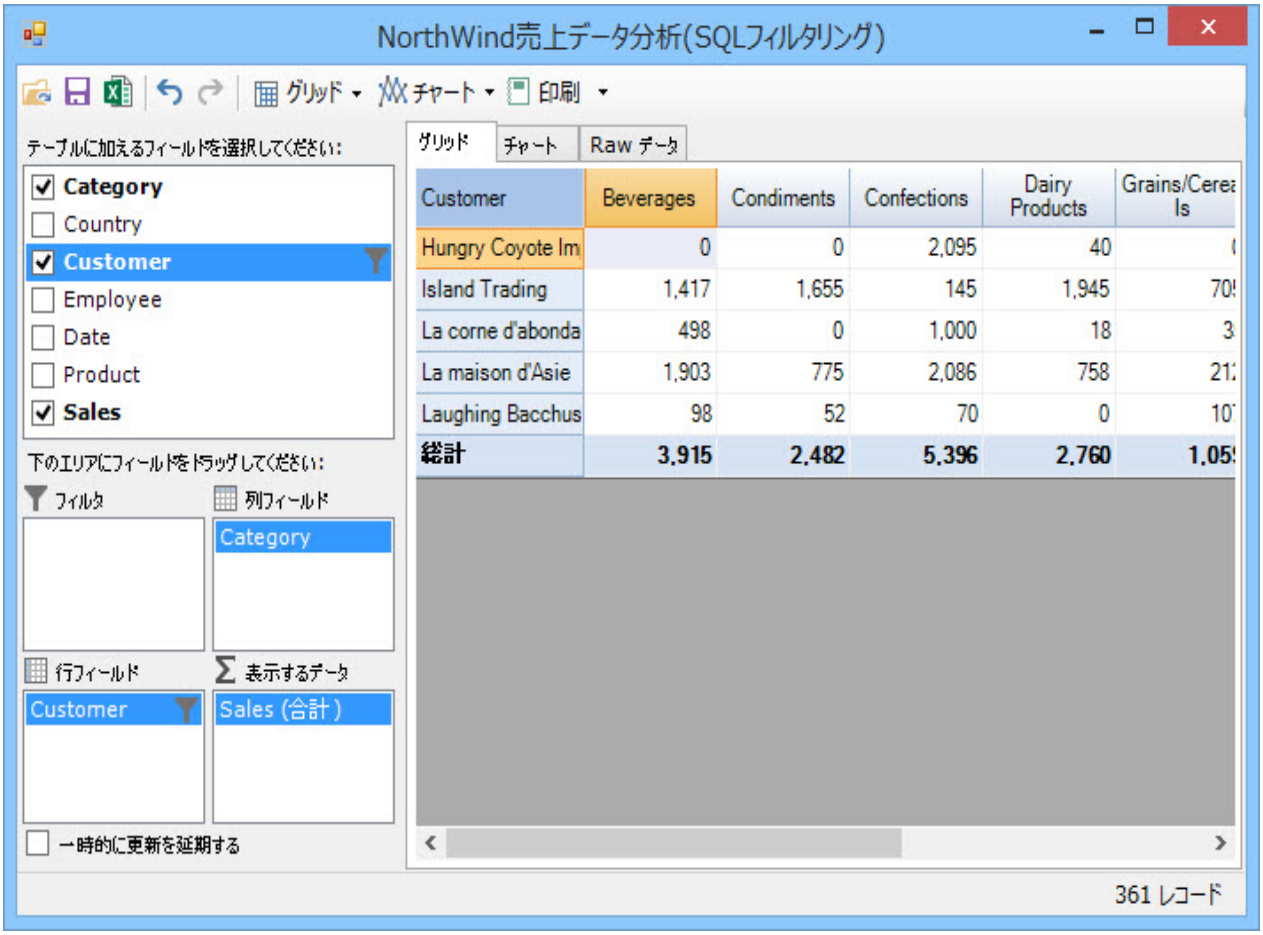
ここでは、ユーザーによって選択された顧客のリストを構築するために、C1OlapFilter(_customFilter)を使用して、その **Apply** メソッドを呼び出します。次のイベントでは、フィルタ処理された請求書データを返す Web サービスへの別の非同期呼び出しを行います。

C#

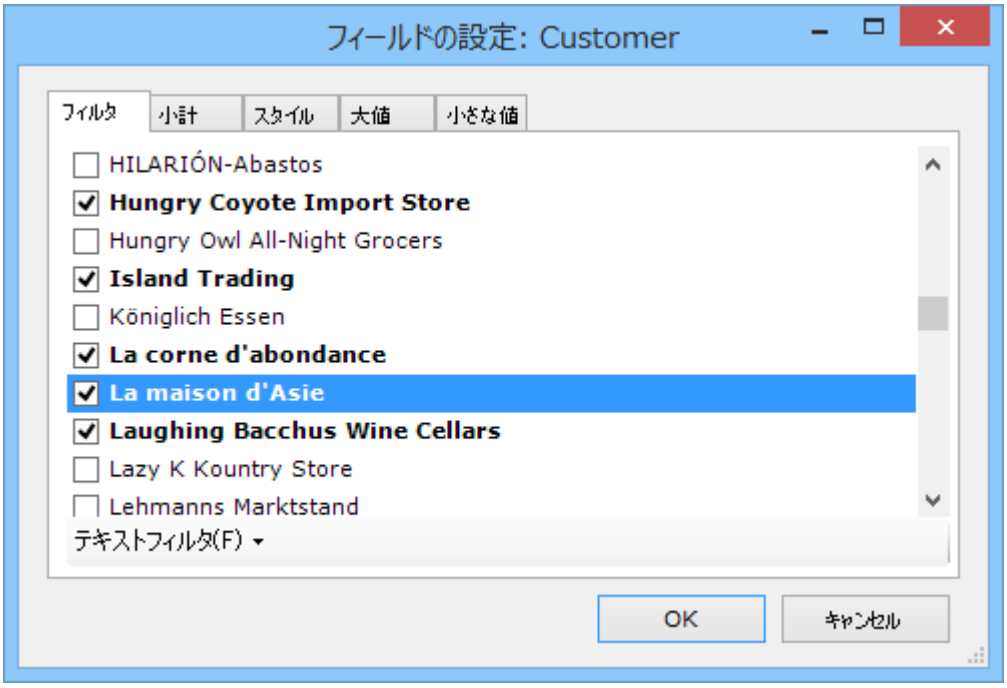
```
// 新しいデータを取得し、C1OlapPage に表示します
void sc_GetCustomerInvoicesCompleted(object sender,
SqlFilter.NorthwindDataServiceReference.GetCustomerInvoicesCompletedEventArgs e)
{
    if (e.Cancelled || e.Error != null)
    {
        _lblStatus.Text = string.Format("** Error: {0}", e.Error != null ?
e.Error.Message : "Canceled");
    }
    else
    {
        _lblStatus.Text = string.Format("Received {0} invoices ({1} customers).",
            e.Result.Count,
            _customerFilter.ShowValues.Length);
```

```
        // 更新を開始します          var olap = _c1OlapPage.OlapEngine;
        olap.BeginUpdate();
        // データソースを更新します          _invoices.Clear();
        foreach (var invoice in e.Result)
        {
            _invoices.Add(invoice);
        }
        // 更新を終了します          olap.EndUpdate();
    }
}
```

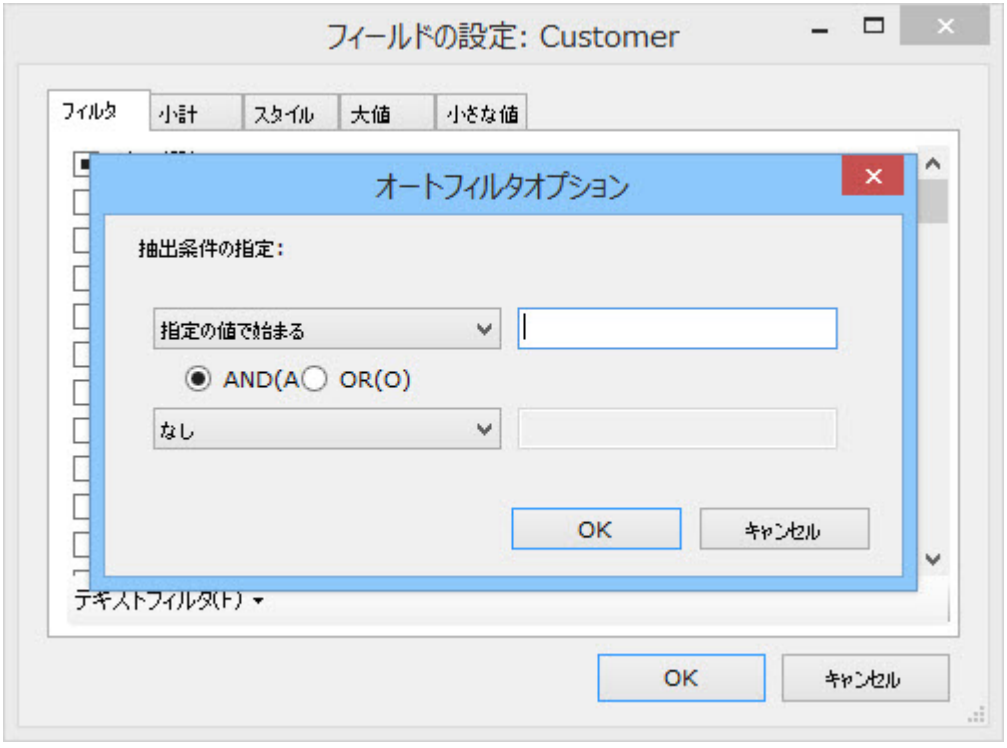
ここでアプリケーションを実行すると、"CustomerName" 設定に含まれる顧客だけがビューに含まれていることがわかります。



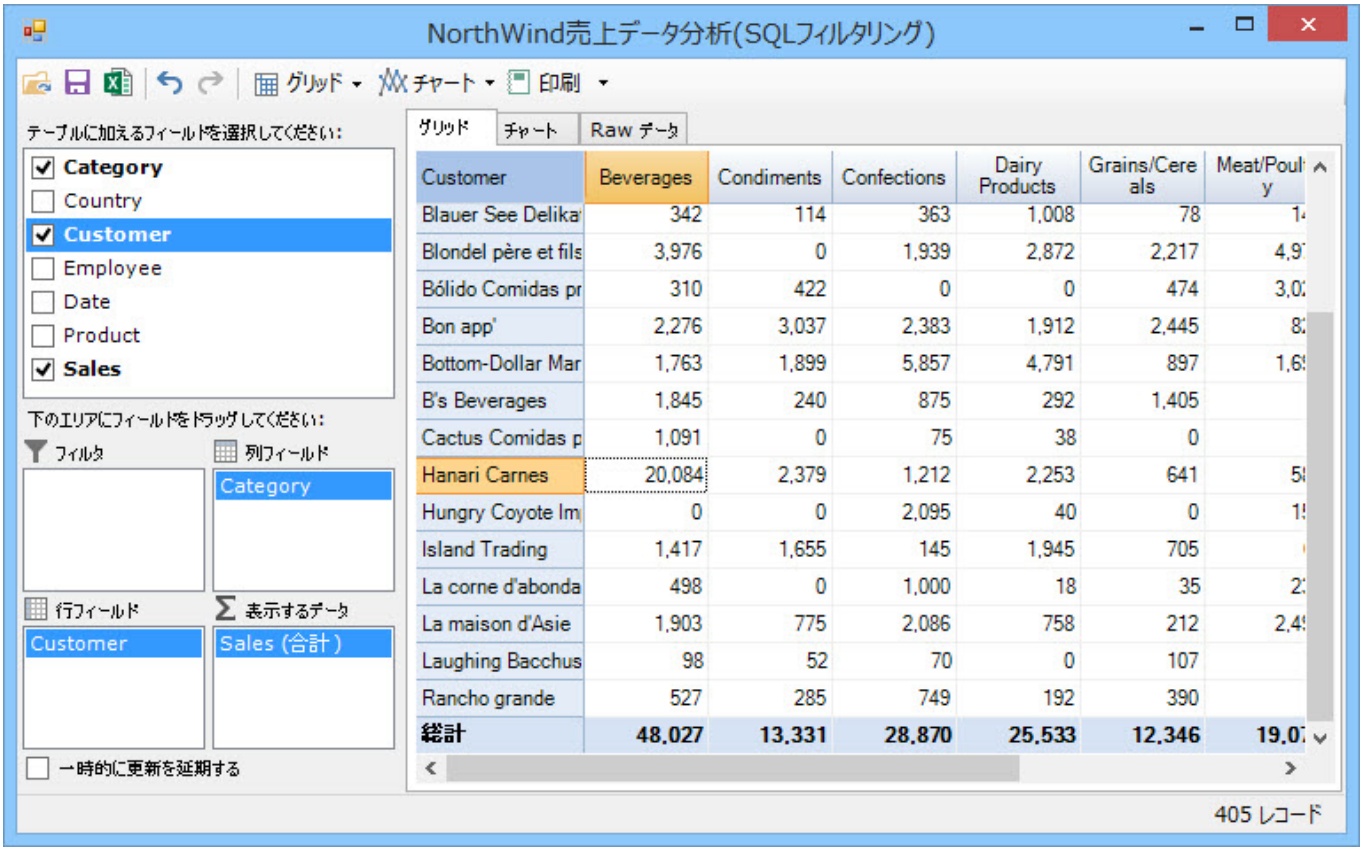
その他の顧客を表示するには、[CustomerName]フィールドをダブルクリックし、[フィールドの設定]を選択して、[フィルタ]設定を開きます。



