ComponentOne

FlexPivot for WinForms

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

Website: http://www.componentone.com
Sales: sales@componentone.com
Telephone: 1.800.858.2739 or 1.412.681.4343 (Pittsburgh, PA USA Office)

Trademarks

Warranty

ComponentOne warrants that the media on which the software is delivered is free from defects in material and workmanship, assuming normal use, for a period of 90 days from the date of purchase. If a defect occurs during this time, you may return the defective media to ComponentOne, along with a dated proof of purchase, and ComponentOne will replace it at no charge. After 90 days, you can obtain a replacement for the defective media by sending it and a check for $2 5 (to cover postage and handling) to ComponentOne.

Except for the express warranty of the original media on which the software is delivered is set forth here, ComponentOne makes no other warranties, express or implied. Every attempt has been made to ensure that the information contained in this manual is correct as of the time it was written. ComponentOne is not responsible for any errors or omissions. ComponentOne's liability is limited to the amount you paid for the product. ComponentOne is not liable for any special, consequential, or other damages for any reason.

Copying and Distribution

Table of Contents

- FlexPivot for WinForms Overview 3
  - What is FlexPivot 3
- Introduction to FlexPivot 4-5
- Key Features 6
- FlexPivot Architecture 7
  - FlexPivotPage 7-8
  - FlexPivotPanel 8-10
  - FlexPivotGrid 10
  - FlexPivotChart 10-11
  - FlexPivotPrintDocument 11
- Adding Data to FlexPivot Application 12
- Design Time Support through Smart Tag 13
  - FlexPivotPanel Smart Tag 13-14
  - FlexPivotPage Smart Tag 14-15
  - FlexPivotChart Smart Tag 15-16
  - FlexPivotGrid Smart Tag 16
- Quick Start: FlexPivot for WinForms 17
  - Using FlexPivot Controls with C1DataEngine 17
    - Step 1: Setting up the Application 17-18
    - Step 2: Connecting FlexPivotPage Control with DataEngine 18
    - Step 3: Creating Different Views at Runtime 18-22
  - Updating Data 22-23
  - Using FlexPivot Controls with Data Source 23
    - Binding FlexPivot to Data Source at Design-Time 23-30
    - Binding FlexPivot to Data Source in Code 30-31
- Using FlexPivotPage ToolStrip 32
  - Grid Menu 32
  - Chart Menu 32-35
  - Report Menu 35-38
- Data Blending Features 39
  - Joining 39
  - Grouping 39-42
  - Drilling Down Data 42-43
  - Applying Conditional Formatting 43-45

Creating Reports with FlexPivot 45-46
Copying Data to Excel 46-47
Filtering 47
Using Value Filters 47-48
Using Range Filters 48-50
Sorting 50-51
Formatting Numeric Data 51-52
Specifying Subtotal Function 52-54
Calculating Weighted Averages and Sums 54-55
FlexPivot Cube 56
Setting Microsoft SQL Server Analysis Services 56
Connecting to a Cube 56-57
Using the Cube 57-58
Task-Based Help 59
Configuring Fields in Code 59-60
Adding Multiple Fields in Values List 60-61
Applying Themes 61-64
Using LINQ Queries to Add Data in FlexPivot 64-65
Creating Custom FlexPivot Application in Code 65
Creating Default View 65-67
Adding Predefined Views 67-74
Persisting Views 74-75
Importing Data from Excel 75-78
C1DataEngine Overview 79
Using C1DataEngine 79
Base Tables 79-80
Simple Operations 80-81
Aggregation 81-82
Combining Operations and Queries 82
Filter and Range 82-84
Join 84-86
Excel Add-in 86
Using Excel Add-in 86

FlexPivot for WinForms Overview

ComponentOne Studio introduces **FlexPivot for WinForms**, a powerful data analysis product designed and developed to cater high-end business intelligence needs. Being a data summarization product, FlexPivot enables end users in breaking heavy and complex database into smaller, modular data chunks presented in multiple formats including grids, charts and reports.	**Getting Started** To get started, review the following topics: • Key Features • Quick Start: FlexPivot for WinForms

What is FlexPivot

In most of the real-world business scenarios, organizing large datasets into smaller, comprehensible formats and extracting valuable information and insights from them remain a common concern. Although a variety of powerful business intelligence tools exists, end users still prefer tools that combine high-performance with smaller footprint to offer faster data analysis and summarization capabilities.

FlexPivot is designed to offer high-end business intelligence capabilities and optimized data analysis to end users. **FlexPivot** controls offer superior analytical processing features. **FlexPivot** is powered by a platform-independent data engine that performs data analytics locally in the application, without locking into another data analysis tool.

FlexPivot controls also deliver simple, code-free data analysis as it creates dynamic summarized views merely by dragging data fields into respective lists available on its User Interface. In addition, the C1FlexPivot class library provides well-defined extensions and API to fetch complex data through code so that end users can analyze and summarize the data in creative ways.

FlexPivot also supports multi-dimensional data analysis through simple drag-and-drop operations. You can create interactive grids, charts and reports that can be saved, exported, shared and printed in various file formats (such as XLS, PDF and XLSX) to cater diverse business needs. FlexPivot also offers the capability to connect with online or local cubes so that users can easily slice and dice the cube data.

Enhanced Data Engine

The C1DataEngine, the core component that powers FlexPivot, is a C# component with low footprint. This data engine stores data in memory-mapped files using column-oriented technology that helps deliver higher processing speeds of up to millions of records in a fraction of a second. The C1DataEngine can also be used separately, independent of FlexPivot controls for data analysis and visualization as described in Using C1DataEngine section.


The data engine fully supports all the features on a 64-bit OS machine.

Introduction to FlexPivot

FlexPivot is designed similar to an online analytical processing tool. This product is a set of technologies that enable dynamic visualization and analysis of data.

Typical analytical processing tools include **OLAP cubes** and pivot tables such as the ones provided by Microsoft Excel. These tools take large set of data and summarize it by grouping records based on a set of criteria. For example, an OLAP cube might summarize sales data by grouping it on the basis of product, region and period. In this case, each grid cell would display the total sales for a particular product, in a particular region, and for a specific period. This cell would normally represent data from several records in the original data source.

Data analysis and processing tools allow users to redefine grouping criteria dynamically (online). This makes it easy to perform ad-hoc data analysis and discover hidden patterns.

For example, consider the following table:

Date	Product	Region	Sales
Oct 2015	Product A	North	12
Oct 2015	Product B	North	15
Oct 2015	Product C	South	4
Oct 2015	Product A	South	3
Nov 2015	Product A	South	6
Nov 2015	Product C	North	8
Nov 2015	Product A	North	10
Nov 2015	Product B	North	3

Now suppose you were asked to analyze this data and answer questions such as:

- Are sales going up or down?
- Which products are most important to the company?
- Which products are most popular in each region?

In order to answer these simple questions, you would have to summarize the data to obtain tables such as these:

Sales by Date and by Product

Date	Product A	Product B	Product C	Total
Oct 2007	15	15	4	34
Nov 2007	16	3	8	27
Total	**31**	**18**	**12**	**61**

Sales by Product and by Region

Product	North	South	Total

Product A	22	9	31
Product B	18		18
Product C	8	4	12
Total	**48**	**13**	**61**

Each cell in the summary tables represents several records in the original data source, where one or more values fields are summarized (sum of sales in this case) and categorized on the basis of other fields (date, product, or region in this case).

This can be done easily in a spreadsheet, but the work is tedious, repetitive, and error-prone. Even if you write a custom application to summarize the data, probably you spend a lot of time maintaining it to add new views, and users might get constrained in their analyses to the views implemented by you.

FlexPivot allows users to define the views they want in an interactive, and ad-hoc fashion. They can use pre-defined views or create and save new ones. Any changes to the underlying data are reflected automatically in the views, and users can create and share reports showing these views. In short, FlexPivot control provides flexible and efficient data analysis.

Key Features

Below mentioned are some of the key features of **FlexPivot** that would help developers realize their complex data analysis needs.

- **Multiple Value Fields**
  FlexPivot supports multiple fields in the Values list. Users can drag multiple fields into the Values list and see the results in grid and chart. See the topic Adding Multiple Fields in Values List to know more about this.

- **Enhanced and Powerful Data Engine**
  FlexPivot is built with a new, powerful data engine that stores data in memory-mapped files using column-oriented technology. The C1DataEngine offers high-speed processing of very large datasets that makes it possible to reach high performance of up to hundreds of millions records in a fraction of a second. The engine also supports query operations such as Aggregation, Joins, Grouping, Filter, Unary/Binary operations, etc. See Using C1DataEngine section for an overview of query related operations.

- **Enhanced User Interface**
  FlexPivot comes with an enhanced, Excel-like user interface (UI) offering modern color schemes in toolbar, drop-down menus, tables and charts.

- **Multiple Theme Support**
  FlexPivot allows developers in choosing a theme that suits their application requirements. To support multiple themes, FlexPivot includes C1ThemeController support.

- **Plot Information on Grid View**
  FlexPivot provides the FlexPivotGrid control to display data in a grid-like view. The C1FlexPivotGrid class library enables you to customize your FlexPivot application. See the FlexPivot Architecture section for an overview of the FlexPivotGrid control.

- **Choose how to display the information at runtime**
  Use the FlexPivotPanel to determine which fields of your data source should be used to display data and how. Drag fields between the lower areas of the FlexPivotPanel to create a filter, column headers, row headers, or get the sum of values from a column or row. For more information, see FlexPivotPanel section.

- **Implements C1 Controls for Customization**
  FlexPivot comes with C1 control support (such as check boxes, tabs and toolstrip) instead of standard Microsoft controls. This way, the FlexPivot product provides users with complete flexibility in customizing the look and feel of their application according as per their requirements.

- **Query Operation Support**
  FlexPivot supports query operations including aggregation, joins, filters, ranges, sorting, grouping, unary and binary operations, calculated fields, projections and grouping, etc.


FlexPivot controls fully support all the features on a **64-bit OS machine**.

FlexPivot Architecture

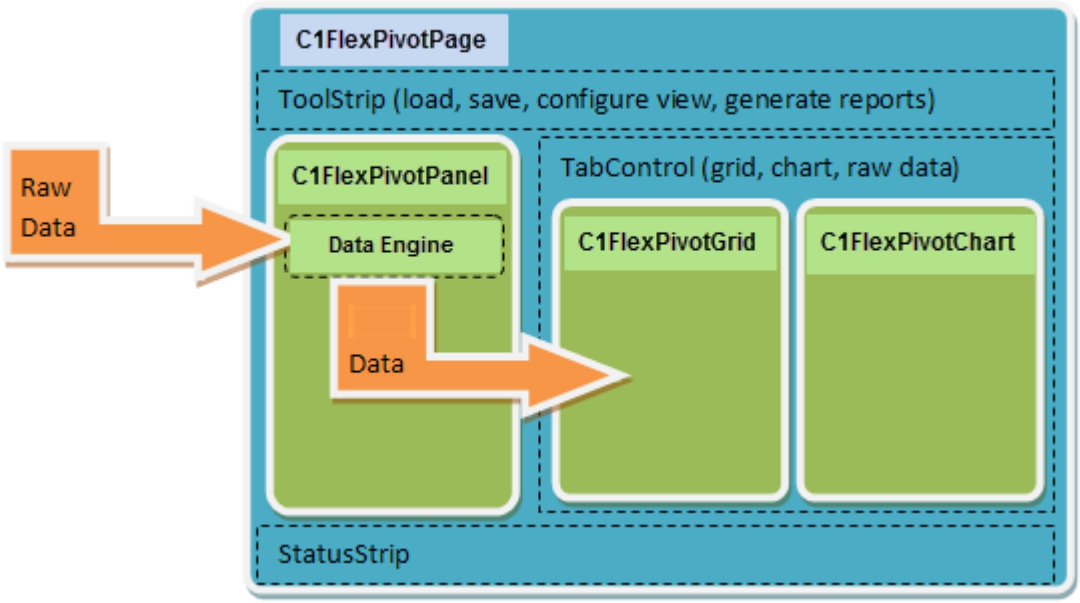
This section highlights the architectural features of **FlexPivot**. The section contains a series of subtopics that cover information on various components integrated within FlexPivot including FlexPivotPage, FlexPivotPanel, FlexPivotGrid, FlexPivotChart and FlexPivotPrintDocument.

FlexPivotPage

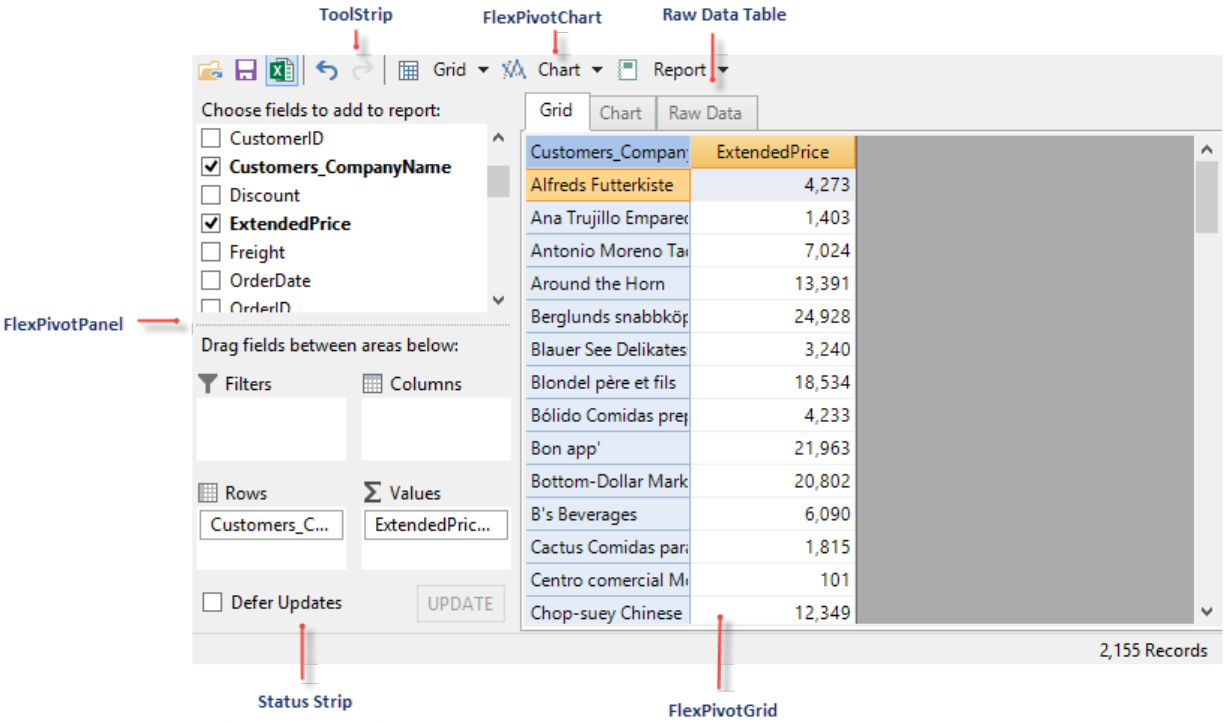
The FlexPivotPage control is designed for easy and quick development of FlexPivot applications. The control provides a complete user interface (UI) built using other controls.

The C1FlexPivotPage object model exposes the integrated controls so that developers can easily customize them by adding or removing the interface elements as per their requirements. Developers can also use the source code of the control for more extensive customization and implement them according to the specifications of their application.

The image below depicts the architectural organization of the FlexPivotPage control.



In Visual Studio, the FlexPivotPage control appears similar to the image below.



FlexPivotPanel

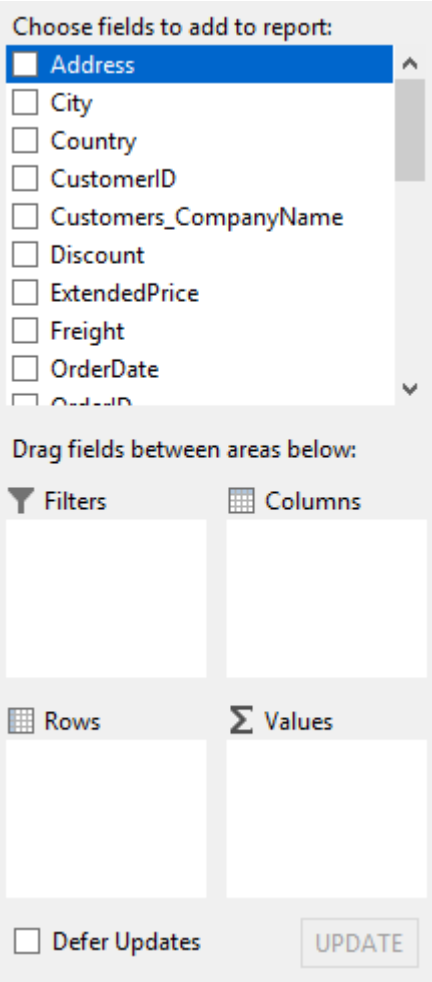
The FlexPivotPanel forms the core control available with **FlexPivot**.

The C1FlexPivotPanel control is designed to provide an Excel-like drag-and-drop UI that enables users to define custom views of the data. The control displays a list containing all the fields that exist in the data. Users can drag-and-drop these fields to lists that represent the row and column dimensions of the output table, the values summarized in the output data cells, and the fields used for filtering the data.

At the core of this control is a powerful data engine that is responsible for fetching raw data from the data source as per the criteria selected by the user. The criteria for data selection is determined by the fields enlisted in the various lists appearing on the panel.

FlexPivot features an open architecture. It can accept any regular collection as data, including tables and generic lists; and add LINQ enumerations to summarize the data and produce a regular data table as output.

The C1FlexPivotPanel control bound to a data source appears similar to the image below.



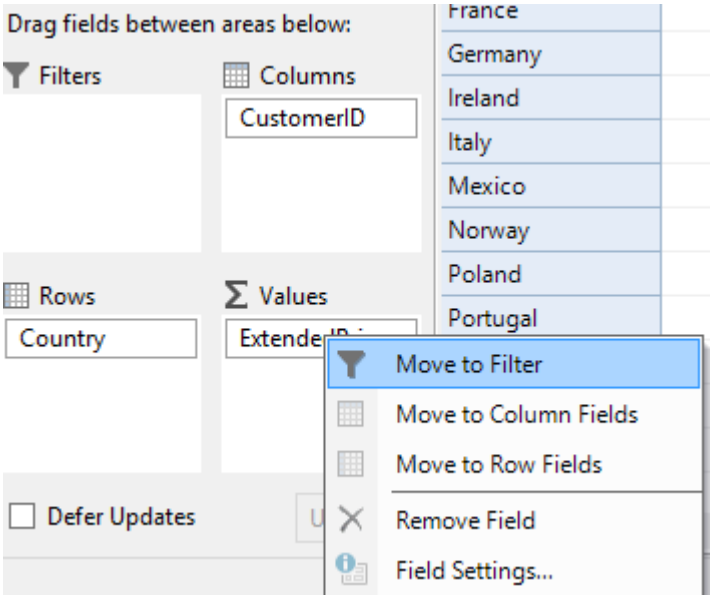
The following table describes the lists that are included in the C1FlexPivotPanel control and their purpose.

C1FlexPivotPanel Area	Description
Filters	Specifies the field to filter.
Rows	The items added to the Rows list become the row headers of a grid. These items populate the Y-axis in a chart.
Columns	The items added to the Column list become the column headers of a grid. These items populate the legend in a chart.
Values	Shows the summation of the specified field.
Defer Updates	Suspends automatic updates that occur when the user modifies the view definition when this checkbox is selected.

Upon right-clicking the Filter, Rows, Columns, or Values lists at runtime, a context menu appears that allows you to do the following.

- Move the field to a different area
- Remove the field
- Click field settings to format
- Apply a filter to the field

The image below illustrates the above stated options.



FlexPivotGrid

The FlexPivotGrid displays data fetched from the data source in the form of Pivot tables. FlexPivotGrid can be bound to a data source for populating it with an ordered dataset. The control provides automatic data binding with C1FlexPivotPanel objects, grouped row and column headers, and custom behaviors for resizing columns, copying data to the clipboard, and showing details for a given cell.

Extending the features of C1FlexGrid control, the C1FlexPivotGrid control allows users to export the grid content to Excel sheet or use styles and owner-draw cells to customize the grid's overall appearance.

The image below shows how a FlexPivotGrid control populated with ordered data looks.

CustomerID	Country	Customers_Compan	ExtendedPrice
ALFKI	Germany	Alfreds Futterkiste	4,273
ANATR	Mexico	Ana Trujillo Emparec	1,403
ANTON		Antonio Moreno Ta	7,024
AROUT	UK	Around the Horn	13,391
BERGS	Sweden	Berglunds snabbköp	24,928
BLAUS	Germany	Blauer See Delikates	3,240
BLONP	France	Blondel père et fils	18,534
BOLID	Spain	Bólido Comidas prep	4,233
BONAP	France	Bon app'	21,963
BOTTM	Canada	Bottom-Dollar Mark	20,802
BSBEV	UK	B's Beverages	6,090
CACTU	Argentina	Cactus Comidas para	1,815
CENTC	Mexico	Centro comercial Mo	101
CHOPS	Switzerland	Chop-suey Chinese	12,349

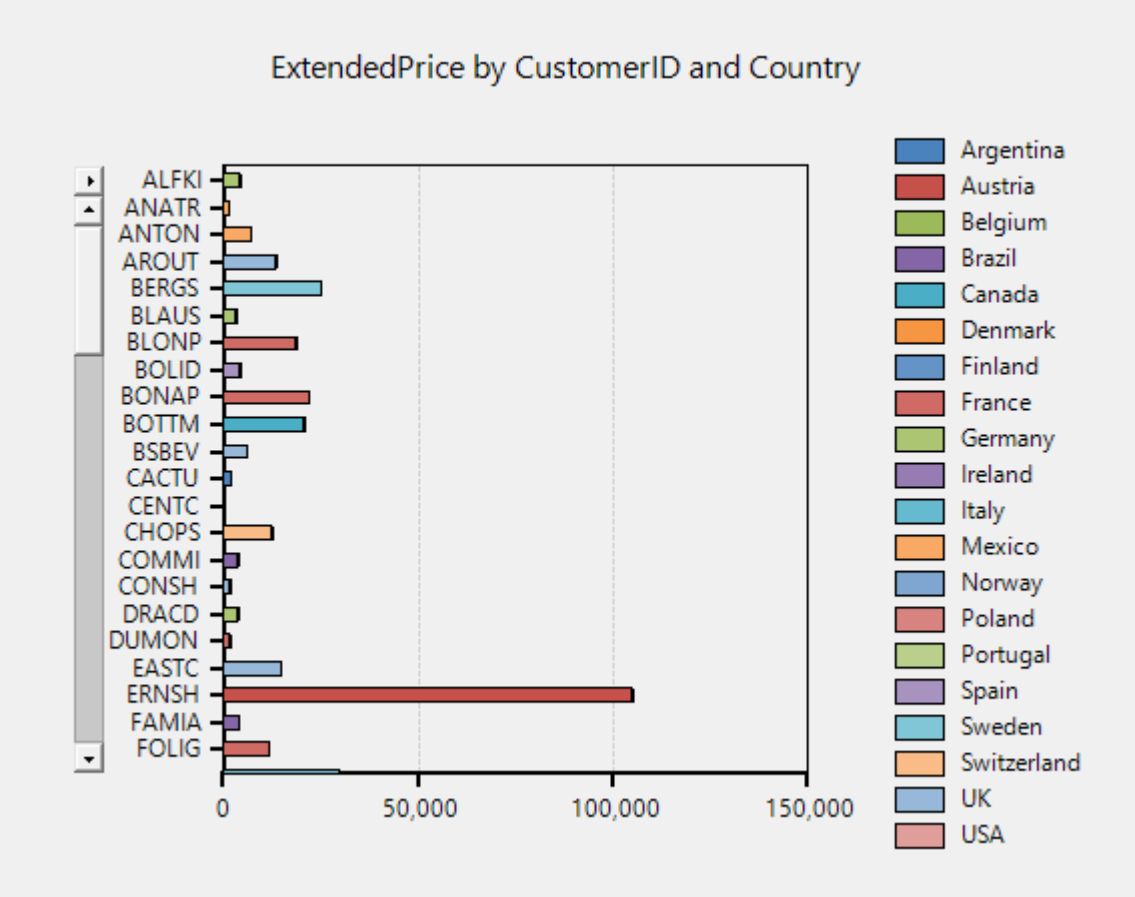
FlexPivotChart

The FlexPivotChart displays data fetched from the data source in the form of Pivot charts. The C1FlexPivotChart

control can be bound to a data source to populate it with data. The control provides automatic data binding with FlexPivotPanel objects, automatic tooltips, chart types and integrated palettes.

Extending the features of **C1Chart** control, the C1FlexPivotChart control allows users to export the chart to different file formats including PNG and JPG or customize chart styles and interactivity.

The image below shows how a FlexPivotChart control looks.



FlexPivotPrintDocument

The C1FlexPivotPrintDocument control creates and displays reports based on various views available in the FlexPivot control. The control extends the PrintDocument class and provides properties that can be used for specifying and formatting content, displaying grids and charts, and create ad-hoc reports using raw data.

Adding Data to FlexPivot Application

FlexPivot comes with three options to add data as mentioned below.

- Using C1DataEngine
- Using a Data Source
- Using Cube

Using C1DataEngine

Using the C1DataEngine is the most effective way to add data to FlexPivot controls. The data engine is not a database or server but a low-footprint C# component that can be easily integrated into FlexPivot applications. The engine stores data in memory-mapped files, which gets fetched instantly without any delay in importing these files.
The C1DataEngine does not put any restriction on the size of the dataset that you wish to analyze or display, and that too without compromising the performance. To know more about using FlexPivot with C1DataEngine, see Using FlexPivot Controls with C1DataEngine topic.

Using Data Source

FlexPivot controls support binding with data sources to display and analyze data in multiple ways. To know more about using data sources with FlexPivot controls, see Using FlexPivot Controls with Data Source topic.

Using Cube

FlexPivot includes cube support that enables users in slicing and dicing the cube in several ways. You can connect to online cubes, or to **Analysis Services, Microsoft SQL Server Analysis Services (SSAS)**, and **SQL Server 2008, 2012**, etc for extracting valuable information. For more information on cube support, see FlexPivot Cube topic.

Design Time Support through Smart Tag

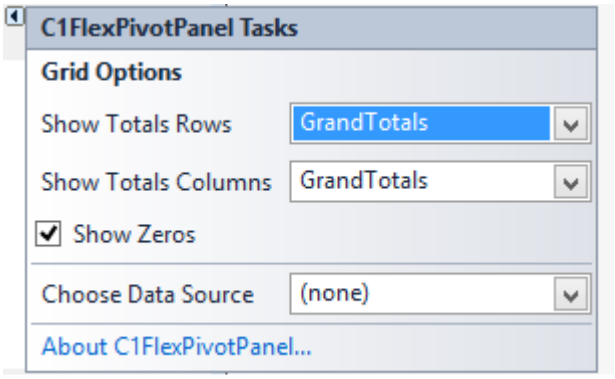
FlexPivot comes with smart tags as a capability to configure the controls at design-time. A smart tag represents a short-cut **Tasks** menu that provides the most commonly used properties in each control.
The C1FlexPivotPanel, C1FlexPivotPage, C1FlexPivotChart, and C1FlexPivotGrid controls offer smart tag support at design-time so you can easily access their properties and configure them.

The following sub topics give a detailed description of the smart tags available in various controls.

FlexPivotPanel Smart Tag

The C1FlexPivotPanel control includes a smart tag in Visual Studio to provide access to its commonly used properties and tasks. The FlexPivotPanel smart tag appears when the control is bound to a data source.

To access the FlexPivotPanel Tasks menu, click the smart tag in the upper-right corner of the FlexPivotPanel control. This smart tag appears similar to the image below.



The FlexPivotPanel Tasks menu provides the following options:

- **Show Totals Row**
  Clicking the **Show Totals Row** check box adds a row at the bottom of your grid which totals all the data in the column.

- **Show Totals Column**
  Clicking the **Show Totals Column** check box adds a column to the right of the last column in your grid which totals all the data in the row.

- **Show Zeros**
  Clicking the **Show Zeros** check box shows any cells containing zero in the grid.

- **Choose Data Source**
  Clicking the drop-down arrow in the **Choose Data Source** box opens a list of available data sources and allows you to add a new data source. To add a new data source to the project, click **Add Project Data Source** option. This opens the **Data Source Configuration Wizard**.

- **Plot Information on Grid View**
  FlexPivot provides a FlexPivotGrid control to display data. FlexPivotGrid is also available as separate control so you can customize your FlexPivot application. See the **FlexPivot Architecture** section for an overview of the control.

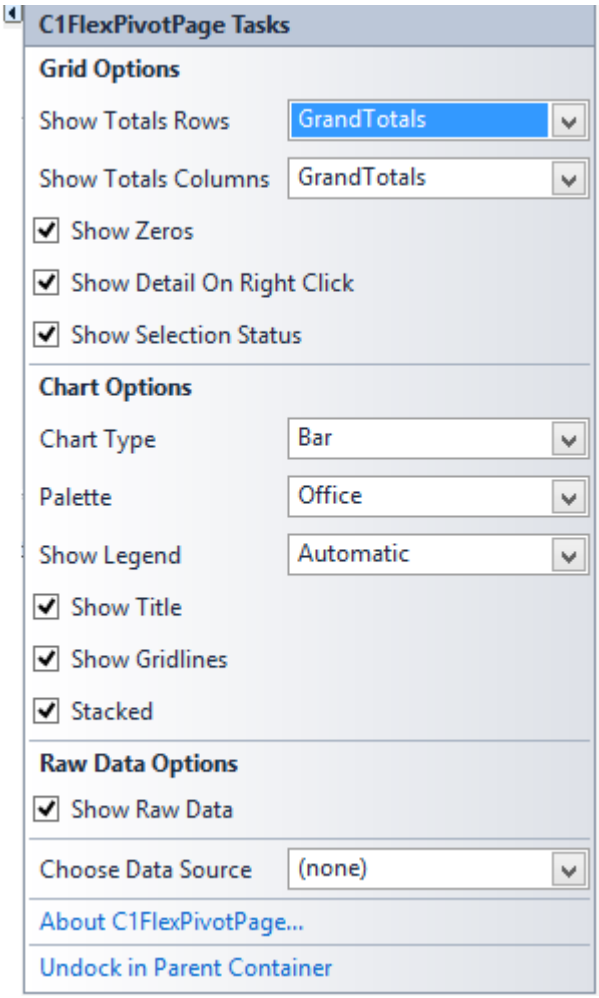
- **About FlexPivotPanel**
  Clicking **About FlexPivotPanel** displays a dialog box, which is helpful in finding the version number of the

product and other resources.

FlexPivotPage Smart Tag

The C1FlexPivotPage control includes a smart tag in Visual Studio to provide access to its commonly used properties and tasks.

To access the **FlexPivotPage Tasks** menu, click the smart tag in the upper-right corner of the FlexPivotPage control. This smart tag appears similar to the image below.



The FlexPivotPage Tasks menu provides the following options:

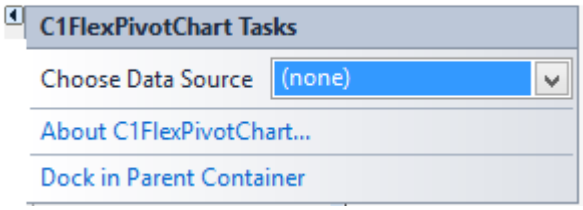
- **Show Totals Row**
  Clicking the Show Totals Row check box adds a row at the bottom of your grid that totals all the data in the column.
- **Show Totals Column**
  Clicking the Show Totals Column check box adds a column to the right of the last column in your grid that totals all the data in the row.
- **Show Zeros**
  Clicking the Show Zeros check box shows any cells containing zero in the grid.
- **Show Detail on Right Click**
  Clicking the Show Detail on Right Click check box allows a detail view to be shown when the user right-clicks a cell in the grid.
- **Show Selection Status**
  Clicking the Show Selection Status check box causes the control to display the sum of the values selected on

the grid in the status bar along the bottom of the control. This corresponds to setting the **ShowSelectionStatus** property to True.

- **Chart Type**
  Clicking the drop-down arrow next to **ChartType** allows you to select the chart type options such as Bar, Column, Area, Line, and Scatter.
- **Palette**
  Clicking the drop-down arrow next to **Show Legend** allows you to choose whether to always, never, or automatically show the legend.
- **Show Legend**
  Clicking the drop-down arrow next to **Show Legend** allows you to choose whether to always, never, or automatically show the legend.
- **Show Title**
  Clicking the **Show Title** check box places a title above the chart.
- **Show Gridlines**
  Clicking the **Show Gridlines** check box places gridlines in the chart.
- **Stacked**
  Clicking the **Stacked** check box creates a chart view where the data is stacked.
- **Show Raw Data**
  Clicking the **Show Raw Data** check box adds a raw data table, which contains the raw data from your data sources, to the views.
- **Choose Data Source**
  Clicking the drop-down arrow in the **Choose Data Source** box opens a list of available data sources and allows you to add a new data source. To add a new data source to the project, click **Add Project Data Source** to open the **Data Source Configuration Wizard**.
- **About C1FlexPivotPage**
  Clicking About C1FlexPivotGrid displays a dialog box, which provides the version number of the product and other online resources.
- **Undock in parent container**
  Clicking **Undock in parent container** sets the **Dock** property to **None** so that none of the borders of the control are bound to the container. The menu option then changes to **Dock in parent container**. When you click this option, the **Dock** property of the control sets to **Fill** and the control bounds to the container.

FlexPivotChart Smart Tag

The C1FlexPivotChart control includes a smart tag in Visual Studio to provide access to its commonly used properties and tasks. This smart tag appears similar to the image below.



