ComponentOne

List 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

List for WinForms Overview

List for WinForms is a set of advanced list box controls that allow end users to browse, edit, add, and delete data in a tabular format. With two robust controls, C1List a full-featured list box control, and C1Combo, a multicolumn drop-down column list box, **List for WinForms** completely manages the database interface, allowing you to concentrate on more important application-specific tasks.

With features like Microsoft Excel-like splits, incremental searching, in-cell objects, Office 2007 visual styles, and more, **List for WinForms** is a powerful, versatile, and easy-to-use data presentation tool that adapts to your requirements and is customizable to meet your users' needs. Whether simple or advanced, you can use **List for WinForms** to create a fully functional database browser with little or no code or use **List for WinForms**' many properties and events to create sophisticated and user-friendly database front-end applications.

Help with WinForms Edition

Getting Started

For information on installing **ComponentOne Studio WinForms Edition**, licensing, technical support, namespaces and creating a project with the control, please visit Getting Started with WinForms Edition.

Key Features

List for WinForms includes dozens of advanced data access, data presentation, and user interface features that enable developers to build intuitive, professional-looking applications:

- **Incremental Search**

  End users can quickly locate list items by typing. Supports both single-character matching and an incremental search mode that concatenates typed characters to further refine the search.

- **Automatic Field Completion**

  Works in concert with incremental search to fill the text portion of a combo with matching field data as characters are typed.

- **Excel and Word-Like Styles**

  Hierarchical style objects encapsulate font, color, picture and formatting information, facilitating easy customization of **List for WinForms** components at design time and run time.

- **Excel-Like Splits**

  Developers and end users can divide the list into separate vertical and horizontal panes to provide multiple views of the data. The splits can scroll independently or simultaneously.

- **Fixed, Nonscrolling Columns**

  Splits can also be used to create nonscrolling columns anywhere in the list (at the left or right edges, or in the middle).

- **In-Cell Objects**

  The list supports a variety of in-cell objects for data display, including check boxes and radio buttons.

- **Automatic Data Translation**

  Database values can be automatically translated into alternate text or graphics without coding. For example, numeric codes can be rendered as words or even bitmaps.

- **Data-Sensitive Display**

  A powerful regular expression facility can be used to apply different styles to individual cells depending upon their contents. For example, negative numbers can be shown in red, or fields containing a particular substring can be shown in bold.

- **Interactive Visual Editing**

  Programmers can create columns, retrieve field layouts from a bound data source, resize **List for WinForms** components and configure all aspects of the list layout at design time no coding is required.

- **Unbound Columns**

  The list supports unbound columns while other columns are bound to a data control.

- **Run-Time CellTips**

  Provides context-sensitive help for end users.

- **Alternating Row Colors**

Enhances the readability of the list's display.

- **Excellent Documentation**

  List for WinForms includes an extensive manual and online help with plenty of tutorials and examples. If you have suggestions on how we can improve our documentation, please email the Documentation team. Please note that e-mail sent to the Documentation team is for documentation feedback only. Technical Support and Sales issues should be sent directly to their respective departments.

- **Free Run-Time Distribution**

  No royalty fees required.

- **New Vertical Splits**

  Like Microsoft Excel, the **List for WinForms** now has the ability to split both horizontally and vertically allowing for a list matrix as well as singular horizontal and vertical views. In addition, a user-friendly matrix dereferencing notation makes splits quite accessible even with multiple horizontal and vertical splits.

- **New Column Objects**

  List for WinForms now has two separate column objects to help simplify the sometimes daunting object model. The **C1DataColumn** object contains all of the properties related to data and data processing, while the **C1DisplayColumn** object contains all of the properties related to the column's display.

- **Style Border Properties**

  The expanded style object now includes properties that let you customize the appearance, size, color, and type of cell borders.

- **Scroll Tracking and ScrollTips**

  The vertical scroll bar thumb can now scroll records as it is moved. You can also use the ScrollTips feature to provide an informational pop-up during scrolling.

- **Visual Styles**

  Visual Styles, including Office 2007 Visual Styles, simplify the visual customization process by allowing you to change styles for the **List for WinForms** control quickly and easily.

- **And Much More**

  Auto completion for drop-downs, tag property for column objects, right to left support, and a wide variety of print enhancements.

Object Model

List for WinForms was developed using the latest .NET technologies. The C1List control and its programmable components are all .NET objects designed according to Microsoft specifications. If you are already familiar with the Microsoft .NET object and collection models, you will have no problem using C1List.

If you are new to Visual Studio, please read Working with Objects and Collections, which illustrates how to manipulate C1List objects in code. Although individual objects are designed to perform different tasks, the techniques used to manipulate them are the same. Once you have mastered these common programming constructs, using Visual Studio and .NET controls will be quite easy and intuitive.

Regardless of your experience level, please read the following section, as it provides a thumbnail sketch of all C1List objects and collections.

List Objects and Collections

List for WinForms provides a rich set of properties, methods, and events that enable you to develop sophisticated database applications. The organization imposed by **List for WinForms**' object model makes it easier to work with such a large feature set.

Objects and collections that refer to visual entities, such as columns, can be customized in the designer or in code. Objects and collections that refer to abstract entities, such as arrays and bookmarks, are only available in code.

Two controls are available in the .NET Toolbox for addition into a project:

Control	Description
C1List	List .NET control.
C1Combo	Combo .NET control.