次に、特定の顧客を選択するか、条件を定義して、フィルタを編集します。カスタムフィルタ条件を定義するには、[フィールドの設定]の[フィルタ]タブの下にある[テキストフィルタ]をクリックし、条件タイプ(次の値と等しい、次の値で始まるなど)を選択してから、次のように条件を入力します。



[OK]をクリックすると、アプリケーションによって変更が検出され、**GetData** メソッドから追加データが要求されます。新しいデータがロードされると、**C1OlapPage** によって変更が検出され、OLAP テーブルが自動的に更新されます。



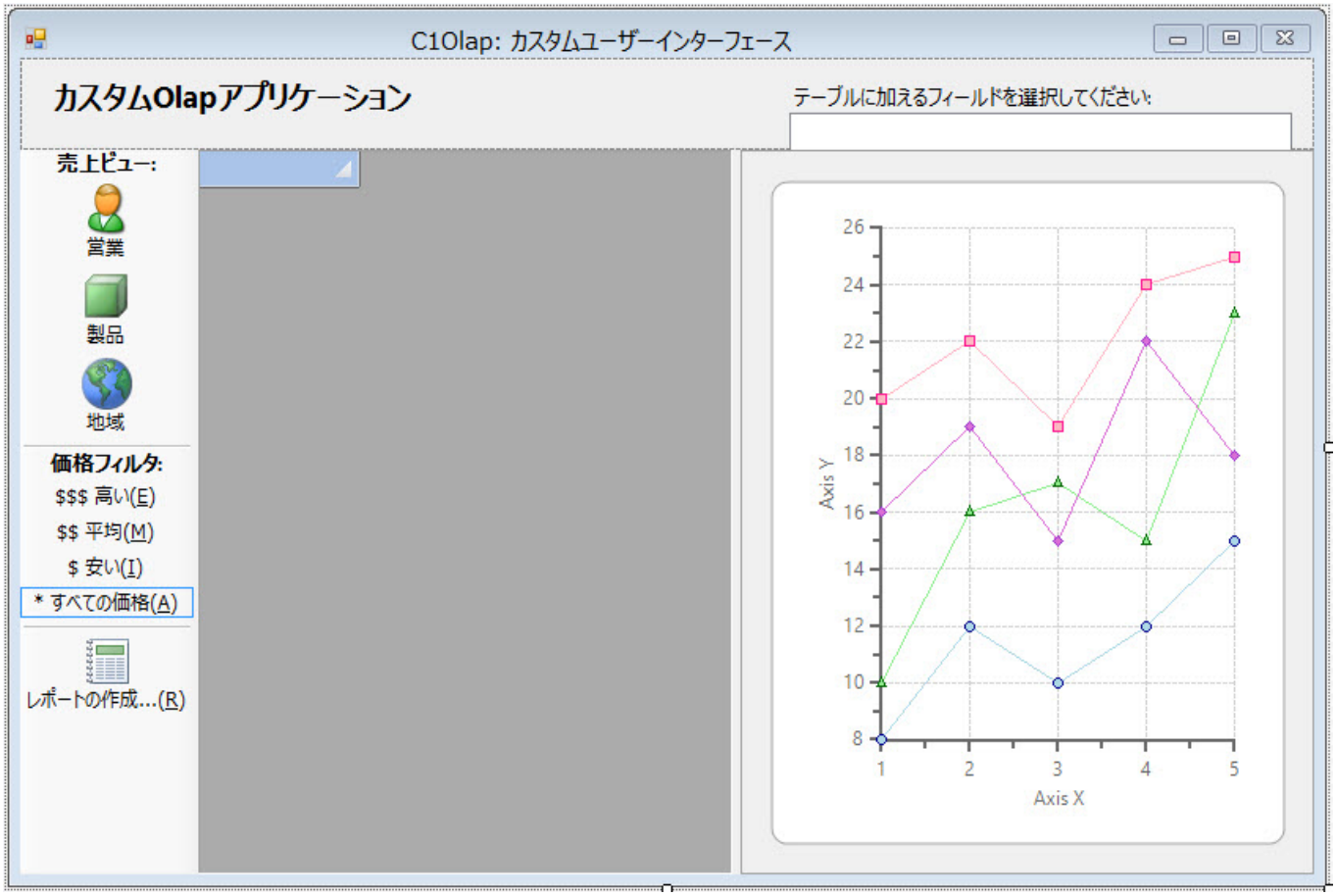
前のセクションで説明した完全な実装については、付属しているサンプル "SqlFilter" を参照してください。このサンプルを拡張して、フィルタを含む OLAP ビューをローカルストレージに格納することもできます。「ローカルストレージ内の永続的な OLAP ビュー」を参照してください。

カスタムユーザーインタフェースの作成

前のセクションのすべての例で使用した **C1OlapPage** コントロールは、すべての必要な UI を含み、コードをほとんどまたはまったく必要としません。このセクションで作成する OLAP アプリケーションでは、**C1OlapPage** を使用しません。代わりに、**C1OlapGrid**、**C1OlapChart**、およびいくつかの標準 Silverlight コントロールを使用して、完全なカスタム UI を作成します。

このアプリケーションの完全なソースコードは、**OLAP for Silverlight and WPF** でインストールされる "CustomUI" のサンプルに含まれています。

次の図は、アプリケーションをデザインビュー内に示しています。



グリッドレイアウトは、2行4列で構成されています。上端の行全体を満たす TextBlock があり、アプリケーションのタイトルを示しています。左端の列に垂直方向の StackPanel コントロールがあり、2つのグループのボタンが含まれています。上端のグループでは、定義済みの3つのビュー、営業担当者別、製品別、国別の売上高から1つを選択できます。次のグループでは、データに対して製品価格に基づいてフィルタ(高価、中程度、安価)を適用できます。

残りの列には、それぞれ空の **C1OlapGrid**、GridSplitter、および空の **C1OlapChart** があります。これらは、現在選択されているビューを表示するコントロールです。

すべてのコントロールが揃ったら、それらをすべて結合するコードを追加して、アプリケーションを動作させます。

コードで **C1OlapPanel** を宣言します。前の例では、エンドユーザーから C1OlapPanel パーツが見えます。しかし、このサンプルでは、それを背後で使用するため、ユーザーからは見えなくなります。この不可視のコントロールは、グリッドとチャートのデータソースとして使用され、また、データのフィルタ処理と要約を処理します。グリッドとチャートの両方で、その **DataSource** プロパティが **C1OlapPanel** に設定されています。

```
C1OlapPanel _olapPanel = new C1OlapPanel();
```

下記のコードは、最初に Northwind データを XML データスキーマファイルから読み込みます。**Data for Silverlight** を使用します。これは、データを読み込むために、よく知られた DataSet オブジェクトと DataTable オブジェクトを提供します。また、**Zip for Silverlight** をクライアント上で使用して、zip 形式で圧縮された XML ファイルをアンパックします。その結果として得られた **DataTable** を **C1OlapPanel.DataSource** プロパティに割り当てます。また、**C1OlapPanel** コントロールを **C1OlapGrid** に、**C1OlapChart** コントロールを **DataSource** プロパティに割り当てます。最後に、現在のビューとフィルタを初期化する2つのボタンのクリックをシミュレートします。

C#

```
public MainPage()
{
    InitializeComponent();

    var ds = new DataSet();
```

```
    var asm = Assembly.GetExecutingAssembly();
    using (var s = asm.GetManifestResourceStream("CustomUI.Resources.nwind.zip"))
    {
        var zip = new C1ZipFile(s);
        using (var zr = zip.Entries[0].OpenReader())
        {
            // データを読み込みます            ds.ReadXml(zr);
        }
    }
    // olap グリッド/チャートをパネルに連結します    _olapChart.DataSource = _olapPanel;
    _olapGrid.DataSource = _olapPanel;
    // olap パネルをデータに連結します    _olapPanel.DataSource = ds.Tables[0].DefaultView;
    // どの製品も SalesPerson ビューで開始します    _btnSalesperson_Click(this, null);
    _btnAllPrices_Click(this, null);
}
```

現在のビューを選択するボタンのイベントハンドラは、次のようになります。

```
void _btnSalesperson_Click(object sender, RoutedEventArgs e)
{
    BuildView("SalesPerson");
}
void _btnProduct_Click(object sender, RoutedEventArgs e)
{
    BuildView("ProductName");
}
void _btnCountry_Click(object sender, RoutedEventArgs e)
{
    BuildView("Country");
}
All handlers use a BuildView helper method given below:
// ボタンがクリックされた後、ビューをリビルドします
void BuildView(string fieldName)
{
    // olap エンジンを取得します    var olap = _olapPanel.OlapEngine;
    // 完了するまで更新を停止します    olap.BeginUpdate();
    // すべてのフィールドをクリアします    olap.RowFields.Clear();
    olap.ColumnFields.Clear();
    olap.ValueFields.Clear();
    // 注文日付の書式を設定して、年別にグループ化します    var f = olap.Fields["OrderDate"];
    f.Format = "yyyy";
    // ビューを構築します    olap.ColumnFields.Add("OrderDate");
    olap.RowFields.Add(fieldName);
    olap.ValueFields.Add("ExtendedPrice");
    // 更新を復元します    olap.EndUpdate();
}
```

BuildView メソッドは、**C1OlapPanel** によって提供される **C1OlapEngine** オブジェクトへの参照を取得し、直ちに **BeginUpdate** メソッドを呼び出して、新しいビューが完全に定義されるまで更新を停止します。これは、パフォーマンスの向上のために実行されます。

次に、コードで[OrderDate]フィールドの書式を "yyyy" に設定して売上高を年別にグループ化し、エンジンの **RowFields**、**ColumnFields**、および **ValueFields** コレクションをクリアし、さらに表示するフィールドを追加することによってビューをリビルドします。呼び出し元によって渡される "fieldName" パラメータには、この例のビューの間で変化するフィールドの名前だけが含まれます。

これがすべて完了すると、コードが **EndUpdate** を呼び出して、**C1OlapPanel** が出力テーブルを更新します。

アプリケーションを実行する前に、フィルタ処理を実装するコードを見ておきましょう。イベントハンドラは次のようになります。

C#

```
void _btnExpensive_Click(object sender, RoutedEventArgs e)
{
    SetPriceFilter("Expensive Products (price > $50)", 50, double.MaxValue);
}
void _btnModerate_Click(object sender, RoutedEventArgs e)
{
    SetPriceFilter("Moderately Priced Products ($20 < price < $50)", 20, 50);
}
void _btnInexpensive_Click(object sender, RoutedEventArgs e)
{
    SetPriceFilter("Inexpensive Products (price < $20)", 0, 20);
}
void _btnAllPrices_Click(object sender, RoutedEventArgs e)
{
    SetPriceFilter("All Products", 0, double.MaxValue);
}
```

すべてのハンドラは、下記に示す **SetPriceFilter** ヘルパーメソッドを使用します。

C#

```
// 製品価格にフィルタを適用します
void SetPriceFilter(string footerText, double min, double max)
{
    // olap エンジンを取得します    var olap = _olapPanel.OlapEngine;
    // 完了するまで更新を停止します    olap.BeginUpdate();
    // ビュー内の単価フィールドがアクティブであることを確認します    var field =
olap.Fields["UnitPrice"];
    olap.FilterFields.Add(field);
    // フィルタをカスタマイズします    var filter = field.Filter;
    filter.Clear();
    filter.Condition1.Operator = C1.Olap.ConditionOperator.GreaterThanOrEqualTo;
    filter.Condition1.Parameter = min;
    filter.Condition2.Operator = C1.Olap.ConditionOperator.LessThanOrEqualTo;
    filter.Condition2.Parameter = max;
    // 更新を復元します    olap.EndUpdate();
}
```