To access **FlexPivotChart Tasks** menu, click the smart tag in the upper-right corner of the FlexPivotChart control.

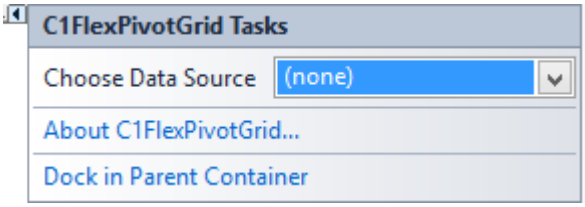
The **FlexPivotChart Tasks** menu provides the following options:

- **Choose Data Source**
  Clicking the drop-down list next to **Choose Data Source** allows you to select a FlexPivotPanel to bind the chart to.
- **About C1FlexPivotChart**
  Clicking **About C1FlexPivotChart** displays a dialog box, which provides the version of **C1Chart** and online resources.
- **Dock in Parent Container**

Clicking **Dock in Parent Container** sets the **Dock** property to **Fill** so that the control bounds to the container. The menu option then changes to **Undock in parent container.**

FlexPivotGrid Smart Tag

The C1FlexPivotGrid control includes a smart tag in Visual Studio to provide access to its commonly used properties and tasks. This smart tag appears similar to the image given below.



To access the **FlexPivotGrid Tasks** menu, click the smart tag in the upper-right corner of the FlexPivotGrid control.

The **FlexPivotGrid Tasks** menu provides the following options:

- **Choose Data Source**
  Clicking the drop-down list next to **Choose Data Source** allows you to select a FlexPivotPanel to bind the grid to.
- **About C1FlexPivotGrid**
  Clicking **About C1FlexPivotGrid** displays a dialog box, which provides the product version.
- **Dock in parent container**
  Clicking **Dock in Parent Container** sets the **Dock** property to **Fill** so that the control bounds to the container. The menu option then changes to **Undock in parent container.**

Quick Start: FlexPivot for WinForms

This quick start guide is intended to get you up and running with FlexPivot. In this section, you begin by connecting the C1FlexPivotPage control to C1DataEngine. You also understand how to bind C1FlexPivotPage control to a local data source, and then continue exploring various features available in the control at runtime.

Using FlexPivot Controls with C1DataEngine

Using C1DataEngine is the most recommended way to add data into FlexPivot controls and fetch it for analysis. All you need to do is create an empty Windows Forms Application, add the FlexPivotPage control to the form, and connect it to the C1DataEngine. Add a few lines of code and that's it. You are ready to add a million records to your application.

To know how the C1DataEngine can be used independent of the FlexPivot controls for data visualization and analysis, see Using C1DataEngine topic.

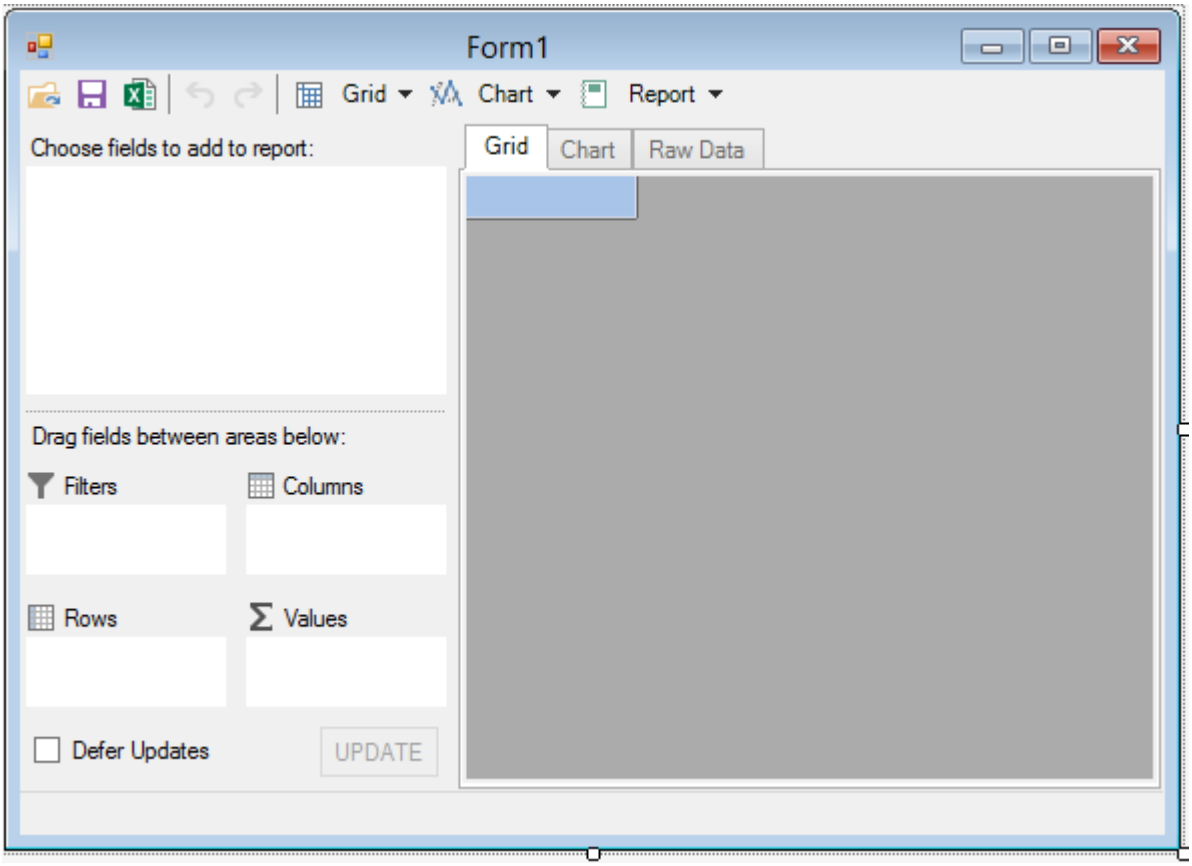
Step 1: Setting up the Application

In this step, you begin with creating a Windows Forms application in Visual Studio and then adding a FlexPivotPage control to it.

At Design-Time

Complete the following steps to add FlexPivotPage control to your Windows Forms application.

1. Create a new Windows Forms Application project in Visual Studio.
2. Navigate to the Toolbox and locate the C1FlexPivotPage icon.
3. Double-click or drag-and-drop the C1FlexPivotPage icon to add the control to the Form. The design view looks similar to the following image.



Step 2: Connecting FlexPivotPage Control with DataEngine

In the previous step, you created a basic Windows Forms application and added FlexPivotPage control to it. In this step, you begin with connecting the FlexPivotPage control to C1DataEngine.

To connect the FlexPivotPage control to C1DataEngine, perform the following steps.

1. Switch to the code view i.e. Form1.cs and initialize workspace within the form's constructor.
   - **Visual Basic**

```
Type your example code here. It will be automatically colorized when you switch to Preview or build the help system.
```

   - **C#**

```
Type your example code here. It will be automatically colorized when you switch to Preview or build the help system.
```

2. Initialize full path to the folder where the DataEngine stores data in files, and an SQL connection above the Form's constructor using the following code.
   - **Visual Basic**

```
Type your example code here. It will be automatically colorized when you switch to Preview or build the help system.
```

   - **C#**

```
Type your example code here. It will be automatically colorized when you switch to Preview or build the help system.
```

   Initially, the DataEngine is empty but it automatically fills with files as and when the data is added to it.

3. Initialize a standard connection string to the database file being used.
   - **Visual Basic**

```
Type your example code here. It will be automatically colorized when you switch to Preview or build the help system.
```

   - **C#**

```
Type your example code here. It will be automatically colorized when you switch to Preview or build the help system.
```

4. Switch to the Design view and subscribe **Form1_Load** event from the **Properties** window.
5. Add the following code to the event handler created for **Form1_Load** event in the code view.
   - **Visual Basic**

```
Type your example code here. It will be automatically colorized when you switch to Preview or build the help system.
```

   - **C#**

```
Type your example code here. It will be automatically colorized when you switch to Preview or build the help system.
```

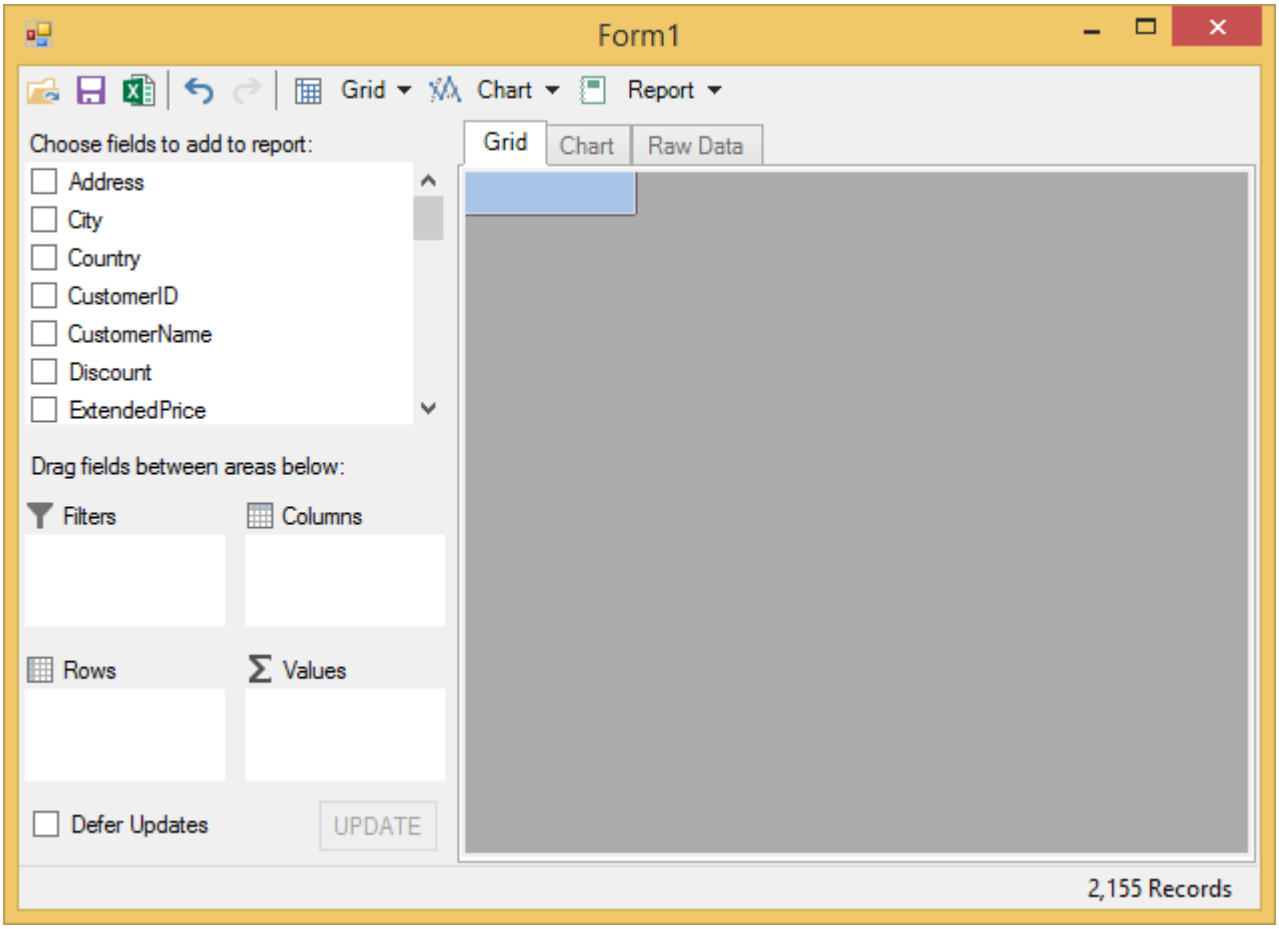
With this, you have successfully connected the FlexPivotPage control to the DataEngine.

Step 3: Creating Different Views at Runtime

Let us now take a look at how we can create different views of the data added using the DataEngine in the previous

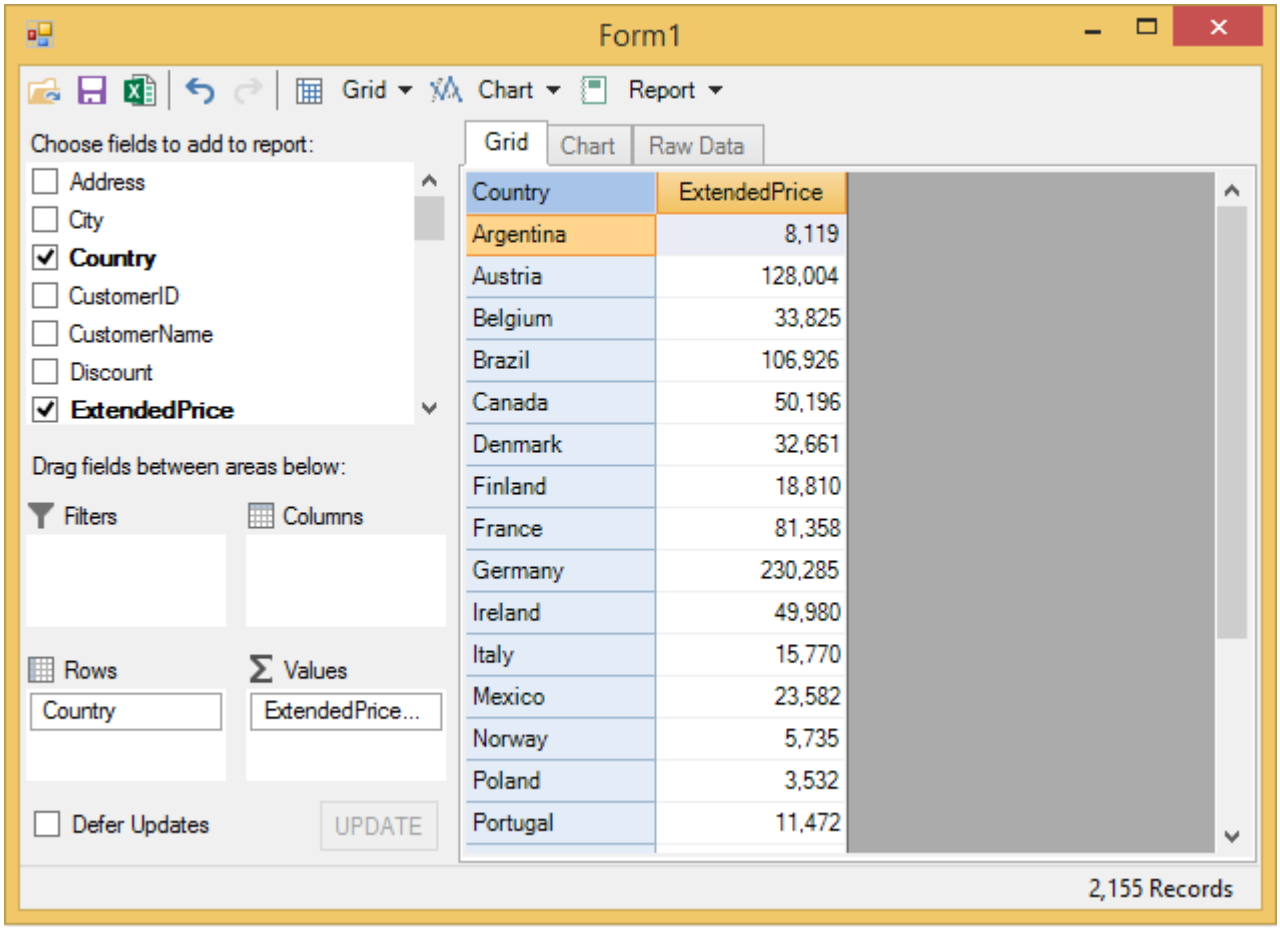
step.

Press **F5** to run the application. FlexPivotPage is now connected to C1DataEngine and you can see the control appears similar to the image below with various data fields that you can choose for data analysis.

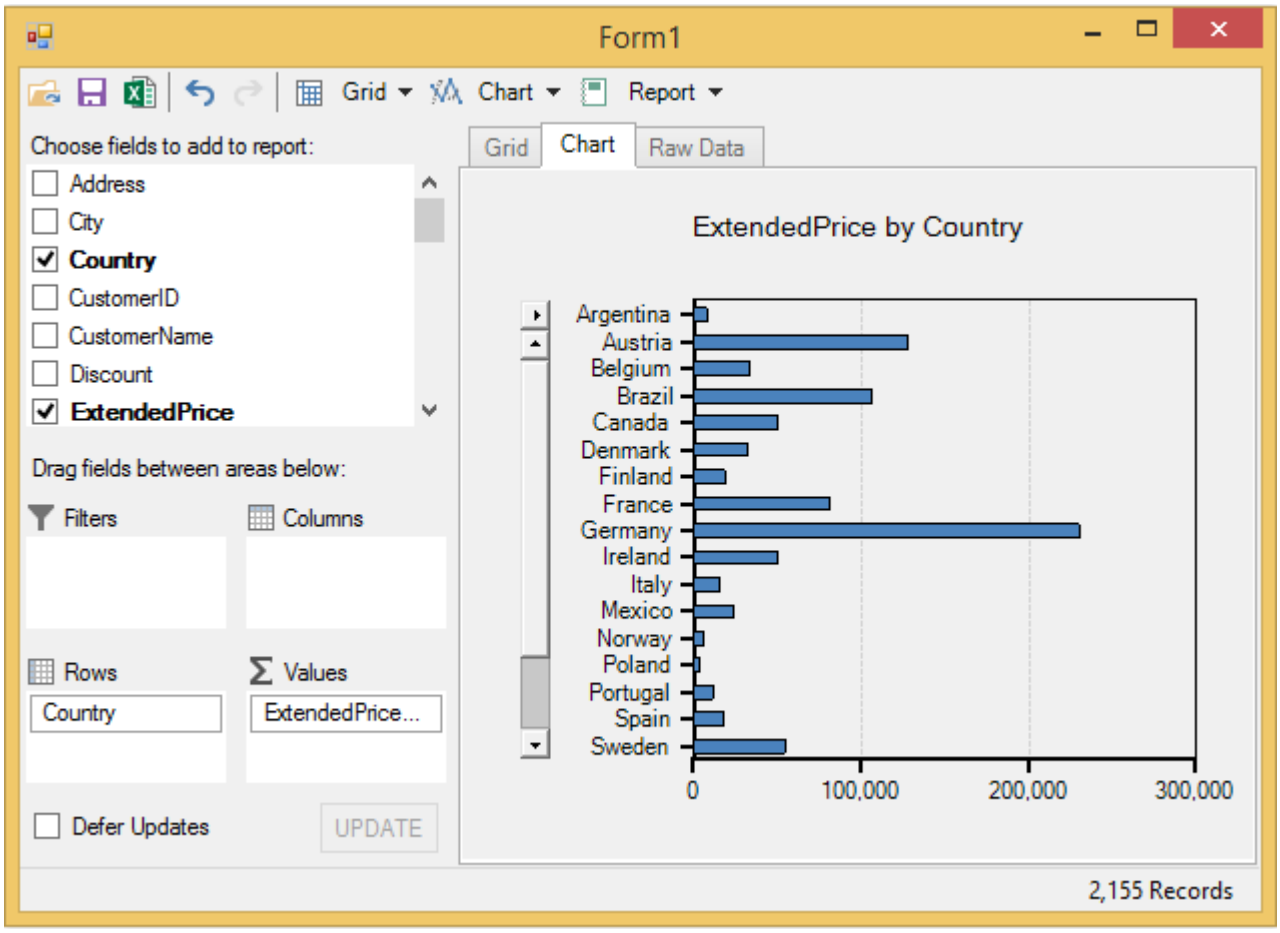


You can create different views by dragging the data fields available in the FlexPivotPanel to Rows, Columns and Values lists as illustrated in the steps below.

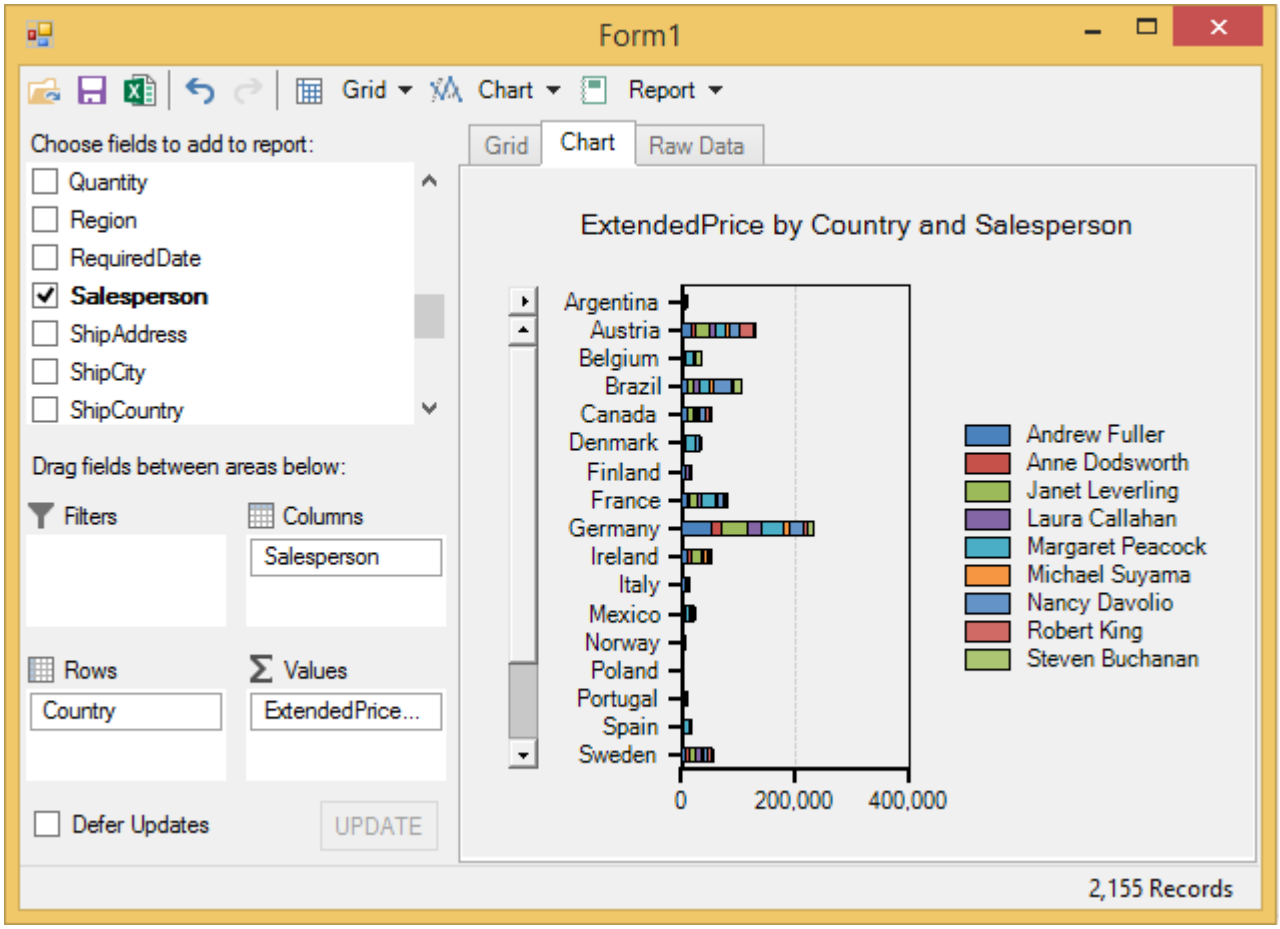
1. Drag the **Country** field from the FlexPivotPanel to **Rows** list, and **Extended Price** field to **Values** list. A summarized view of Extended Price against Countries appears in the form of a grid as shown in the image below.



2. Click the **Chart** tab to view the same data in chart format.



3. Drag **Salesperson** field in the **Columns** list to slightly enhance the view. The new view shows Extended Price by Country and Salesperson.



With this, you have completed all the steps of this Quick Start guide.

Updating Data

FlexPivot enables users to add and update data incrementally through the C1DataEngine. This feature saves users from loading the entire data in every session since the data added in the first session persists. You can achieve this functionality by adding a few lines of code to your application. The data gets loaded into the view instantly without any delay in importing data since it is stored in memory-mapped files.

> Refer to the product sample **DataUpdate** stored at the following location in your system to see the C1DataEngine in full action
>
> ***C:\...\Documents\ComponentOne Samples\WinForms\DataUpdate.***

Complete the following steps to incrementally update data in the FlexPivotPage control.

1. Add the following code in the **Form1_Load** event. The code given below immediately loads the data into the C1DataEngine if the folder pointed out by the data path is not empty. The FlexPivotPage control also displays the loaded data into the view once it is connected to the C1DataEngine.

   C#

   ```
   Workspace.Init(dataPath);
   ```

2. To add new rows of data to the persisted view, use connect.AppendData method as mentioned below.

   C#

   ```
   connector.AppendData(tableName);
   ```

3. By default, a FlexPivot application automatically updates the results once the AppendData method finishes. To delay updates, use BeginUpdate and EndUpdate methods. You can call the EndUpdate method to update FlexPivot results when you are finished with adding data using connectors.

C#

```
c1FlexPivotPage.FlexPivotPanel.FlexPivotEngine.BeginUpdate();
c1FlexPivotPage.FlexPivotPanel.FlexPivotEngine.EndUpdate();
```

Using FlexPivot Controls with Data Source

This section is intended to get you up and running in creating an application that adds data by binding the FlexPivotPage control to a data source at design-time. This way users can add data to the FlexPivot control at design-time without writing even a single line of code.

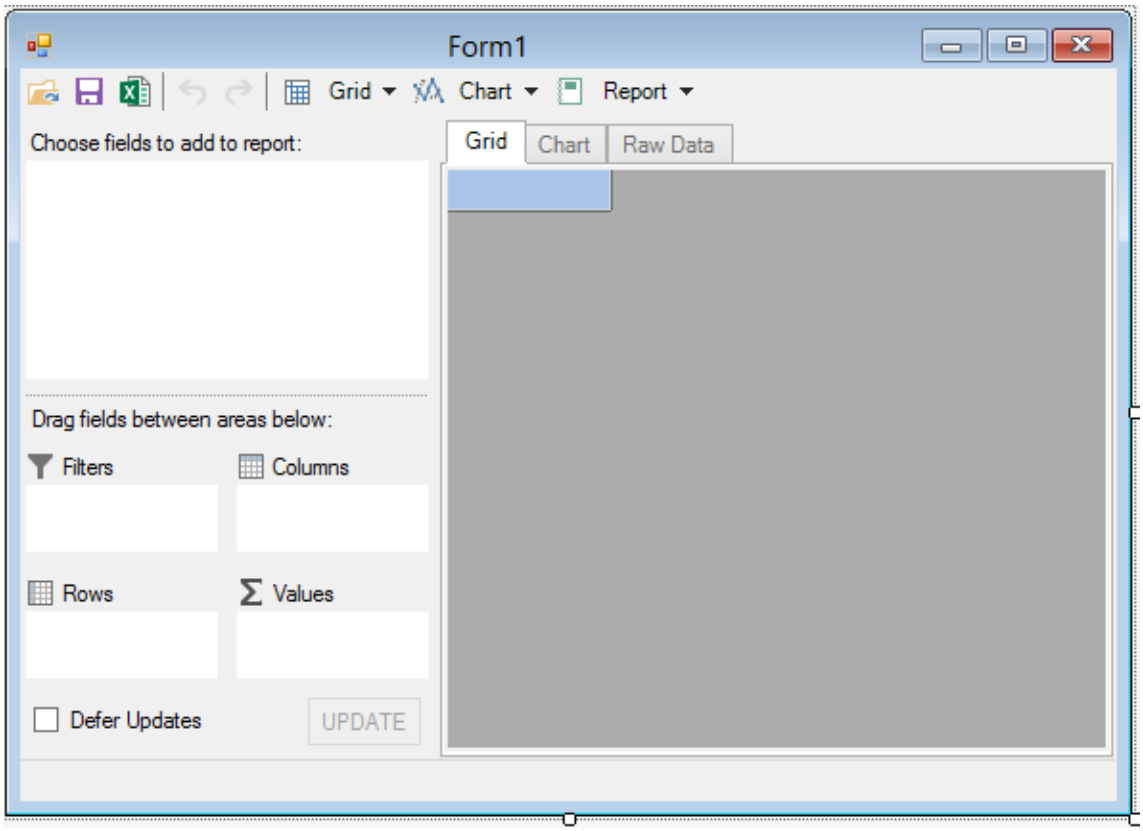
This section also highlights how to bind the FlexPivotPage control to a data source in code.

Binding FlexPivot to Data Source at Design-Time

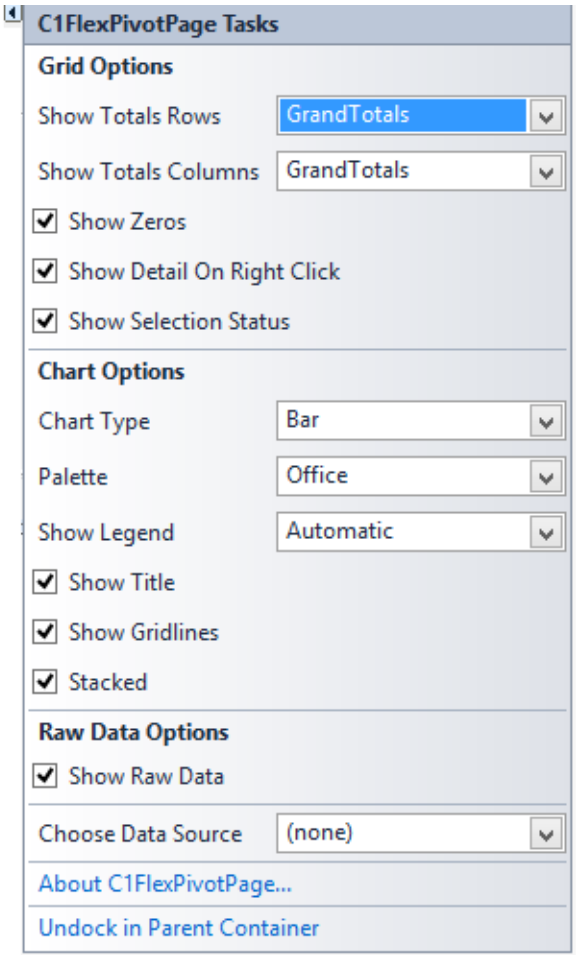
FlexPivot provides an array of default features that can be directly used without writing even a single line of code. You start by adding the FlexPivotPage control to an empty Windows Forms application and binding it to a data source using the Smart tag.

At Design-Time

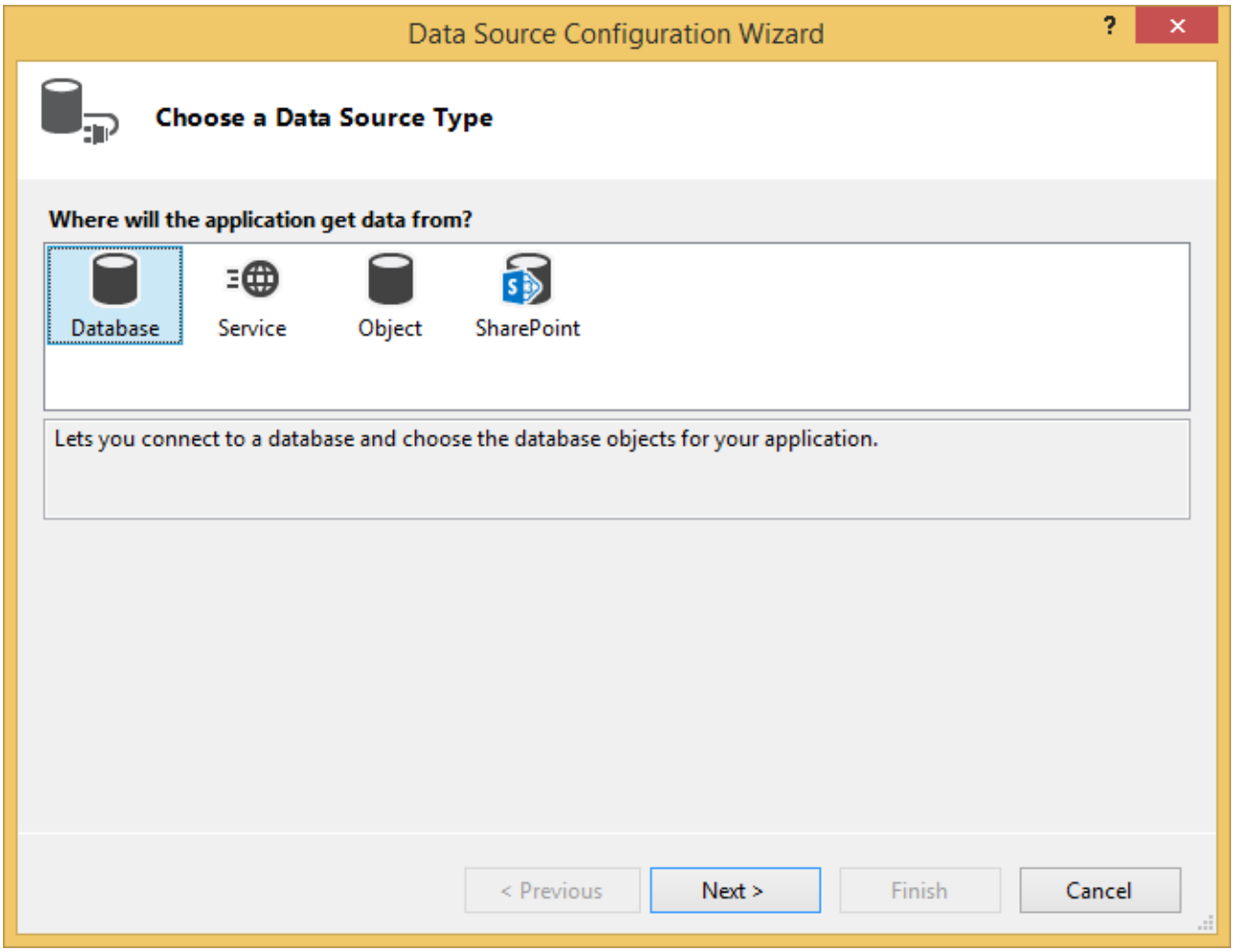
1. Create a new **Windows Forms Application** in Visual Studio.
2. Navigate to the Toolbox and locate the C1FlexPivotPage control icon.
3. Double-click or drag-and-drop the C1FlexPivotPage icon to the Form.
4. The C1FlexPivotPage control docks to fill the form in the designer as shown in the image below.