The namespace for **List for WinForms** also contains definitions for the following objects:

Object	Description
C1DataColumn	Represents a column of data within a list.
C1DisplayColumn	Represents a column of data relative to a split.
GridLines	Represents the gridlines which separate items in the list.
HBar	Represents the horizontal scroll bar and its properties.
Split	Represents a group of adjacent columns that scroll as a unit.
Style	Encapsulates font, color, picture, and formatting information.
ValueItems	Encapsulates both the Values collection and valueitem properties.
ValueItem	Allowable column input values, with optional translation.
VBar	Represents the vertical scroll bar

Object	Description
	and its properties.

A *collection* is an object used to group similar data items, such as list columns or styles. In general, a group of similar items in **List** is implemented as a collection. Since a collection is an object, you can manipulate it in code just as you would any other object. **List for WinForms** exposes the following collections:

Collection	Description
C1DataColumnCollection	Contains zero or more C1DataColumn objects in a list.
C1DisplayColumnCollection	Contains zero or more C1DisplayColumn objects in a list.
SelectedRowCollection	Contains zero or more selected row bookmarks
SelectedColumnCollection	Contains zero or more selected row bookmarks
SplitCollection	Contains one or more Style objects in a list.
GridStyleCollection	Contains built-in and user-defined Style objects for a list.
ValueItemCollection	Contains zero or more ValueItem objects for a column.

The following sections provide a brief overview of **List for WinForms**' objects and collections.

List for WinForms Object

Using **List for WinForms** C1DataColumnCollection and C1DisplayColumnCollection objects, you can create, access, and modify the column objects that define the mappings between the list's physical columns and the underlying database fields. Using its SplitCollection object, you can divide the list into multiple vertical panes to provide different views of the same data source.

C1Combo Object

The C1Combo control is used as a multicolumn drop-down list box. C1Combo is independent of C1List, so it can be used alone or in conjunction with C1List. You can place a C1Combo control on a Visual Studio form at design time as you would a C1List control.

The C1Combo control also supports incremental search.

C1DataColumnCollection Object

The C1List control and the C1Combo control both maintain a C1DataColumnCollection object to hold and manipulate C1DataColumn objects. This collection is contained under the C1List object and can be modified through .NET's collection editor. This collection is available through the C1ListBase.Columns property of the **List for WinForms**. In addition, this collection can be modified through .NET's C1List Designer.

C1DataColumn Object

Each column within a C1List control, C1Combo control, or **Split** object is represented by two column objects, one global and one split specific. All of the properties related to data access and formatting are contained under the C1DataColumn object. The properties of the C1DataColumn object are global in scope, which means changing a C1DataColumn property changes that value for all columns, even across splits. You can access the C1DataColumn object as follows:

To write code in Visual Basic

Visual Basic

```
Me.C1List1.Columns(0).Caption = "Region"
```

To write code in C#

C#

```
this.c1List1.Columns[0].Caption = "Region";
```

C1DisplayColumnCollection Object

The C1List control and the C1Combo control both maintain a C1DataColumnCollection object to hold and manipulate C1DisplayColumn objects. This collection is contained under the SplitCollection object, and is available through the Split's **DisplayColumns** property. In addition, this collection can be modified in .NET through the C1DisplayColumn Collection Editor.

C1DisplayColumn Object

Each split within the list contains at least one C1DisplayColumn object. All of the properties related to a column's display are contained under this object. Unlike the C1DataColumn properties, the properties of the C1DisplayColumn object are split-specific. Changing a C1DisplayColumn property only changes that value for only the specified column inside the specified split. You can access the object as follows:

To write code in Visual Basic

Visual Basic

```
Me.C1List1.Splits(0, 0).DisplayColumns(0).Style.ForeColor = System.Drawing.Color.Blue
```

To write code in C#

C#

```
this.c1List1.Splits[0, 0].DisplayColumns[0].Style.ForeColor =
System.Drawing.Color.Blue;
```

SplitCollection Object

The C1List control maintains a SplitCollection object to hold and manipulate **Split** objects. A list has one split by default, but may contain multiple splits. This collection is accessed using the C1ListBase.Splits property of the C1List control. In addition, this collection can be modified in .NET through the Split Collection Editor.

Split Object

List for WinForms supports Excel-like splits that divide the list into vertical and horizontal panes to provide users with different views of the data source. Each split is represented by a **Split** object and contains a group of adjacent columns that scroll as a unit.

When a C1List control is created, it contains one **Split** object by default. Many of the properties of the **Split** object also apply to the C1List control as a whole, so you do not need to concern yourself with splits until you actually need to use them, such as when creating fixed, nonscrolling columns. You can access the object as follows:

To write code in Visual Basic

```
Visual Basic
Me.C1List1.Splits(0).Caption = "Split00"
```

To write code in C#

```
C#
this.c1List1.Splits[0].Caption = "Split00";
```

GridStyleCollection Object

The C1List and C1Combo controls store all built-in and user-defined **Style** objects in the GridStyleCollection object. You can access the members of this collection by name in code, then apply them to a grid, column or split in order to control the appearance of the object in question. This collection is accessed using the Styles property in the C1List. In addition, this collection and its members can be modified through the Style Collection Editor.