前のように、コードが **C1OlapEngine** に対する参照を取得して、直ちに **BeginUpdate** を呼び出します。

次に、データのフィルタ処理に使用される[UnitPrice]フィールドに対する参照を取得します。[UnitPrice]フィールドがエンジンの **FilterFields** コレクションに追加され、フィルタが現在のビューに適用されます。

この細部の処理は重要です。フィールドがどのビューコレクション(**RowFields**、**ColumnFields**、**ValueFields**、**FilterFields**)にも含まれない場合、そのフィールドがビューに含まれることはなくなり、また、ビューはその **Filter** プロパティによる一切の影響を受けなくなります。

コードは、ビューに含まれる必要のある値の範囲を指定する2つの条件を設定することにより、[UnitPrice]フィールドの **Filter** プロパティの設定に進みます。範囲は、"min" パラメータと "max" パラメータによって定義されます。条件を使用する代わりに、包める値のリストを提供することもできます。通常、数値を扱う場合には条件の方が便利であり、文字列値や列挙ではリストの方が便利です。

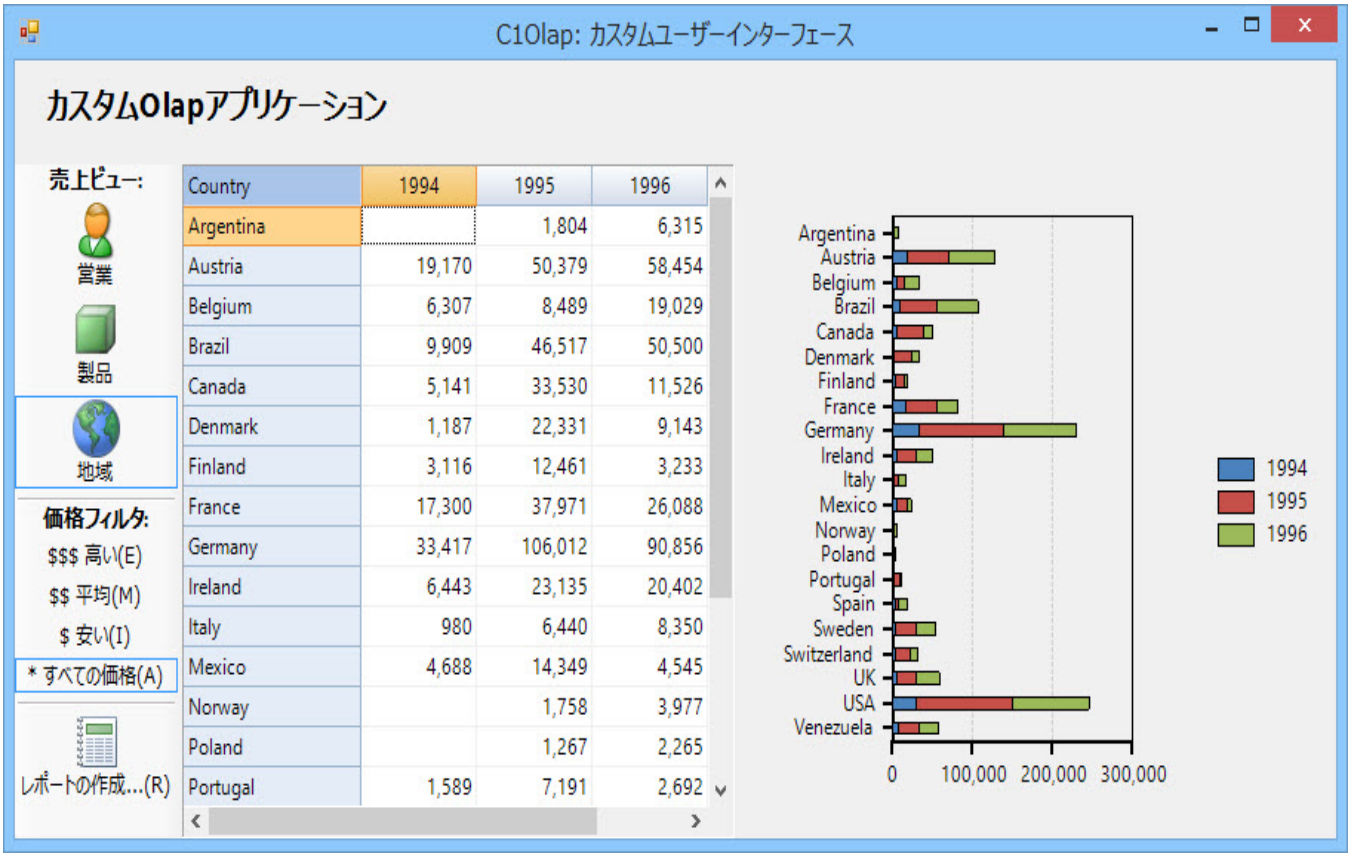
最後に、コードが **EndUpdate** を呼び出します。

最後に、**C1OlapGrid** 内の列をソートするときは必ず **C1OlapChart** を更新します。これにより、データ値が同じ順序で表示されます。

C#

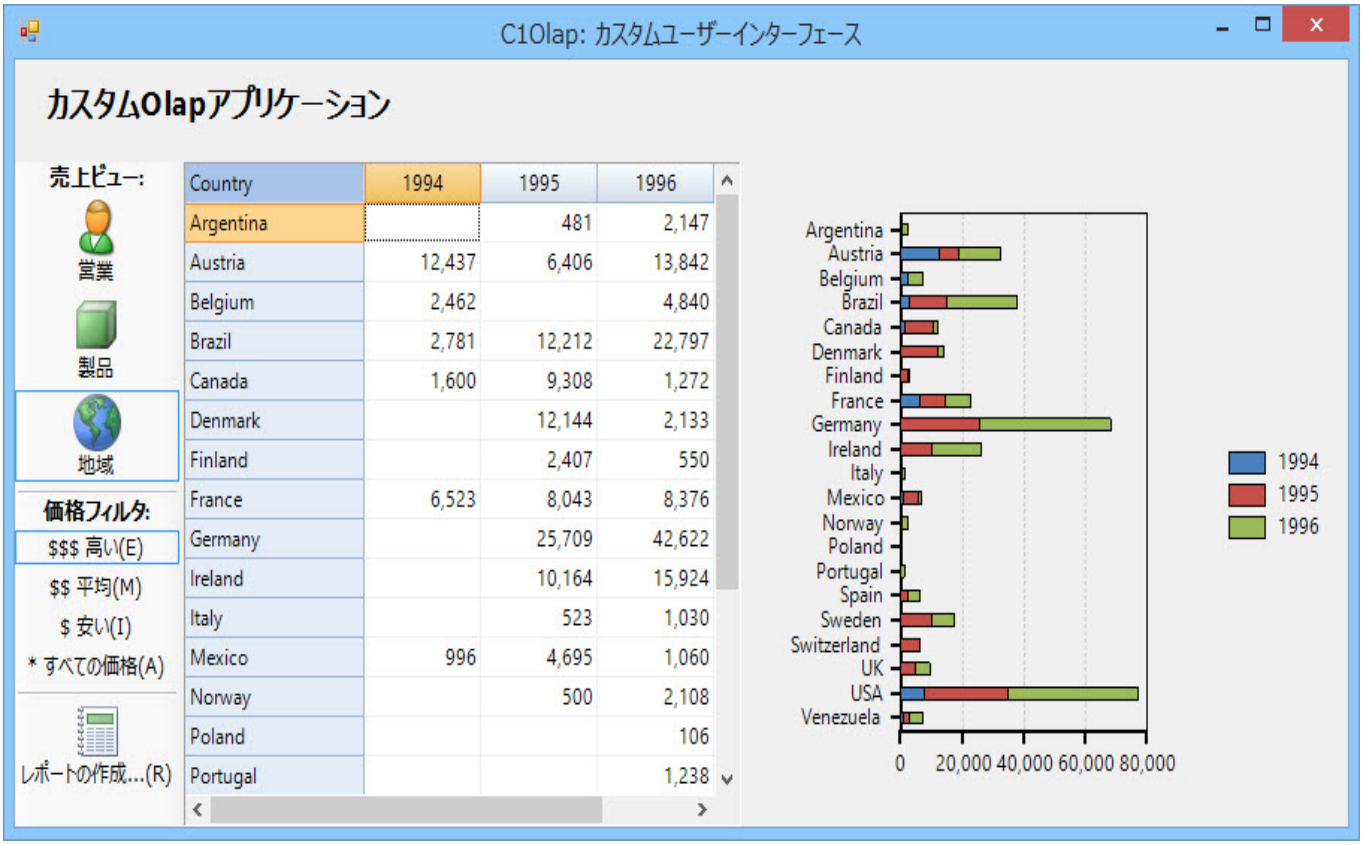
```
void _olapGrid_SortedColumn(object sender, C1.Silverlight.FlexGrid.CellRangeEventArgs
e)
{
    _olapChart.UpdateChart();
}
```

これで、アプリケーションを使用する準備が整います。次のように実行して、アプリケーションのさまざまなビューやフィルタ処理機能をテストできます。



このビューでは、すべての製品の売上高が年別および国別にグループ化して表示されます。300,000 ドル弱の値がチャートにどのように表示されているか注目してください。

[$$$ 高額]ボタンをクリックすると、フィルタが適用されて、直ちにビューが変化します。今度は、80,000 ドル弱の値がチャートにどのように表示されているか注目してください。高額の値が売上高の3分の1を占めています。



XAML クイックリファレンス

このトピックでは、**OLAP for WPF/Silverlight** コントロールの作成に使用される XAML の概要を提供します。

開発を開始するには、ルート要素タグに **c1** 名前空間宣言を追加します。

```
xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml"
```

次にその XAML を示します。

XAML

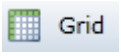
```
<!-- olap page -->
<c1:C1OlapPageHorizontalAlignment="Left"Margin="28,12,0,0"Name="c1OlapPage1"VerticalAlignment="Top"Height="426"Width="528"/>
<!-- olap panel -->
<c1:C1OlapPanelHorizontalAlignment="Left"Margin="26,11,0,0"Name="c1OlapPanel1"VerticalAlignment="Top"Height="307"Width="393"/>
<!-- olap grid -->
<BorderStyle="{StaticResource_border}"Grid.Row="1"Grid.Column="1">
<c1:C1OlapGridx:Name="_olapGrid"Margin="4"SortedColumn="_olapGrid_SortedColumn"/>
</Border>
<!-- olap chart -->
<BorderStyle="{StaticResource_border}"Grid.Row="1"Grid.Column="3">
<c1:C1OlapChartx:Name="_olapChart"Margin="4"/>
</Border>
```

OLAP for WPF/Silverlight の設計時サポート

以下のセクションでは、**OLAP for Silverlight and WPF** 設計時環境を使用して、コントロールを設定する方法について説明します。

C1OlapPage ツールストリップの使用

C1OlapPage コントロールには、.xml ファイルとしての **C1OlapPage** のロードまたは保存、グリッドまたはチャートでのデータの保存、または、レポートの設定および印刷に使用できるツールストリップがあります。次のテーブルでは、ツールストリップ内のボタンについて説明します。

ボタン		説明
ロード		以前に保存した **C1Olap** ビュー定義ファイル(*.olapx)を **C1OlapPage** にロードできます。
保存		**C1Olap** ビュー定義ファイル(*.olapx)を保存できます。
エクスポート		**C1OlapGrid** をさまざまな形式(.xlsx、.xls、.csv、.txt など)にエクスポートできます。
元に戻す		**[元に戻す]**ボタンをクリックすると、**C1OlapPage** で最後に実行されたアクションが取り消されます。
やり直し		**[やり直し]**ボタンをクリックすると、**[元に戻す]**ボタンを使用して取り消された最後のアクションが実行されます。
グリッド		**C1OlapGrid** に表示する列および行を選択できます。
グラフ		データの表示に使用するチャートをカスタマイズできます。チャートタイプ、パレットまたはテーマ、タイトルを表示するかどうか、チャートを積み重ねるかどうか、およびグリッド線を表示するかどうかを決定できます。
レポート		レポートの各ページのヘッダーまたはフッターの指定、レポート、OLAP グリッド、チャート、または生データグリッドに何を含めるかの決定、ページレイアウト(方向、用紙サイズ、マージンなど)の指定、印刷前のレポートのプレビュー、レポートの印刷を行うことができます。