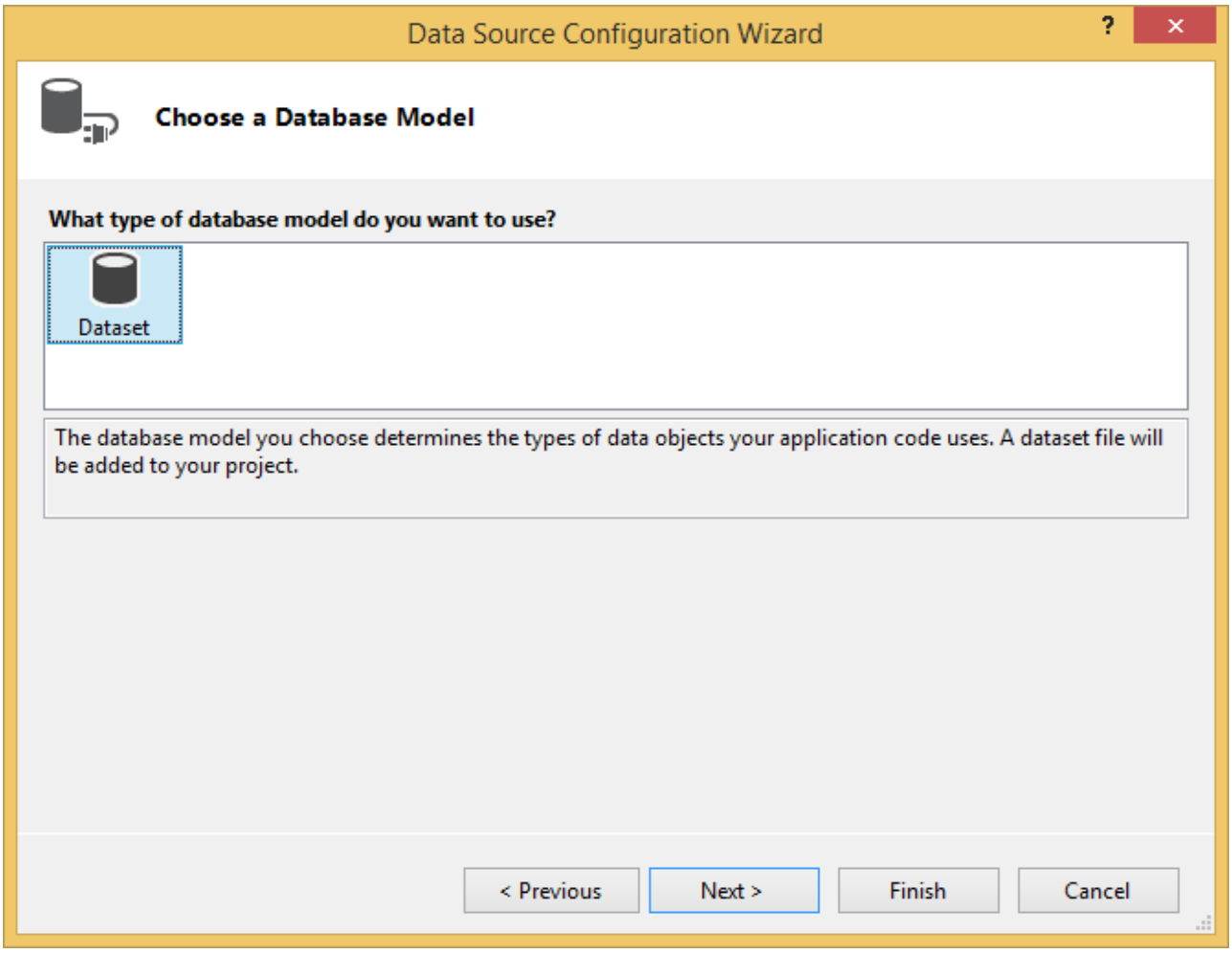
5. Click the smart tag icon (▶) appearing in the upper right corner of the C1FlexPivotPage control. The C1FlexPivotPage Tasks smart tag panel appears similar to the image below.



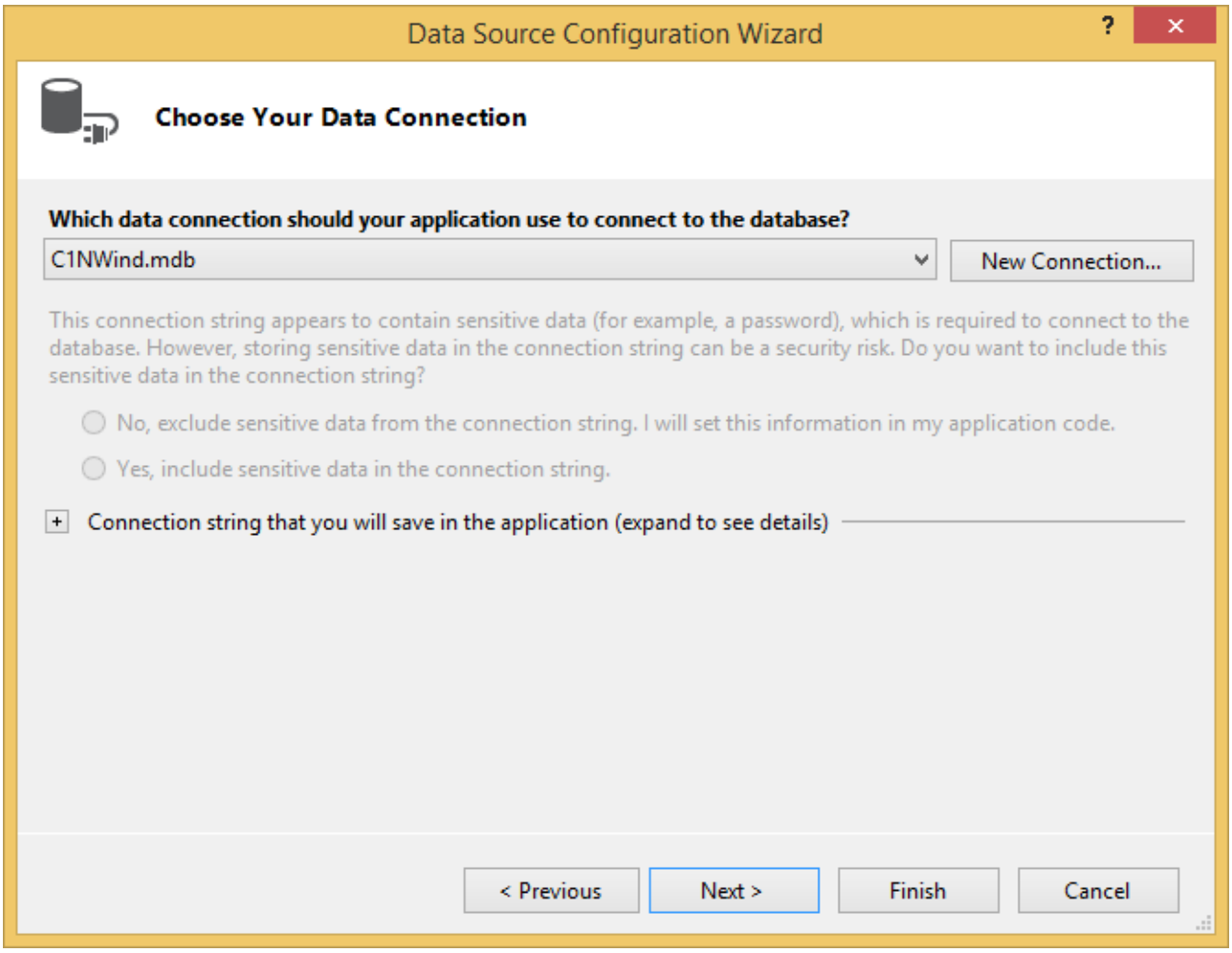
6. Click the drop-down menu appearing against the **Choose Data Source** option in the C1FlexPivotPage Tasks smart tag panel.
7. Click the 'Add Project Data Source' link to open **Data Source Configuration Wizard** window.
8. Select Database as the Data Source type in the **Data Source Configuration Wizard** window and click Next.



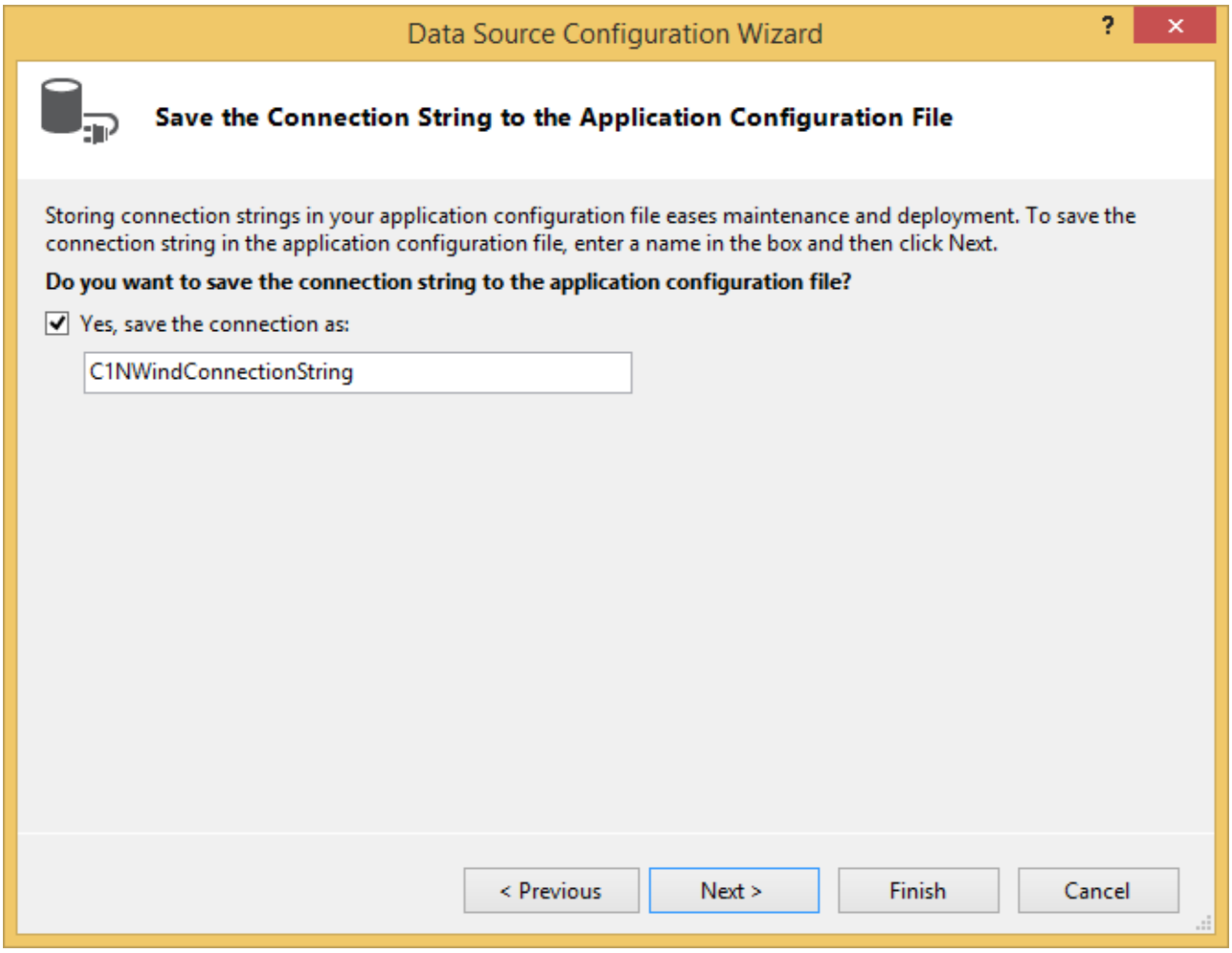
9. Choose Dataset as the database model in the **Data Source Configuration Wizard** window and click Next.



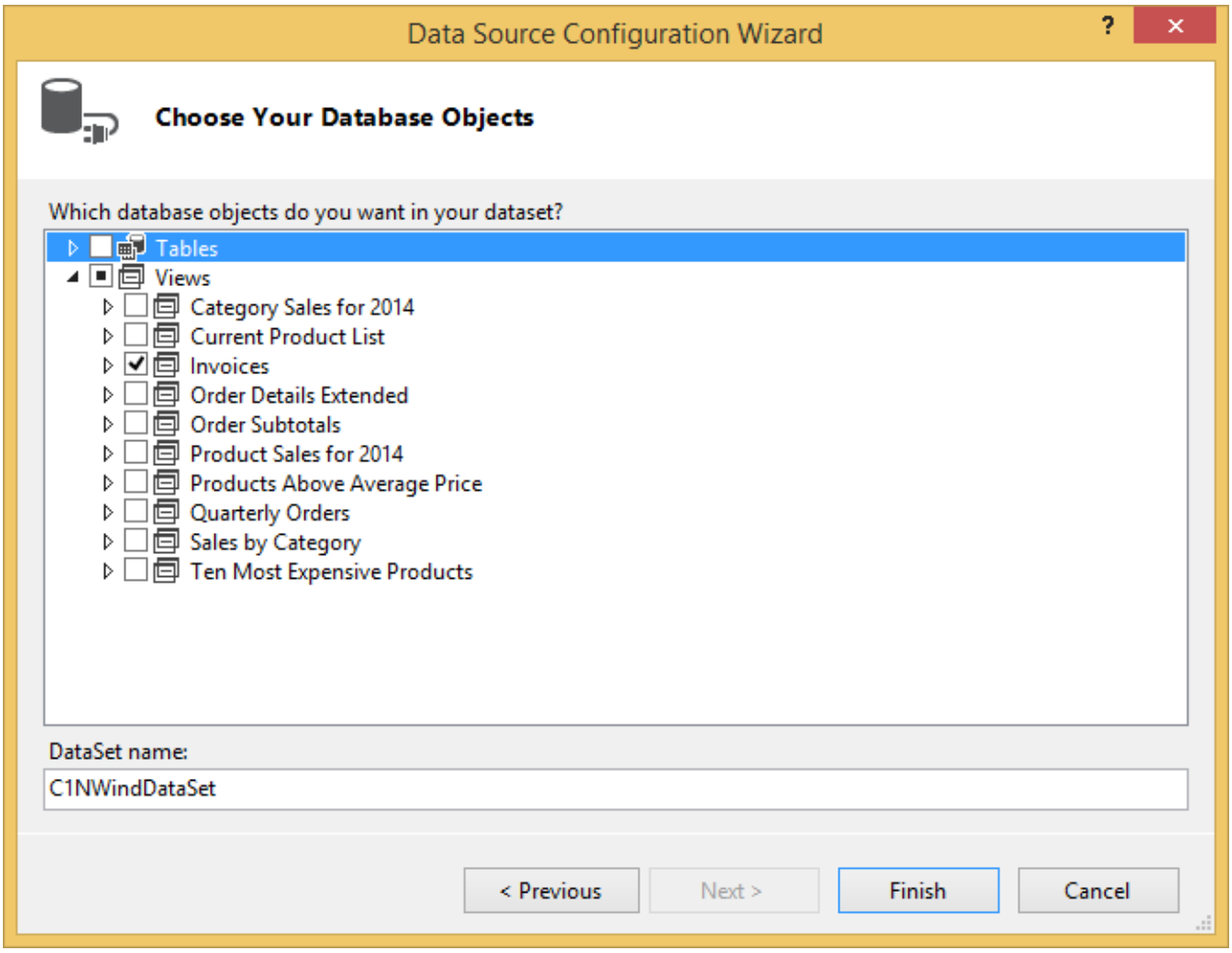
10. Choose data connection string for your application to connect to the database and click Next. In this example, we are using C1NWind.mdb file as the data source. This file is kept at **Documents\ComponentOne Samples\Common\C1NWind.mdb** location on your system.



11. Save the connection string by selecting the checkbox and click Next.



12. Choose database objects from the **Data Source Configuration Wizard** window that you want in your dataset. In this example, we are selecting Invoices view for data binding as shown in the image. Click Finish to complete the data binding.



13. Press F5 to run the application. FlexPivotPage control is now connected to the C1NWind.mdb file and you can see that the control appears with various data fields available in the FlexPivotPanel.

You can now create different views by dragging the data fields to Rows, Columns and Values list for data analysis. Click here to know how to create different views.

Binding FlexPivot to Data Source in Code

You can also bind the FlexPivot controls to a data source kept locally at your machine in code. Follow the given steps to bind the FlexPivotPage control to data source (c1nwind.mdb) in code.

In Code

1. Create a Windows Forms Application project in Visual Studio.
2. Drag-and-drop C1FlexPivotPage control to the Form.
3. Switch to the code view and add the following code to initialize a standard connection string.
   - **Visual Basic**

```
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
```

   - **C#**

```
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);
}
```

   This code creates a connection with the **c1nwind.mdb** database file installed on your system at **Documents\ComponentOne Samples\Common** location on your system.

4. Add the following code in the Form's constructor to fetch data from the data source through Oledb adapters.

- **Visual Basic**

```
' load data
Dim da = New OleDbDataAdapter("select * from invoices", GetConnectionString())
Dim dt = New DataTable()
da.Fill(dt)
```

- **C#**

```
// load data
var da = new OleDbDataAdapter("select * from invoices", GetConnectionString());
var dt = new DataTable();
da.Fill(dt);
```

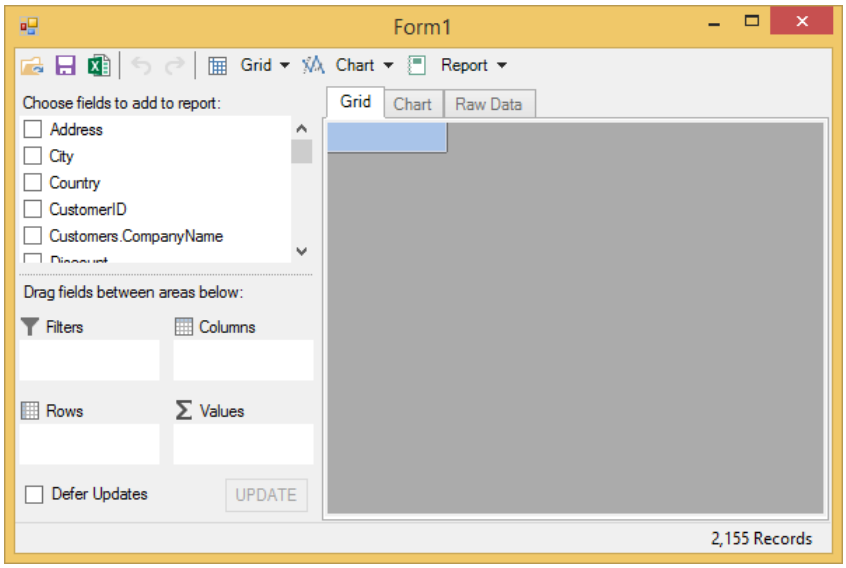
5. Bind the FlexPivotPage control to the data source using DataSource property of C1FlexPivotPage class in the Form's constructor.
   - **Visual Basic**

```
' bind data
Me.C1FlexPivotPage1.DataSource = dt
```

   - **C#**

```
// bind data
this.c1FlexPivotPage1.DataSource = dt;
```

6. Press **F5** to run the application. FlexPivotPage is now connected to the c1nwind.mdb file with various data fields available in the FlexPivotPanel.



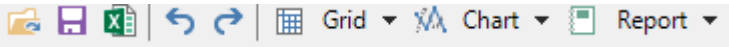
You can now create different views by dragging the data fields to Rows, Columns and Values list for data analysis.

Using FlexPivotPage ToolStrip

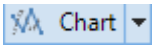
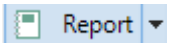
The C1FlexPivotPage control provides a ToolStrip that you can use to:

- load or save a C1FlexPivotPage as an .xml file.
- display data in a grid or chart.
- setup and print a report.

The ToolStrip appears similar to the image below.

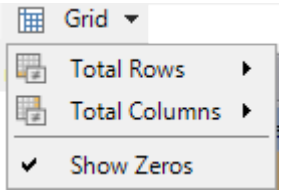


The following tables describes the buttons that appear in the ToolStrip and their purpose.

Button		Description
Load		Load a FlexPivot view from a file.
Save		Save a FlexPivot view to a file.
Export		Export data appearing on the view to .xlsx format.
Undo		Cancel the last action performed in C1FlexPivotPage**.**
Redo		Perform the last action(s) cancelled using the **Undo** button.
Grid		Choose the columns and rows to display data in the C1FlexPivotGrid**.**
Chart		Customize the chart that displays the selected data. You can determine: the chart type, the palette or theme using this button. You can also determine whether the title appears, whether the chart is stacked, and whether the gridlines appear or not.
Report		Customize the report by specifiying a header or footer for each page; determine what to include in the report, the grid, the chart, or raw data grid; specify the page layout, including its orientation, paper size, margins, etc.; preview the report before printing; and print the report.

Grid Menu

The **Grid** menu appears similar to the image below and provides three options listed in the table below.

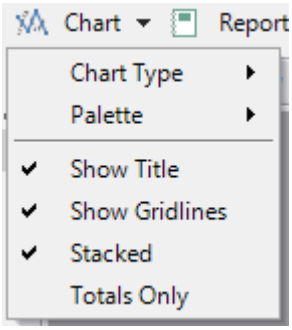


Total Rows	Allows you to choose from **Grand Totals**, **Subtotals**, or **None**.
Total Columns	Allows you to choose from **Grand Totals**, **Subtotals**, or **None**.
Show Zeros	If checked, shows any cells containing zero in the grid.

Simply uncheck any of these items to hide the total rows, total columns, or any zeros in the grid.

Chart Menu

From the **Chart** menu, you can determine: the chart type, the palette, whether to show the chart title above the chart, whether to show a stacked chart, whether to show chart gridlines, and whether to show totals only.

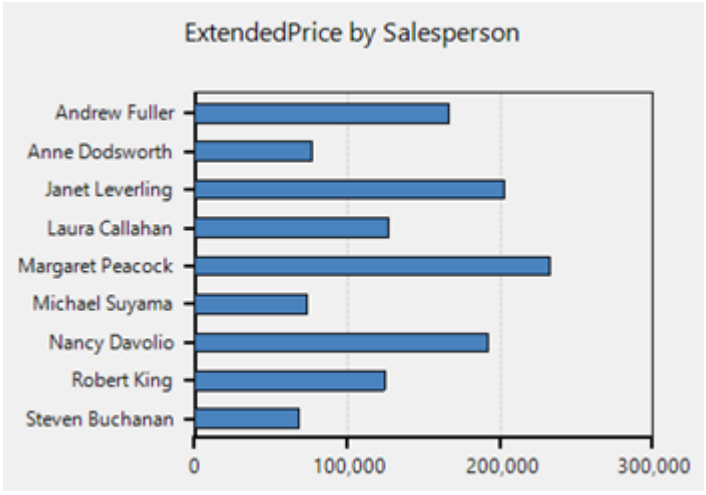


The following table illustrates the option available in the **Chart** drop-down menu.

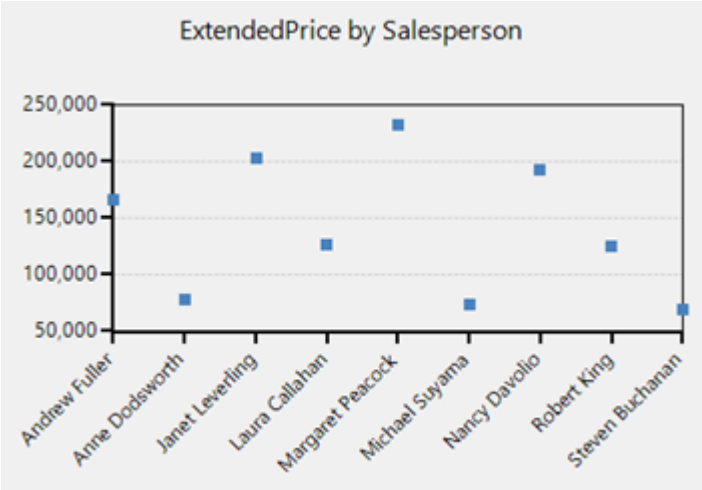
Chart Type	Click **Chart Type** to select from five common chart types shown below.
Palette	Click **Palette** to select from twenty-two palette options that define the colors of the chart and legend items. See the options in the **Palette** topic below.
Show Title	When selected, shows a title above the chart.
Stacked	When selected, creates a chart view where the data is stacked.
Show Gridlines	When selected, shows gridlines in the chart.
Totals Only	When selected, shows only totals as opposed to one series for each column in the data source.

Chart Types

FlexPivot offers five common chart types that are shown in the table below.

Bar	

Column	
Area	
Line	

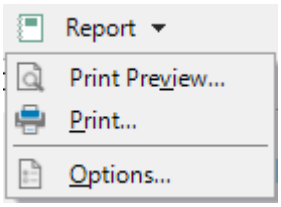
Scatter	

Palette

The FlexPivotChart controls comes with 22 different palette options that offer different color schemes to present the data chart more dynamically. Users can choose the from these palettes as per their styling requirements.

Report Menu

The **Report Menu** provides you with options to preview or print the report, setup report pages, add header and/or footers, and specify items to be shown in the report. On clicking the **Report** button available in the toolstrip, a context menu appears with Print Preview, Print and Options as follows.



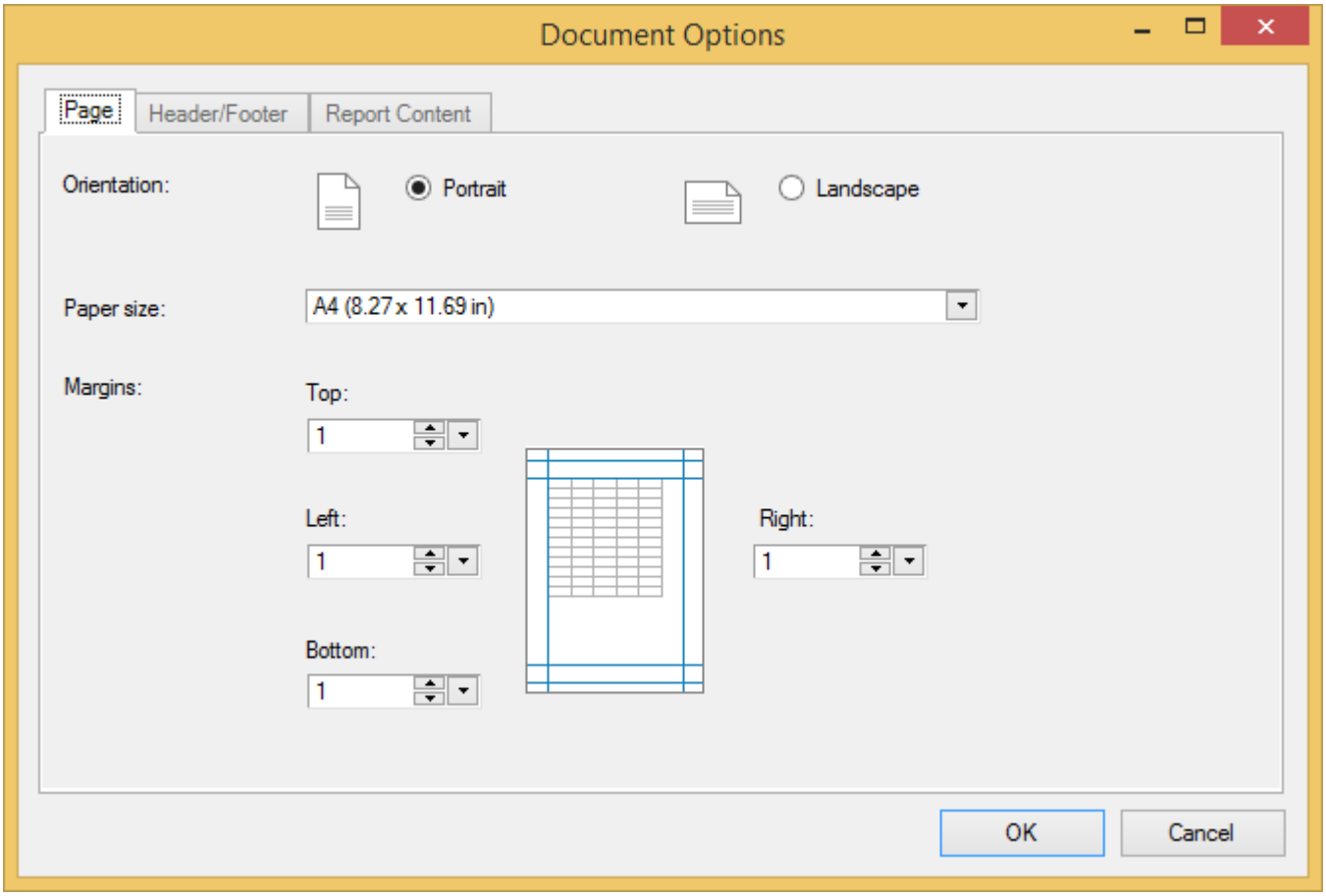
The following table illustrates the options available in the Report context menu.

Print Preview	Select Print Preview to preview your report before printing or to export to a PDF file.
Print	Click **Print** to print the C1FlexPivotGrid, C1FlexPivotChart, or both.
Options	Click **Options** to open the **Document Options** dialog box.

Document Options

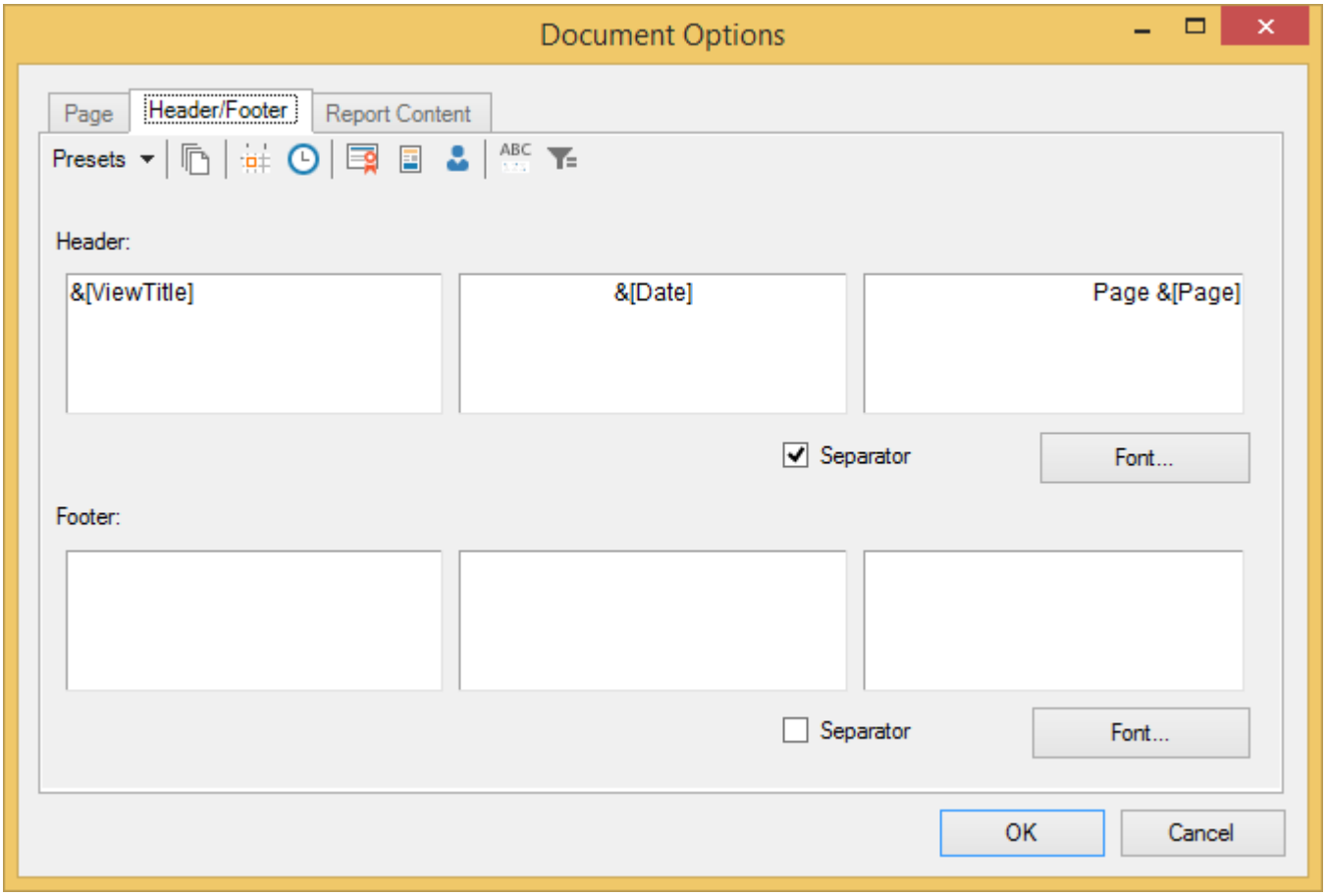
Page Tab

The page tab allows you to specify the Orientation, Paper Size, and Margins. See the image below to visualize the option more clearly.