Style Object

Style objects encapsulate font, color, picture, and formatting information for a C1List, C1Combo, **Split**, or C1DataColumn object. The Style object is a very flexible and powerful tool that provides Excel and Word-like formatting capabilities for controlling the appearance of the list's display.

When a C1List or C1Combo control is created, it contains ten built-in styles. You can modify the built-in styles or add your own styles either in the designer or in code. Both controls also support several optional events that use Style objects to convey formatting information on a per cell or per row basis. You can access the object as follows:

To write code in Visual Basic

```
Visual Basic
Me.C1List1.Styles("Normal").Backcolor = System.Drawing.Color.Gray
```

To write code in C#

```
C#
this.c1List1.Styles["Normal"].Backcolor = System.Drawing.Color.Gray;
```

ValueItems Object

The ValueItems object contains a collection and a couple of properties that can create alternate display values for database values in the list. It can specify an allowable input value for a given C1DataColumn object or it can also be used to translate raw data values into alternate text or graphics for display. For example, you may want to display *Balance Due* and *Paid in Full* instead of the numeric data values 0 and 1. The ValueItems object contains display properties and a collection of ValueItem objects, the ValueItemCollection. You can access this object as follows:

To write code in Visual Basic

```
Visual Basic
Me.C1List1.Columns(0).ValueItems.Values
```

To write code in C#

```
C#
this.c1List1.Columns[0].ValueItems.Values;
```

ValueItemCollection Object

Each C1DataColumn object within a C1List or C1Combo control stores its set of display value/value pairs in objects called ValueItem objects. The ValueItemCollection object is a collection of these pairs. This collection can be accessed through the Values property of the ValueItems object. For instance, in order to alter the first ValueItem in the collection, the code would look like:

To write code in Visual Basic

```
Visual Basic
Me.C1List1.Columns(0).ValueItems.Values(0).DisplayValue = "Canada"
```

To write code in C#

```
C#
this.c1List1.Columns[0].ValueItems.Values[0].DisplayValue = "Canada";
```

ValueItem Object

The ValueItem object consists of two properties: DisplayValue and Value. The Value property specifies the underlying value in the database and the DisplayValue property specifies the value you would like displayed in the list. These objects are contained in the ValueItemCollection object, and can be edited in .NET's ValueItemCollection Editor. This editor is available in the C1List Designer under the ValueItems object.

HBar Object

The HBar object is used to specify properties of the horizontal scrollbar. With the HScrollBar property, you can specify the height of the scroll bar, and whether it shows up automatically or not at all.

VBar Object

The VBar object is used to specify properties of the vertical scrollbar. With the VScrollBar property, you can specify the

width of the scroll bar, and whether it shows up automatically or not at all.

GridLines Object

The GridLines object is used to specify characteristics of the ColumnDivider and RowDivider properties. Both the color and style of the column and row lines are able to be manipulated at run-time or design-time with the GridLines object.

CellBorders Object

The **CellBorders** object is used to specify the characteristics of the BorderStyle property of a Style. This property sets the column borders for the cell. With this object, you can specify the width of each border around a cell and the color of the cell border.

SelectedRowCollection Object

When the user selects and highlights one or more rows of a C1List control at run time, the bookmarks of the selected rows are stored in the SelectedRowCollection object. In code, you can use the Item property and IndexOf method of the collection to determine which rows are selected. You can also select and deselect records programmatically using its Insert and RemoveAt methods.

SelectedColumnCollection Object

When the user selects and highlights one or more columns of a C1List control at run time, the C1DataColumn objects for those rows are stored in the SelectedColumnCollection object. In code, you can use the **Item** property and **IndexOf** method of the collection to determine which rows are selected. You can also select and deselect records programmatically using its Insert and RemoveAt methods.

Working with Objects and Collections

This section describes how to work with objects and collections in code, with an emphasis on efficiency. Although the concepts are illustrated with C1List objects and collections, you can apply the same fundamentals to all Visual Studio objects and collections.

A C1List object is created when you place a C1List control on a .NET form. C1List objects created in Visual Studio will have default names of C1List1, C1List2, and so forth. You can change the control name in the Properties window at design time.

Working with Collections

A C1List object has eight separate collections which govern its diverse objects. Each of these collections has an associated property within the C1List object which returns the collection object. This prevents the need for the programmer to enter the entire collection name when using the list in code. The following table outlines these mappings:

Collection	Associated Property
C1DataColumnCollection	Columns property

Collection	Associated Property
C1DisplayColumnCollection	**DisplayColumns** property
GridStyleCollection	Styles property
SelectedColumnCollection	SelectedCols property
SelectedRowCollection	SelectedIndices property
SplitCollection	Splits property
ValueItemCollection	Values property

By default, the SplitCollection object contains one **Split** object. The GridStyleCollection object contains these default Style objects: Normal, Heading, Footing, Selected, Caption, HighlightRow, EvenRow, OddRow, and RecordSelector.