[グリッド]メニューの使用

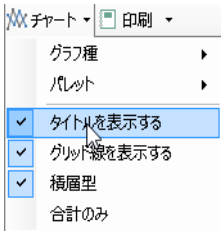
[グリッド]メニューには、3つのオプションがあります。

合計行	**[総計]**、**[小計]**、**[なし]**の中から選択できます。
合計列	**[総計]**、**[小計]**、**[なし]**の中から選択できます。
ゼロを表示する	オンにすると、グリッド内のゼロを含むセルが表示されます。

これらの項目のいずれかをオフにするだけで、合計行、合計列、またはグリッド内のゼロが非表示になります。

[チャート]メニューの使用

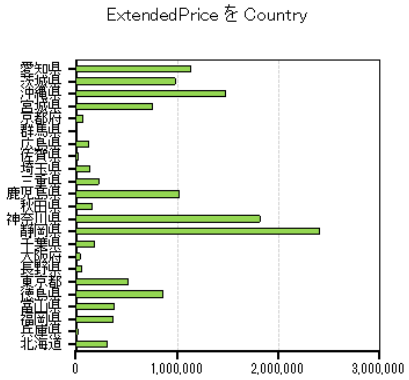
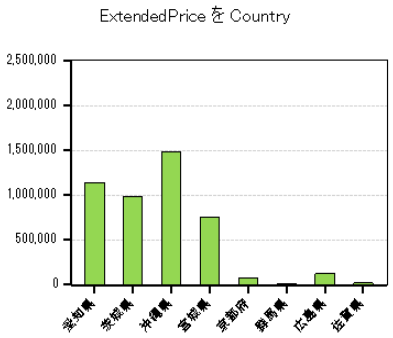
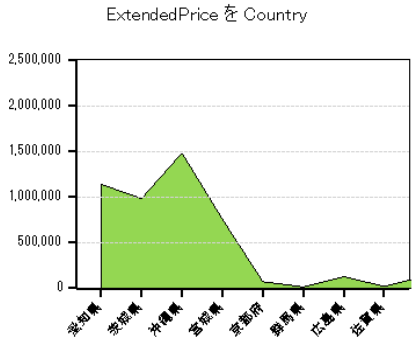
[チャート]メニューからは、チャートタイプ、パレット、チャートの上にチャートタイトルを表示するかどうか、チャートのグリッド線を表示するかどうか、積層グラフを表示するかどうか、および合計だけを表示するかどうかを決定できます。

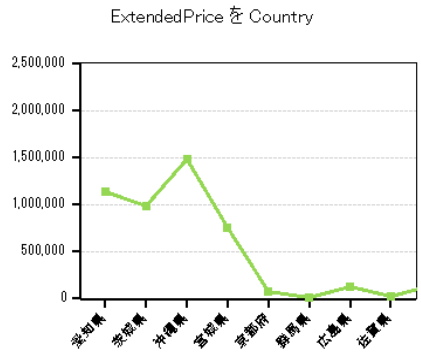
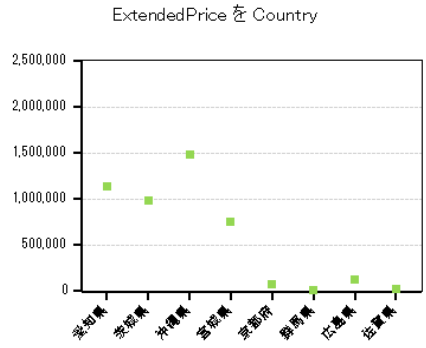


チャートタイプ	**[チャートタイプ]**をクリックして、次に示す5つの一般的なチャートタイプの中から選択します。
パレット	**[パレット]**をクリックし、22 のパレットオプションの中から選択して、チャートおよび凡例項目の色を定義します。以下の「**パレット**」トピックのオプションを参照してください。
タイトルの表示	選択すると、チャートの上にタイトルが表示されます。
積層	選択すると、データが積み重ねられたチャートビューが作成されます。
グリッド線の表示	選択すると、チャートにグリッド線が表示されます。
合計のみ	選択すると、データソースの列ごとに1つの系列ではなく、合計だけが表示されます。

チャートタイプ

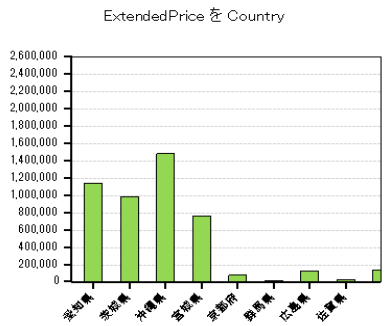
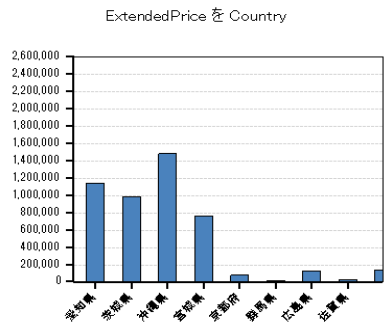
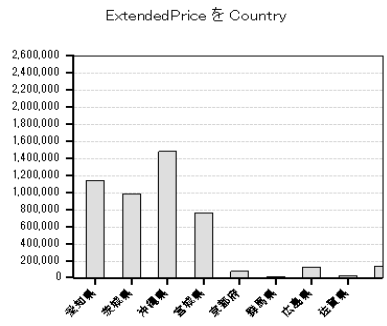
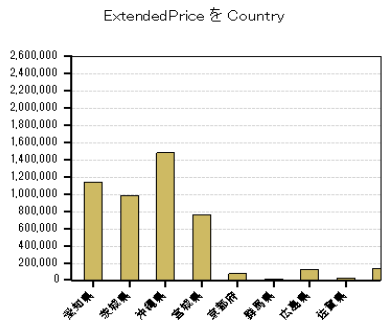
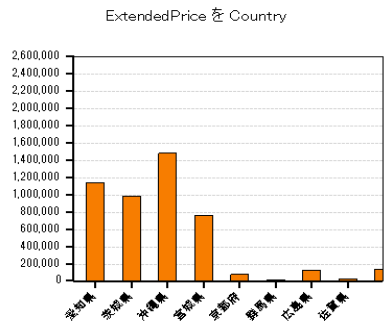
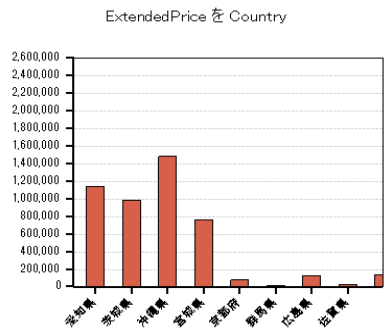
OLAP for WPF/Silverlight には、最も一般的な5つのチャートタイプが用意されています。次の表は、タイプごとの例を示します。

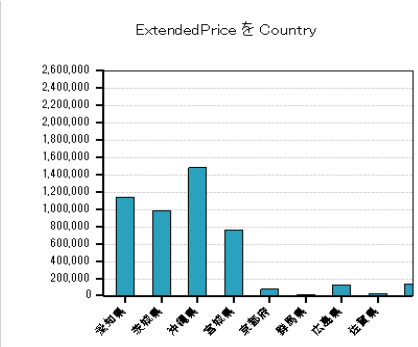
Bar	
Column	
Area	

Line	
散布	

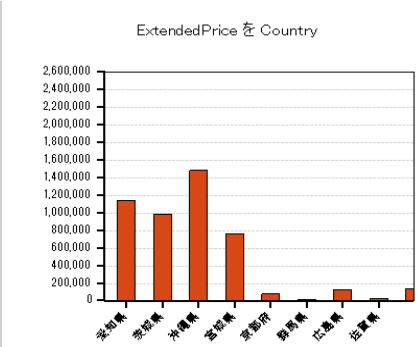
パレット

C1OlapChart パレットは、チャートおよび凡例項目の色を定義する 22 のオプションで構成されます。次の表に、各パレットオプションの色を示します。

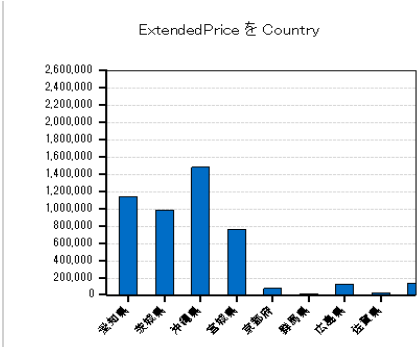
Standard（標準）	**Office（Office）**	**GrayScale**
		