Header/Footer Tab

The header/footer tab allows you to add a header and/or footer to each page of the report. See the image below to visualize the option more clearly.



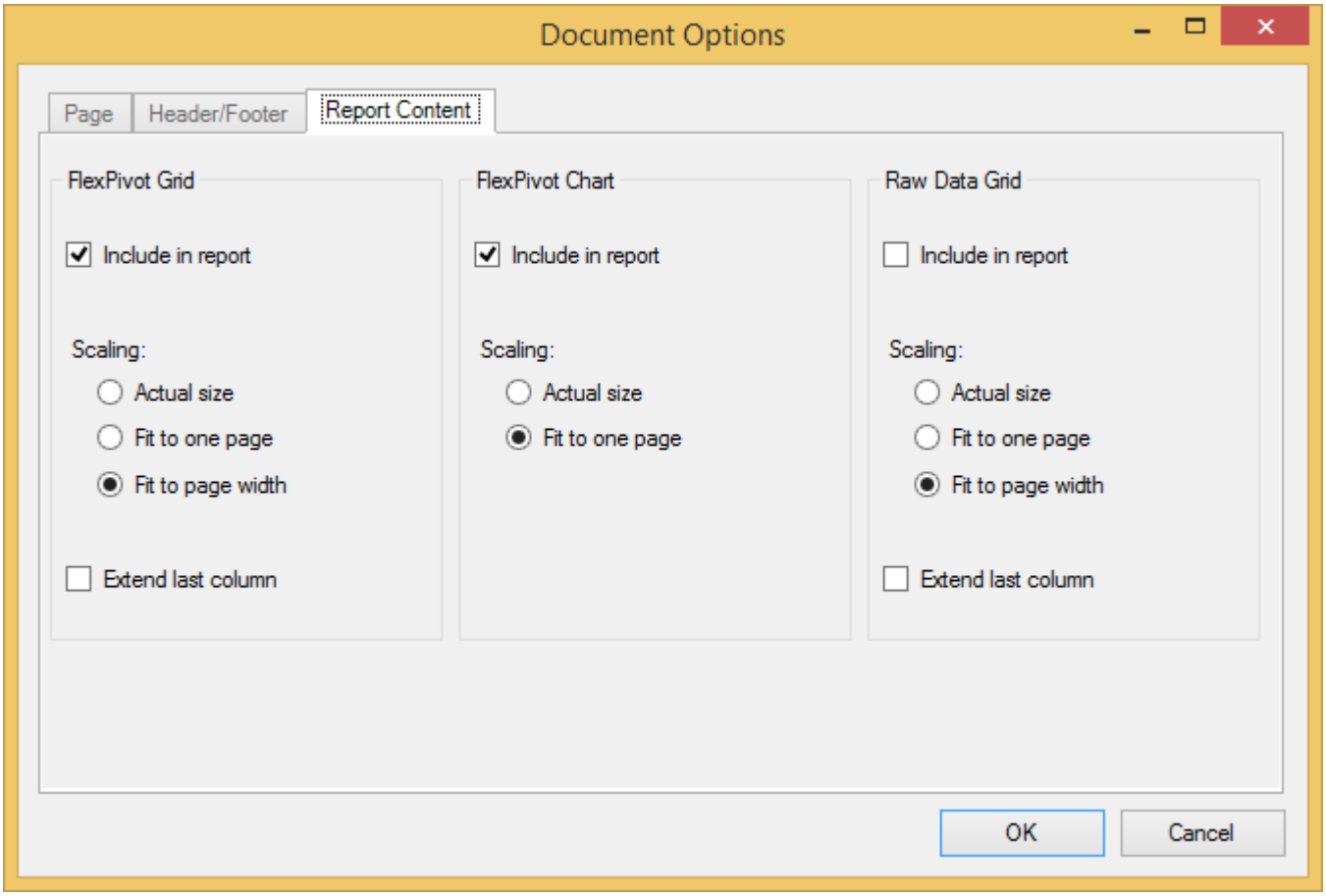
Click one of the buttons on the toolbar to insert fields into the header or footer.

Button	Field
Presets	Choose from the three predefined options containing groups of fields to be inserted in the header or footer.
Page Number	&[Page]
Current Date	&[Date]
Current Time	&[Time]
Document Name	$[DocName]
View Description	&[ViewTitle]
Author Name	&[UserName]

Check the **Separator** box to show a separator line below the header or above the footer. Click the **Font** button to change the font, style, size, or effects.

Report Content Tab

On the **Report Content** tab, you can determine whether to include the grid, chart, and/or raw data grid in your report. You can also scale the items as desired and extend the last columns of the grids.



Data Blending Features

Designed as powerful data analysis tools to deliver high-end business intelligence capabilities, the C1FlexPivot controls can be used for various data blending operations such as filtering, grouping, sorting, drill-down, applying conditional formatting, creating interactive reports, etc. Once connected to data (Invoices view of C1NWind.mdb file), users can implement all these data blending features in the FlexPivot controls as illustrated in this section.

Joining

Using the C1DataEngine, you can combine (join, blend) data from different data sources (tables) into a single table and analyze the combined data in FlexPivot. See Using C1DataEngine > Join to know in detail how to join multiple data sources.

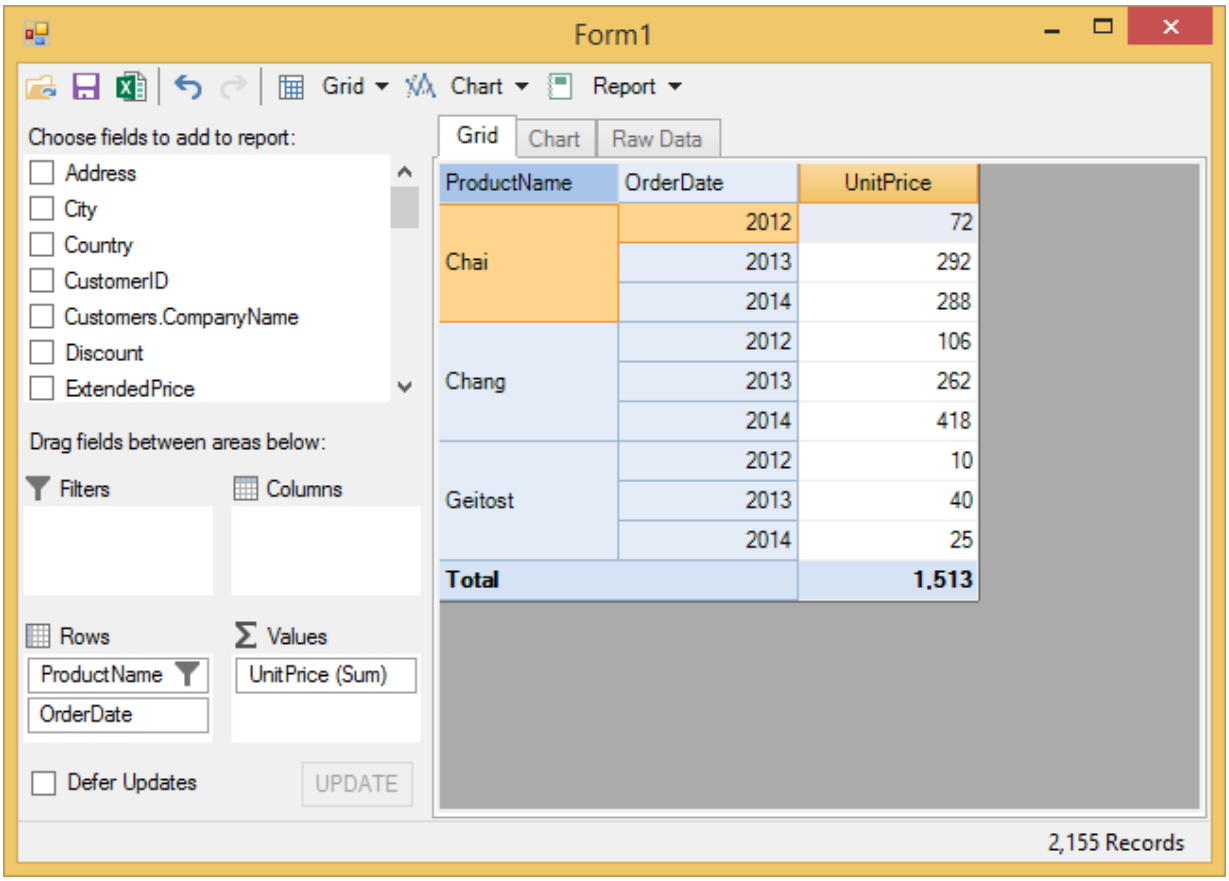
> Refer to the product sample DataJoin stored at the following location in your system to see how to build a join query dynamically based on our user selection of table and fields:
>
> ***C:\...\Documents\ComponentOne Samples\WinForms\C1FlexPivot\CS or VB\DataJoin.***

Grouping

Grouping enables users to organize data in groups based on certain set of criteria. The FlexPivotGrid control lets users group data based on specific conditions.

Consider a user wants to analyze the Unit Price for three consecutive years, that is 2012, 2013 and 2014, of a few products. The user wants to know these price points so that he can maximize his profits by making some adjustments in the Unit Price. In such a case, the user can firstly group the data by Order Date and then format data to obtain yearly figures.

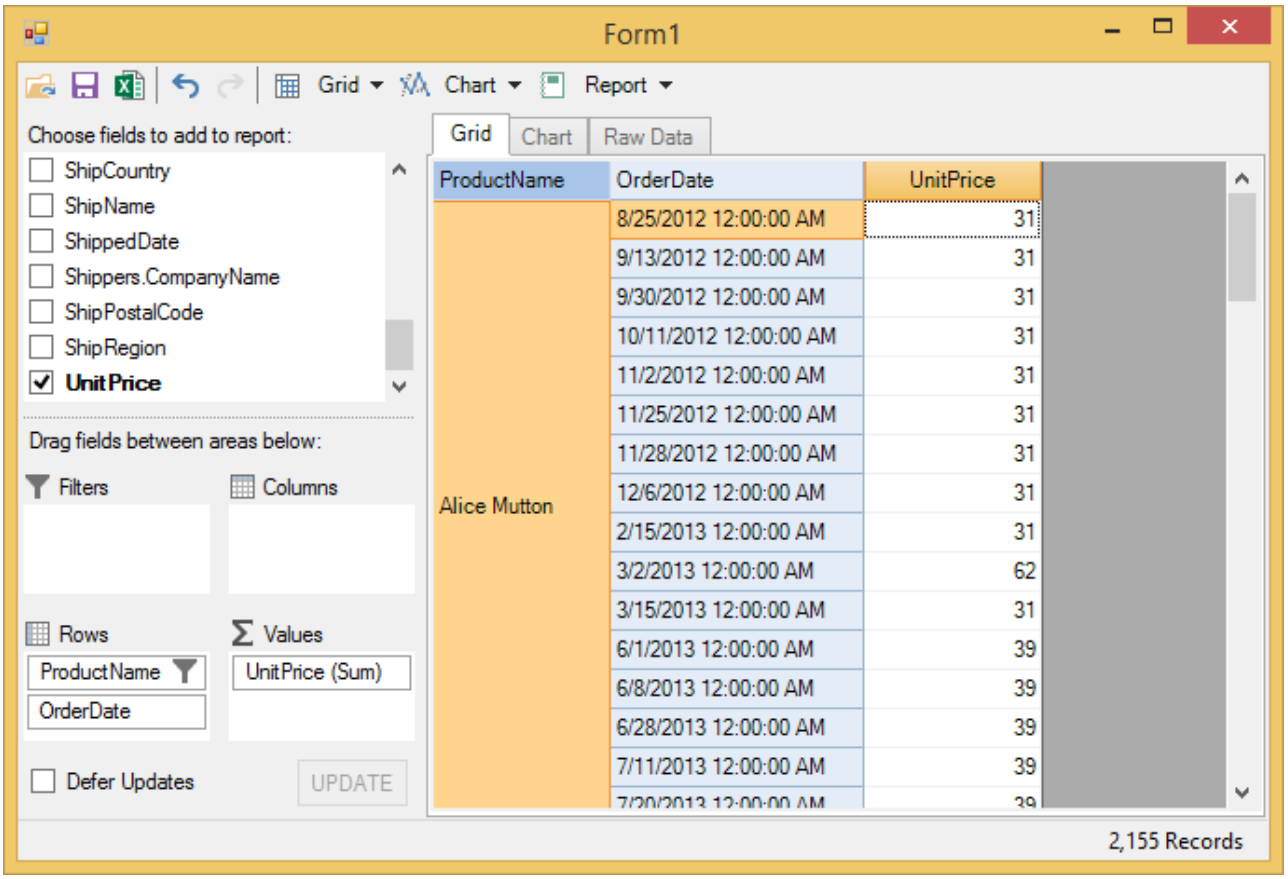
The image given below shows a FlexPivotGrid displaying the Unit Price of three products, namely Chai, Chang and Geitost.



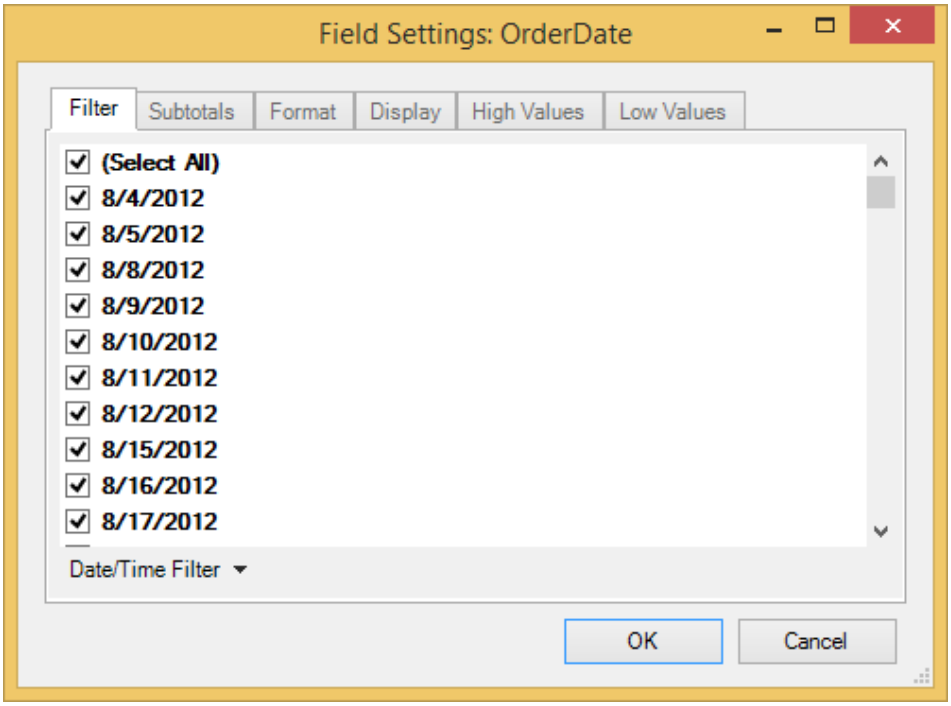
Complete the following steps to implement grouping in C1FlexPivotGrid control. This implementation uses the sample created in Binding FlexPivot to Data Source in Code topic.

At Runtime

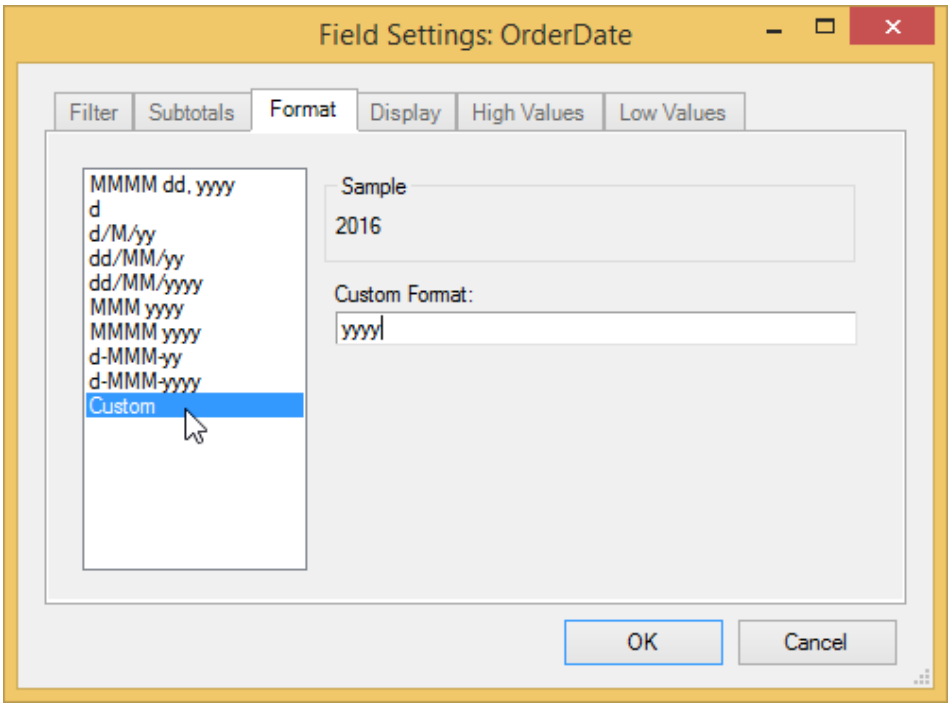
1. Drag-and-drop **ProductName** and **OrderDate** fields in the **Rows** list, and **UnitPrice** in the **Values** list.
2. The data before grouping appears similar to the image given below.



3. Right-click the OrderDate field from the Rows list and select Field Settings option from the context menu. The Field Settings dialog for OrderDate field appears.



4. Click the Format tab and select Custom format**.** Enter "yyyy" in the Custom Format textbox and click OK.



5. Right-click the ProductName field from the Rows list and select Field Settings option from the context menu.
6. Select Chai, Chang and Geitost products in the filter tab and click OK.

In Code

You can also group the data in code. Use the following code to apply the same grouping as above.

Visual Basic

```
Dim fp = Me.c1FlexPivotPage1.FlexPivotEngine
fp.RowFields.Add("ProductName", "OrderDate")
fp.ValueFields.Add("UnitPrice")

Dim field = fp.Fields("OrderDate")
field.Format = "yyyy"

Dim filter As C1.FlexPivot.C1FlexPivotFilter =
fp.Fields("ProductName").Filter
filter.Clear()
filter.ShowValues = "Chai,Chang,Geitost".Split(","C)
```

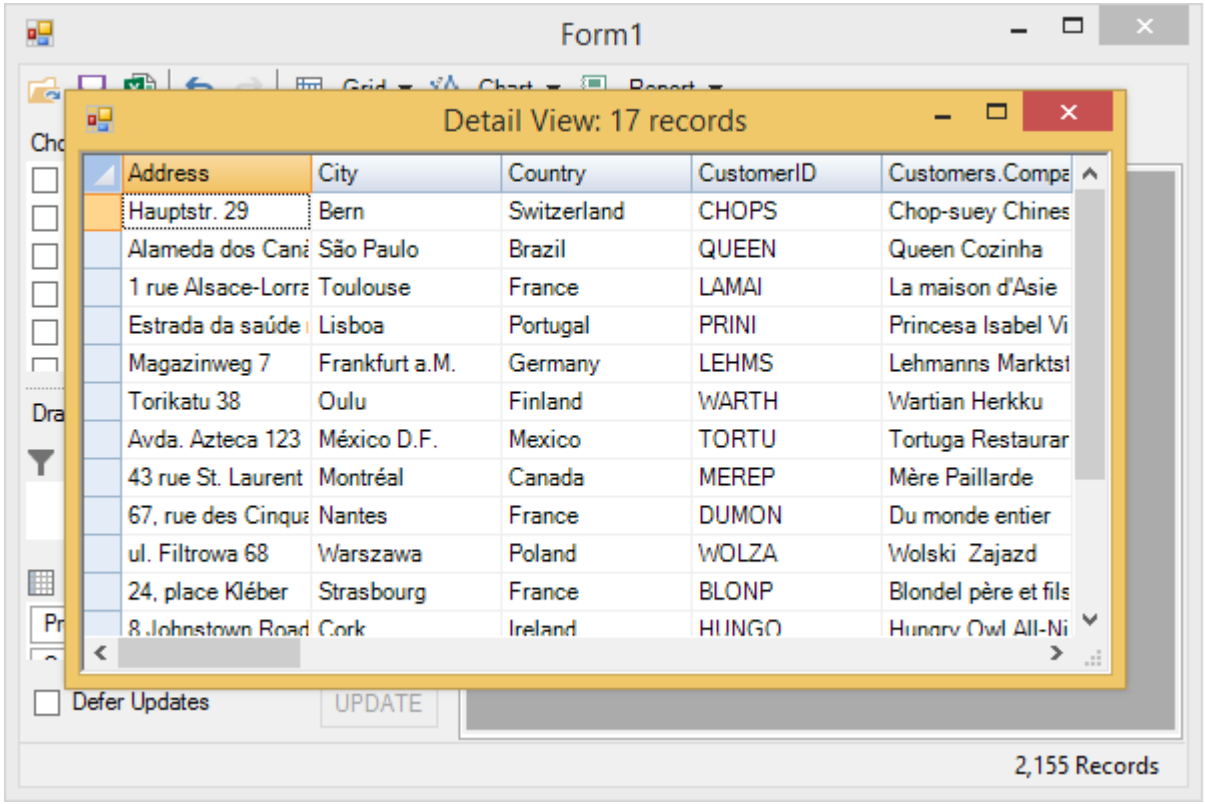
C#

```
var fp = this.c1FlexPivotPage1.FlexPivotEngine;
fp.RowFields.Add("ProductName", "OrderDate");
fp.ValueFields.Add("UnitPrice");

var field = fp.Fields["OrderDate"];
field.Format = "yyyy";

C1.FlexPivot.C1FlexPivotFilter filter = fp.Fields["ProductName"].Filter;
filter.Clear();
filter.ShowValues = "Chai,Chang,Geitost".Split(',');
```

Drilling Down Data

Drilling down feature provides users with access to a structured database placed at a lower level in the hierarchy. The FlexPivotGrid control enables users in drilling down the database to see the underlying records below each aggregate value appearing in the grid. Users can view the underlying data by simply double-clicking any cell.

1. In the Grid tab, right-click the first cell appearing under the **UnitPrice** column.
2. You notice a new grid appears with a summary of records displayed as follows.

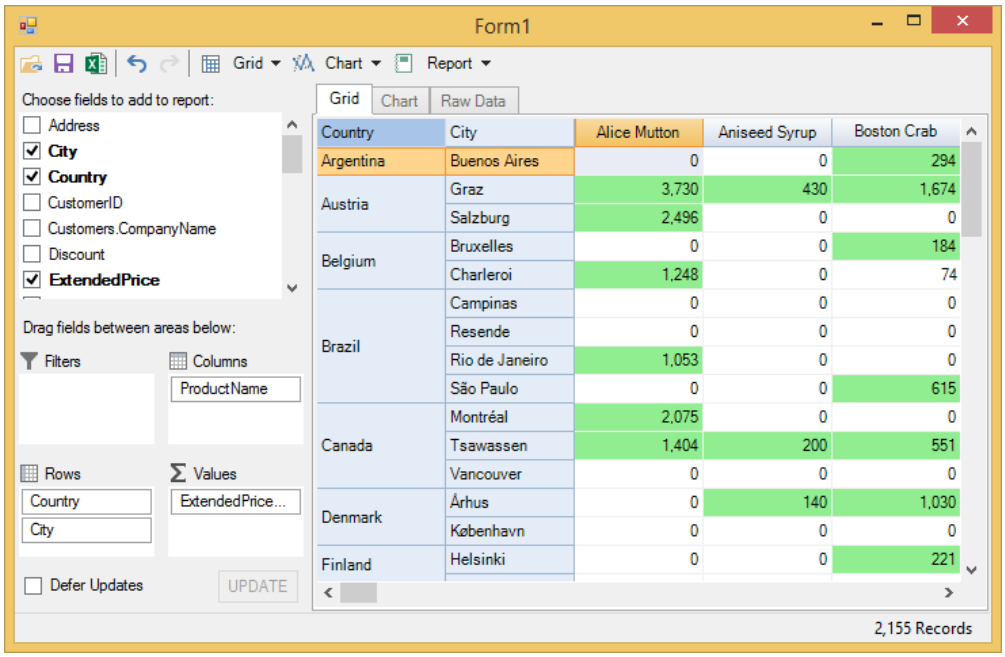


Applying Conditional Formatting

Conditional formatting enables users to highlight cells with a certain color, depending upon the cell's value. The FlexPivotGrid control lets users apply conditional formatting to an individual cell or a range of cells to let them visualize data more clearly for analysis and comparison. The C1FlexPivotGrid class extends this functionality from the OwnerDraw feature of C1FlexGrid.

Let's say a user wants to analyze the variation in Extended Price of products on the basis of Geography, that is City and Country. The user wants to know the cities in which the Extended Price of the product is less than 100 so that he/she can make price adjustments in future. In such a case, the user can apply conditional formatting to highlight and compare Extended Price.

The image given below shows a FlexPivotGrid highlighting Extended prices greater than 100 and segregated by Product, Country and City.



Implementation

Complete the following steps for applying conditional formatting to C1FlexPivotGrid. This implementation uses the sample created in Binding FlexPivot to Data Source in Code topic.

1. Create an instance of CellStyle class, and initialize a constant field Value with 100 in code view.
   - **Visual Basic**

```
Private cellValue As CellStyle
Const Value As Integer = 100
```

   - **C#**

```
CellStyle cellValue;
const int Value = 100;
```

2. Add the following code in Form's constructor to create a default view that displays **Extended Price** in Values list, **Product Name** in Columns list, **City** and **Country** fields in Columns list.
   - **Visual Basic**

```
Dim fp = Me.c1FlexPivotPage1.FlexPivotEngine
fp.ValueFields.Add("ExtendedPrice")
fp.RowFields.Add("Country", "City")
fp.ColumnFields.Add("ProductName")
```

   - **C#**

```
var fp = this.c1FlexPivotPage1.FlexPivotEngine;
fp.ValueFields.Add("ExtendedPrice");
fp.RowFields.Add("Country", "City");
fp.ColumnFields.Add("ProductName");
```

3. Add the following code to the Form's constructor for configuring grid and styling grid cells.
   - **Visual Basic**

```
' configure grid
Dim grid = Me.c1FlexPivotPage1.FlexPivotGrid

' style used to show 'big values'
cellValue = grid.Styles.Add("cellValue")
cellValue.BackColor = Color.LightGreen

' owner draw to apply the style
grid.DrawMode = DrawModeEnum.OwnerDraw
```

   - **C#**

```
// configure grid
var grid = this.c1FlexPivotPage1.FlexPivotGrid;

// style used to show 'big values'
cellValue = grid.Styles.Add("cellValue");
cellValue.BackColor = Color.LightGreen;

// owner draw to apply the style
grid.DrawMode = DrawModeEnum.OwnerDraw;
```

4. Subscribe **grid_OwnerDrawCell** event to apply conditional formatting.
   - **Visual Basic**

```
AddHandler grid.OwnerDrawCell, AddressOf grid_OwnerDrawCell
```

   - **C#**

```
grid.OwnerDrawCell += grid_OwnerDrawCell;
```

5. Add the following code to the event handler created for **grid_OwnerDrawCell** event.
   - **Visual Basic**

```
Private Sub grid_OwnerDrawCell(sender As Object, e As OwnerDrawCellEventArgs)
    Dim grid = TryCast(sender, C1.Win.C1FlexGrid.C1FlexGrid)
    If e.Row >= grid.Rows.Fixed AndAlso e.Col >= grid.Cols.Fixed AndAlso TypeOf grid(e.Row, e.Col) Is Double Then
        Dim value__1 = CDbl(grid(e.Row, e.Col))
        If value__1 > Value Then
            e.Style = cellValue
        End If
    End If
End Sub
```

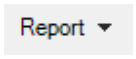
- **C#**

```
private void grid_OwnerDrawCell(object sender, OwnerDrawCellEventArgs e)
{
    var grid = sender as C1.Win.C1FlexGrid.C1FlexGrid;
    if (e.Row >= grid.Rows.Fixed &&
        e.Col >= grid.Cols.Fixed &&
        grid[e.Row, e.Col] is double)
    {
        var value = (double)grid[e.Row, e.Col];
        if (value > Value)
        {
            e.Style = cellValue;
        }
    }
}
```

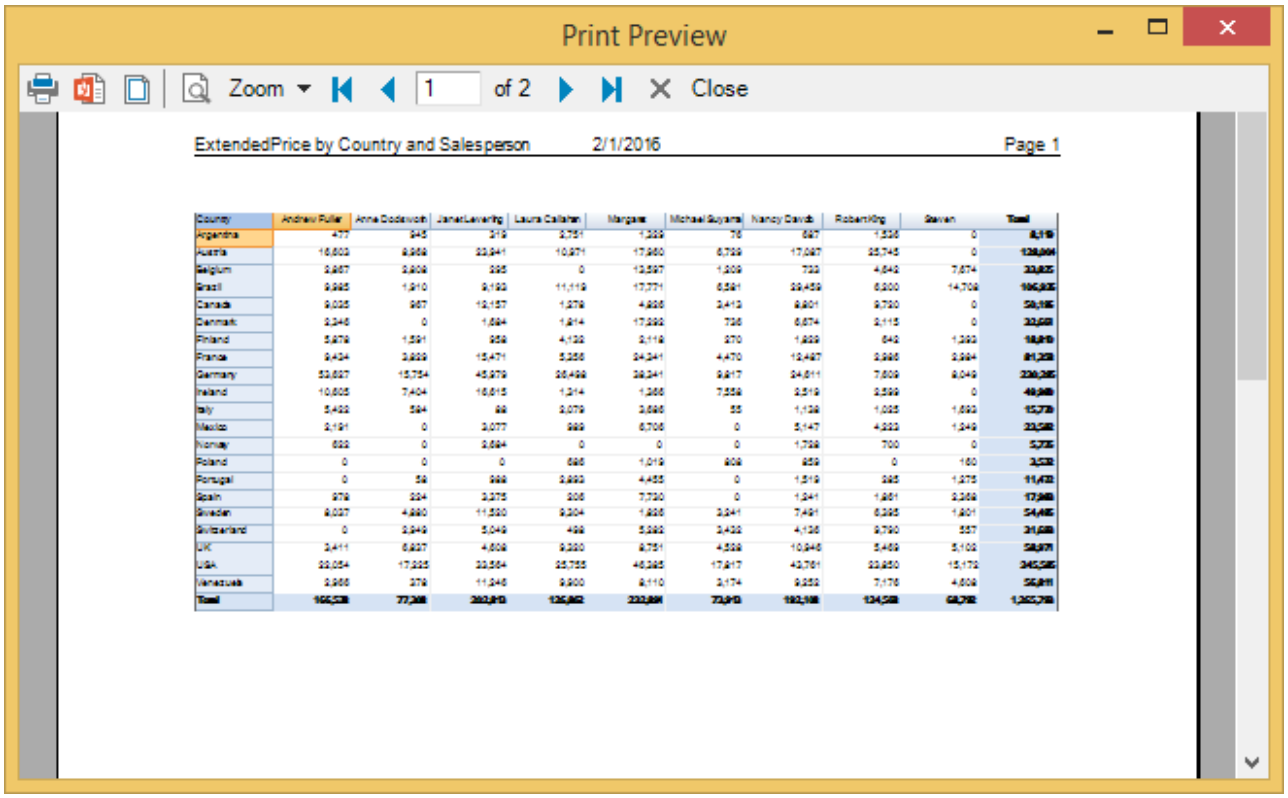
6. Run the application to see that the extended prices greater than 100 highlighted in green color.

Creating Reports with FlexPivot

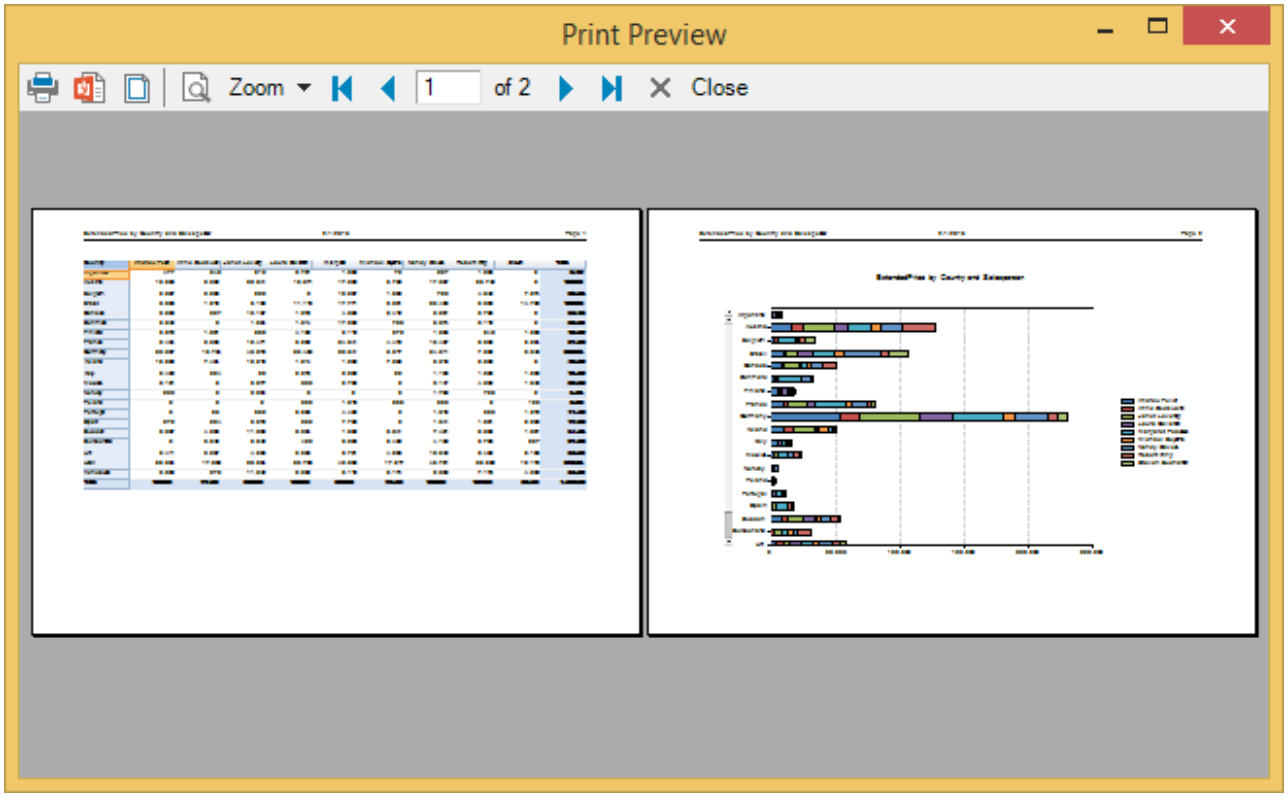
FlexPivot includes report design functionality so that the end users can make full use of full-featured reports to cater diverse business needs. The report design functionality enables end users to share the report generated by the control with others as well. The generated report shows data in tabular format on one page and a chart representation on the other as depicted in the image in Step 2 below.