You can reference an object in a collection using its zero-based index. You can read or set the **Split** object's properties as follows:

To write code in Visual Basic

Visual Basic

```
' Read a Split object property.
variable = Me.C1List1.Splits(0).Property

' Set a Split object property.
Me.C1List1.Splits(0).Property = variable
```

To write code in C#

C#

```
// Read a Split object property.
variable = this.c1List1.Splits[0].Property;

// Set a Split object property.
this.c1List1.Splits[0]. = variable;
```

You can create a reference to an object in a collection using the collection's **Item** method. The following code creates a reference to a list's default **Split** object:

To write code in Visual Basic

Visual Basic

```
' Declare Split0 as a Split object.
Dim Split0 As C1.Win.C1List.Split

' Set Split0 to reference the first Split in the collection.
Split0 = Me.C1List1.Splits.Item(0)
```

To write code in C#

C#

```
// Declare Split0 as a Split object.
C1.Win.C1List.Split Split0;
```

```
// Set Split0 to reference the first Split in the collection.
Split0 = this.c1List1.Splits.Item[0];
```

Note the use of the namespace qualifier in the preceding example. Using the namespace qualifier is recommended in order to resolve potential naming conflicts with other controls. For example, if you use another control in the same project that also defines an object named **Split**, the C1List namespace qualifier is required, as is the namespace qualifier for the other control.

Since the **Item** method is implicit for collections, you can omit it:

To write code in Visual Basic

Visual Basic

```
' Declare Split0 as a Split object.
Dim Split0 As C1.Win.C1List.Split

' Set Split0 to reference the first Split in the collection.
Split0 = Me.C1List1.Splits(0)
```

To write code in C#

C#

```
// Declare Split0 as a Split object.
C1.Win.C1List.Split Split0;

// Set Split0 to reference the first Split in the collection.
Split0 = this.c1List1.Splits[0];
```

You can now use Split0 to read or set the **Split** object's properties or to execute its methods:

To write code in Visual Basic

Visual Basic

```
' Read a Split object property.
variable = Split0.Property

' Set a Split object property.
Split0.Property = variable

' Execute a Split object method.
Split0.Method (arg1, arg2, ...)
```

To write code in C#

C#

```
// Read a Split object property.
variable = Split0;

// Set a Split object property.
Split0 = variable;
```

```
// Execute a Split object method.
Split0.Method(arg1, arg2, ...);
```

Very often, you need to read and set more than one of an object's properties. For example:

To write code in Visual Basic

Visual Basic

```
' Read a Split object's properties.
variable1 = Me.C1List1.Splits(0,0).Property1
variable2 = Me.C1List1.Splits(0,0).Property2

' Set a Split object's properties.
Me.C1List1.Splits(0,0).Property1 = variable1
Me.C1List1.Splits(0,0).Property2 = variable2
```

To write code in C#

C#

```
// Read a Split object's properties.
variable1 = this.c1List1.Splits[0, 0].Property1;
variable2 = this.c1List1.Splits[0, 0].Property2;

// Set a Split object's properties.
this.c1List1.Splits[0, 0].Property1 = variable1;
this.c1List1.Splits[0, 0].Property2 = variable2;
```

This code is very inefficient because of the number of times the object **C1List1.Splits(0,0)** is accessed. It is more efficient to create a single reference to the object up front and use it repeatedly:

To write code in Visual Basic

Visual Basic

```
' Declare Split0 as a Split.
Dim Split0 As C1List.Split

' Set Split0 to reference the first Split in the collection.
Split0 = Me.C1List1.Splits.Item(0,0)

' Read a Split object's properties.
variable1 = Split0.Property1
variable2 = Split0.Property2

' Set a Split object's properties.
Split0.Property1 = variable1
Split0.Property2 = variable2
```

To write code in C#

C#

```
// Declare Split0 as a Split.
```

```
C1List.Split Split0;

// Set Split0 to reference the first Split in the collection.
Split0 = this.c1List1.Splits[0,0];

// Read a Split object's properties.
variable1 = Split0.Property1;
variable2 = Split0.Property2;

// Set a Split object's properties.
Split0.Property1 = variable1;
Split0.Property2 = variable2;
```

This code is much more efficient and easier to read. If your Visual Studio application accesses collection objects frequently, you can improve the performance of your code significantly by adhering to these guidelines.

Similarly, you can apply this technique to other objects and collections of C1List, and of Visual Studio in general. Of particular importance to the list are the C1DataColumn object and C1DataColumnCollection object (also applies to C1DisplayColumn object):

To write code in Visual Basic

Visual Basic

```
' Declare Cols as a Columns collection object, then set it to reference C1List1's
C1DataColumnCollection object.
Dim Cols As C1List.C1DataColumnCollection
Cols = Me.C1List1.Columns

' Declare Col0 as a DataColumn object, then set it to reference the first Column
object in the collection.
Dim Col0 As C1List.C1DataColumn
Col0 = Cols(0)

' Read and set the DataColumn object's Property1.
variable1 = Col0.Property1
Col0.Property1 = variable1

' Execute the DataColumn object's Method1 (declared as a Sub).
Col0.Method1(arg1, arg2, ...)

' Execute the DataColumn object's Method2 (declared as a Function).
variable2 = Col0.Method2(arg1)
```

To write code in C#

C#

```
// Declare Cols as a Columns collection object, then set it to reference C1List1's
C1DataColumnCollection object.
C1List.C1DataColumnCollection Cols;
Cols = this.C1List1.Columns;

// Declare Col0 as a DataColumn object, then set it to reference the first Column
```

```
object in the collection.
C1List.C1DataColumn Col0;
Col0 = Cols[0];

// Read and set the DataColumn object's Property1.
variable1 = Col0.Property1;
Col0.Property1 = variable1;

// Execute the DataColumn object's Method1 (declared as a Sub).
Col0.Method1(arg1, arg2, ...);

// Execute the DataColumn object's Method2 (declared as a Function).
variable2 = Col0.Method2(arg1);
```