Apex（ひらめき）	**Aspect（シック）**	**Civic（クール）**
		
Concourse（ビジネス）	**Equity（ジャパネスク）**	**Flow（リゾート）**



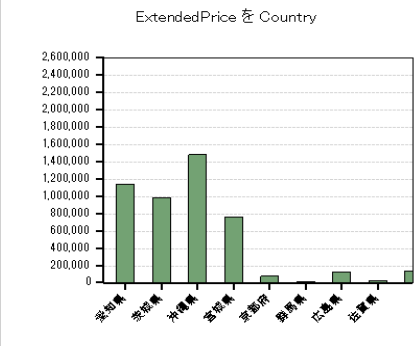
Foundry（エコロジー）



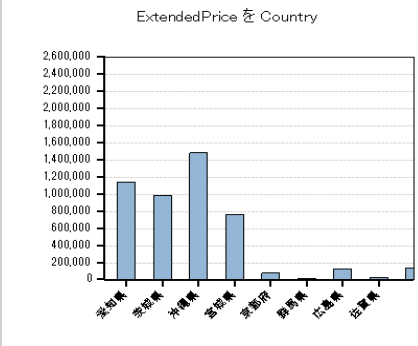
Median（デザート）



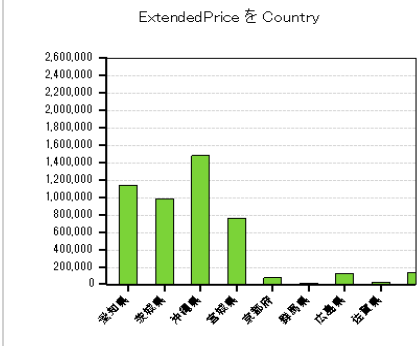
Metro



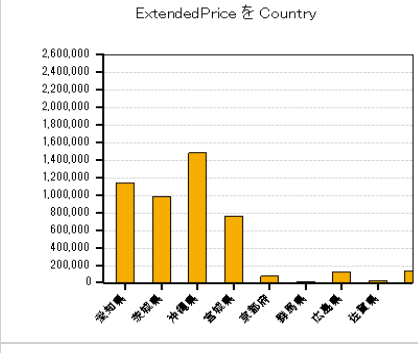
Module（モジュール）



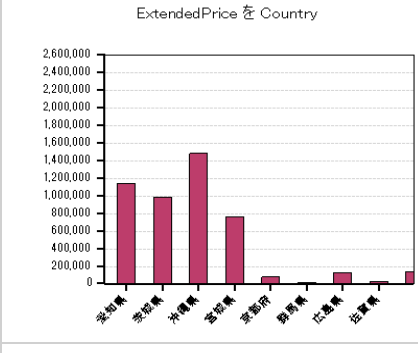
Opulent（キュート）



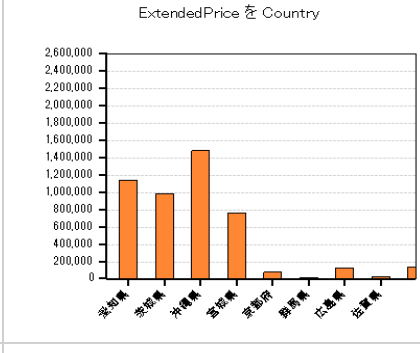
Oriel（スパイス）



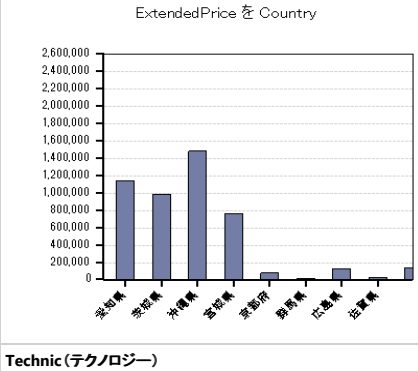
Origin（アース）



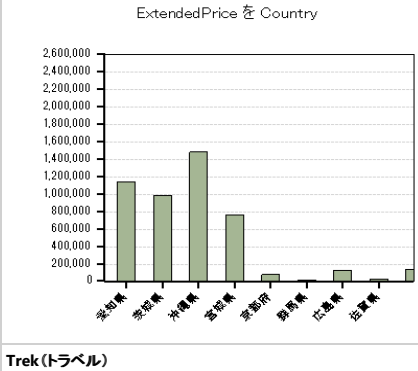
Paper（ペーパー）



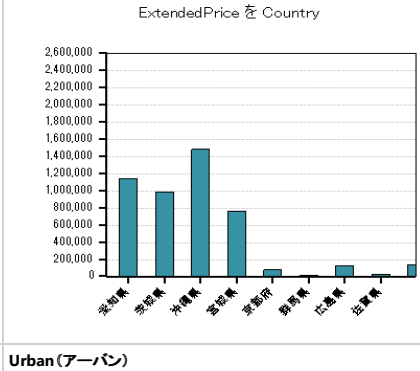
Solstice（フレッシュ）



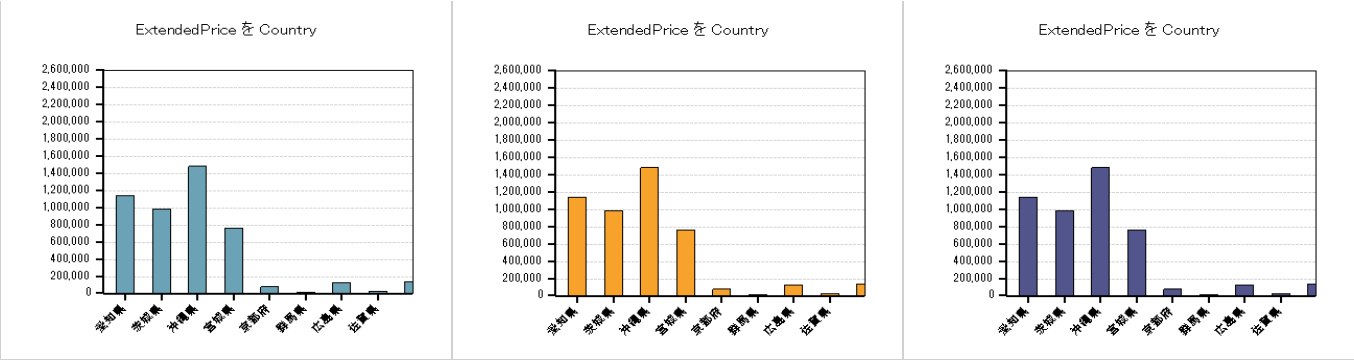
Technic（テクノロジー）



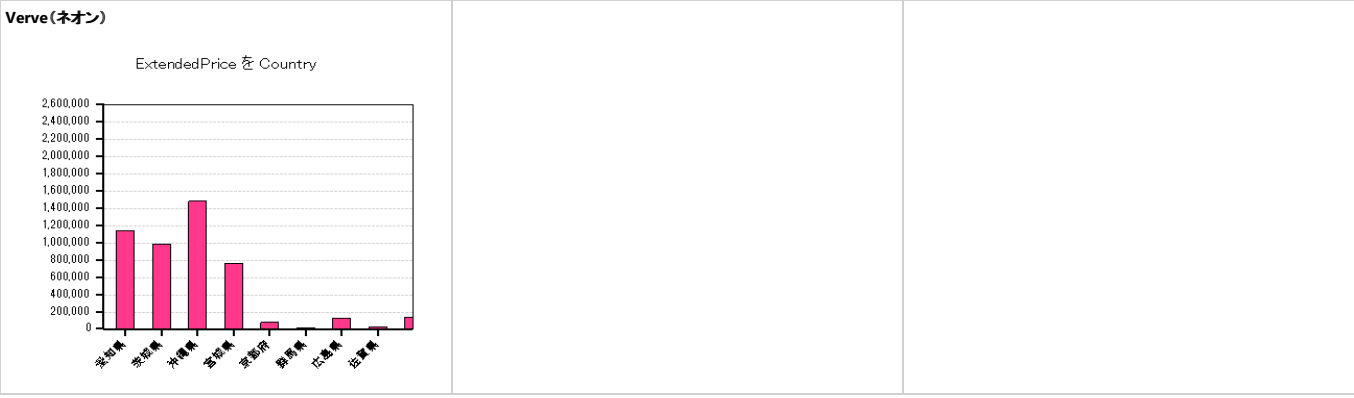
Trek（トラベル）



Urban（アーバン）

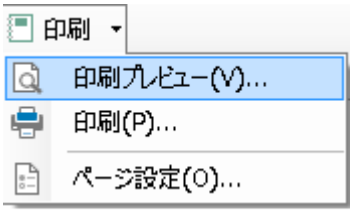


Verve（ネオン）



[レポート]メニューの使用

[レポート]メニューからは、レポートのプレビューまたは印刷、レポートのページの設定、ヘッダーまたはフッター（あるいはその両方）の追加、レポートに表示する項目の指定を行うことができます。

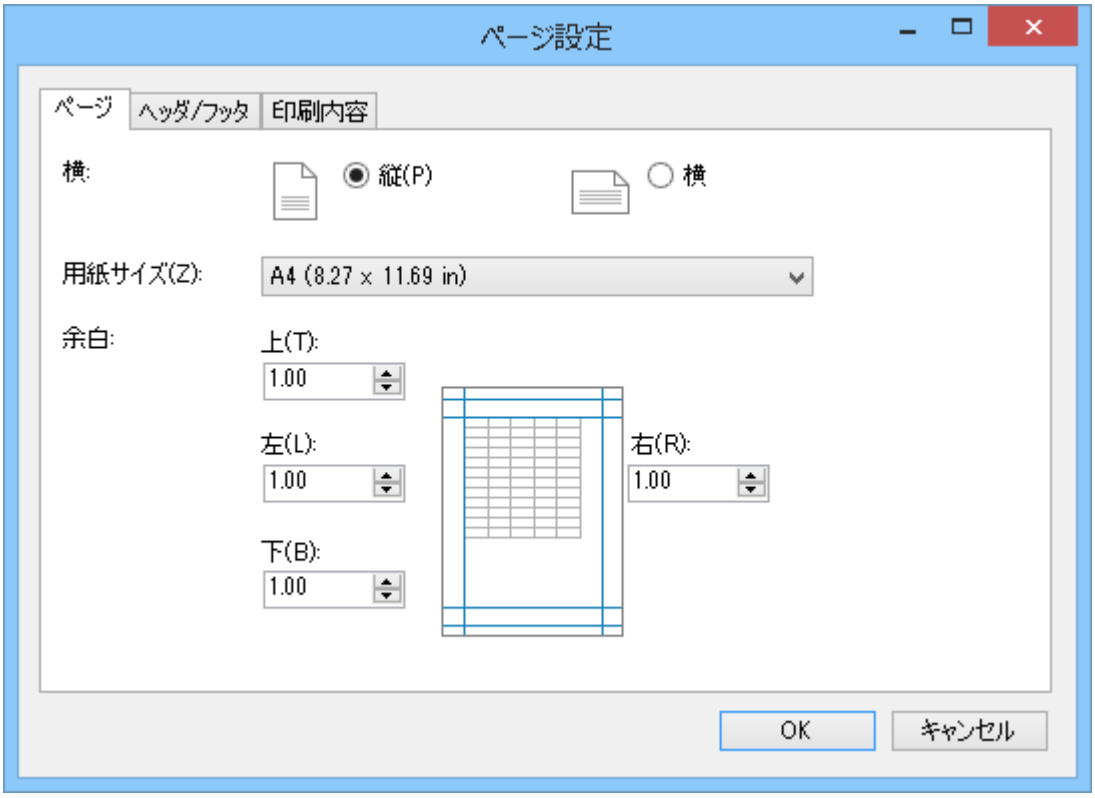


印刷	**[印刷]**をクリックして、**C1OlapGrid**、**C1OlapChart**、またはその両方を印刷します。
オプション	**[オプション]**をクリックして、**[ドキュメントオプション]**ダイアログボックスを開きます

ドキュメントオプション

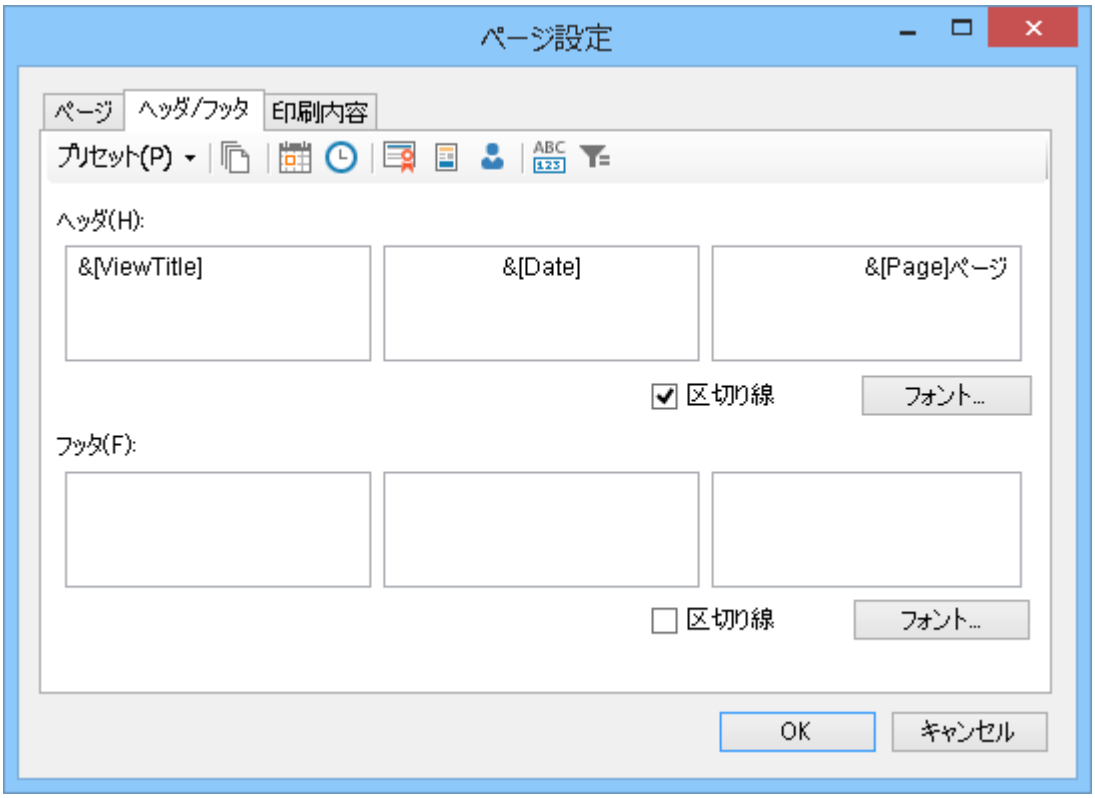
[ページ]タブ

[ページ]タブでは、マージンおよびパディングを指定できます。



[ヘッダー/フッター]タブ

[ヘッダー/フッター]タブでは、レポートの各ページにヘッダーまたはフッター(あるいはその両方)を追加できます。



ツールバーのボタンの1つをクリックして、フィールドをヘッダーまたはフッターに挿入します。

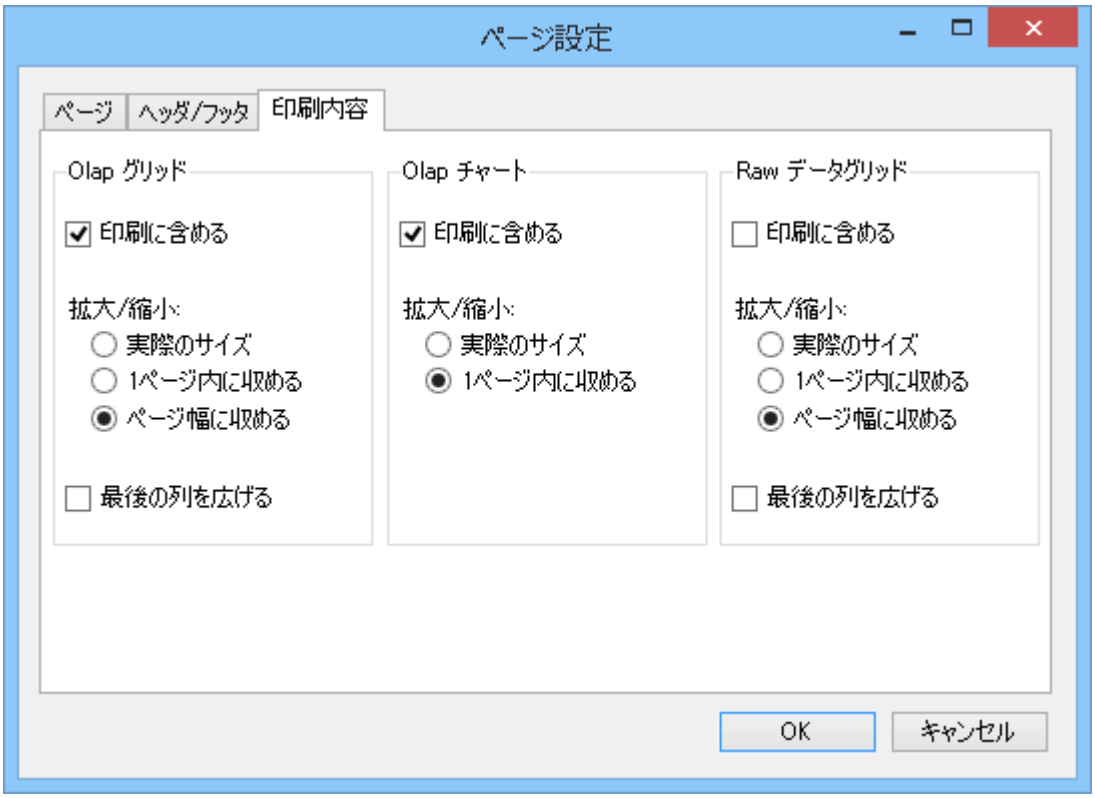
ボタン	フィールド

ページ番号	&[Page]
合計ページ数	&[PageCount]
現在の日付	&[Date]
現在の時刻	&[Time]
タイトル	&[ViewTitle]