1. Click the **Report** 

button appearing in the toolstrip on the control's interface.
2. A Print Preview dialog box appears similar to the image below.



3. In the Print Preview dialog, click the **Page Setup** button and change the orientation to **Landscape**, and Click the Zoom drop-down menu and select Two-Pages option. The report appears similar to the image below.



You can print the generated report by clicking the **Print Document** button appearing on the toolstrip. Users can also export the generated report in PDF format using the **Export to PDF** button on the toolstrip. The exported PDF format report can be easily shared with others through e-mail or can be hosted on the web.

Copying Data to Excel

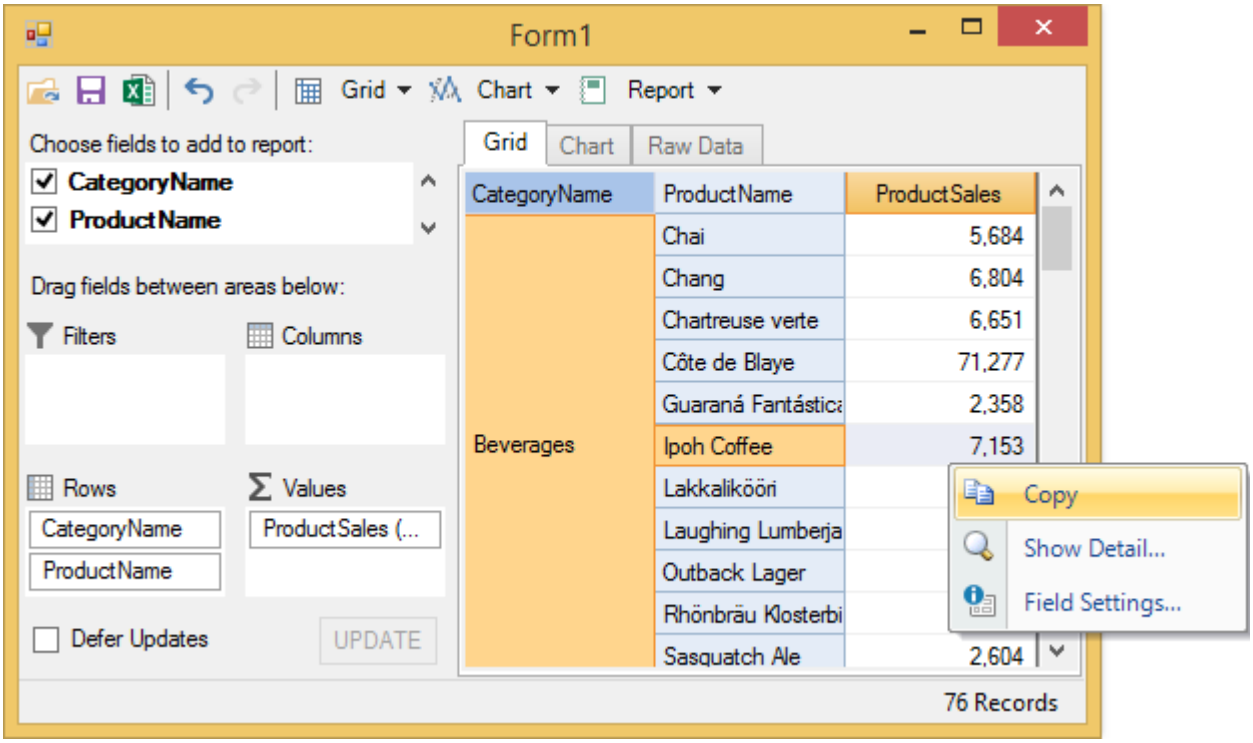
Though the built-in reports are convenient and handy to use, end users may wish to copy the data appearing on the FlexPivotGrid to an Excel sheet for performing additional analysis such as regression. Besides, end users may also want to create customized reports by annotating the data or using custom charts.

To cater such needs, the C1FlexPivotGrid control comes with default clipboard support that allows users to copy and paste data on an Excel sheet. Modeled on ComponentOne's popular FlexGrid control, the C1FlexPivotGrid enables users in directly copying and pasting the data using **Ctrl+C** and **Ctrl+V** shortcuts, respectively.

Complete the following steps to copy the data to an Excel sheet.

1. Select the entire data appearing on the FlexPivotGrid by clicking once on the very first cell of the grid.
2. Press Ctrl+C to copy the entire data.
3. Open an Excel sheet on your system.
4. Press Ctrl+V to paste the data. The data gets pasted in the Excel sheet.

FlexPivotGrid also displays a context menu at runtime that provides various options including copying data. Right-click on the grid to open the context menu at runtime as shown in the image below.



The context menu provides the following options:

Options	Description
Copy	Copies the data appearing in the grid
Show Detail	Drills down the record on the selected value
Field Settings	Opens the Field Settings Dialog

Filtering

Filtering is one of the core features available in the FlexPivot control. The filtering feature enables users to display a specified set of data as per the filters applied on the fields.

FlexPivot supports two types of filters, that are **Value Filter** and **Range Filter**. The Value Filter allows users to filter specific values in a list. The Range Filter allows users to filter data based on some condition(s). The two filters are independent of each other, and values must pass both filters to be included in the grid and chart.

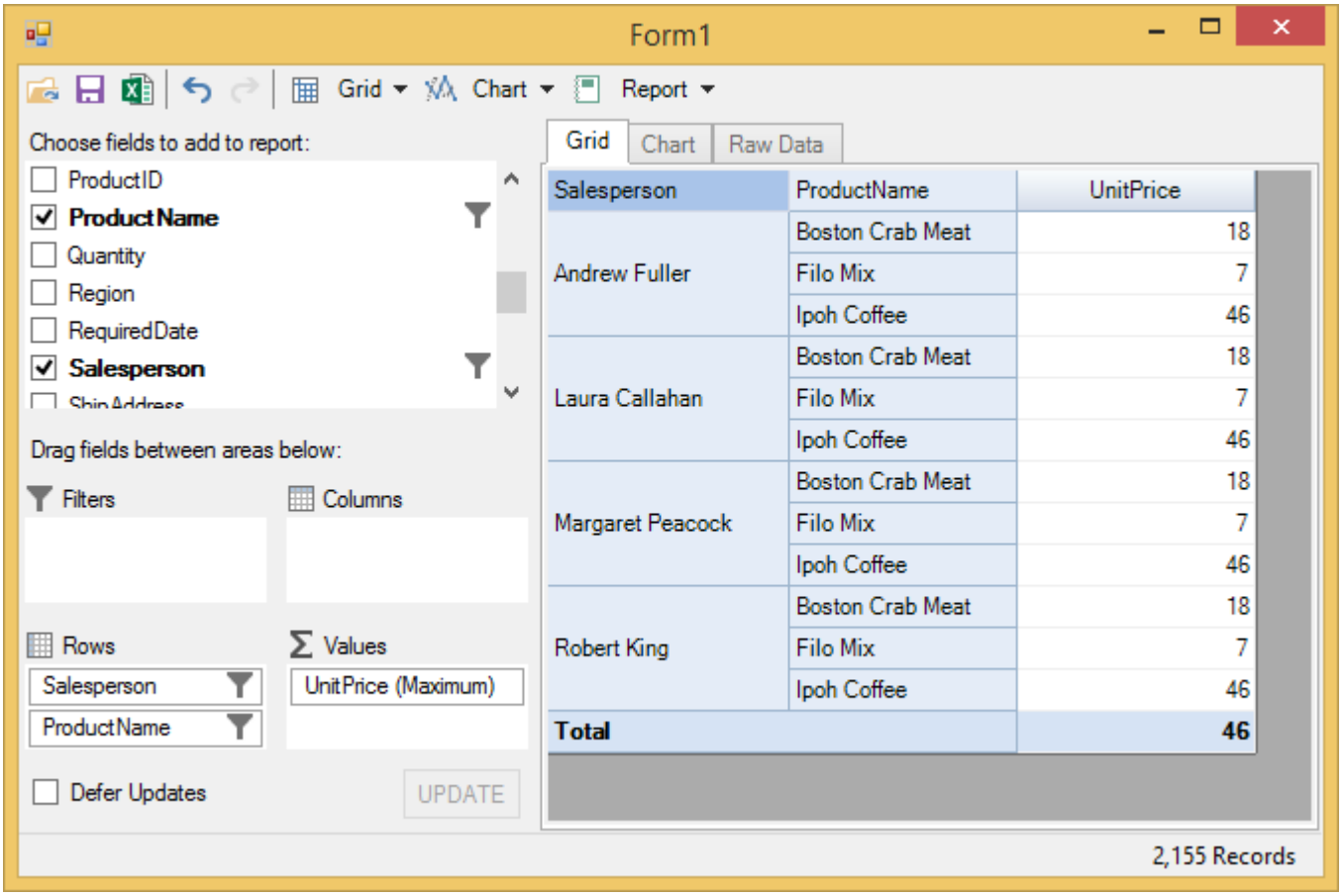
To know how to apply these filters on fields in code behind, see the topic Configuring Fields in Code that describes implementation similar to the above example through code.

Using Value Filters

Let's say a user wants to view the maximum unit prices of three products namely Boston Crab Meat, Filo Mix and Ipoh Coffee offered by four salespersons (Andrew Fuller, Laura Callahan, Margaret Peacock and Robert King). For this, the user needs to slice the Salesperson and Product Name fields in the database using value filters.

The image given below shows a FlexPivotGrid displaying the maximum unit price of the three products offered by

these salespersons.



Complete the following steps to implement value filtering in the FlexPivotGrid control. This implementation uses the sample created in Binding FlexPivot to Data Source in Code topic.

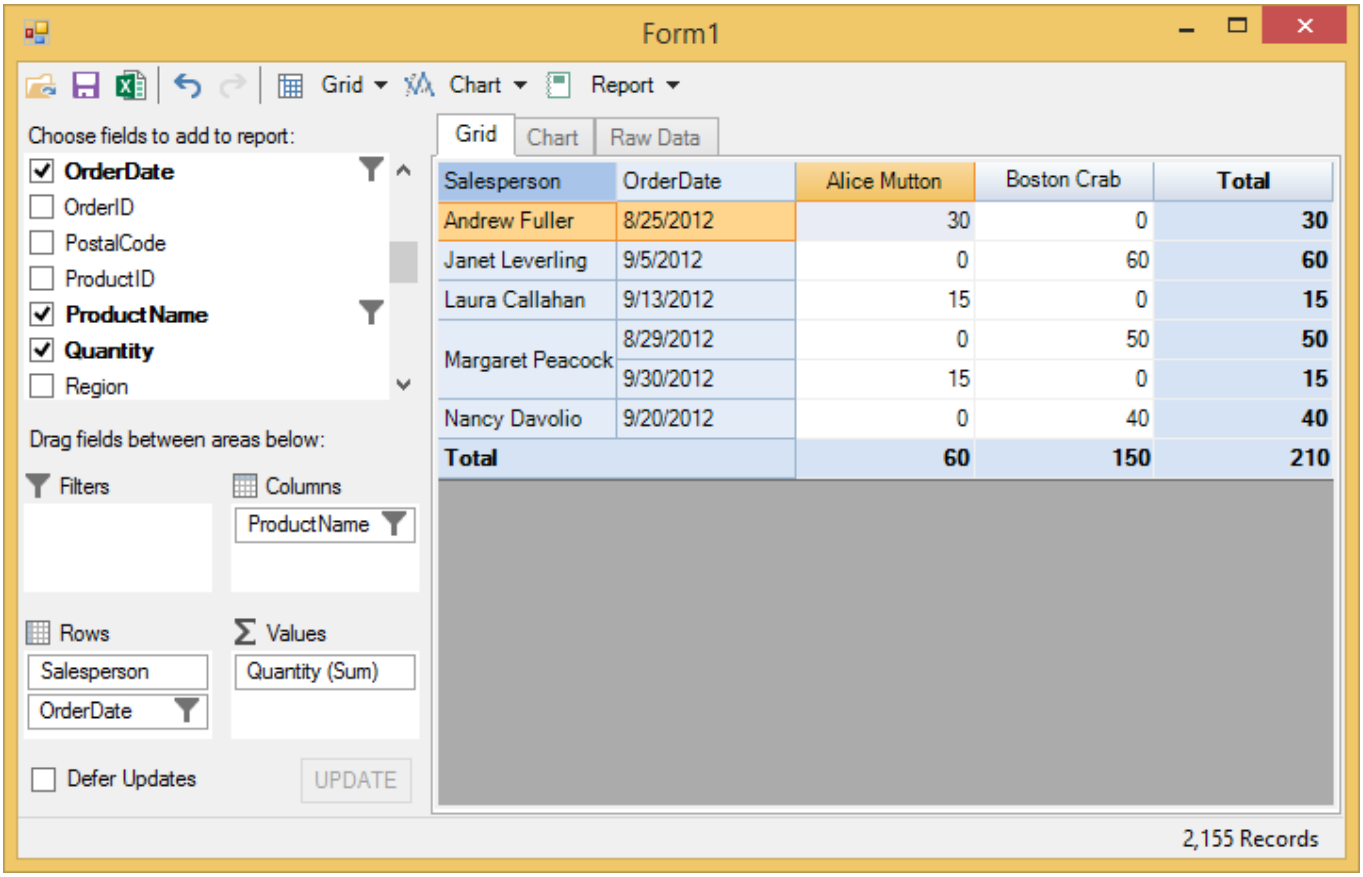
1. Add Salesperson and ProductName fields in the Rows list and UnitPrice field in the Values list.
2. Right-click the Salesperson field from the Rows list and click Field Settings option once to open the Field Settings dialog.
3. Select the following salespersons from the given list
    - Andrew Fuller
    - Laura Callahan
    - Margaret Peacock
    - Robert King
4. Right-click the ProductName field from the Rows list and click Field Settings option once to open the Field Settings dialog.
5. Select the following products from the given list
    - Boston Crab Meat
    - Filo Mix
    - Ipoh Coffee
6. Right-click the UnitPrice field from the Values list and click Field Settings option once to open the Field Settings dialog.
7. Navigate to the Subtotals tab and select Maximum.

Using Range Filters

Let's say a user wants to view the quantity of three products namely Alice Mutton, Boston Crab Meat, Filo Mix ordered by respective salespersons in the first nine months of the year 2012. For this, the user needs to slice the Salesperson

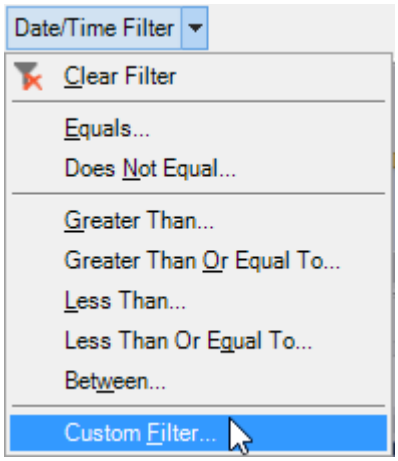
and Product Name fields in the database using range filters.

The image given below shows a FlexPivotGrid displaying the quantity of the three products ordered by respective salespersons for the first nine months of the year 2012.

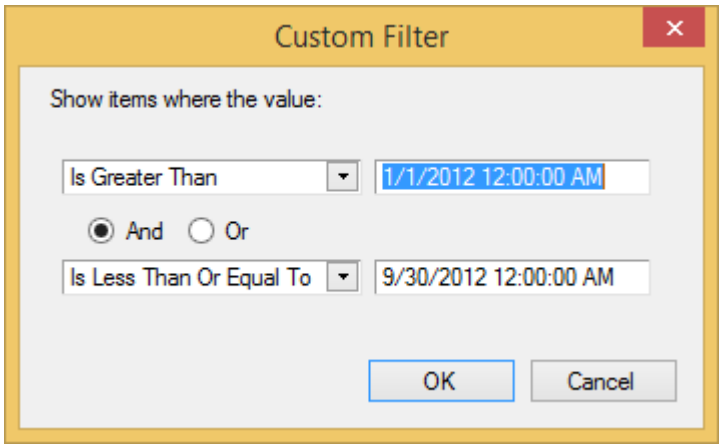


Complete the following steps to implement value filtering in the FlexPivotGrid control. This implementation uses the sample created in Binding FlexPivot to Data Source in Code topic.

1. Add Salesperson and OrderDate fields in the Rows list, Quantity field in the Values list, and ProductName field in the Columns lists.
2. Right-click the ProductName field in the Columns list and click Field Settings option once to open the Field Settings dialog.
3. Select the following products from the given list.
    - Alice Mutton
    - Boston Crab Meat
4. Right-click the OrderDate field in the Rows list and click Field Settings option once to open the Field Settings dialog.
5. Click the Date/Time Filter once and then click Custom Filter to apply conditions.



6. Set the condition in the Custom Filter dialog as illustrated below.



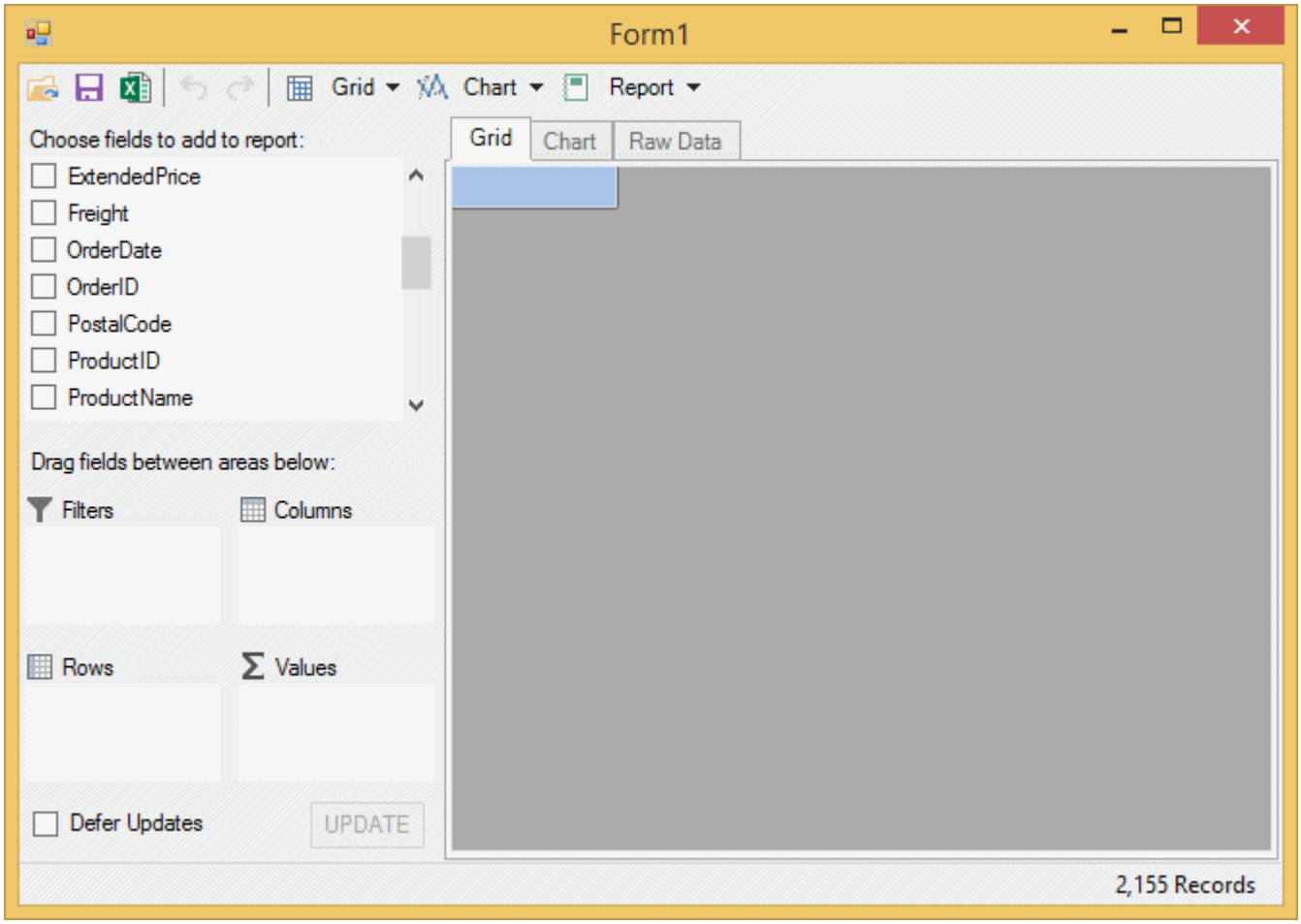
7. Click OK to see that the FlexPivotGrid control displays relevant data.

Sorting

Sorting is an important requirement when it comes to data analysis. Sorting helps in listing or arranging data in a specified order. The FlexPivotGrid control allows users to sort numeric values in ascending or descending data.

By default, data appearing in output grids are sorted by key. For example, the Country column in the FlexPivotGrid control appears sorted alphabetically. However, this is not always very useful for data presentation as users may prefer to sort the grid by value fields such as Unit Price, Extended Price or Discount.

To enable sorting, the C1FlexPivotGrid.AllowSorting property can be set **true**. This allows users to sort data by clicking the arrow appearing on the column headers, just like a regular grid. See the image below to visualize this behavior at runtime.



Clicking the arrow appearing on the column header changes the sort order from ascending to descending to unsorted.

By default, the AllowSorting property is set to True for C1FlexPivotGrid class.

Formatting Numeric Data

FlexPivot provides the option to format and represent numeric data in various formats such as number, currency, scientific, and percentage. You can also use custom format so that the numbers appears the way you want.

At Runtime

1. Right-click a field in the Values list in the pivot panel.
2. Click Field Settings option in the context menu to open Field Settings dialog.
3. Click the Format tab and select one of the following options.

Numeric	Formats the data as a number such as 1,235. You can specify the number of decimal places, and whether to use thousand separator (,) or not.
Currency	Formats the data as currency. You can specify the number of decimal places.
Percentage	Formats the data as percentage. You can specify the number of decimal places.
Scientific	Formats the data in scientific notation. You can specify the number of decimal places.
Custom	Applies a custom format to the data.

4. Click **OK** and observe how the format of numeric data changes.

In Code

You can format numeric data using the Format property and standard numeric format strings in code. Following enlisted are the accepted format strings to be used in code.

"N" or "n"	Numeric	Formats data as h numbers such as 1,k235. You can specify the number of decimal places and whether to use thousand separator (,) or not.
"C" or "c"	Currency	Formats the data as currency. You can specify the number of decimal places.
"P" or "p"	Percentage	Formats the data as percentage. You can specify the number of decimal places.
"E" or "e"	Scientific	Formats the data in scientific notation. You can specify the number of decimal places.
Non-standard numeric strings	Custom	Apply a custom format to the data.

Use the following code to format numeric data in Currency format.

Visual Basic

```
//formatting numeric data in code
Dim field = fp.Fields("ExtendedPrice")
field.Format = "c"
```

C#

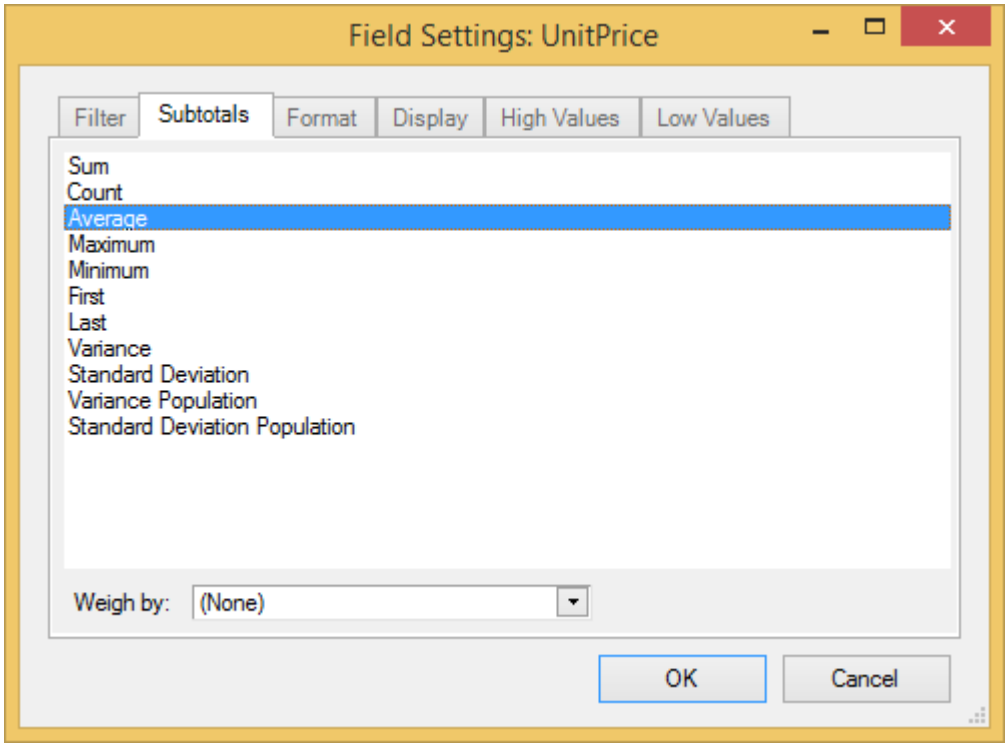
```
//formatting numeric data in code
var field = fp.Fields["ExtendedPrice"];
field.Format = "c";
```

Specifying Subtotal Function

FlexPivot control allows users to specify subtotals using various aggregate functions such as Sum, Count, Average, etc. This can either be done at runtime through the **Field Settings** dialog, or in code.

At Runtime

1. Right-click the field in the Values area of the FlexPivotPanel control and select Field Settings option.
2. Click the **Subtotals** tab and select the type of aggregate function you want to apply.



Sum	Gets the sum of a group
Count	Gets the number of values in a group.
Average	Gets the average of a group.
Maximum	Gets the maximum value in a group.
Minimum	Gets the minimum value in a group.
First	Gets the first value in a group.
Last	Gets the last value in a group.
Variance	Gets the sample variance of a group.
Standard Deviation	Gets the sample standard deviation of a group.
Variance Population	Gets the population variance of a group.
Standard Deviation Population	Gets the population standard deviation of a group.

3. Click **OK** and observe the change in the values.

In Code

You can use the Subtotal property to specify subtotals in code.

Visual Basic

```
//formatting numeric data in code
Dim field = fp.Fields("ExtendedPrice")
field.Subtotal = C1.FlexPivot.Subtotal.Average
```

C#

```
// show average price
var field = fp.Fields["ExtendedPrice"];
field.Subtotal = C1.FlexPivot.Subtotal.Average;
```

The C1FlexPivotPage class also provides the TotalsBeforeData property to determine whether row and column totals should be displayed before or after regular data rows and columns. The TotalsBeforeData property accepts Boolean values. If the value is set to true, total rows appear above data rows and total columns appear to the left of the regular data columns. By default, this property is set to false.

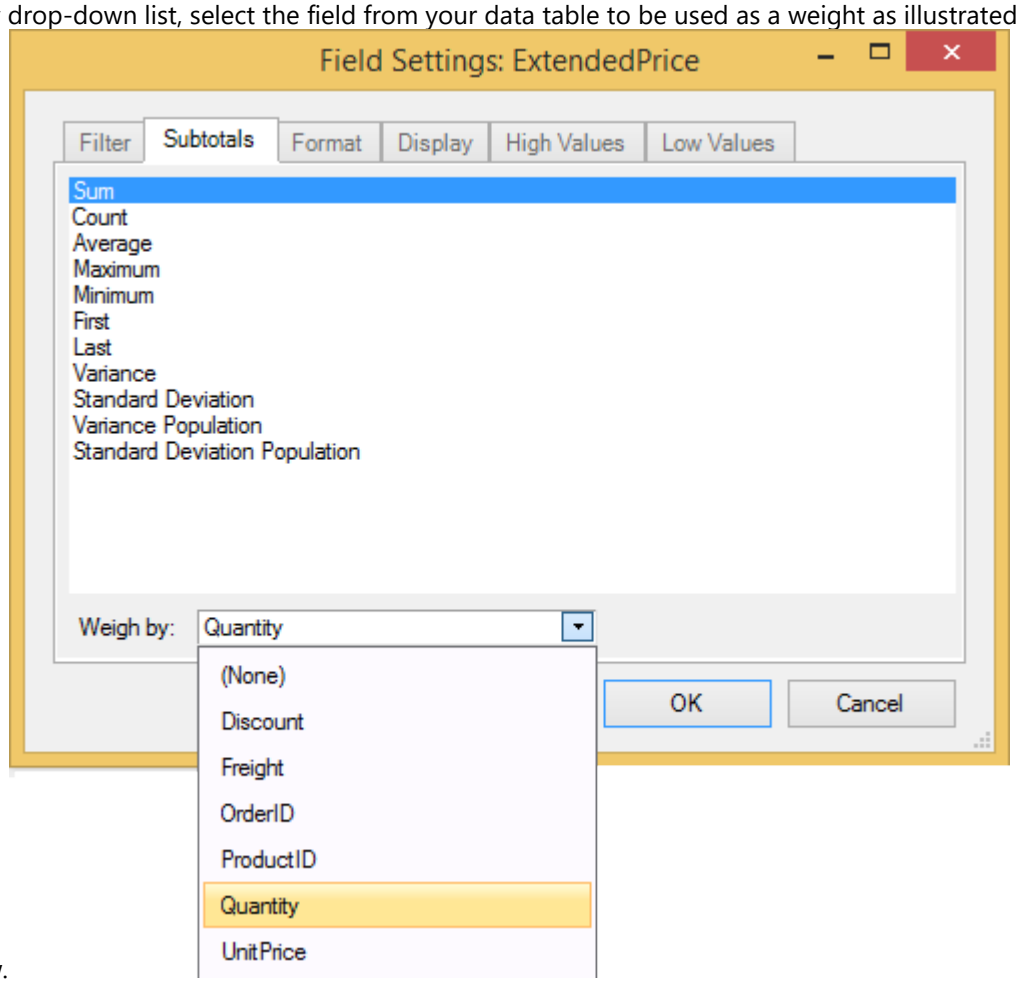
Calculating Weighted Averages and Sums

The FlexPivot control allows users to calculate average or sum of the data displayed on the FlexPivotGrid control. This can be done both at runtime as well as in code.

Consider a scenario where the user wants to find the average price for a group of products, taking into account the quantity of each purchased product. You can weigh the price average by the number of units purchased.

At Runtime

1. Right-click the field in the Values area of the FlexPivotPanel control and select Field Settings option**.**
2. Click the **Subtotals** tab and select the type of subtotal you want to calculate.
3. In the **Weigh by** drop-down list, select the field from your data table to be used as a weight as illustrated in the image below.



4. Click **OK** to close the Field Settings dialog.

In Code

Use the WeightField property to specify the field to be used as weight. The code given below uses Quantity field as the Weight.

Visual Basic

```
//setting weight field in code
var fp = this.c1FlexPivotPage1.FlexPivotEngine;
var field = fp.Fields["Quantity"];
field.WeightField = fp.Fields["Quantity"];
```

C#

```
//setting weight field in code
var fp = this.c1FlexPivotPage1.FlexPivotEngine;
var field = fp.Fields["Quantity"];
field.WeightField = fp.Fields["Quantity"];
```

FlexPivot Cube

In computing, cube is a multidimensional, logically-arranged dataset. More specifically, FlexPivot Cube is a data structure that provides faster data analysis in multiple dimensions.

FlexPivot features cube support that enables users in extracting valuable information by slicing and dicing the cube in ways pertinent for data analysis. You can connect to various data sources such as **Microsoft SQL Server Analysis Services (SSAS)**, or online cubes, or attach to a local cube at runtime. FlexPivot control works with **Analysis Services** and **SQL Server 2008**, **2012** and **2014**.

Setting Microsoft SQL Server Analysis Services

This guide is intended to provide users with information on setting up SQL Server Analysis Service (SSAS). To analyse cube data, you need to setup SSAS. The following steps explain how to setup the database.

1. Install a full version of **SQL Server.**
2. Download the Adventure Works database compatible with the version of SQL Server you installed. You can select the database from http://msftdbprodsamples.codeplex.com/releases.
3. Install Adventure Works database in the SQL Server.

Connecting to a Cube

Users can connect to a cube database through ConnectCube method. This method accepts two parameters: the name of the cube and the connection string to the installed SSAS.