Visual Basic also provides an efficient With...End With statement for setting multiple properties of an object without explicitly assigning it to a variable. For example, the following code sets multiple properties of the first column of a list (recall that collections are zero-based):

To write code in Visual Basic

Visual Basic

```
With Me.C1List1.Columns(0)
 .Property1 = variable1
 .Property2 = variable2
End With
```

To write code in C#

C#

```
this.c1List1.Columns[0].Property1 = variable1;
this.c1List1.Columns[0].Property2 = variable2;
```

Adding Members

To create and add an object to a collection, use the collection's **Add** method. The method takes an object as its only argument. For example, you can create more valueitems for a column by adding new ValueItem objects to the ValueItemCollection object:

To write code in Visual Basic

Visual Basic

```
' Create a ValueItem object.
Dim S As New C1List.ValueItem
Me.C1List1.Columns(0).ValueItems.Values.Add(S)
```

To write code in C#

C#

```
// Create a ValueItem object.
C1List.ValueItem S;
```

```
this.c1List1.Columns[0].ValueItems.Values.Add(S);
```

This code adds a ValueItem object to the ValueItemCollection of C1List1. Alternatively, you can create a ValueItem object with index 1 with the **Insert** method:

To write code in Visual Basic

Visual Basic

```
' Create a Split object with index 1.
Dim S As New C1List.ValueItem
Me.C1List1.Columns(0).ValueItems.Values.Insert(1, S)
```

To write code in C#

C#

```
// Create a Split object with index 1.
C1List.ValueItem S;
this.c1List1.Columns[0].ValueItems.Values.Insert(1, S);
```

The only object which is unable to add or remove members using the **Add** or **RemoveAt** methods is the **Split** object. To correctly add or remove Splits you must use the InsertHorizontalSplit/RemoveHorizontalSplit and InsertVerticalSplit/RemoveVerticalSplit methods of the **Split** object. These methods are also available in the list's right-click context menu at design time

Removing Members

Regardless of how a collection implements the **Add** or **Insert** methods, the syntax for removing items is the same. To remove an existing item from a collection, use the **RemoveAt** method:

To write code in Visual Basic

Visual Basic

```
' Remove the Split object with index 1.
Me.C1List1.Columns(0).ValueItems.Values.RemoveAt(1)
```

To write code in C#

C#

```
// Remove the Split object with index 1.
this.c1List1.Columns[0].ValueItems.Values.RemoveAt(1);
```

After this statement is executed, all splits with collection indexes greater than 1 will be shifted down by 1 to fill the place of the removed split.

Working with the Count Property

You can determine the number of objects in a collection using the collection's **Count** property:

To write code in Visual Basic

Visual Basic

```
' Set a variable equal to the number of Splits in C1List1.
variable = Me.C1List1.Splits.Count
```

To write code in C#

```
C#
// Set a variable equal to the number of Splits in C1List1.
variable = this.c1List1.Splits.Count;
```

You can also iterate through all objects in a collection using the **Count** property as in the following example, which prints the Caption string of each C1DataColumn object in a list:

To write code in Visual Basic

```
Visual Basic
For n = 0 To Me.C1List1.Columns.Count - 1
    Debug.WriteLine(Me.C1List1.Columns(n).Caption)
Next n
```

To write code in C#

```
C#
for (int n = 0; n <= this.c1List1.Columns.Count - 1; n++)
{
     Console.WriteLine(this.c1List1.Columns[n].Caption);
}
```

The **Count** property is also useful for appending and removing columns:

To write code in Visual Basic

```
Visual Basic
' Determine how many columns there are.
Dim NumCols As Integer
NumCols = Me.C1List1.Columns.Count

' Append a column to the end of the Columns collection.
Dim C As New C1List.C1DataColumn()
Me.C1List1.Columns.Insert(NumCols, C)

' The following loop removes all columns from the list.
While Me.C1List1.Columns.Count
      Me.C1List1.Columns.RemoveAt(0)
End While
```

To write code in C#

```
C#
// Determine how many columns there are.
int NumCols;
NumCols = this.c1List1.Columns.Count;
```

```
// Append a column to the end of the Columns collection.
C1List.C1DataColumn C = new C1List.C1DataColumn();
this.c1List1.Columns.Insert(NumCols, C);

// The following loop removes all columns from the list.
while (this.c1List1.Columns.Count)
this.c1List1.Columns.RemoveAt(0);
```

Visual Basic also provides an efficient For Each...Next statement that you can use to iterate through the objects in a collection without using the **Count** property:

To write code in Visual Basic

Visual Basic

```
Dim C As C1List.C1DataColumn
For Each C In Me.C1List1.Columns
     Debug.WriteLine(C.Caption)
Next
```

To write code in C#

C#

```
C1List.C1DataColumn C;
foreach (C in this.c1List1.Columns )
{
     Console.WriteLine(C.Caption);
}
```

In fact, using the For Each...Next statement is the preferred way to iterate through the objects in a collection.

List for WinForms Quick Start

This quick start guide will walk you through the steps of adding C1List to a project, binding the list to a data source, and customizing the **List for WinForms** application. You will find that **List for WinForms** is very easy to use, and it enables you to create powerful list applications.