[区切り]ボックスをオンにすると、ヘッダーの下またはフッターの上に分割線が表示されます。フォント、スタイル、サイズ、またはエフェクトを変更するには、**[フォント]**ボタンをクリックします。

[レポートコンテンツ]タブ

[レポートコンテンツ]タブでは、レポートに OLAP グリッド、OLAP グラフ、または生データグリッドを含めるかどうかを決定します。また、必要に応じて項目を拡大縮小することもできます。



OLAP キューブ

Olap for WPF では、**Microsoft SQL Server Analysis Services**（**SSAS**）のように OLAP データソースに接続できます。オンラインキューブに接続することも、実行時にローカルキューブをアタッチすることもできます。**C1Olap** は、**Analysis Services** と **SQL Server 2008**、**2012**、および **2014** と共に機能します。

メモ: キューブサポートは WPF のみで提供されます。

OLAP キューブへの接続

キューブに接続するには、**C1OlapPanel.ConnectCube** メソッドを使用する必要があります。このメソッドは、2つのパラメータを受け取ります。Analysis Services がインストールされた SQL Server への接続文字列と、キューブの名前です。実行時に例外をキャッチすることで、ユーザーにエラーをレポートできます。ここでは、キューブに接続するための完全なコード例を示します。

C#

```
// キューブに接続します
stringconnectionString = @"Data Source=myServerAddress;Catalog=myDataBase";
stringcubeName = "Adventure Works";

try
{
    _c1OlapPage.OlapPanel.ConnectCube(cubeName, connectionString);
}
catch(Exceptionex)
{
    MessageBox.Show(ex.Message);
}
```

接続文字列には、**Data Source** および **Initial Catalog** を設定する必要があります。複数の Microsoft OLE DB Provider for OLAP をインストールしてある場合は、接続文字列でプロバイダのバージョンを指定できます。たとえば、**Provider** を **MSOLAP** に設定すると、システムにインストールされている **OLE DB for OLAP** の最新バージョンが使用されます。

例:

C#

```
Provider=MSOLAP;Data Source=myServerAddress;Initial Catalog=myDataBase;
```

メモ: カスタム UI を作成している場合、または **C1OlapPage** コントロールを使用しない場合は、**C1OlapPanel** コントロールと、その同じ **C1OlapPanel.ConnectCube** メソッドを使用できます。

ローカルキューブファイルのロード

C1Olap は、ローカルキューブファイル（.cub）と共に使用できます。たとえば、キューブファイルを **Data** という名前のプロジェ

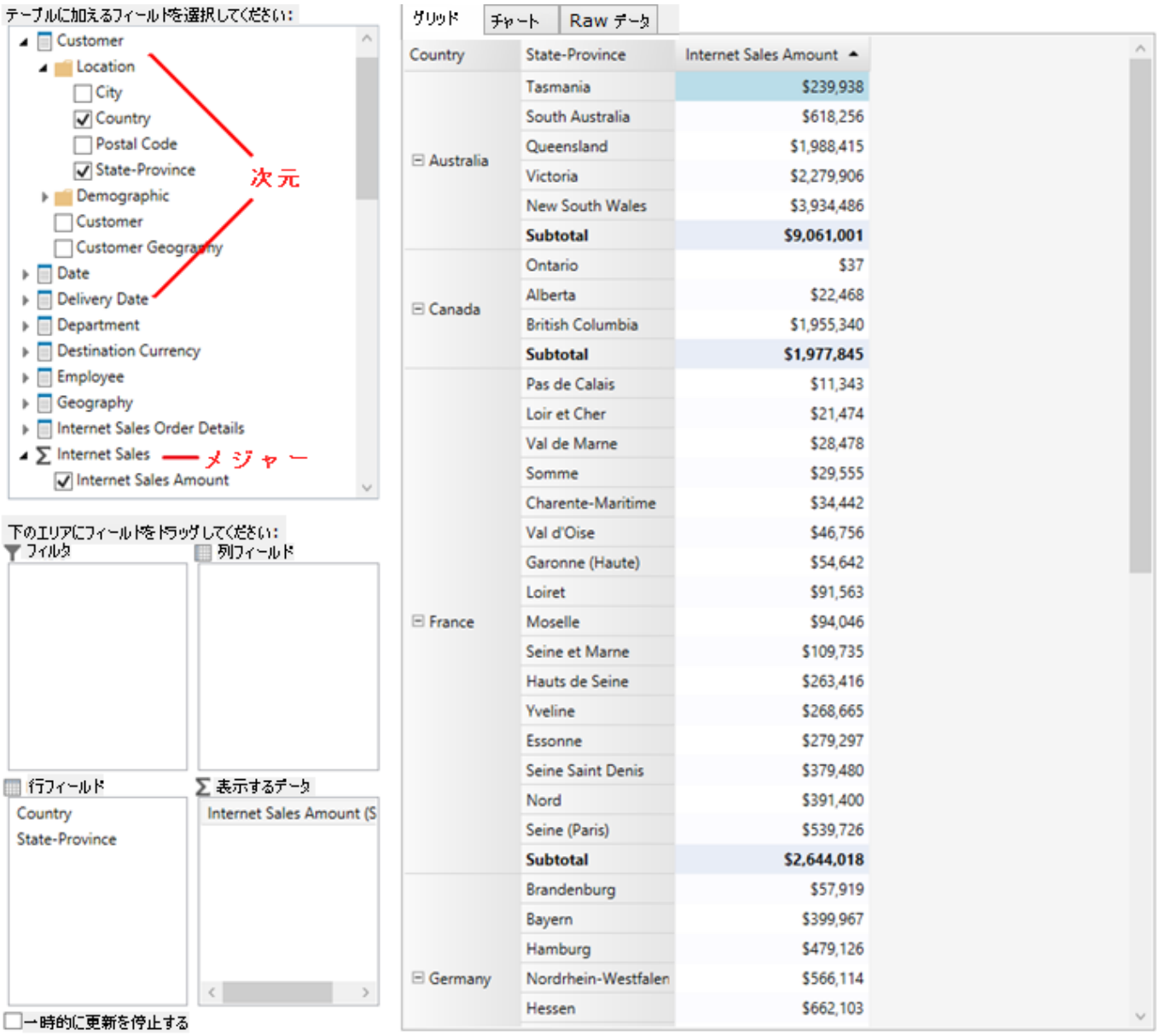
クト内のディレクトリに配置してある場合、この接続文字列は次のようになります。

C#

```
stringconnectionString = @"Data Source="+
System.AppDomain.CurrentDomain.BaseDirectory +
@"\Data\LocalCube.cub;Provider=msolap";
stringcubeName = "LocalCube";
c1OlapPage1.OlapPanel.ConnectCube(cubeName, connectionString);
```

キューブデータソースの使用

ユーザーは、実行時に、通常のデータセットからの場合とほぼ同じようにして、キューブデータからレポートを構築できます。主な違いとして、キューブデータセットは、**C1OlapPanel** コントロールではツリーによって表されます。各ノードは、次元エンティティまたはメジャー用のオブジェクトを表します。レポートに追加できるすべてのフィールドには、チェックボックスが隣に表示されます。合計記号(∑)で表されるオブジェクトはメジャーであり、**Values** コレクションに追加できます。エンティティのフィールドは、**Rows** または **Columns** コレクションに追加できます。



OLAP for WPF/Silverlight のタスク別ヘルプ

タスク別のヘルプは、ユーザーの皆様が Visual Studio .NET のプログラミングに精通しており、連結および非連結コントロールを使用する一般的な方法を理解していることを前提としています。各トピックは、**OLAP for WPF および Silverlight** 製品を使用して特定のタスクを実行するためのソリューションを提供します。ヘルプに示される手順に従って作業を進めるだけで、さまざまな **OLAP for WPF/Silverlight** の機能を具体的に示すプロジェクトを作成できます。

また、タスク別の各ヘルプトピックは、新しい WPF または Silverlight プロジェクトが既に作成されていることを前提としています。

データソースへの C1OlapPage または C1OlapPanel の連結

C1OlapPage.DataSource または **C1OlapPanel.**DataSource プロパティを使用することにより、**C1OlapPage** または **C1Olap** をデータソースに簡単に連結できます。この例では、Northwind 製品データを XML データスキーマファイルからロードします。**nwind.zip** は、**OlapQuickStart** サンプルでインストールされることに注意してください。**ComponentOne Data** を使用します。これは、データを読み込むために、よく知られた DataSet オブジェクトと DataTable オブジェクトを提供します。また、**C1Zip** をクライアント上で使用して、zip 形式で圧縮された XML ファイルをアンパックします。

C1OlapPage コントロールを連結するには、次の手順に従います。

1. 次のコードを追加します。

C#

```
// 埋め込まれた zip リソースからデータを読み込みます
var ds = new DataSet();
var asm = Assembly.GetExecutingAssembly();
using (var s = asm.GetManifestResourceStream("OlapQuickStart.nwind.zip"))
{
var zip = new C1ZipFile(s);
using (var zr = zip.Entries[0].OpenReader())
{
// データを読み込みます
ds.ReadXml(zr);
}
}
```

2. **C1OlapPage** コントロールに対して **C1OlapPage.DataSource** プロパティを設定します。このコントロールでは、任意のデータ連結方法を使用できます。

C#

```
// olap ページをデータに連結します
_c1OlapPage.DataSource = ds.Tables[0].DefaultView;
```

C1OlapPanel へのC1OlapChart の連結

データソースに連結された **C1OlapPanel** に連結することにより、**C1OlapChart** コントロールを挿入できます。このトピックでは、フォームに **C1OlapPanel** コントロールが連結されていることを想定しています。

C1OlapChart の **C1OlapChart.DataSource** プロパティに、OLAP データを提供する **C1OlapPanel** を設定します。

C1OlapPanel への C1OlapGrid の連結

データソースに連結された **C1OlapPanel** に連結することにより、**C1OlapGrid** コントロールを挿入できます。このトピックで

は、フォームに **C1OlapPanel** コントロールが連結されていることを想定しています。

C1OlapGrid の **C1OlapGrid.DataSource** プロパティに、OLAP データを提供する **C1OlapPanel** を設定します。

データビューからフィールドを削除

C1OlapPanel コントロールまたは **C1OlapPage** コントロールの **C1OlapPanel** 領域では、フィールドが **C1OlapGrid** または **C1OlapChart** データビューに表示されないように、フィールド全体をフィルタ処理することができます。これは、実行時に行えます。