The connection string must specify the **Data Source**, that is the Server name, and the **Initial Catalog**, that is the database name. The version of the **Provider** must also be specified if more than one **Microsoft OLE DB** provider for FlexPivot is installed. For instance, if the Provider is set to **MSOLAP**, the latest version of **OLE DB for FlexPivot** installed on your system is used.

The code given below illustrates an example of connecting to a cube.

Visual Basic

```
'prepare to build view
Dim connectionString As String = "Data Source=ServerAddress;Provider=msolap;Initial
Catalog=DatabaseName;User Id=ValidUserID; Password=ValidPassword"

Dim cubeName As String = "Adventure Works"
Try
c1FlexPivotPage1.FlexPivotPanel.ConnectCube(cubeName, connectionString)
' show some data.
Dim fp = c1FlexPivotPage1.FlexPivotEngine
fp.BeginUpdate()
fp.ColumnFields.Add("Color")
fp.RowFields.Add("Category")
fp.ValueFields.Add("Order Count")
fp.EndUpdate()
Catch ex As Exception
MessageBox.Show(ex.Message)
End Try
```

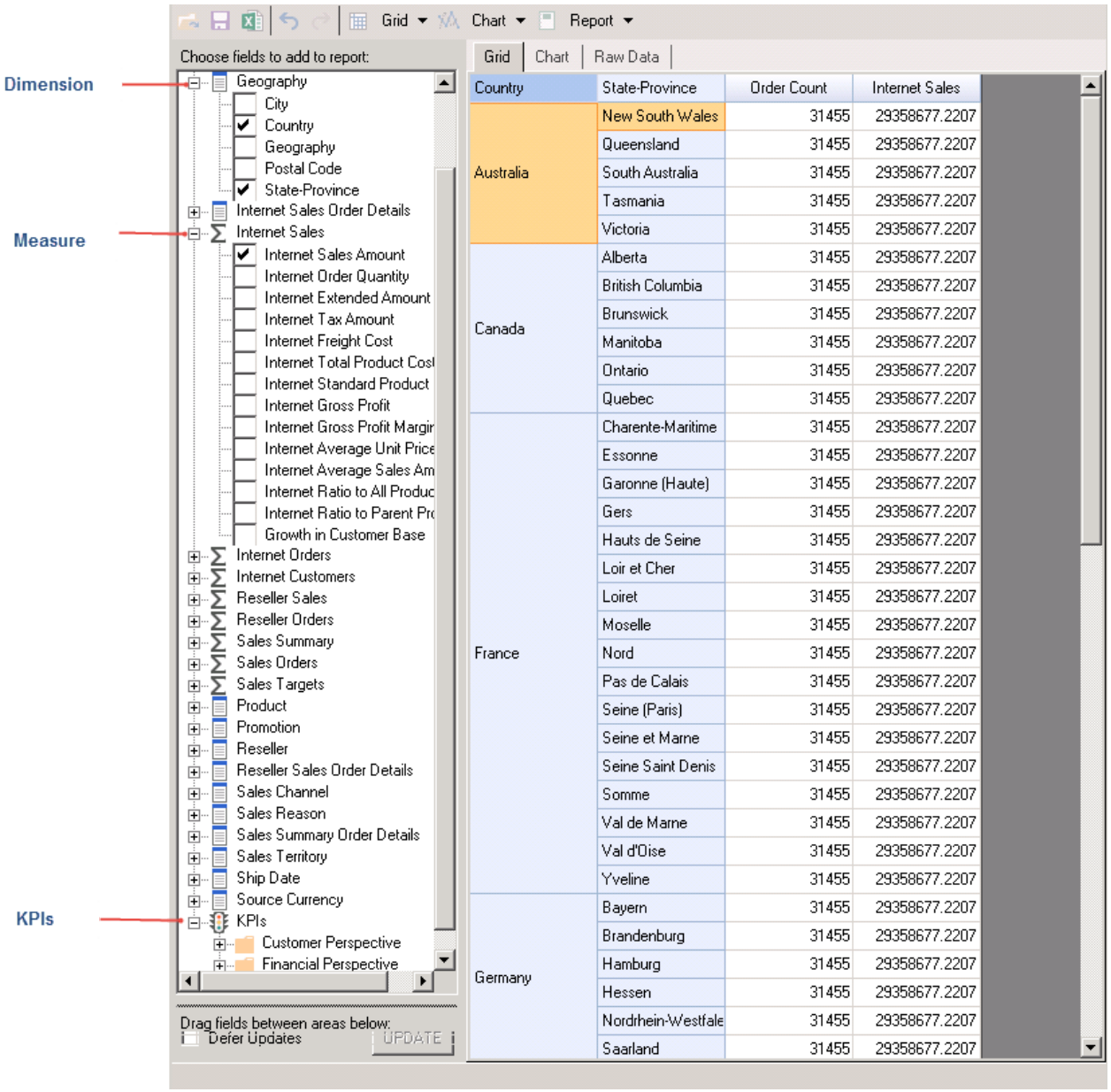
C#

```
//prepare to build view
string connectionString = @"Data Source=ServerAddress;Provider=msolap;Initial
Catalog=DatabaseName;User Id=ValidUserID; Password=ValidPassword";

string cubeName = "Adventure Works";
try
{
c1FlexPivotPage1.FlexPivotPanel.ConnectCube(cubeName, connectionString);
// show some data.
var fp = c1FlexPivotPage1.FlexPivotEngine;
fp.BeginUpdate();
fp.ColumnFields.Add("Color");
fp.RowFields.Add("Category");
fp.ValueFields.Add("Order Count");
fp.EndUpdate();
}
catch (Exception ex)
{
MessageBox.Show(ex.Message);
}
```

Using the Cube

In the previous step, you connected your application with a cube. Now, its time to run the application and see how the data appears on the FlexPivot control.

1. Press **F5** to run the project.
2. You notice that the control appears displaying some random data fetched from the database **Adventure Works.**
3. Now, drag-and-drop **Country** and and **State-Province** fields from the pivot panel to the **Rows** list, and **Order Count** and **Internet Sales Amount** to the **Values** list.
4. The control now displays a grid summarizing **Order Count** and **Internet Sales Amount** by Country and State Province as follows.



5. The cube data bound to the FlexPivot control in the above image consists of **Dimensions**, **Measures**, and **Key Performance Indicators** (KPIs). Dimensions are used to categorize the data cube, while Measures are the values for the dimensions. KPIs evaluate the measures in cube so as to present different perspectives of performance.
6. The installed cube **Adventure Works** consists of **Geography** as one of the many dimensions, **Internet Sales** as one of the measures, and Customer Perspective and Financial Perspective as the KPIs.
7. Users can use the cube data to build reports much like they would using regular datasets. The major difference between cube data and regular dataset is that data in cubes is represented in a tree-like structure in the C1FlexPivotPanel control.
8. Each node in the tree-like structure represents a dimensional entity or an object for measure.
9. Moreover, dimensions comprises **Heirarchies**, **Levels**, and **Attributes**
    - **Heirarchies**: Organizes levels in which the dimensions of a cube are structured.
    - **Level**: Describes position in a heirarchy.
    - **Attribute**: Gives additional information about the corresponding data.

Task-Based Help

The task-based help section assumes that you are familiar with programming in Visual Studio.NET environment. Each help topic in this section lets you accomplish a specific task such as configuring fields in code, using LINQ queries, applying themes, importing data from an Excel sheet in FlexPivot. For more information, click the following links:

- Configuring Fields in Code
- Adding Multiple Fields in Values List
- Applying Themes
- Using LINQ Queries to Add Data in FlexPivot
- Creating Custom FlexPivot Application in Code
- Importing Data from Excel

Configuring Fields in Code

FlexPivot allows users to configure fields programmatically. The control comes with a powerful object model that enables developers in configuring fields, applying filters, and specifying format of data fields in code.

To configure fields in code, complete the following steps.

1. Create a new Windows Forms Application project.
2. Drag-and-drop FlexPivotPage control (see the C1FlexPivotPage icon in the Toolbox) onto the form.
3. Switch to the code view and add the following code to set up a connection string with **c1nwind.mdb** database.
   - **Visual Basic**

```
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
```

   - **C#**

```
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);
}
```

4. Add the following code within the Form's constructor to load data (Invoices view) from the database, assign it to the FlexPivotPage control, and initialize a default view.
   - **Visual Basic**

```
' get data
Dim da = New OleDbDataAdapter("select * from invoices", GetConnectionString())
Dim dt = New DataTable()
da.Fill(dt)

' bind to FlexPivot page
Me.C1FlexPivotPage1.DataSource = dt

' build view
Dim fp = Me.C1FlexPivotPage1.FlexPivotEngine
fp.ValueFields.Add("ExtendedPrice")
fp.RowFields.Add("OrderDate", "ProductName")
```

   - **C#**

```
// get data
var da = new OleDbDataAdapter("select * from invoices", GetConnectionString());
var dt = new DataTable();
da.Fill(dt);

// bind to FlexPivot page
this.c1FlexPivotPage1.DataSource = dt;

// build view
var fp = this.c1FlexPivotPage1.FlexPivotEngine;
fp.ValueFields.Add("ExtendedPrice");
fp.RowFields.Add("OrderDate", "ProductName");
```

5. Use the following code to format the ExtendedPrice and OrderDate fields. This code sets the format of the ExtendedPrice field to Currency and that of the OrderDate field to Year.
   - **Visual Basic**

```
' format order date and extended price
Dim field = fp.Fields("OrderDate")
field.Format = "yyyy"
field = fp.Fields("ExtendedPrice")
field.Format = "c"

' show average price (instead of sum)
field = fp.Fields("ExtendedPrice")
field.Subtotal = C1.FlexPivot.Subtotal.Average
```

- **C#**

```
// format order date and extended price
var field = fp.Fields["OrderDate"];
field.Format = "yyyy";
field = fp.Fields["ExtendedPrice"];
field.Format = "c";

// show average price (instead of sum)
field = fp.Fields["ExtendedPrice"];
field.Subtotal = C1.FlexPivot.Subtotal.Average;
```

6. Add the following code to apply filter on products. This code applies filter to display only 4 products that include Chai, Chang, Geitost and Ikura.
   - **Visual Basic**

```
' apply value filter to show only a few products
Dim filter As C1.FlexPivot.C1FlexPivotFilter = fp.Fields("ProductName").Filter
filter.Clear()
filter.ShowValues = "Chai,Chang,Geitost,Ikura".Split(",c)
```

   - **C#**

```
// apply value filter to show only a few products
C1.FlexPivot.C1FlexPivotFilter filter = fp.Fields["ProductName"].Filter;
filter.Clear();
filter.ShowValues = "Chai,Chang,Geitost,Ikura".Split(',');
```

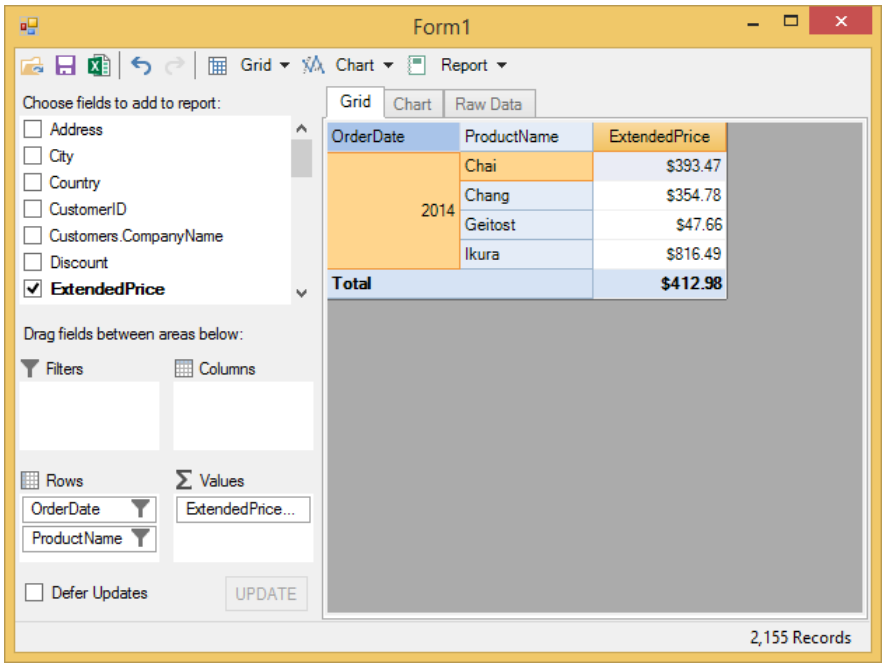
7. Add the following code to apply filter on OrderDate field. This code filters OrderDate from January 1st, 2013 to December 31st, 2014.
   - **Visual Basic**

```
' apply range filter to show only some dates
filter = fp.Fields("OrderDate").Filter
filter.Clear()
filter.Condition1.[Operator] = C1.FlexPivot.ConditionOperator.GreaterThanOrEqualTo
filter.Condition1.Parameter = New DateTime(2014, 1, 1)
filter.Condition2.[Operator] = C1.FlexPivot.ConditionOperator.LessThanOrEqualTo
filter.Condition2.Parameter = New DateTime(2014, 12, 31)
filter.AndConditions = True
```

   - **C#**

```
// apply range filter to show only some dates
filter = fp.Fields["OrderDate"].Filter;
filter.Clear();
filter.Condition1.Operator = C1.FlexPivot.ConditionOperator.GreaterThanOrEqualTo;
filter.Condition1.Parameter = new DateTime(2014, 1, 1);
filter.Condition2.Operator = C1.FlexPivot.ConditionOperator.LessThanOrEqualTo;
filter.Condition2.Parameter = new DateTime(2014, 12, 31);
filter.AndConditions = true;
```

8. Run the application. The form appears with a custom view showing fields set in the code.



Adding Multiple Fields in Values List

FlexPivot provides the option for adding more than one value fields to the **Values** list. All you need to do is set MaxItems property to some numeric value in the code. This example uses the sample created in Configuring Fields in Code.

1. Set the MaxItems property to 3 and add three value fields namely Extended Price, Discount and Freight to the

ValueFields.Add() method as illustrated in the following code.

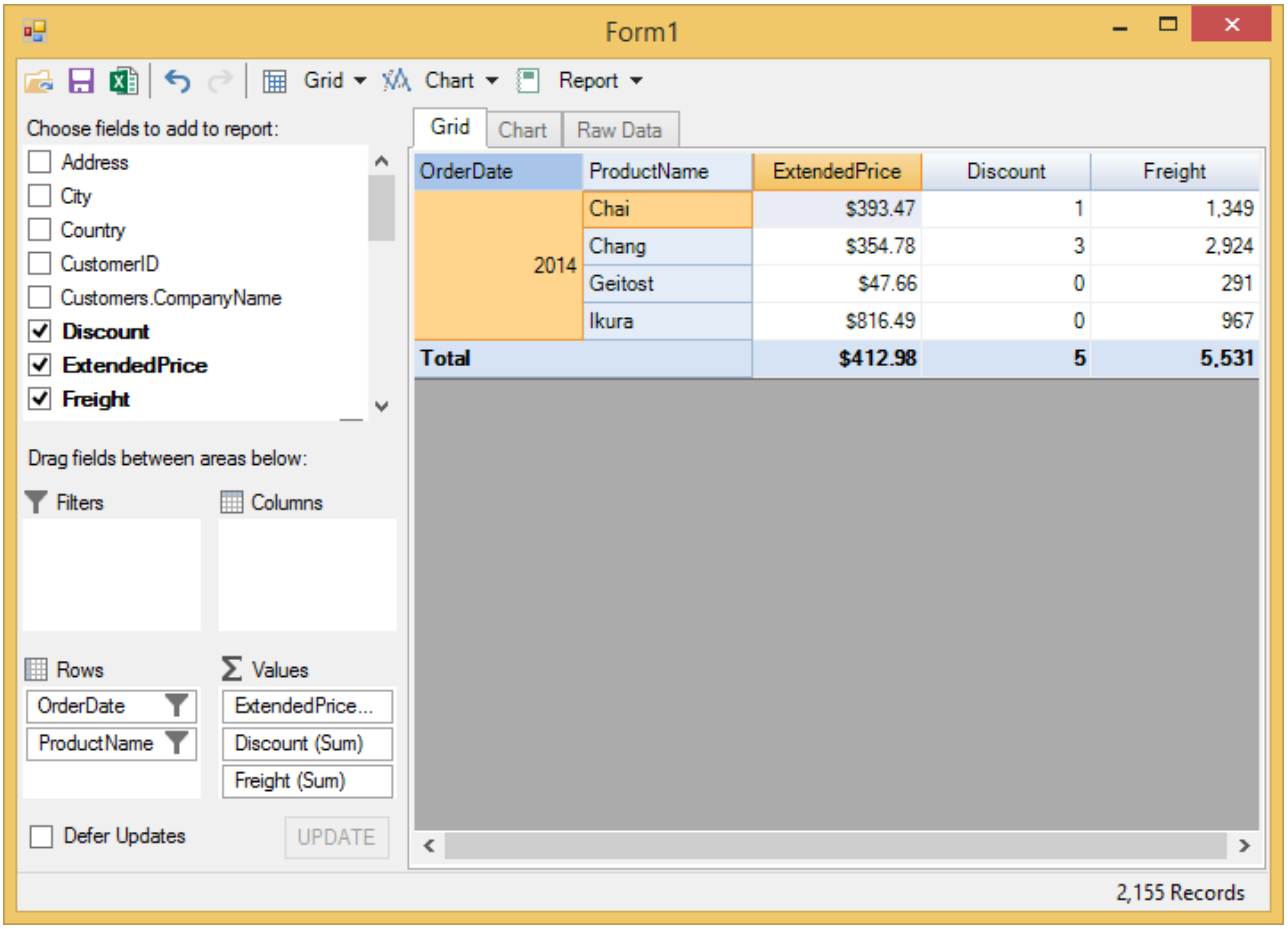
- **Visual Basic**

```
'apply multiple value fields
fp.ValueFields.MaxItems = 3
fp.ValueFields.Add("ExtendedPrice", "Discount", "Freight")
```

- **C#**

```
//apply multiple value fields
fp.ValueFields.MaxItems = 3;
fp.ValueFields.Add("ExtendedPrice", "Discount", "Freight");
```

2. Press **F5** to run the application and observe that Extended Price, Discount and Freight fields are displayed in the grid.

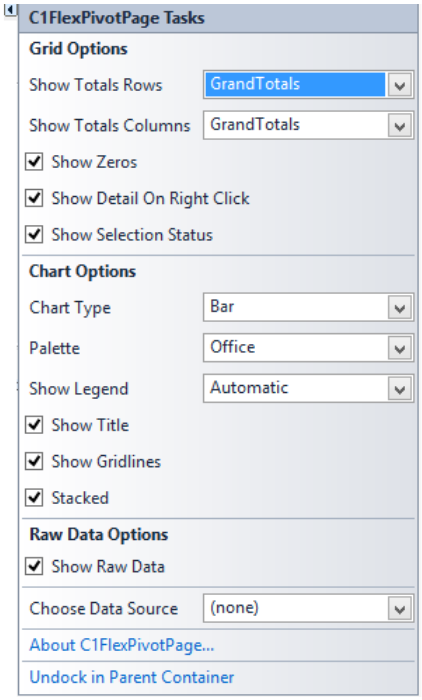


Applying Themes

The overall appearance of FlexPivot controls can also be customized by applying themes through C1ThemeController. Developers can choose from a collection of predefined built-in themes to customize the control's overall appearance.

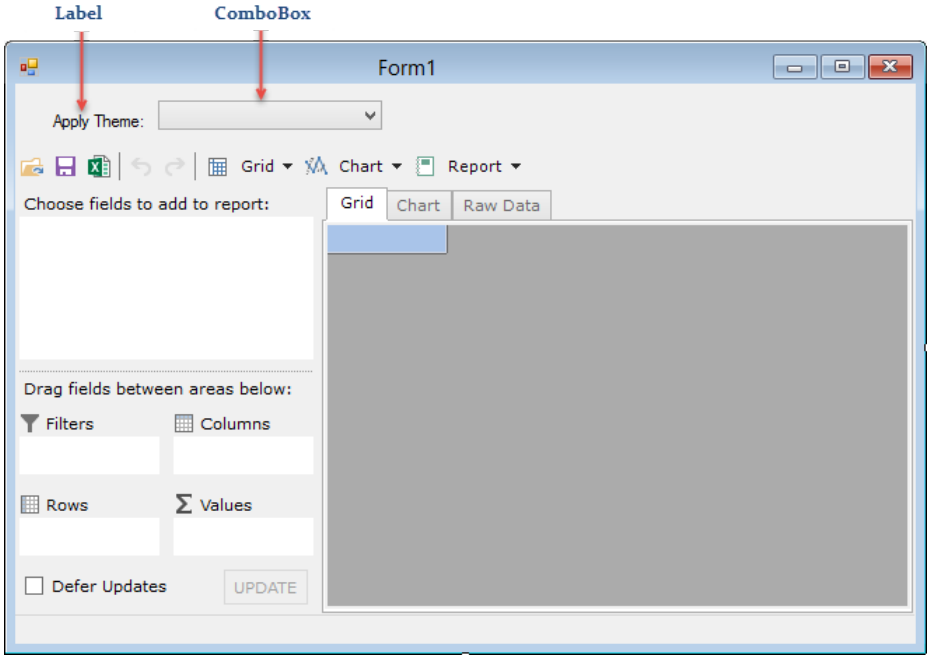
To apply built-in themes, perform the following steps.

1. Create a new Windows Forms Application project in Visual Studio.
2. Drag-and-drop the C1FlexPivotPage control onto the form from the Toolbox.
3. Add **C1.Win.C1Themes.4** reference to your project to access built-in themes through C1ThemeController.
4. Click once on the smart tag icon ( ). The C1FlexPivotPageTasks smart panel appears as illustrated in the image below.



5. Select **Undock in Parent Container** option to undock the FlexPivotPage control in the parent container i.e. Form.
6. Navigate to the toolbox and add a standard label control to the form.
7. Set some of the properties of the label control from the Properties Window as follows:
    - AutoSize = True
    - TabIndex = 0
    - Text = "Apply Theme"
8. Add a standard Combobox control from the Toolbox and set some its properties as follows:
    - Name = "cbTheme"
    - FormattingEnabled = True
    - DropDownStyle = DropDownList
    - TabIndex = 1
    - Text = "Apply Theme"

The Design View appears similar to the following image:



9. Switch to the code view add the following Import statement.
    - **Visual Basic**

```
Imports C1.Win.C1Themes
Imports System.Data.OleDb
```

    - **C#**

```
using C1.Win.C1Themes;
```

```
using System.Data.OleDb;
```

10. Add the following code to set up a connection string with the **C1NWind.mdb** database file.
    - **Visual Basic**

```
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
```

    - **C#**

```
// get standard nwind mdb connection string
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);
}
```

11. Add the following code in the Form's constructor to fetch data from the **C1NWind.mdb** database file and create a view.
    - **Visual Basic**

```
' get data
Dim da = New OleDbDataAdapter("Select * from Invoices", GetConnectionString())
Dim dt = New DataTable("NorthWind Sales Data")
da.Fill(dt)

' assign data to C1FlexPivotPage control
C1FlexPivotPage1.DataSource = dt

Dim fp = C1FlexPivotPage1.FlexPivotEngine
fp.ValueFields.MaxItems = 3
fp.BeginUpdate()
fp.RowFields.Add("Country")
fp.ColumnFields.Add("Product")
fp.ValueFields.Add("Sales")
fp.EndUpdate()
```

    - **C#**

```
// get data
var da = new OleDbDataAdapter("Select * from Invoices", GetConnectionString());
var dt = new DataTable("NorthWind Sales Data");
da.Fill(dt);

// assign data to C1FlexPivotPage control
c1FlexPivotPage1.DataSource = dt;

var fp = c1FlexPivotPage1.FlexPivotEngine;
fp.ValueFields.MaxItems = 3;
fp.BeginUpdate();
fp.RowFields.Add("Country");
fp.ColumnFields.Add("Product");
fp.ValueFields.Add("Sales");
fp.EndUpdate();
```

12. Add the following code in the Form's constructor to subscribe **SelectedIndexChanged** event for Combobox control, and implement logic for applying themes to the Form on selecting built-in themes from the dropdown list.
    - **Visual Basic**

```
For Each theme As String In C1ThemeController.GetThemes()
    cbTheme.Items.Add(theme)
Next
AddHandler cbTheme.SelectedIndexChanged, AddressOf cbTheme_SelectedIndexChanged
```

    - **C#**

```
cbTheme.SelectedIndexChanged += cbTheme_SelectedIndexChanged;
foreach (string theme in C1ThemeController.GetThemes())
    cbTheme.Items.Add(theme);
```

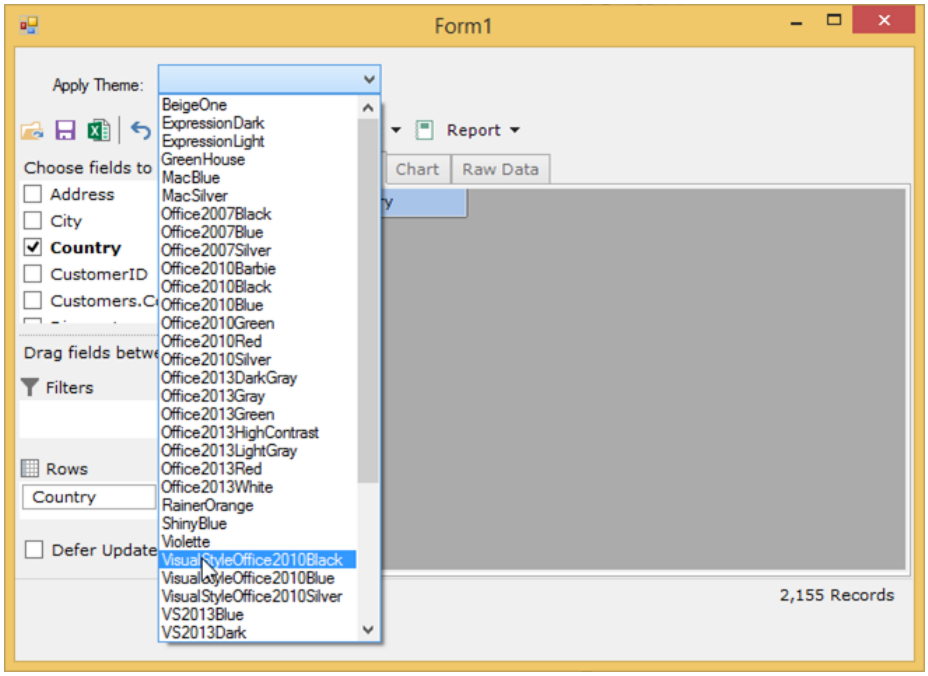
13. Add the following code to the event handler created for **cbTheme.SelectedIndexChanged** event.
    - **Visual Basic**

```
Private Sub cbTheme_SelectedIndexChanged(sender As Object, e As EventArgs)
    Dim theme As C1Theme = C1ThemeController.GetThemeByName(cbTheme.Text, False)
    If theme IsNot Nothing Then
        C1ThemeController.ApplyThemeToObject(c1FlexPivotPage1, theme)
    End If
End Sub
```

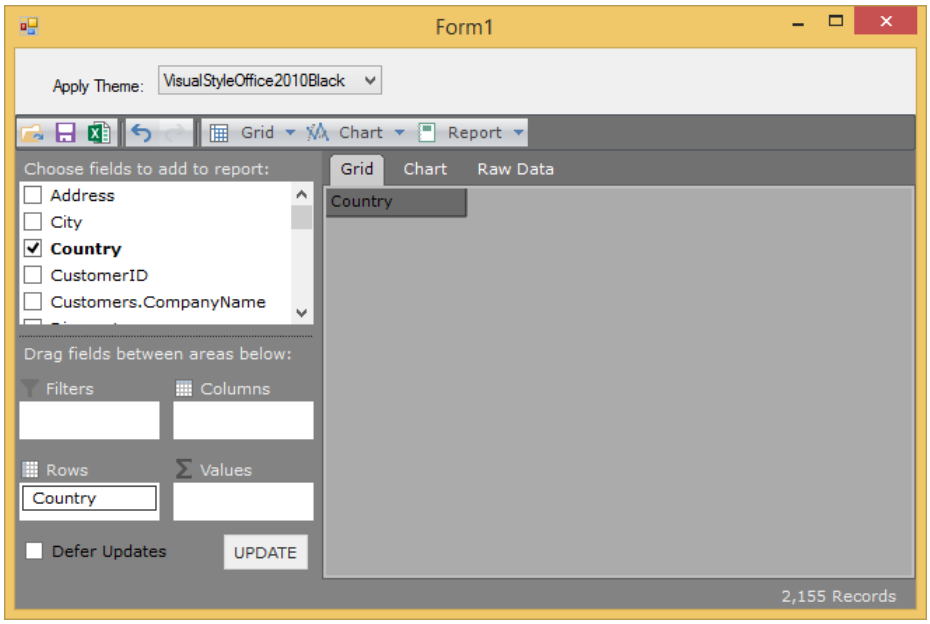
    - **C#**

```
private void cbTheme_SelectedIndexChanged(object sender, EventArgs e)
{
    C1Theme theme = C1ThemeController.GetThemeByName(cbTheme.Text, false);
    if (theme != null)
        C1ThemeController.ApplyThemeToObject(c1FlexPivotPage1, theme);
}
```

14. Press **F5** to run the application and select a predefined theme, for example, VisualStyleOffice2010Black from the dropdown list.



15. The theme applies to the forms as illustrated in the following image.



Using LINQ Queries to Add Data in FlexPivot

FlexPivot can use various collections such as LINQ queries to add data. LINQ provides a flexible and efficient data querying model to create new queries within the client application without modifying the database. As a result, FlexPivot control can use LINQ queries as a data source so that end users can create their own views for data analysis.

Complete the following steps to create a LINQ query and use it as a data source for FlexPivot.

1. Create a new Windows Forms Application project in Visual Studio.
2. Drag-and-drop FlexPivotPage control (see C1FlexPivotpage icon in the Toolbox) onto the form.
3. Switch to the code view and add the following namespaces.
   - **Visual Basic**

```
Imports System.Data.OleDb
Imports System.Linq
```

   - **C#**

```
using System.Data.OleDb;
using System.Linq;
```

4. Add the following code in the Forms1.cs constructor to load the data using LINQ query.
   - **Visual Basic**

```
Dim ds = New DataSet()
For Each table As String In "Products,Categories,Employees,Customers,Orders,Order Details".Split(","c)
    Dim sql As String = String.Format("select * from [{0}]", table)
    Dim da = New OleDbDataAdapter(sql, GetConnectionString())
    da.Fill(ds, table)
Next table
```

   - **C#**

```
var ds = new DataSet();
foreach (string table in "Products,Categories,Employees,Customers,Orders,Order Details".Split(','))
{
```

```
    string sql = string.Format("select * from [{0}]", table);
    var da = new OleDbDataAdapter(sql, GetConnectionString());
    da.Fill(ds, table);
}
```

5. Initialize a connection string to connect with a database installed on your system. This example uses **C1NWind.mdb** file as the database. You can find this file at **Documents\ComponentOne Samples\Common** location on your system. In case your database file is kept at a different location then make changes in the path defined in the **GetConnectionString** method.
   - **Visual Basic**

```
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
```

   - **C#**

```
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);
}
```

6. Add the following LINQ query to connect the FlexPivotPage control with the tables loaded from the database.
   - **Visual Basic**

```
Dim q = From detail In ds.Tables("Order Details").AsEnumerable() _
        Join product In ds.Tables("Products").AsEnumerable() On detail.Field(Of Integer)("ProductID") Equals product.Field(Of Integer)("ProductID") _
        Join category In ds.Tables("Categories").AsEnumerable() On product.Field(Of Integer)("CategoryID") Equals category.Field(Of Integer)("CategoryID") _
        Join order In ds.Tables("Orders").AsEnumerable() On detail.Field(Of Integer)("OrderID") Equals order.Field(Of Integer)("OrderID") _
        Join customer In ds.Tables("Customers").AsEnumerable() On order.Field(Of String)("CustomerID") Equals customer.Field(Of String)("CustomerID") _
```

   - **C#**

```
var q =
   from detail in ds.Tables["Order Details"].AsEnumerable()
   join product in ds.Tables["Products"].AsEnumerable()
       on detail.Field<int>("ProductID") equals product.Field<int>("ProductID")
   join category in ds.Tables["Categories"].AsEnumerable()
       on product.Field<int>("CategoryID") equals category.Field<int>("CategoryID")
   join order in ds.Tables["Orders"].AsEnumerable()
       on detail.Field<int>("OrderID") equals order.Field<int>("OrderID")
   join customer in ds.Tables["Customers"].AsEnumerable()
       on order.Field<string>("CustomerID") equals customer.Field<string>("CustomerID")
   join employee in ds.Tables["Employees"].AsEnumerable()
       on order.Field<int>("EmployeeID") equals employee.Field<int>("EmployeeID")
```

   Each table connects to the query by joining its primary key to a given field. For instance, Products table is joined using ProductID, Categories is joined using CategoryID, and so on.