The tutorials use an Access database, **C1NWind.mdb**. This quick start assumes that the database file **C1NWind.mdb** is in the **\common** directory installed with **WinForms Edition** (**Documents\ComponentOne Samples\Common**) and refer to it by filename instead of the full pathname for the sake of brevity.

Step 1 of 3: Creating the List for WinForms Application

In this step, you'll add the C1List control to your project and create a simple **List for WinForms** application. Complete the following steps:

1. Create a new WinForms project. For more information, see Creating a .NET Project.
2. From the Toolbox double-click the **C1List** icon. The C1List control is added to the project and the **C1List Tasks** menu appears.
3. Navigate to the Properties window and set the **Dock** property to **Fill**.

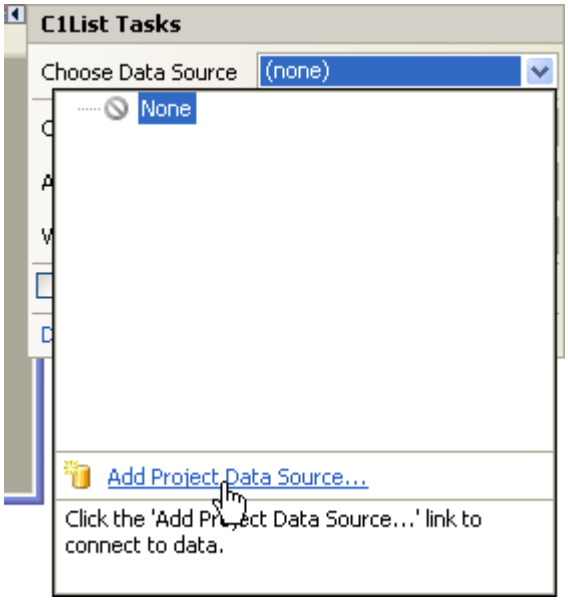
The form will appear similar to the following:

You have successfully created a simple list application. In the next step, you will learn how to bind the C1List control to a data source.

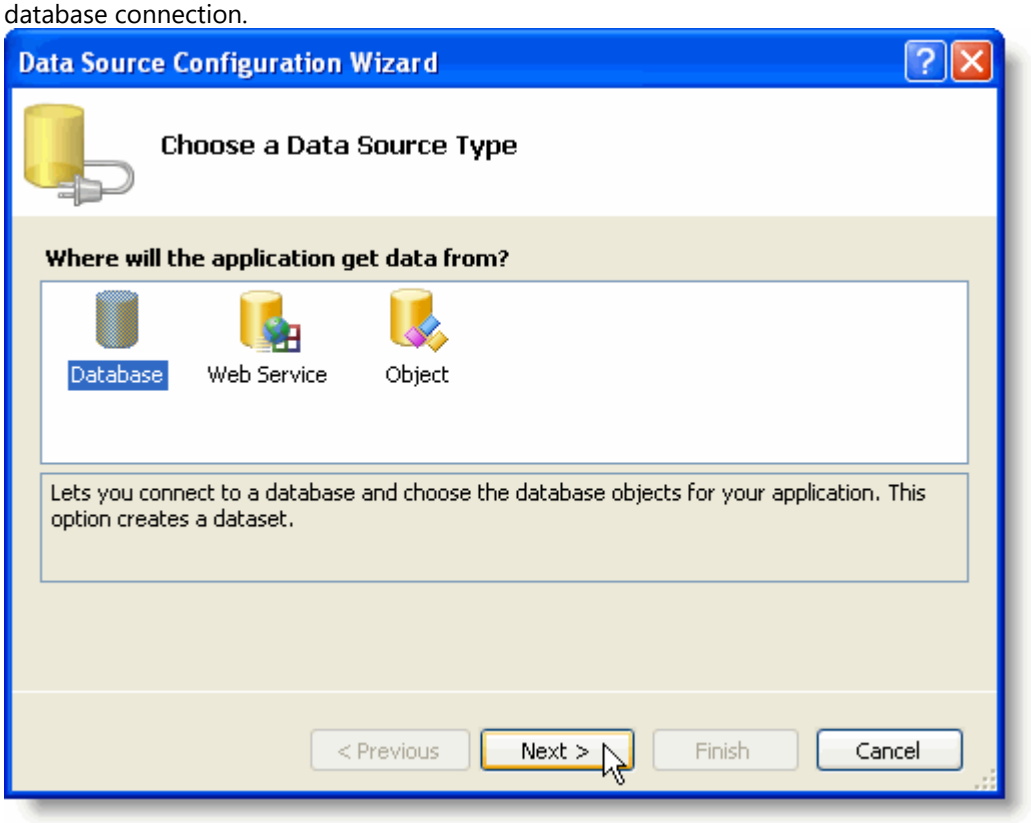
Step 2 of 3: Binding the List for WinForms Application to a DataSet

In the last step you added the C1List control to your project and created a simple list application. In this step you'll bind the C1List control to a DataSet. Complete the following steps:

1. Click C1List1's smart tag to open the **C1List Tasks** menu, select the drop-down arrow next to **Choose Data Source**, and click **Add Project Data Source** to open the **Data Source Configuration Wizard**.