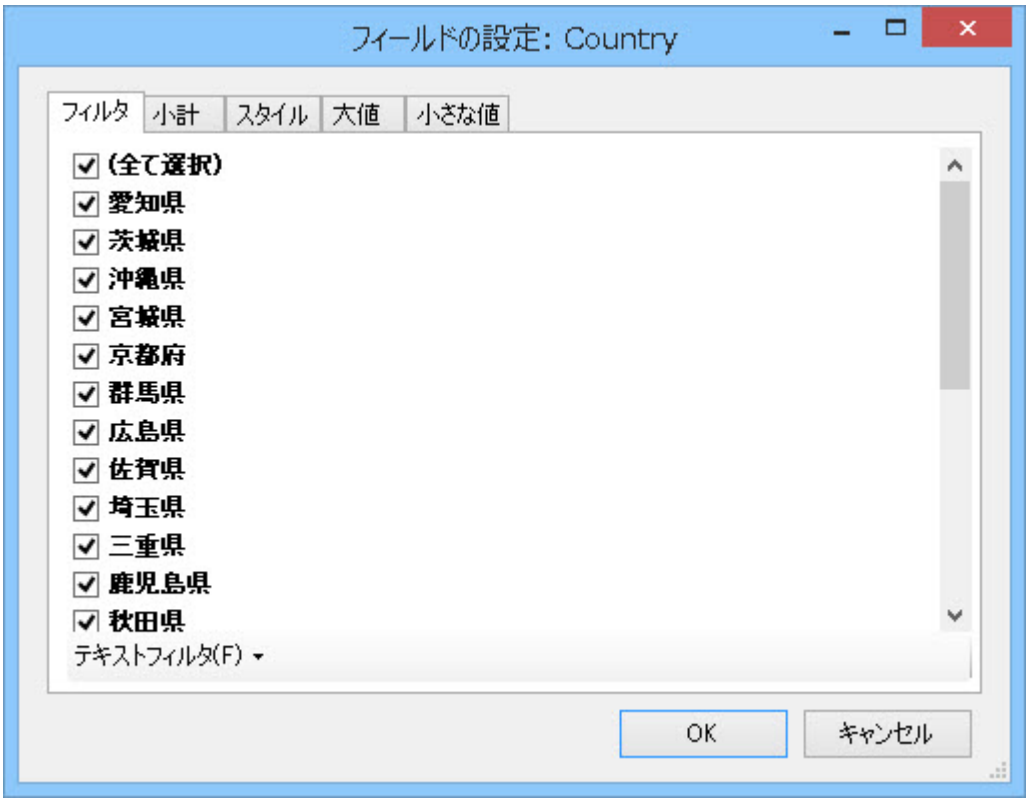
1. パネルの**[次のボックス間でフィールドをドラッグしてください]**セクションで、ビューから除外するフィールドを選択します。
2. パネルの**[フィルタ]**領域にそのフィールドをドラッグします。このフィールド内のデータが **C1OlapGrid** または **C1OlapChart** データビューから削除されます。

フィールド内のデータをフィルタ処理

C1OlapPanel コントロール、または **C1OlapPage** コントロールの **C1OlapPanel** 領域では、実行時にパネルの**[次のボックス間でフィールドをドラッグしてください]**セクションで、フィールドのデータをフィルタ処理できます。各フィールドには2つのフィルタがあります。リスト内の特定の値をチェックできる値フィルタと、1つまたは2つの基準を指定できる範囲フィルタです。この2つのフィルタは独立しており、値が OLAP テーブルに表示されるためには、この2つのフィルタを通過する必要があります。

値フィルタの使用

1. **[フィルタ]**、**[列フィールド]**、**[行フィールド]**、または**[値]**領域で、1つのフィールドを右クリックします。
2. コンテキストメニューの**[フィールドの設定]**をクリックします。**[フィールドの設定]**ダイアログボックスが開きます。
3. **[フィルタ]**タブをクリックします。これは値フィルタです。OLAP テーブルに表示しないフィールドについては、選択をクリアできます。



テーブルに表示するフィールドを選択したら、ウィンドウの下部にある**[テキストフィルタ]**または**[数値フィルタ]**ボタンをクリックして、範囲フィルタを指定できます。

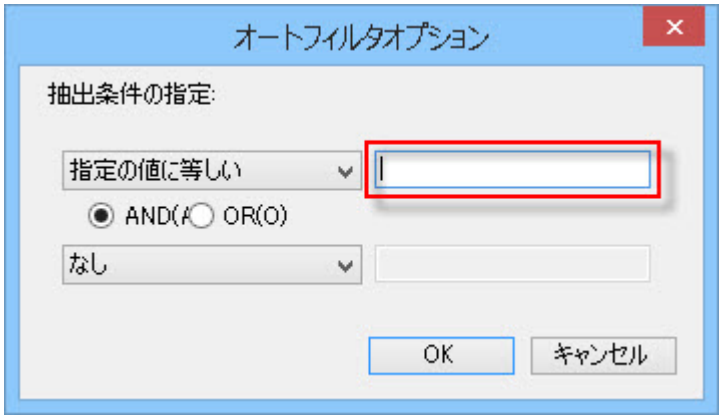
メモ: フィルタ処理するフィールドに数値データが含まれている場合は、**[テキストフィルタ]**の代わりに**[数値フィルタ]**が表示されます。

範囲フィルタの使用

1. **[フィルタ]**、**[列フィールド]**、**[行フィールド]**、または**[値]**領域で、1つのフィールドを右クリックします。
2. コンテキストメニューの**[フィールドの設定]**をクリックします。**[フィールドの設定]**ダイアログボックスが開きます。
3. **[フィルタ]**タブをクリックし、必要に応じて値フィルタを指定します。OLAP テーブルに表示しないフィールドについては、選択をクリアできます。
4. **[テキストフィルタ]**または**[数値フィルタ]**ボタンをクリックして、範囲フィルタを設定します。
5. 次の項目のうちの1つを選択します。

フィルタのクリア	すべてのフィルタ設定をクリアします。
次の値と等しい	**[カスタムフィルタ]**ダイアログボックスを開きます。ここでは、指定された値と等しい項目が表示されるフィルタを作成できます。
次の値に等しくない	**[カスタムフィルタ]**ダイアログボックスを開きます。ここでは、指定された値と等しくない項目が表示されるフィルタを作成できます。
次の値で始まる	**[カスタムフィルタ]**ダイアログボックスを開きます。ここでは、指定された値で開始される項目が表示されるフィルタを作成できます。
次の値で終わる	**[カスタムフィルタ]**ダイアログボックスを開きます。ここでは、指定された値で終了される項目が表示されるフィルタを作成できます。
次の値を含む	**[カスタムフィルタ]**ダイアログボックスを開きます。ここでは、指定された値が含まれる項目が表示されるフィルタを作成できます。
次の値を含まない	**[カスタムフィルタ]**ダイアログボックスを開きます。ここでは、指定された値が含まれない項目が表示されるフィルタを作成できます。
カスタムフィルタ	**[カスタムフィルタ]**ダイアログボックスを開きます。ここでは、独自の条件を使用したフィルタを作成できます。

6. 最初の空白のテキストボックスに、フィルタ処理する項目を追加します。



7. **[および]**または**[または]**を選択します。
8. 必要に応じて、2つめのフィルタ条件を追加します。**[なし]**以外のオプションを選択すると、2つめのテキストボックスがアクティブになり、項目を入力できます。
9. **[OK]**をクリックして**[カスタムフィルタ]**ダイアログボックスを閉じ、もう一度**[OK]**をクリックして**[フィールドの設定]**ダイアログボックスを閉じます。

小計機能の指定

データのカスタムビューを作成する際、列または行で "Sum" 以外の集計関数を実行したい場合があります。たとえば、データの平均値または最大値を検索する場合です。これは、**[フィールドの設定]**ダイアログボックスを使用するか、コードで簡単に実行できます。

データに対して実行する関数を実行時に指定するには:

1. **C1OlapPanel** の**[値]**フィールドを右クリックします。
2. コンテキストメニューの**[フィールドの設定]**をクリックします。**[フィールドの設定]**ダイアログボックスが開きます。
3. **[小計]**タブをクリックします。
4. 次のオプションのうちの1つを選択します。

Sum	グループの合計を取得します。
Count	グループ内の値の数を取得します。
Average	グループの平均を取得します。
Maximum	グループ内の最大値を取得します。
Minimum	グループ内の最小値を取得します。
First	グループ内の最初の値を取得します。
Last	グループ内の最後の値を取得します。
Variance	グループの標本分散を取得します。
Standard Deviation	グループの標本標準偏差を取得します。
Variance Population	グループの母分散を取得します。
Standard Deviation Population	グループの母標準偏差を取得します。

5. **[OK]**をクリックして、**[フィールド設定]**ダイアログボックスを閉じます。サマリーテーブルの値が変化していることを確認してください。

データに対して実行する関数をコードで指定するには:

フィールドの **C1OlapField.Subtotal** プロパティを使用して、関数を指定します。このサンプルコードでは、最初にビューが作成され、次に製品別に平均単価が計算されます。

C#

```
// ビューを構築します
var olap = this.c1OlapPage1.OlapEngine;
olap.ValueFields.Add("UnitPrice");
olap.RowFields.Add("OrderDate", "ProductName");
// 単価を書式設定し、平均を計算します
var field = olap.Fields["UnitPrice"];
field.Subtotal = Subtotal.Average;
field.Format = "c";
```

数値データの書式設定

数値データに対しては、通貨やパーセンテージとして書式設定したり、独自のカスタム書式を作成することができます。

実行時に数値データを書式設定するには:

1. **C1OlapPanel** の**[値]**フィールドを右クリックします。
2. コンテキストメニューの**[フィールドの設定]**をクリックします。**[フィールドの設定]**ダイアログボックスが開きます。
3. **[書式]**タブをクリックします。
4. 次のオプションのうちの1つを選択します。

数値	データを 1,235 のような数値として書式設定します。小数点以下の桁数を指定し、桁区切り(,)を使用するかどうかを指定できます。
通貨	データを通貨として書式設定します。小数点以下の桁数を指定できます。
パーセンテージ	データをパーセンテージとして書式設定します。小数点以下の桁数を指定できます。
指数	データを指数表記として書式設定します。小数点以下の桁数を指定できます。
カスタム	データに対して独自のカスタム書式を入力します。

5. **[OK]**をクリックして、**[フィールド設定]**ダイアログボックスを閉じます。サマリーテーブルの値が変化していることを確認してください。

コードで数値データを書式設定するには:

フィールドの **C1OlapField.Format** プロパティと Microsoft 標準の数値書式文字列を使用して、書式を指定します。

適用される書式文字列:

"N" または "n"	**数値**	データを 1,235 のような数値として書式設定します。小数点以下の桁数を指定し、桁区切り(,)を使用するかどうかを指定できます。
"C" または "c"	**通貨**	データを通貨として書式設定します。小数点以下の桁数を指定できます。
"P" または "p"	**パーセンテージ**	データをパーセンテージとして書式設定します。小数点以下の桁数を指定できます。
"E" または "e"	**指数**	データを指数表記として書式設定します。小数点以下の桁数を指定できます。
任意の非標準の数値書式文字列	**カスタム**	データに対して独自のカスタム書式を入力します。

このサンプルコードでは、最初にビューが作成され、次に通貨書式で平均単価が計算されます。

C#

```
// ビューを構築します
   var olap = this.c1OlapPage1.OlapEngine;
   olap.ValueFields.Add("UnitPrice");
   olap.RowFields.Add("OrderDate", "ProductName");
// 単価を書式設定し、平均を計算します
   var field = olap.Fields["UnitPrice"];
   field.Subtotal = Subtotal.Average;
   field.Format = "c";
```

加重平均と合計

データの加重平均や合計の計算が必要な場合があります。加重平均や合計では、いくつかのデータポイントが他のデータポイントよりも大きく影響します。

製品の連結リストがあり、各製品の購入数量を考慮して製品グループの平均価格を計算するとします。購入された単位数によって価格の平均を重み付けします。これは、実行時にユーザーが行うかコードで行います。

実行時に計算の重み付けを行う場合:

1. **C1OlapPanel** の**[値]**領域でフィールドを右クリックして、**[フィールド設定]**を選択します。
2. **[小計]**タブをクリックして、計算する小計のタイプを選択します。
3. **[重み]**ドロップダウンリストで、重みとして使用されるデータテーブルからフィールドを選択します。
4. **[OK]**をクリックして、**[フィールド設定]**ダイアログボックスを閉じます。

コードで計算の重み付けを行う場合:

C1OlapField.WeightField プロパティを使用して、重みとして使用するフィールドを指定します。この例では、重みは**[Quantity]**フィールドです。

Visual Basic

```
Dim olap = Me.C1OlapPage1.OlapEngine
Dim field = olap.Fields("Quantity")
field.WeightField = olap.Fields(“Quantity”)
```

C#

```
var olap = this.c1OlapPage1.OlapEngine;
var field = olap.Fields["Quantity"];
field.WeightField = olap.Fields["Quantity"];
```

グリッドのエクスポート

OLAP for WPF/Silverlight では、**C1OlapGrid** を .xlsx、.xls、.csv、および .txt 形式にエクスポートすることができます。**[ツールストリップ]**の**[エクスポート]**ボタンをクリックすると、エクスポートが開始されます。

1. フォームの **C1OlapPage** で、[ツールストリップ]の**[エクスポート]**ボタン  をクリックします。
2. **[名前を付けて保存]**ダイアログボックスで、**[ファイル名]**を入力し、ファイル形式の1つを選択したら、**[OK]**をクリックします。