7. Add the following Select statement to build a new anonymous class containing fields specified in query above. Note that the fields may map directly to the table fields or may be calculated.
   - **Visual Basic**

```
Select New With
  {
    Key .Sales = (detail.Field(Of Short)("Quantity") * CDbl(detail.Field(Of Decimal)("UnitPrice"))) * (1 - CDbl(detail.Field(Of Single)("Discount"))),
    Key .OrderDate = order.Field(Of Date)("OrderDate"),
    Key .Product = product.Field(Of String)("ProductName"),
    Key .Customer = customer.Field(Of String)("CompanyName"),
    Key .Country = customer.Field(Of String)("Country"),
    Key .Employee = employee.Field(Of String)("FirstName") & " " & employee.Field(Of String)("LastName"),
    Key .Category = category.Field(Of String)("CategoryName")}
```

   - **C#**

```
select new
{
    Sales = (detail.Field<short>("Quantity") * (double)detail.Field<decimal>("UnitPrice")) * (1 - (double)detail.Field<float>("Discount")),
    OrderDate = order.Field<DateTime>("OrderDate"),
    Product = product.Field<string>("ProductName"),
    Customer = customer.Field<string>("CompanyName"),
    Country = customer.Field<string>("Country"),
    Employee = employee.Field<string>("FirstName") + " " + employee.Field<string>("LastName"),
    Category = category.Field<string>("CategoryName")
};
```

8. Convert the LINQ query to a list using the **ToList()** method and assign the resultant to the DataSource property of C1FlexPivotPage class.
   - **Visual Basic**

```
c1FlexPivotPage1.DataSource = q.ToList()
```

   - **C#**

```
c1FlexPivotPage1.DataSource = q.ToList();
```

9. Create a default view that gets loaded as the initial view when the application runs.
   - **Visual Basic**

```
Dim fp = c1FlexPivotPage1.FlexPivotPanel.FlexPivotEngine
fp.BeginUpdate()
fp.RowFields.Add("Country")
fp.ColumnFields.Add("Category")
fp.ValueFields.Add("Sales")
fp.EndUpdate()
```

   - **C#**

```
var fp = c1FlexPivotPage1.FlexPivotPanel.FlexPivotEngine;
fp.BeginUpdate();
fp.RowFields.Add("Country");
fp.ColumnFields.Add("Category");
fp.ValueFields.Add("Sales");
fp.EndUpdate();
```

10. Run the project to observe that the FlexPivotPage control bound to data from tables defined in the **C1NWind.mdb** database file appears with an initial view.

Creating Custom FlexPivot Application in Code

FlexPivot controls can be easily customized to suit the requirements of developers as well as end users. From setting default views to adding predefined views, developers can perform high-level customizations in these controls through code.

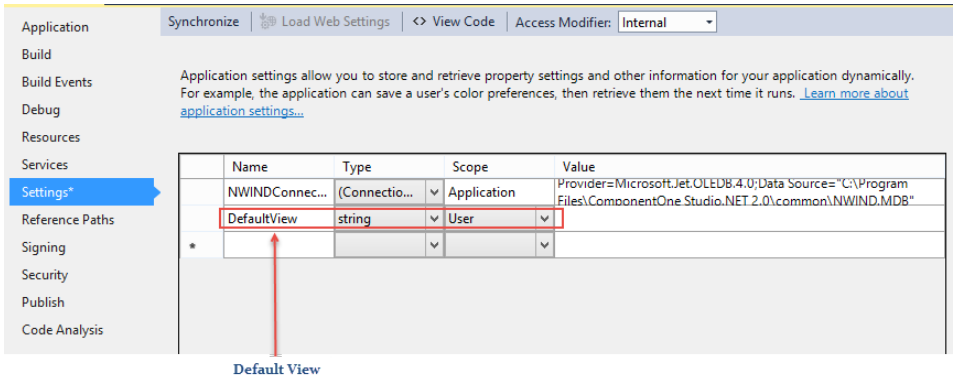
Creating Default View

Data (view or table) added to the FlexPivotPage control can be visualized by creating views. You can create summarized views by dragging data fields into various lists at runtime as illustrated in Creating Different Views at Runtime topic. To create a default view that appears automatically when your FlexPivot application runs, you need to perform some design-time settings and add set the ViewDefinition property in code.

Complete the following steps to create an application that displays a default view with **ProductName** in the Rows list, **Country** in the Columns list, and **ExtendedPrice** in

the Values list.

1. Create a new Windows Forms Application project in Visual Studio.
2. Add the C1FlexPivotPage control to your project and bind it to **Invoices** view of the c1NWind data source file as illustrated in Binding FlexPivot to Data Source topic.
3. In Solution Explorer, right-click your project and click Properties to open Project designer.
4. In the Project Design, click Settings option.
5. Create a new setting and name it, for example, DefaultView.



6. Switch to code view and add the following code in Form1_Load event to initialize the default view and set the data fields.
   - **Visual Basic**

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    ' modify the connection string to make it work for the current user on this machine
    Me.InvoicesTableAdapter.Connection.ConnectionString = GetConnectionString()

    'loads data into the table adapter
    Me.InvoicesTableAdapter.Fill(Me.C1NWindDataSet.Invoices)

    ' show default view:
    ' this assumes an application setting of type string called "DefaultView"
    Dim view = My.Settings.DefaultView
    If Not String.IsNullOrEmpty(view) Then
        c1FlexPivotPage1.ViewDefinition = view
    Else
        ' build default view now
        Dim fp = c1FlexPivotPage1.FlexPivotEngine
        fp.BeginUpdate()
        fp.RowFields.Add("ProductName")
        fp.ColumnFields.Add("Country")
        fp.ValueFields.Add("ExtendedPrice")
        fp.EndUpdate()
    End If
End Sub
```

   - **C#**

```
private void Form1_Load(object sender, EventArgs e)
{
    //modify the connection string to make it work for the current user on this machine
    this.invoicesTableAdapter.Connection.ConnectionString = GetConnectionString();

    //loads data into the table adapter
    this.invoicesTableAdapter.Fill(this.c1NWindDataSet.Invoices);

    // show default view:
    // this assumes an application setting of type string called "DefaultView"
    var view = Properties.Settings.Default.DefaultView;
    if(!string.IsNullOrEmpty(view))
    {
        c1FlexPivotPage1.ViewDefinition = view;
    }
    else
    {
        // build default view
        var fp = c1FlexPivotPage1.FlexPivotEngine;
        fp.BeginUpdate();
        fp.RowFields.Add("ProductName");
        fp.ColumnFields.Add("Country");
        fp.ValueFields.Add("ExtendedPrice");
        fp.EndUpdate();
    }

}
```

7. Initialize a standard connection string to the database file being used.
   - **Visual Basic**

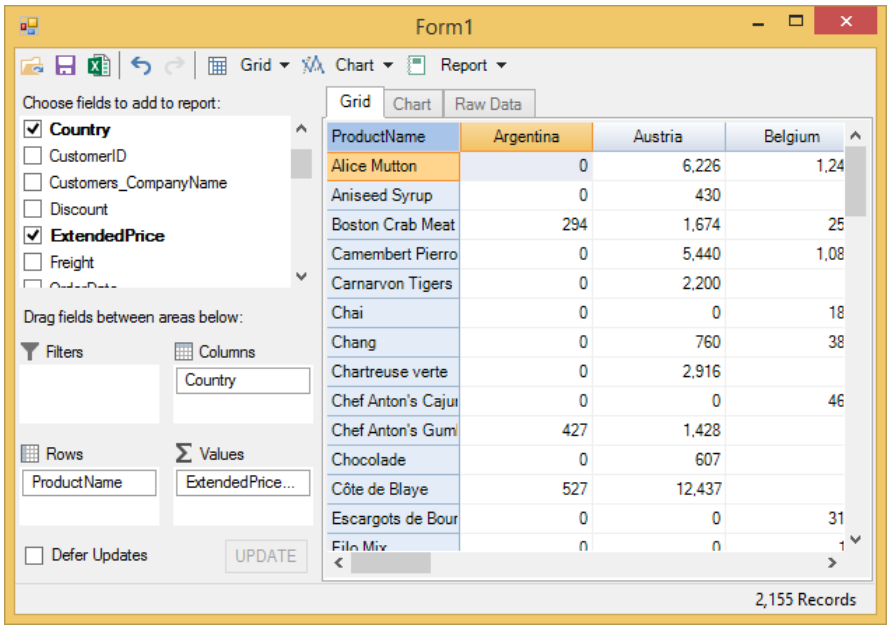
```
'initializing the connection string
Private Shared Function GetConnectionString() As String
    Dim path As String = Environment.GetFolderPath(Environment.SpecialFolder.Personal) + "\ComponentOne Samples\Common"
    Dim conn As String = "provider=microsoft.jet.oledb.4.0;data source={0}\c1nwind.mdb;"
```

```
    Return String.Format(conn, path)
End Function
```

- **C#**

```
//initializing the connection string
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);
}
```

8. Press F5 to run the application. The default view gets displayed with **ProductName** appearing in the Rows list, **Country** in the Columns list, and **ExtendedPrice** in the Values list.



Adding Predefined Views

Users can add predefined views to their FlexPivot application by defining the views in an XML file and adding the same as a resource to the Project Designer. The C1FlexPivotPage class includes ReadXml and WriteXml methods to add files as streams. These methods are automatically invoked by the FlexPivotPage control to apply predefined views.

1. Create a new XML file, for example FlexPivotViews.xml, and add the following code to define 5 view definitions, each showing **Extended Price** (a field in the Invoices View of the c1NWind.mdb database file) by
   - Product vs Country
   - Salesperson vs Country
   - Salesperson vs Month
   - Salesperson vs Weekday
   - Salesperson vs Year

XML copyCode

```
<FlexPivotViews>

<C1FlexPivotPage id="Product vs Country">
  <C1FlexPivotEngine>
    <Fields>
      <Field name="Address" subtotal="Count" />
      <Field name="City" subtotal="Count" />
      <Field name="Country" subtotal="Count" />
      <Field name="CustomerID" subtotal="Count" />
      <Field name="Customers_CompanyName" subtotal="Count" />
      <Field name="Discount" subtotal="Sum" format="n0" />
      <Field name="ExtendedPrice" subtotal="Sum" format="n0" />
      <Field name="Freight" subtotal="Sum" format="n0" />
      <Field name="OrderDate" subtotal="Count" format="d" />
      <Field name="OrderID" subtotal="Sum" format="n0" />
      <Field name="PostalCode" subtotal="Count" />
      <Field name="ProductID" subtotal="Sum" format="n0" />
      <Field name="ProductName" subtotal="Count" />
      <Field name="Quantity" subtotal="Sum" format="n0" />
```

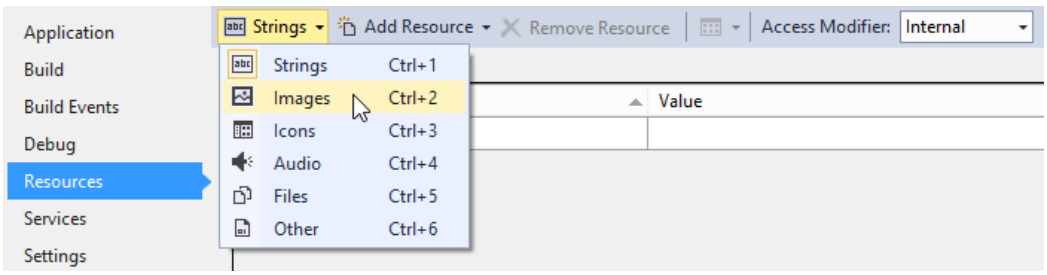
```
        <Field name="Region" subtotal="Count" />
        <Field name="RequiredDate" subtotal="Count" format="d" />
        <Field name="Salesperson" subtotal="Count" />
        <Field name="ShipAddress" subtotal="Count" />
        <Field name="ShipCity" subtotal="Count" />
        <Field name="ShipCountry" subtotal="Count" />
        <Field name="ShipName" subtotal="Count" />
        <Field name="ShippedDate" subtotal="Count" format="d" />
        <Field name="Shippers_CompanyName" subtotal="Count" />
        <Field name="ShipPostalCode" subtotal="Count" />
        <Field name="ShipRegion" subtotal="Count" />
        <Field name="UnitPrice" subtotal="Sum" format="n0" />
      </Fields>
      <RowFields>
        <Field name="ProductName" />
      </RowFields>
      <ColumnFields>
        <Field name="Country" />
      </ColumnFields>
      <ValueFields>
        <Field name="ExtendedPrice" />
      </ValueFields>
    </C1FlexPivotEngine>
    <C1FlexPivotPrintDocument>
      <Header Text="&amp;[ViewTitle]&#x9;&amp;[Date]&#x9;Page &amp;[Page]" Separator="True"
FontName="Arial" FontSize="9" FontStyle="Regular" />
      <Footer Text="" Separator="False" FontName="Arial" FontSize="9" FontStyle="Regular" />
      <Grid ShowGrid="True" GridOptions="1" />
      <Chart ShowChart="True" ChartFillsPage="True" />
      <RawData ShowRawData="False" RawDataOptions="1" />
      <PageSettings Landscape="False" Margins="100,100,100,100" />
    </C1FlexPivotPrintDocument>
    <C1FlexPivotChart ChartType="Bar" ColorGeneration="Office" ShowTitle="True"
ShowLegend="Automatic" ShowGridLines="True" Stacked="True" />
  </C1FlexPivotPage>

  <C1FlexPivotPage id="SalesPerson vs Country">
    <C1FlexPivotEngine>
      <Fields>
        <Field name="Address" subtotal="Count" />
        <Field name="City" subtotal="Count" />
        <Field name="Country" subtotal="Count" />
        <Field name="CustomerID" subtotal="Count" />
        <Field name="Customers_CompanyName" subtotal="Count" />
        <Field name="Discount" subtotal="Sum" format="n0" />
        <Field name="ExtendedPrice" subtotal="Sum" format="n0" />
        <Field name="Freight" subtotal="Sum" format="n0" />
        <Field name="OrderDate" subtotal="Count" format="d" />
        <Field name="OrderID" subtotal="Sum" format="n0" />
        <Field name="PostalCode" subtotal="Count" />
        <Field name="ProductID" subtotal="Sum" format="n0" />
        <Field name="ProductName" subtotal="Count" />
        <Field name="Quantity" subtotal="Sum" format="n0" />
        <Field name="Region" subtotal="Count" />
        <Field name="RequiredDate" subtotal="Count" format="d" />
        <Field name="Salesperson" subtotal="Count" />
        <Field name="ShipAddress" subtotal="Count" />
        <Field name="ShipCity" subtotal="Count" />
        <Field name="ShipCountry" subtotal="Count" />
        <Field name="ShipName" subtotal="Count" />
        <Field name="ShippedDate" subtotal="Count" format="d" />
        <Field name="Shippers_CompanyName" subtotal="Count" />
        <Field name="ShipPostalCode" subtotal="Count" />
        <Field name="ShipRegion" subtotal="Count" />
        <Field name="UnitPrice" subtotal="Sum" format="n0" />
      </Fields>
      <RowFields>
```

```
        <Field name="Country" />
      </RowFields>
      <ColumnFields>
        <Field name="Salesperson" />
      </ColumnFields>
      <ValueFields>
        <Field name="ExtendedPrice" />
      </ValueFields>
    </C1FlexPivotEngine>
    <C1FlexPivotPrintDocument>
      <Header Text="&amp;[ViewTitle]&#x9;&amp;[Date]&#x9;Page &amp;[Page]" Separator="True"
FontName="Arial" FontSize="9" FontStyle="Regular" />
      <Footer Text="" Separator="False" FontName="Arial" FontSize="9" FontStyle="Regular" />
      <Grid ShowGrid="True" GridOptions="1" />
      <Chart ShowChart="True" ChartFillsPage="True" />
      <RawData ShowRawData="False" RawDataOptions="1" />
      <PageSettings Landscape="False" Margins="100,100,100,100" />
    </C1FlexPivotPrintDocument>
    <C1FlexPivotChart ChartType="Bar" ColorGeneration="Office" ShowTitle="True"
ShowLegend="Automatic" ShowGridLines="True" Stacked="True" />
  </C1FlexPivotPage>

  <C1FlexPivotPage id="SalesPerson vs Month">
    <C1FlexPivotEngine>
      <Fields>
        <Field name="Address" subtotal="Count" />
        <Field name="City" subtotal="Count" />
        <Field name="Country" subtotal="Count" />
        <Field name="CustomerID" subtotal="Count" />
        <Field name="Customers_CompanyName" subtotal="Count" />
        <Field name="Discount" subtotal="Sum" format="n0" />
        <Field name="ExtendedPrice" subtotal="Sum" format="n0" />
        <Field name="Freight" subtotal="Sum" format="n0" />
        <Field name="OrderDate" subtotal="Count" format="MMMM" />
        <Field name="OrderID" subtotal="Sum" format="n0" />
        <Field name="PostalCode" subtotal="Count" />
        <Field name="ProductID" subtotal="Sum" format="n0" />
        <Field name="ProductName" subtotal="Count" />
        <Field name="Quantity" subtotal="Sum" format="n0" />
        <Field name="Region" subtotal="Count" />
        <Field name="RequiredDate" subtotal="Count" format="d" />
        <Field name="Salesperson" subtotal="Count" />
        <Field name="ShipAddress" subtotal="Count" />
        <Field name="ShipCity" subtotal="Count" />
        <Field name="ShipCountry" subtotal="Count" />
        <Field name="ShipName" subtotal="Count" />
        <Field name="ShippedDate" subtotal="Count" format="d" />
        <Field name="Shippers_CompanyName" subtotal="Count" />
        <Field name="ShipPostalCode" subtotal="Count" />
        <Field name="ShipRegion" subtotal="Count" />
        <Field name="UnitPrice" subtotal="Sum" format="n0" />
      </Fields>
      <RowFields>
        <Field name="Salesperson" />
      </RowFields>
      <ColumnFields>
        <Field name="OrderDate" />
      </ColumnFields>
      <ValueFields>
        <Field name="ExtendedPrice" />
      </ValueFields>
    </C1FlexPivotEngine>
    <C1FlexPivotPrintDocument>
      <Header Text="&amp;[ViewTitle]&#x9;&amp;[Date]&#x9;Page &amp;[Page]" Separator="True"
FontName="Arial" FontSize="9" FontStyle="Regular" />
      <Footer Text="" Separator="False" FontName="Arial" FontSize="9" FontStyle="Regular" />
      <Grid ShowGrid="True" GridOptions="1" />
```

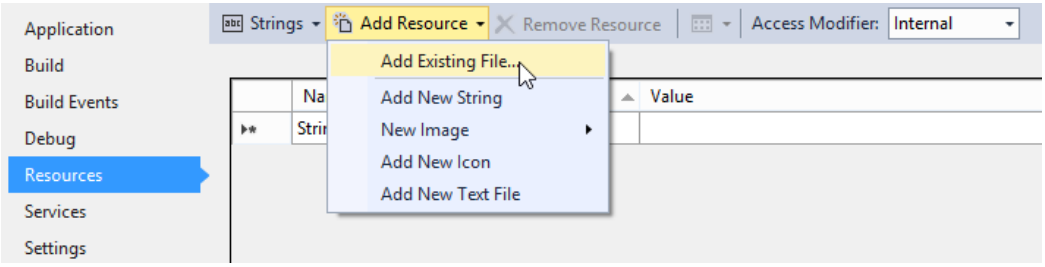
```
      <Chart ShowChart="True" ChartFillsPage="True" />
      <RawData ShowRawData="False" RawDataOptions="1" />
      <PageSettings Landscape="False" Margins="100,100,100,100" />
    </C1FlexPivotPrintDocument>
    <C1FlexPivotChart ChartType="Bar" ColorGeneration="Office" ShowTitle="True"
ShowLegend="Automatic" ShowGridLines="True" Stacked="True" />
  </C1FlexPivotPage>

  <C1FlexPivotPage id="SalesPerson vs Weekday">
    <C1FlexPivotEngine>
      <Fields>
        <Field name="Address" subtotal="Count" />
        <Field name="City" subtotal="Count" />
        <Field name="Country" subtotal="Count" />
        <Field name="CustomerID" subtotal="Count" />
        <Field name="Customers_CompanyName" subtotal="Count" />
        <Field name="Discount" subtotal="Sum" format="n0" />
        <Field name="ExtendedPrice" subtotal="Sum" format="n0" />
        <Field name="Freight" subtotal="Sum" format="n0" />
        <Field name="OrderDate" subtotal="Count" format="dddd" />
        <Field name="OrderID" subtotal="Sum" format="n0" />
        <Field name="PostalCode" subtotal="Count" />
        <Field name="ProductID" subtotal="Sum" format="n0" />
        <Field name="ProductName" subtotal="Count" />
        <Field name="Quantity" subtotal="Sum" format="n0" />
        <Field name="Region" subtotal="Count" />
        <Field name="RequiredDate" subtotal="Count" format="d" />
        <Field name="Salesperson" subtotal="Count" />
        <Field name="ShipAddress" subtotal="Count" />
        <Field name="ShipCity" subtotal="Count" />
        <Field name="ShipCountry" subtotal="Count" />
        <Field name="ShipName" subtotal="Count" />
        <Field name="ShippedDate" subtotal="Count" format="d" />
        <Field name="Shippers_CompanyName" subtotal="Count" />
        <Field name="ShipPostalCode" subtotal="Count" />
        <Field name="ShipRegion" subtotal="Count" />
        <Field name="UnitPrice" subtotal="Sum" format="n0" />
      </Fields>
      <RowFields>
        <Field name="Salesperson" />
      </RowFields>
      <ColumnFields>
        <Field name="OrderDate" />
      </ColumnFields>
      <ValueFields>
        <Field name="ExtendedPrice" />
      </ValueFields>
    </C1FlexPivotEngine>
    <C1FlexPivotPrintDocument>
      <Header Text="&amp;[ViewTitle]&#x9;&amp;[Date]&#x9;Page &amp;[Page]" Separator="True"
FontName="Arial" FontSize="9" FontStyle="Regular" />
      <Footer Text="" Separator="False" FontName="Arial" FontSize="9" FontStyle="Regular" />
      <Grid ShowGrid="True" GridOptions="1" />
      <Chart ShowChart="True" ChartFillsPage="True" />
      <RawData ShowRawData="False" RawDataOptions="1" />
      <PageSettings Landscape="False" Margins="100,100,100,100" />
    </C1FlexPivotPrintDocument>
    <C1FlexPivotChart ChartType="Bar" ColorGeneration="Office" ShowTitle="True"
ShowLegend="Automatic" ShowGridLines="True" Stacked="True" />
  </C1FlexPivotPage>

  <C1FlexPivotPage id="SalesPerson vs Year">
    <C1FlexPivotEngine>
      <Fields>
        <Field name="Address" subtotal="Count" />
        <Field name="City" subtotal="Count" />
        <Field name="Country" subtotal="Count" />
```

```
        <Field name="CustomerID" subtotal="Count" />
        <Field name="Customers_CompanyName" subtotal="Count" />
        <Field name="Discount" subtotal="Sum" format="n0" />
        <Field name="ExtendedPrice" subtotal="Sum" format="n0" />
        <Field name="Freight" subtotal="Sum" format="n0" />
        <Field name="OrderDate" subtotal="Count" format="yyyy" />
        <Field name="OrderID" subtotal="Sum" format="n0" />
        <Field name="PostalCode" subtotal="Count" />
        <Field name="ProductID" subtotal="Sum" format="n0" />
        <Field name="ProductName" subtotal="Count" />
        <Field name="Quantity" subtotal="Sum" format="n0" />
        <Field name="Region" subtotal="Count" />
        <Field name="RequiredDate" subtotal="Count" format="d" />
        <Field name="Salesperson" subtotal="Count" />
        <Field name="ShipAddress" subtotal="Count" />
        <Field name="ShipCity" subtotal="Count" />
        <Field name="ShipCountry" subtotal="Count" />
        <Field name="ShipName" subtotal="Count" />
        <Field name="ShippedDate" subtotal="Count" format="d" />
        <Field name="Shippers_CompanyName" subtotal="Count" />
        <Field name="ShipPostalCode" subtotal="Count" />
        <Field name="ShipRegion" subtotal="Count" />
        <Field name="UnitPrice" subtotal="Sum" format="n0" />
      </Fields>
      <RowFields>
        <Field name="Salesperson" />
      </RowFields>
      <ColumnFields>
        <Field name="OrderDate" />
      </ColumnFields>
      <ValueFields>
        <Field name="ExtendedPrice" />
      </ValueFields>
    </C1FlexPivotEngine>
    <C1FlexPivotPrintDocument>
      <Header Text="&amp;[ViewTitle]&#x9;&amp;[Date]&#x9;Page &amp;[Page]" Separator="True"
FontName="Arial" FontSize="9" FontStyle="Regular" />
      <Footer Text="" Separator="False" FontName="Arial" FontSize="9" FontStyle="Regular" />
      <Grid ShowGrid="True" GridOptions="1" />
      <Chart ShowChart="True" ChartFillsPage="True" />
      <RawData ShowRawData="False" RawDataOptions="1" />
      <PageSettings Landscape="False" Margins="100,100,100,100" />
    </C1FlexPivotPrintDocument>
    <C1FlexPivotChart ChartType="Bar" ColorGeneration="Office" ShowTitle="True"
ShowLegend="Automatic" ShowGridLines="True" Stacked="True" />
  </C1FlexPivotPage>

</FlexPivotViews>
```

2. Add the view definitions created above in an XML file (i.e. FlexPivotViews.xml) along with two image files CollapseAll.png () and Views_small.png () in a folder named, for example, **Resources**.
3. Add the **Resources** folder at the back end of your project and include it in your project from the Solution Explorer.
4. Right-click your project in the Solution Explorer and select Properties to open the Project Designer.
5. Select **Resources** and click the **Strings** drop-down menu. Choose **Images** option from the drop-down as illustrated in the image below.



6. Select the Add Resource drop-down menu and choose **Add Existing File** option.



7. Browse to the Resources folder where the two image files are added in Step 2 and add them in the Project Designer.
8. Click the **Strings** drop-down menu. Choose **Files** option from the drop-down.
9. Select the Add Resources drop-down menu and choose **Add Existing File** option.
10. Browse to the Resources folder where FlexPivotViews.xml file is added in Step 2 and add it in the Project Designer.
11. Switch to the code view (i.e. Form1.cs) and add the following code to build menu with predefined views.
    - **Visual Basic**

```
' build menu with predefined views:
Dim doc = New System.Xml.XmlDocument()
doc.LoadXml(My.Resources.FlexPivotViews)
Dim menuView = New C1.Win.C1Command.C1CommandMenu()
menuView.Text = "&View"
menuView.Image = My.Resources.Views_small
```

    - **C#**

```
// build menu with predefined views:
var doc = new System.Xml.XmlDocument();
doc.LoadXml(Properties.Resources.FlexPivotViews);
var menuView = new C1.Win.C1Command.C1CommandMenu();
menuView.Text = "&View";
menuView.Image = Properties.Resources.Views_small;
```

12. Add the following code below the above code snippet to apply selected view from the drop-down menu.
    - **Visual Basic**

```
For Each nd As System.Xml.XmlNode In doc.SelectNodes("FlexPivotViews/C1FlexPivotPage")
    Dim cmd = New C1.Win.C1Command.C1Command()
    cmd.Text = nd.Attributes("id").Value
    cmd.UserData = nd
    AddHandler cmd.Click, AddressOf menuView_DropDownItemClicked
    Dim link = New C1.Win.C1Command.C1CommandLink(cmd)
    menuView.CommandLinks.Add(link)
Next nd
```

    - **C#**

```
foreach (System.Xml.XmlNode nd in doc.SelectNodes("FlexPivotViews/C1FlexPivotPage"))
{
    var cmd = new C1.Win.C1Command.C1Command();
    cmd.Text = nd.Attributes["id"].Value;
    cmd.UserData = nd;
    cmd.Click += menuView_DropDownItemClicked;
    var link = new C1.Win.C1Command.C1CommandLink(cmd);
    menuView.CommandLinks.Add(link);
}
```

13. Add the following code to the event handler created for **menuView_DropDownItemClicked** event.
    - **Visual Basic**

```
Private Sub menuView_DropDownItemClicked(ByVal sender As Object, ByVal e As C1.Win.C1Command.ClickEventArgs)
    Dim nd = TryCast(e.CallerLink.Command.UserData, System.Xml.XmlNode)
    If nd IsNot Nothing Then
        ' load view definition from XML
        c1FlexPivotPage1.ViewDefinition = nd.OuterXml

        ' show current view name in status bar
        c1FlexPivotPage1.LabelStatus.Text = nd.Attributes("id").Value
    End If
End Sub
```

    - **C#**

```
private void menuView_DropDownItemClicked(object sender, C1.Win.C1Command.ClickEventArgs e)
{
    var nd = e.CallerLink.Command.UserData as System.Xml.XmlNode;
    if (nd != null)
    {
        // load view definition from XML
        c1FlexPivotPage1.ViewDefinition = nd.OuterXml;

        // show current view name in status bar
        c1FlexPivotPage1.LabelStatus.Text = nd.Attributes["id"].Value;
    }
}
```

14. Add the new View Menu to the toolstrip appearing on C1FlexPivotPage control using the following code.
    - **Visual Basic**

```
' add new view menu to C1FlexPivotPage toolstrip
Dim menuLink = New C1.Win.C1Command.C1CommandLink(menuView)
c1FlexPivotPage1.ToolBar.CommandLinks.Insert(3, menuLink)
```

- **C#**

```
// add new view menu to C1FlexPivotPage toolstrip
var menuLink = new C1.Win.C1Command.C1CommandLink(menuView);
c1FlexPivotPage1.ToolBar.CommandLinks.Insert(3, menuLink);
```

15. Initialize a new variable, for example collapseAllView, of C1Command type in the Form1 class.
    - **Visual Basic**

```
Dim collapseAllView As C1.Win.C1Command.C1Command
```

    - **C#**

```
C1.Win.C1Command.C1Command collapseAllView;
```

16. Add the CollapseAll Menu to the toolstrip appearing on C1FlexPivotPage control using the following code.
    - **Visual Basic**

```
' add collapseall menu to C1FlexPivotPage toolstrip
collapseAllView = New C1.Win.C1Command.C1Command()
collapseAllView.Text = "&CollapseAll"
collapseAllView.Image = My.Resources.CollapseAll
AddHandler collapseAllView.Click, AddressOf collapseAllView_Click
Dim collapseAllViewLink = New C1.Win.C1Command.C1CommandLink(collapseAllView)
c1FlexPivotPage1.ToolBar.CommandLinks.Add(collapseAllViewLink)
```

    - **C#**

```
// add collapseall menu to C1FlexPivotPage toolstrip
collapseAllView = new C1.Win.C1Command.C1Command();
collapseAllView.Text = "&CollapseAll";
collapseAllView.Image = Properties.Resources.CollapseAll;
collapseAllView.Click += collapseAllView_Click;
C1.Win.C1Command.C1CommandLink collapseAllViewLink = new C1.Win.C1Command.C1CommandLink(collapseAllView);
c1FlexPivotPage1.ToolBar.CommandLinks.Add(collapseAllViewLink);
```

17. Add the following code in the event handler created for **collapseAllView_Click** event.
    - **Visual Basic**

```
Private Sub collapseAllView_Click(ByVal sender As Object, ByVal e As EventArgs)
    c1FlexPivotPage1.FlexPivotGrid.CollapseAllCols()
    c1FlexPivotPage1.FlexPivotGrid.CollapseAllRows()
End Sub
```

    - **C#**

```
private void collapseAllView_Click(object sender, C1.Win.C1Command.ClickEventArgs e)
{
    c1FlexPivotPage1.FlexPivotGrid.CollapseAllCols();
    c1FlexPivotPage1.FlexPivotGrid.CollapseAllRows();
}
```

18. Subscribe **c1FlexPivotPage1.Updated** event using the code below.
    - **Visual Basic**

```
AddHandler c1FlexPivotPage1.Updated, AddressOf c1FlexPivotPage1_Updated
```

    - **C#**

```
c1FlexPivotPage1.Updated += c1FlexPivotPage1_Updated;
```

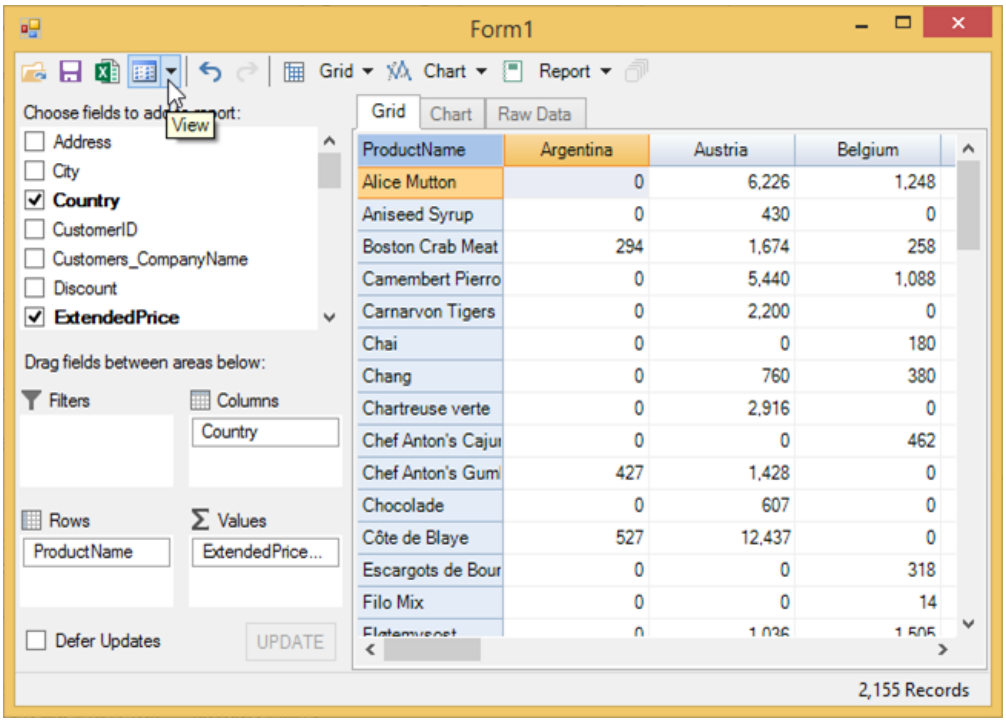
19. Add the following code to the event handler created for **c1FlexPivotPage1.Updated** event.
    - **Visual Basic**