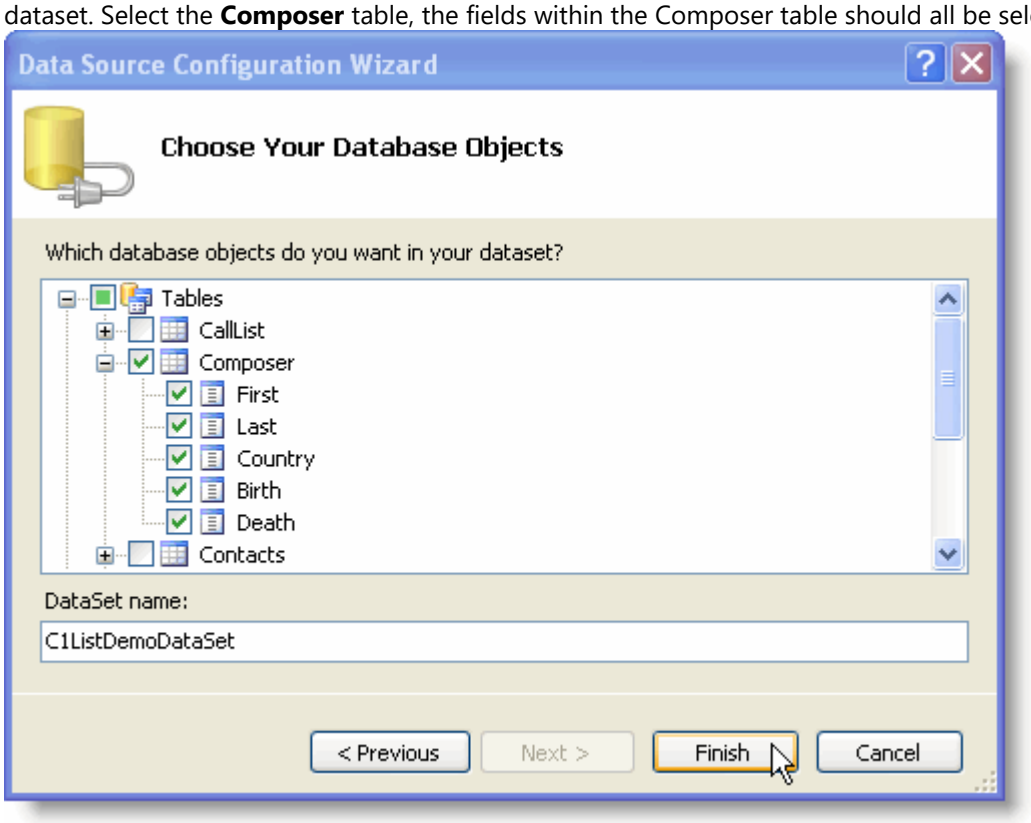
2. The **Data Source Configuration Wizard** appears with a **Database** data source selected. Click **Next** to select a database connection.



3. Click the **New Connection** button to locate and connect to a database.
4. The **Add Connection** dialog box opens. Click the **Browse** button and locate the **C1NWind.mdb** file. Select it and click **Open**.
5. Click the **Test Connection** button to make sure that you have successfully connected to the database or server and click **OK** in the dialog window confirming the connection succeeded.
6. Click **OK** in the **Add Connection** dialog box to add the next connection and return to the **Data Source Configuration Wizard**. The new string appears in the data connection drop-down box.
7. Click the **Next** button to continue. A dialog box will appear asking if you would like to add the data file to your project and modify the connection string. Click **No**.
8. In the next window, note that the **Yes, save the connection as** checkbox is checked by default and "C1ListDemoConnectionString" has been automatically entered in the text box. Click **Next** to add tables and

fields to the DataSet.

9. In the **Choose Your Database Objects** window, you can select the tables and fields that you would like in your dataset. Select the **Composer** table, the fields within the Composer table should all be selected as well.



The DataSet is given a default name in the **DataSet name** textbox.

10. Click **Finish** to exit the wizard. The **DataSet**, **BindingSource** and **TableAdapter** now appear on your form.
11. Double-click the form. Notice that Visual Studio has added the following code to the **Form_Load** event:

To write code in Visual Basic

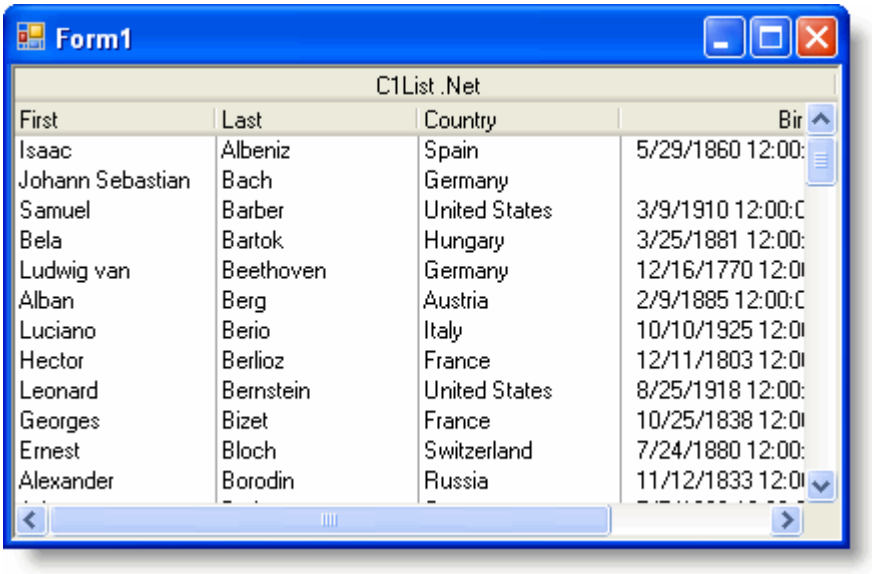
```
Visual Basic
Me.ComposerTableAdapter.Fill(Me.C1ListDemoDataSet.Composer)
```