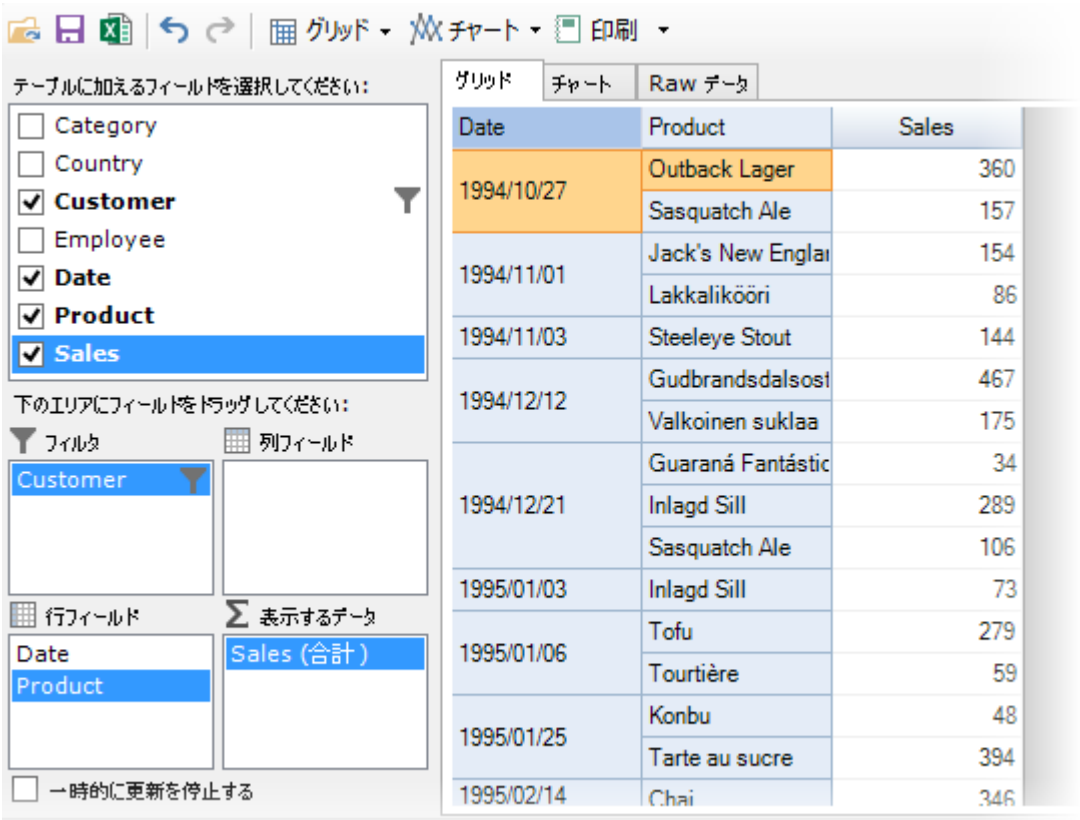
データのグループ化

フィールドの書式設定を使用して、データをグループ化することができます。製品の連結リストがあり、1年以内に注文されたすべての項目をまとめてグループ化するとします。実行時に**[フィールドの設定]**ダイアログボックスを使用するか、コードを使用できます。この例では、製品と共にインストールされる C1Nwind.mdb に連結された **C1OlapPage** コントロールを使用します。

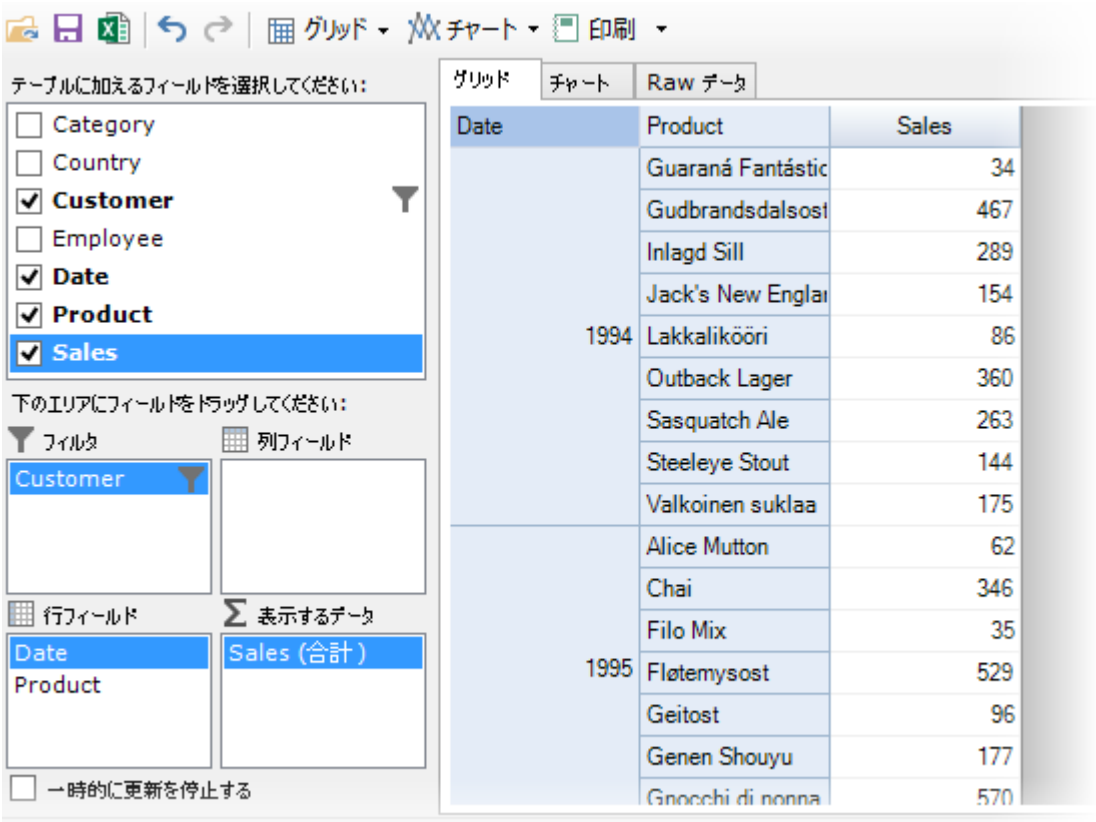
実行時にデータを年別にグループ化するには:

1. **C1OlapPage** の **C1OlapPanel** 領域で、*[OrderDate]*、*[Product]*、および *[Sales]*フィールドを選択し、それらをグリッドビューに追加します。必要に応じて、**[OLAP グリッド]**タブをクリックしてグリッドを表示します。
2. **[行フィールド]**内の**[注文日]**フィールドを右クリックし、**[フィールドの設定]**を選択します。**[フィールドの設定]**ダイアログボックスが表示されます。
3. **[フィルタ]**タブで、**[すべて選択]**が選択されていることを確認します。
4. **[書式]**タブをクリックし、**[カスタム]**を選択します。
5. **[カスタム書式]**テキストボックスに「yyyy」と入力し、**[OK]**をクリックします。

次の画像に、グループ化前とグループ化後のグリッドを示します。*グループ化前*画像では、**[注文日]**はグループ化されていません。グループ化後画像では、製品が購入された年別にまとめてグループ化されています。



グループ化前



グループ化後

コードでデータをグループ化するには:

コードでデータをグループ化することもできます。上の例の場合は、次のコードが使用されます。

VisualBasic

```
Imports C1.Olap
Imports System.Data.OleDb
Namespace WindowsFormsApplication1
    Public Partial Class Form1
        Inherits Form
        Public Sub New()
            InitializeComponent()
            ' データを取得します
            Dim da = New OleDbDataAdapter("select * from invoices",
GetConnectionString())
            Dim dt = New DataTable()
            da.Fill(dt)

            ' OLAP ページに連結します
            Me.c1OlapPage1.DataSource = dt

            ' ビューを構築します
            Dim olap = Me.c1OlapPage1.OlapEngine
            olap.ValueFields.Add("UnitPrice")
            olap.RowFields.Add("OrderDate", "ProductName")

            ' 注文日付の書式を設定して、データをグループ化します
            Dim field = olap.Fields("OrderDate")
            field.Format = "yyyy"
        End Sub
        Private Shared Function GetConnectionString() As String
            Dim path As String =
Environment.GetFolderPath(Environment.SpecialFolder.Personal) + "\ComponentOne
Samples\Common"
            Dim conn As String = "provider=microsoft.jet.oledb.4.0;data source=
{0}\c1nwind.mdb;"
            Return String.Format(conn, path)
        End Function
    End Class
End Namespace
```

C#

```
using C1.Olap;
using System.Data.OleDb;
namespace WindowsFormsApplication1
  {
    public partial class Form1 : Form
     {
       public Form1()
         {
```

```
            InitializeComponent();
             // データを取得します
             var da = new OleDbDataAdapter("select * from invoices",
GetConnectionString());
             var dt = new DataTable();
             da.Fill(dt);
             // OLAP ページに連結します
             this.c1OlapPage1.DataSource = dt;
             // ビューを構築します
             var olap = this.c1OlapPage1.OlapEngine;
             olap.ValueFields.Add("UnitPrice");
             olap.RowFields.Add("OrderDate", "ProductName");
             // 注文日付の書式を設定して、データをグループ化します
             var field = olap.Fields["OrderDate"];
             field.Format = "yyyy";
         }
       static string GetConnectionString()
         {
            string path =
Environment.GetFolderPath(Environment.SpecialFolder.Personal) + @"\ComponentOne
Samples\Common";
            string conn = @"provider=microsoft.jet.oledb.4.0;data source=
{0}\c1nwind.mdb;";
            return string.Format(conn, path);
        }
    }
}
```

グループの折りたたみ/展開

C1OlapGrid は、コードで次のメソッドを使用して、グループの詳細データを表示したり、サマリーだけを表示する機能も提供しています。

- **CollapseAllRows**:このメソッドは、行グループに多くのデータレベルがあるときに、行グループを折りたたむために使用されます。たとえば、**CollapseAllRows** を使用して、次に示すように、年ごとの売上高を表示できます。

OrderDate	Product	Sales
⊞ 1994	**Subtotal**	162,744
⊞ 1995	**Subtotal**	590,927
⊞ 1996	**Subtotal**	512,022

- **CollapseAllCols**:このメソッドは、列グループの多くのデータレベルからサマリーデータだけを表示する必要があるときに、列グループを折りたたむために使用されます。
- **ExpandAllRows**:このメソッドは、行グループを展開して、折りたたまれた行の詳細データを表示するために使用されます。代わりに、実行時に[+]ボタンをクリックして展開することもできます。
- **ExpandAllCols**:このメソッドは、列グループを展開して、折りたたまれた列の詳細データを表示するために使用されます。代わりに、実行時に[+]ボタンをクリックして展開することもできます。

以下のコードは、これらのプロパティの設定方法を示します。

- 行グループを折りたたむには

VB

```
c1OlapPage1.OlapGrid.CollapseAllRows()
```

C# copyCode

```
c1OlapPage1.OlapGrid.CollapseAllRows();
```

- 行グループを展開するには

VB

```
c1OlapPage1.OlapGrid.ExpandAllRows()
```

C# copyCode

```
c1OlapPage1.OlapGrid.ExpandAllRows();
```

列グループを折りたたむ/展開するためのプロパティも、同様に設定できます。

レポートの作成

C1OlapPage コントロールで、実行時に**[レポート]**メニューを使用してレポートの設定と印刷を行うことができます。

レポートを作成するには、次の手順に従います。

1. **C1OlapPage** ツールストリップ内の**[レポート]**の横にあるドロップダウン矢印をクリックします。
2. **[オプション]**を選択します。**[ドキュメントオプション]**ダイアログボックスが表示されます。
3. **[ページ]**タブで、必要に応じて、ページの**[向き]**、**[用紙サイズ]**を選択し、**[マージン]**を設定します。
4. **[ヘッダー/フッター]**タブをクリックします。
5. ヘッダーまたはフッターのテキストボックスで、テキストまたは定義済みのヘッダー/フッター項目を追加する場所にカーソルを置きます。
6. ツールバーのボタンの1つをクリックして、目的のフィールドに入力します。
7. **[レポートコンテンツ]**タブをクリックします。
8. レポートに含める項目の横にあるチェックボックスをオンにします。また、ラジオボタンを選択して、グリッドまたはチャートのスケーリングを選択することもできます。
9. **[OK]**をクリックして、**[ドキュメントオプション]**ダイアログボックスを閉じます。