```
Private Sub c1FlexPivotPage1_Updated(ByVal sender As Object, ByVal e As EventArgs)
    ' clear report name after user made any changes
    c1FlexPivotPage1.LabelStatus.Text = String.Empty

    ' update button status of collapseAllView.
    If (c1FlexPivotPage1.ShowTotalsColumns
    = C1.FlexPivot.ShowTotals.Subtotals Or c1FlexPivotPage1.ShowTotalsRows
    = C1.FlexPivot.ShowTotals.Subtotals) Then
        collapseAllView.Enabled = True
    Else
        collapseAllView.Enabled = False
    End If
End Sub
```

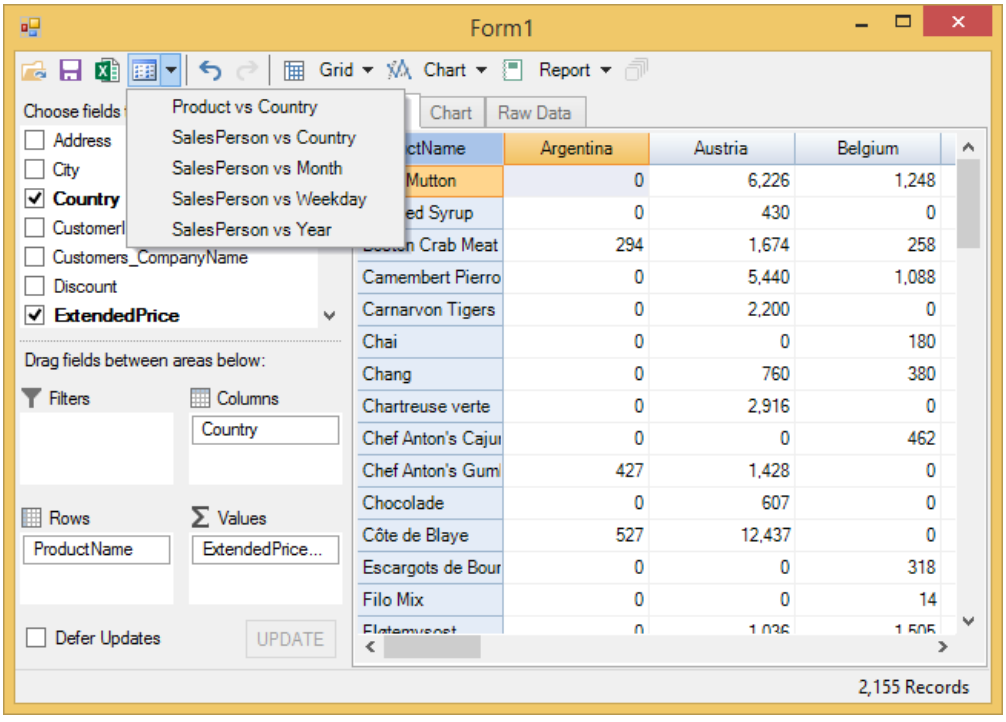
    - **C#**

```
private void c1FlexPivotPage1_Updated(object sender, EventArgs e)
{
    // clear report name after user made any changes
    c1FlexPivotPage1.LabelStatus.Text = string.Empty;

    // update button status of collapseAllView.
    if (c1FlexPivotPage1.ShowTotalsColumns == C1.FlexPivot.ShowTotals.Subtotals
    || c1FlexPivotPage1.ShowTotalsRows == C1.FlexPivot.ShowTotals.Subtotals)
        collapseAllView.Enabled = true;
    else
        collapseAllView.Enabled = false;
}
```

20. Press **F5** to run the application. The toolstrip appears with a View drop-down menu.



21. Click the View drop-down menu and observe that the list shows all the views whose definition is added in the **FlexPivotViews.xml** file.



Persisting Views

Users can also persist the view they created in their FlexPivot application across session. Any customization made by the user on the view can be saved and restored when the application is run next time.

To persist views, perform the following steps.

1. Add the following code to save the current view by reading the ViewDefinition property and assigning it to the DefaultView created.

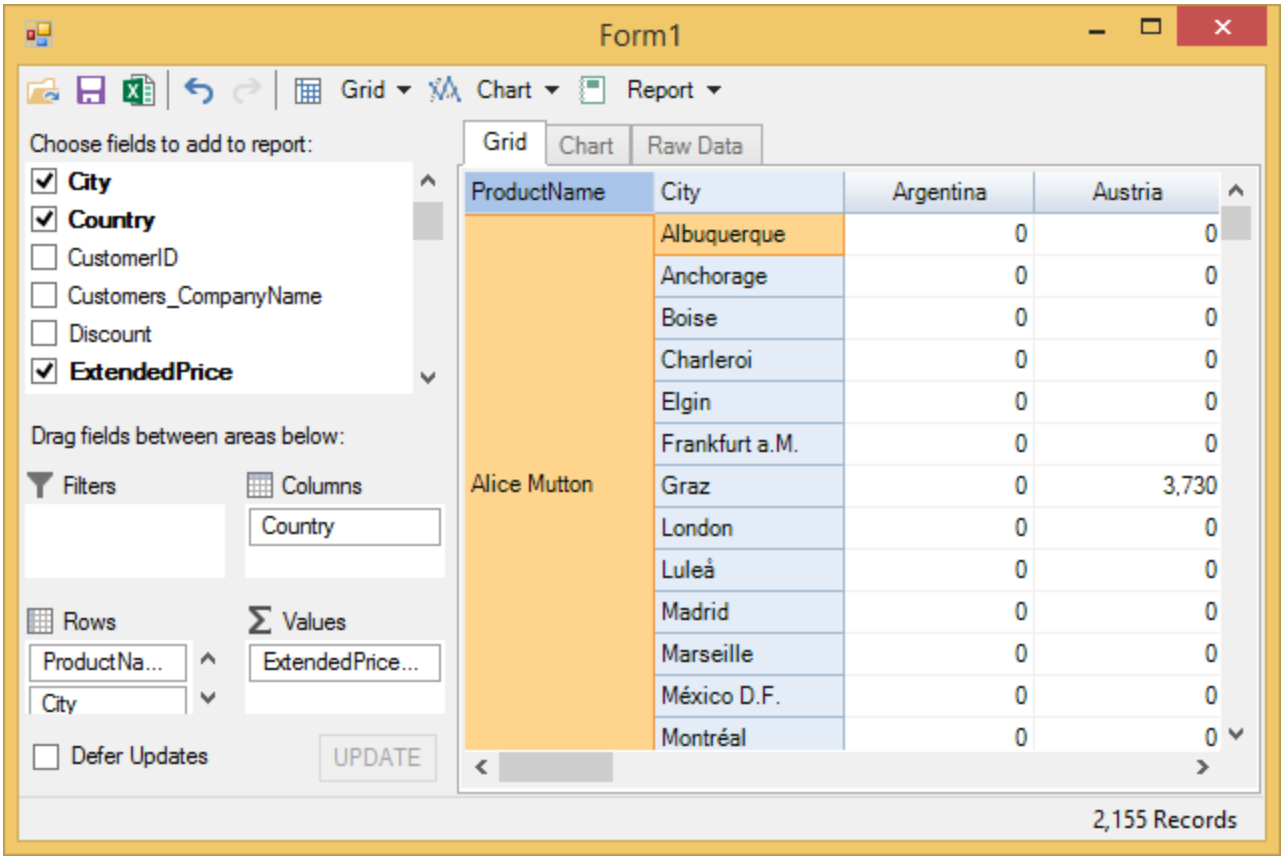
- **Visual Basic**

```
Protected Overrides Sub OnClosing(e As CancelEventArgs)
    'save current view as new default
    My.Settings.DefaultView = c1FlexPivotPage1.ViewDefinition
    My.MySettings.Default.Save()
    MyBase.OnClosing(e)
End Sub
```

- **C#**

```
// saves current view as default for next time when the form closes
protected override void OnClosing(CancelEventArgs e)
{
    // save current view as new default
    Properties.Settings.Default.DefaultView = c1FlexPivotPage1.ViewDefinition;
    Properties.Settings.Default.Save();
    base.OnClosing(e);
}
```

2. Press F5 to run the application and drag **City** data field in the Rows list. The default view appears similar to the image below.



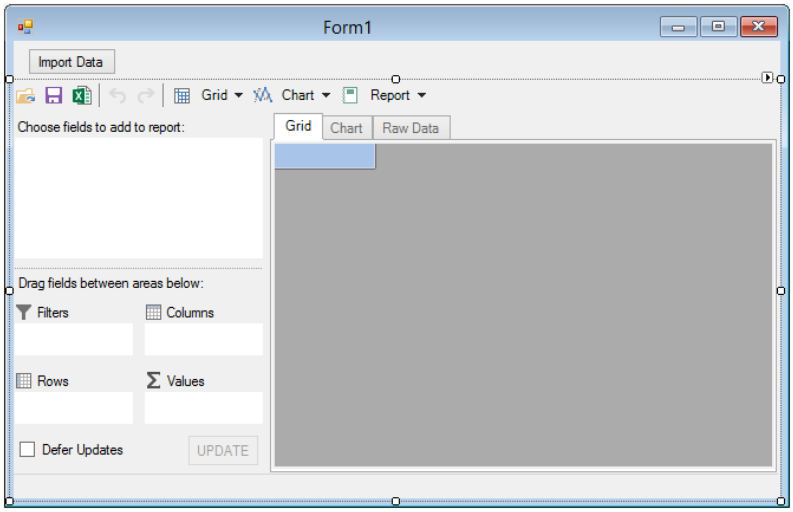
3. Close the application and restart it by pressing F5. The last view that you created persists and appears as the default view.

Importing Data from Excel

You can import data from Excel files (.xls, .xlsx) to your FlexPivot application. The code sample given below illustrates importing Excel files in C1FlexPivotPage.

Complete the following steps to import data from an Excel file to C1FlexPivotPage control. This example uses a sample Excel file named **Sales.xlsx** for importing data.

1. Create a Windows Forms Application project in Visual Studio.
2. Add C1FlexPivotPage control to the form through Toolbox.
3. Click once on the smart tag icon ( ) to open the C1FlexPivotPage Tasks smart tag panel.
4. Select **Undock in Parent Container** option to undock the FlexPivotPage control in the parent container i.e. Form.
5. Navigate to the Toolbox again and add a general button control to the Form.
6. Place the button control above the FlexPivotPage control.
7. Set the Text property for the button control as **Import Data** from the Properties window. The designer appears similar to the image given below.



8. Switch to the code view and add the following code to set up a connection string with the **Sales.xlsx** file.
   - **Visual Basic**

```
'get sample Excel file connection string
Private Function GetConnectionString(Optional firstRowHasNames As Boolean = True, Optional mixedTypesAsText As Boolean = True) As String
    Dim conn As String = "Provider=Microsoft.ACE.OLEDB.12.0;Data Source={0};Extended Properties=""Excel 12.0;HDR={1};IMEX={2};ReadOnly=true"""
    Return String.Format(conn, samplePath, firstRowHasNames, mixedTypesAsText)
End Function
```

   - **C#**

```
//get sample Excel file connection string
private string GetConnectionString(bool firstRowHasNames = true, bool mixedTypesAsText = true)
{
    string conn = "Provider=Microsoft.ACE.OLEDB.12.0;Data Source={0};Extended Properties=\"Excel 12.0;HDR={1};IMEX={2};ReadOnly=true\"";
    return string.Format(conn, samplePath, firstRowHasNames, mixedTypesAsText);

}
```

   You can store this file at Documents\ComponentOne Samples\Common\Sales.xlsx location on your system. In case you want to store the file at a different location then make changes in the path defined in the GetConnectionString method.

9. Switch to the code view and add the following import statements.
   - **Visual Basic**

```
Imports C1.DataEngine
Imports System.Data.OleDb
```

   - **C#**

```
using C1.DataEngine;
using System.Data.OleDb;
```

10. Initialize data path and sample path as illustrated in the following code.
    - **Visual Basic**

```
Dim dataPath As String = Path.Combine(System.Windows.Forms.Application.StartupPath, "Data")
Dim samplePath As String = Environment.GetFolderPath(Environment.SpecialFolder.Personal) + "\ComponentOne Samples\Common\Sales.xlsx"
```

    - **C#**

```
string dataPath = Path.Combine(System.Windows.Forms.Application.StartupPath, "Data");
string samplePath = Environment.GetFolderPath(Environment.SpecialFolder.Personal) + @"\ComponentOne Samples\Common\Sales.xlsx";
```

11. Create a class named **Sales** in the code view to read the data from Excel file.
    - **Visual Basic**

```
Public Class Sales
    Public Property salesperson() As String
        Get
            Return m_salesperson
        End Get
        Set(value As String)
            m_salesperson = Value
        End Set
    End Property
    Private m_salesperson As String
    Public Property region() As String
        Get
            Return m_region
        End Get
        Set(value As String)
            m_region = Value
        End Set
    End Property
    Private m_region As String
    Public Property account_number() As Double
        Get
            Return m_account_number
        End Get
        Set(value As Double)
            m_account_number = Value
        End Set
    End Property
    Private m_account_number As Double
    Public Property amount() As Decimal
        Get
            Return m_amount
        End Get
        Set(value As Decimal)
            m_amount = Value
        End Set
    End Property
    Private m_amount As Decimal
    Public Property month() As String
```

```
        Get
            Return m_month
        End Get
        Set(value As String)
            m_month = Value
        End Set
    End Property
    Private m_month As String

    Public Sub New(reader As IDataReader)
        Dim nv = New NullValue()
        salesperson = If(reader.IsDBNull(0), nv.NullString, reader.GetString(0))
        region = If(reader.IsDBNull(1), nv.NullString, reader.GetString(1))
        account_number = If(reader.IsDBNull(2), nv.NullDouble, reader.GetDouble(2))
        amount = If(reader.IsDBNull(3), nv.NullDecimal, reader.GetDecimal(3))
        month = If(reader.IsDBNull(4), nv.NullString, reader.GetString(4))
    End Sub

    Public Shared Iterator Function GetSalesInfo(reader As IDataReader) As IEnumerable(Of Sales)
        While reader.Read()
            Yield New Sales(reader)
        End While
    End Function

End Class
```

- **C#**

```
public class Sales
{
    public string salesperson { get; set; }
    public string region { get; set; }
    public double account_number { get; set; }
    public decimal amount { get; set; }
    public string month { get; set; }

    public Sales(IDataReader reader)
    {
        var nv = new NullValue();
        salesperson = reader.IsDBNull(0) ? nv.NullString : reader.GetString(0);
        region = reader.IsDBNull(1) ? nv.NullString : reader.GetString(1);
        account_number = reader.IsDBNull(2) ? nv.NullDouble : reader.GetDouble(2);
        amount = reader.IsDBNull(3) ? nv.NullDecimal : reader.GetDecimal(3);
        month = reader.IsDBNull(4) ? nv.NullString : reader.GetString(4);
    }

    public static IEnumerable<Sales> GetSalesInfo(IDataReader reader)
    {
        while (reader.Read())
            yield return new Sales(reader);
    }
}
```

12. Initialize workspace in the Form's constructor.
    - **Visual Basic**

```
Public Sub New()
    InitializeComponent()
    C1.DataEngine.Workspace.Init(dataPath)
End Sub
```

    - **C#**

```
public Form1()
{
    InitializeComponent();
    C1.DataEngine.Workspace.Init(dataPath);
}
```

13. Add the following code to fetch data from the Excel file.
    - **Visual Basic**

```
Private Function GetFirstSalesData() As String
    Using conn As New OleDbConnection(GetConnectionString())
        conn.Open()
        ' get workbook table list
        Dim tables = conn.GetOleDbSchemaTable(OleDbSchemaGuid.Tables, New Object() {Nothing, Nothing, Nothing, "TABLE"})

        Dim name As String = tables.Rows(0)("TABLE_NAME").ToString()

        Dim command = New OleDbCommand((Convert.ToString("select * from [") & name) + "]", conn)
        Using reader = command.ExecuteReader()
            Dim connector = New ObjectConnector(Of Sales)(Sales.GetSalesInfo(reader))
            connector.GetData(name)
        End Using
        Return name
    End Using
End Function
```

    - **C#**

```
private string GetFirstSalesData()
{
    using (OleDbConnection conn = new OleDbConnection(GetConnectionString()))
    {
        conn.Open();
        // get workbook table list
        var tables = conn.GetOleDbSchemaTable(OleDbSchemaGuid.Tables, new object[] { null, null, null, "TABLE" });

        string name = tables.Rows[0]["TABLE_NAME"].ToString();

        var command = new OleDbCommand("select * from [" + name + "]", conn);
        using (var reader = command.ExecuteReader())
        {
            var connector = new ObjectConnector<Sales>(Sales.GetSalesInfo(reader));
            connector.GetData(name);
        }
        return name;
    }
}
```

14. Subscribe button1_click event from the Properties window.
15. Add the following code to the event handler created in the above step.
    - **Visual Basic**

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
```

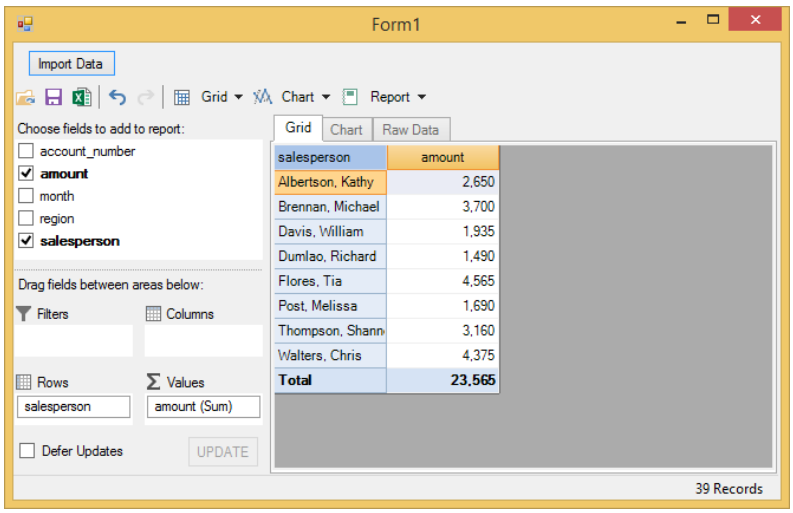
```
    Dim tableName As String = GetFirstSalesData()
    C1FlexPivotPage1.FlexPivotPanel.ConnectDataEngine(tableName)

    'build a default view
    Dim engine = C1FlexPivotPage1.FlexPivotPanel.FlexPivotEngine
    engine.BeginUpdate()
    engine.RowFields.Add("salesperson")
    engine.ValueFields.Add("amount")
    engine.EndUpdate()
End Sub
```

- **C#**

```
private void button1_Click(object sender, EventArgs e)
{
    string tableName = GetFirstSalesData();
    c1FlexPivotPage1.FlexPivotPanel.ConnectDataEngine(tableName);

    //build a default view
    var engine = c1FlexPivotPage1.FlexPivotPanel.FlexPivotEngine;
    engine.BeginUpdate();
    engine.RowFields.Add("salesperson");
    engine.ValueFields.Add("amount");
    engine.EndUpdate();
}
```

This code connects the Data Engine to the sample table and builds a default view to be displayed on running the application.

16. Press **F5** to run the application and click the button control appearing on the form to import data from the sample file.



C1DataEngine Overview

ComponentOne Studio's **DataEngine** powers FlexPivot control, but this is not its only purpose. C1DataEngine can also be used on its own, separate from FlexPivot, to integrate (blend, join), transform, and aggregate (analyze) data. In this section, you learn to use the DataEngine independent of FlexPivot through queries.

Using C1DataEngine

The C1DataEngine can be used independently for data blending and analysis by executing queries. A query takes data from base tables or from other queries, performs operations on that data such as joining multiple tables, filtering, calculations, grouping, aggregation, and saves results as a new table in the DataEngine.

Query results can then be used in various ways including but not limited to:

- Showing results in a grid (for example, aggregation results, for analysis)
- Feeding results to FlexPivot for analysis (for example, joining several tables so FlexPivot can analyze the joined table)
- Further programmatically transform the results to perform any batch tasks on data

The unique advantage of using DataEngine queries compared with other programming techniques is their exceptional performance. A query can often process millions of records in a second or less. In addition, DataEngine queries support custom operations, which allows you to create user-specific queries for both simple and group operations (aggregation).

Refer to the product sample **DataEngineQueries** sample at the following location in your system to see the C1DataEngine in full action.

C:\...\Documents\ComponentOne Samples\WinForms\C1FlexPivot\CS or VB\DataEngineQueries.

Base Tables

Base tables are simply data imported into DataEngine from external sources using connectors. Base tables act as input data for queries. Since a query or query result is also a table, you can distinguish between the two by the fact that base tables are not produced by queries. Every base table has a name (it must be unique in the workspace) and a collection of columns. When a workspace is initialized in code, some base tables can already exist (their data stored in the workspace's folder in memory-mapped files) and some can be added after workspace initialization.

A query is based on a table or on multiple tables if it's a join query. We will consider queries based on a single table first. A simple, non-join query is based on a base table or another query. Let us start with a simple case of a query over a base table first. To define query, you need to get hold of a base table first. This can be done using the syntax below.

Syntax

Visual Basic

```
Dim od As Object = workspace.table("OrderDetails")
```

C#

```
dynamic od = workspace.table("OrderDetails");
```

This is an object representing a base table with the name "OrderDetails". Note that such table must already exist (i.e. must have been created earlier by importing data via a connector). This object is dynamic to make it easy (and type-safe) for you to reference table columns. For example, od.ProductID is the ProductID column, and od.Discount is the Discount column of the OrderDetails table.

Simple Operations

After getting hold of the base tables, you can perform a range of operations by formulating queries. Having the easy way to represent columns, we can formulate a query for simple operations as illustrated below.

Syntax

Visual Basic

```
Dim query1 As Object = workspace.query("prices", New With { _
            Key .Order = od.OrderID, _
            Key .Product = Od.ProductID _
            Key .Price = Op.Mul(od.UnitPrice, od.Quantity), _
        })
```

C#

```
dynamic query1 = workspace.query("prices", new
    {
        Order = od.OrderID,
        Product = od.ProductID,
        Price = Op.Mul(od.UnitPrice, od.Quantity)
    });
```

This is a simple query over a single table without grouping and aggregation. It creates a table (result of any query is a table) having the same number of rows as the base table OrderDetails. It has three columns called Order, Product, and Price. In every row, columns Order and Product contain the same order and Product IDs as in the base OrderDetail row. The Price column is unit price times quantity. **Op.Mul** is multiplication operation. Other binary operations have similar notation:

- **Op.Add** - for addition
- **Op.Sub** - for subtraction
- **Op.Div** - for division
- **Op.Mod** - for modulus

There are also some unary operations. For example, Op.UCase(products.ProductName) converts string to upper case. All operations are listed in the class C1.DataEngine.Op

To execute a query, use the **Execute** method as illustrated below.

Syntax

Visual Basic

```
query1.Query.Execute()
```

C#

```
query1.Query.Execute();
```

Aggregation

Aggregation (subtotals) operations compute a single value from a collection of values using operators such as Sum, Min, Max, Count, etc. The most basic difference between Aggregation operations and Simple operations is that aggregation uses grouping. Aggregation without grouping is also possible but in that case it produces a single row of grand totals. Grouping is needed to produce subtotals. Input rows are grouped by values of one or more columns, and subtotals are calculated for each of those groups.

The following syntax illustrates an aggregation query that calculates subtotals for each products, so the number of resulting row is equal to the number of products. For each product, two subtotals (aggregations) are calculated. These subtotals are number of orders including that product, and maximum discount for that product in those orders.

Syntax

Visual Basic

```
Dim query2 As dynamic = workspace.query("subtotals", New With { _
        Key .Product = od.ProductID, _
        Key .OrdersCount = Op.Count(od.OrderID), _
        Key .MaxDiscount = Op.Max(od.Discount) _
})
```

C#

```
dynamic query2 = workspace.query("subtotals", new
    {
        Product = od.ProductID,
        OrdersCount = Op.Count(od.OrderID),
        MaxDiscount = Op.Max(od.Discount)
    });
```

While using aggregation, you can group more than one column. The following query illustrates this as it has two grouping columns as a result of which there will be more groups in the result.

Syntax

Visual Basic

```
Dim query3 As dynamic = workspace.query("subtotals3", New With { _
        Key .Product = od.ProductID, _
        Key .Discount = od.Discount, _
        Key .OrdersCount = Op.Count(od.OrderID) _
})
```

C#

```
dynamic query3 = workspace.query("subtotals3", new
    {
        Product = od.ProductID,
        Discount = od.Discount,
        OrdersCount = Op.Count(od.OrderID)
    });
```

In the above query results in more than one group each containing orders including a certain product with a certain discount, and the subtotal will show the number of orders in the group.

Combining Operations and Queries

A query line can have only have one operation in it. In simple words, you cannot combine operations to form a complex formula. However, this is not a problem because queries can be easily combined and results of any query can be fed as input to another query. So, you can perform some simple operations first and then apply an aggregation query to calculate subtotals on their results. You can also apply simple calculation operations to results of an aggregation, or build any other combination you need.

One common case is specifically supported so that you can avoid creating multiple queries. You can define a column in a query and then use its string name as the argument of an operation. This combination can be represented in a single query as follows.

Syntax

Visual Basic

```
discount = Op.Max(Op.Mul(od.UnitPrice, od.Discount)),
MaxDiscount = Op.Max("discount")
```

C#

```
discount = Op.Max(Op.Mul(od.UnitPrice, od.Discount)),
MaxDiscount = Op.Max("discount")
```

When you combine queries, that is, define a query that is based on another query, the resulting query contains by default only the columns that are defined in it explicitly. In other words, it does not by default include columns from the query on which it is based. However, sometimes your query just needs to add a few columns to those in the base one (for example, add two columns of the base query) or it might show the same columns only filtered by a condition. You can do it by adding a line for each column as given below:

columnName = baseQuery.columnName

A shortcut for the above case is adding **_base = "*"**

Filter and Range

You can restrict your query calculation to some subset of the input rows using a filter condition. Or maybe the entire

purpose of your query is to select a subset of rows from a base table or another query result based on some filter condition or a range of values of a column or several columns. All this is achieved by filter conditions. There are two kinds of conditions, namely _range and _filter.

Filter(_filter)	Range (_range)
_filter is more general as it can represent any condition. However, it has performance cost because it scans the entire set of input rows and checks the condition on every row.	_range is a special case of filter in which DataEngine does not need to scan all rows, and hence performs faster.

Range

The following query illustrates how to formulate range condition.

Syntax

Visual Basic

```
Dim query4 As dynamic = workspace.query("range", New With { _
        Key ._base = "*", _
        Key ._range = od.UnitPrice.Eq(10) _
})
```

C#

```
dynamic query4 = workspace.query("range", new
    {
        _base = "*",
        _range = od.UnitPrice.Eq(10)
    });
```

The above query selects OrderDetails rows with unit price = 10. This process is very efficient as the query traverses to the desired segment of rows using an index built internally by DataEngine and outputs just those rows with unit price = 10 without checking any other rows. In addition, several condition operations can also be applied to a column. The above query uses is Eq that is "equals". Other common operations are Gte, Lte (greater than or equals, less than or equals), Gt, Lt (greater than, less than), etc. Conditions applied to a column can be combined as follows.

Syntax

```
_range = od.UnitPrice.Gte(10).Lte(20)
```

This kind of 'between' condition is still appropriate for _range, performance optimization is still possible because row scan can be constrained to scanning the from-to segment of rows. You can also specify ranges with conditions imposed on two or more columns as illustrated below.

Syntax

```
_range = od.Discount.Eq(0) + od.UnitPrice.Gte(10).Lte(20)
```

Filter

Some conditions cannot be specified as _range. For example, A "not equal" or Ne condition cannot be used in a range. So, to use Ne condition, you need to apply _filter as illustrated below.

Syntax

```
_filter = od.Discount.Ne(0)
```

DataEngine conditions can also be combined with **'or'**, not only with **'and'**. However, 'or' can only be used in a filter as illustrated below.

Syntax

```
_filter = od.UnitPrice.Lt(10).Or().Gt(20)
```

There is also an And() operation that allows you to combine 'or' and 'and' in a single filter as illustrated below.

Syntax

```
_filter = od.UnitPrice.Lt(10).Or().Gt(20).And().Lt(30)
```

The above query means either less than 10 or between 20 and 30.

You can have any number of Or() and And() connectives, and impose conditions on multiple columns combining them with + as illustrated below.

Syntax

```
_filter = od.Discount.Ne(0) + od.UnitPrice.Lt(10).Or().Gt(20).Or().Gt(30)
```

Join

All the queries that were discussed in the previous topics were based on a single table. However, one cannot restrict the queries to a single table and so applying queries on multiple tables becomes important.

Join is a very important feature of DataEngine queries that allows you to combine (blend) data from several tables (base tables or queries). Join query is a special kind of query that does not have any calculation or aggregation operations. The only purpose it serves is that of joining tables. If you need calculations and/or aggregation subtotals, you can do this on joined tables as well since all the DataEngine queries, including joins, can be combined as without restrictions as discussed in Combining Operations and Queries topic.

Joins in DataEngine are always many-to-one, which corresponds to the common "star schema" convention in data analytics. So there is one main table and one or more 'attached' tables that are linked to the main table. Join result contains the same number of rows as the main table. Every joined row contains a value for each main table column included in the result (not necessarily all columns are included, although including all columns is a common case) and a value for each included column of each linked table (usually only some of the columns of linked tables are included).

The following example illustrates a Join.

Syntax

Visual Basic

```
Dim jq As dynamic = workspace.join("query_join", od, New With { _
        Key .od = od.ProductID + od.OrderID, _
        Key .pr = pr.UnitsInStock Or od.ProductID = pr.ProductID _
})
```

C#

```
 dynamic jq = workspace.join("query_join", od, new
    {
```

```
        od = od.ProductID + od.OrderID, // get ProductID and OrderID
from the OrderDetails table
        pr = pr.UnitsInStock | od.ProductID == pr.ProductID // get
UnitsInStock from the Products table and join OrderDetails and Products
table on the ProductID field
    });
```

Main table is specified in the second parameter, od (it is our old OrderDetails table). First line applies to the main table and defines which columns we take from the main table to the result. In this case we take two columns, ProductID and OrderID. The main table line is optional. If it's not present, all columns from the main table are included in the join.

Second line defines a linked table, Products, and fields we take from it, UnitsInStock. It also defines, after the separator | how we link Products to the main table OrderDetails. A linking (join) condition must be an equality between one or more columns from each side (they are usually called key columns).

Here is a more general example showing that there can be more linked tables than one (but only one main table) and join conditions can have more than one key column (Main table - Orders; Linked tables - Customers, Employees).

Syntax

Visual Basic

```
Dim jq As dynamic = workspace.join("join2", orders, New With { _
        Key .customers = customers.Company + customers.Country Or
orders.CustomerID = customers.customerID, _
        Key .employees = employees.Name + employees.Title Or
orders.EmployeeID = employees.EmployeeID + orders.CustomerID =
customers.customerID _
})
```

C#

```
dynamic jq = workspace.join("join2", orders, new
    {
        customers = customers.Company + customers.Country |
orders.CustomerID == customers.customerID,
        employees = employees.Name + employees.Title | orders.EmployeeID
== employees.EmployeeID + orders.CustomerID == customers.customerID
    });
```

By default, columns in the result keep their names that they had in the main and in the linked tables. However, sometimes these names can collide, so you need to change the name of a column in the result (or might just want to give a column a different name).

Main table columns always keep their names, but names of linked columns can be changed with As("alias") as illustrated below:

Example Title

```
customers = customers.Company.As("CustomerCompany") + customers.Country |
orders.CustomerID ==
    customers.customerID,
```

Refer to the product sample **DataEngineQueries** stored at the following location in your system to see how to use join both with and without FlexPivot.

C:\...\Documents\ComponentOne Samples\WinForms\DataEngineQueries

Also, refer to the product sample **DataJoin** stored at the following location in your system to see how to build a join query dynamically based on user selection of tables and fields.

C:\...\Documents\ComponentOne Samples\WinForms\DataJoin

Excel Add-in

Using Excel Add-in

C1DataEngine has built-in connectors for getting data from databases or any collection. In addition to that, C1DataEngine also includes an add-in to Excel that opens up the wealth of data sources available through Microsoft Power Query.

Power Query, included in Microsoft Excel, provides connectors to multiple data sources, including cloud as well as big data. The DataEngine Excel add-in can be used in two modes illustrated below:

Programmatic Import to DataEngine

In this mode, the excel remains invisible and is used behind the scenes from the user application code via automation to directly fetch data from any of the Power Query data sources. In this scenario, developers or information workers use Excel to develop an application. Once the application is complete, it functions without opening the Excel. This way Power Query serves as a powerful extension of C1DataEngine to connect to a variety of sources.

Exporting Data to Excel

For Excel users, the Excel add-in has a button for exporting data from any of the Power Query data sources to C1DataEngine. This mode makes analytical applications very flexible from the point of view of data acquisition. Developers or information workers can use Excel to make queries to various data sources and the application will analyze any such data without changes to application code.

Refer to the product sample **PowerQueryConnection** sample at the following location in your system to see the C1DataEngine and Power Query in full action.

C:\...\Documents\ComponentOne Samples\WinForms\PowerQueryConnection.

Important Note:

DataEngine Excel add-in is located in ***C:\...\ComponentOne\C1FlexPivot\ExcelAddIn\bin*** directory. To install the add-in, complete the steps given below:

1. First execute the file registerCOM.bat
2. Double-click the file C1.DataEngine.ExcelAddIn.vsto in the folder.

The Excel add-in is compatible with Excel 2016 and Excel 2013. For Excel 2013, Power Query add-in must be installed.