To write code in C#

```
C#
this.ComposerTableAdapter.Fill(this.C1ListDemoDataSet.Composer);
```

Run the program and observe the following:

Notice that the data from the **Composers** table is reflected in the list:



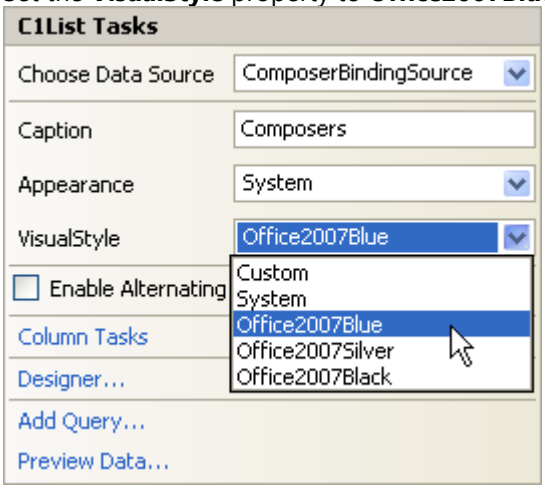
List for WinForms retrieves the database schema information from the DataSet and automatically configures itself to display all of the fields contained in the database table. Note that the field names are used as the default column headings.

You have successfully completed binding **List for WinForms** to a DataSet. In the next step you will customize the C1List control's appearance settings.

Step 3 of 3: Customizing List for WinForms' Settings

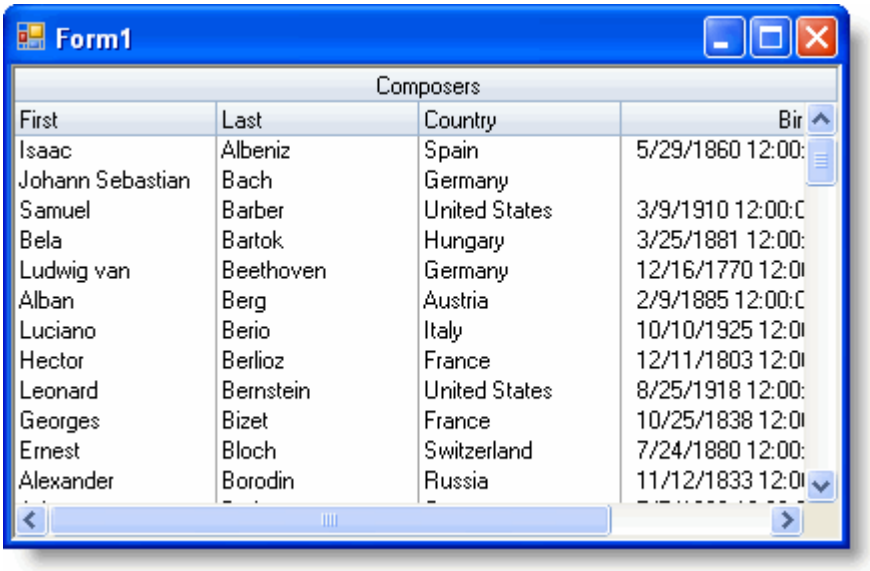
In the previous steps you've added the C1List control to the form and bound the C1List control to a data source. In this step you will customize the C1List control's appearance settings. Complete the following steps:

1. Switch to **Design** view and select **C1List1**'s smart tag to open the **C1List Tasks** menu.
2. In the **C1List Tasks** menu set the following properties:
   - Set **Caption** property to **Composers** to add a caption to the list.
   - Set the **VisualStyle** property to **Office2007Blue**.



Run the program and observe the following:

Notice that the list appears with a caption and in the Office 2007 Blue style:



Congratulations, you've successfully completed the **List for WinForms** quick start tutorial! You've created a simple list application, bound the C1List to a data source, and customized the list's appearance without writing a single line of code!

Run-Time Interaction

This chapter describes how users of your application interact with **List for WinForms** at run time. You can give your users the ability to perform any or all of the following:

- Navigate within the list using the mouse or keyboard.
- Select rows or columns.
- Configure the list's layout.

In the following sections, the properties and events associated with a particular user action are noted where applicable.

Navigation and Scrolling

The following sections describe the list's default navigation and scrolling behavior. You always have complete control over the behavior of the Tab and arrow keys as well as the position of the current cell when a row or split boundary is reached.

Mouse Navigation

When the user clicks a row, that row becomes current, and the RowChange event is fired.

The user can also use the mouse to manipulate the list's scroll bars, bringing cells that lie outside the list's display area into view. The vertical scroll bar governs rows; the horizontal scroll bar governs columns. The HScrollBar property controls whether the horizontal scroll bars are displayed, while the VScrollBar property controls the vertical scroll bar.

Scrolling always occurs in discrete cell units; the user cannot scroll on a per-pixel basis in either direction. Note that the scroll bars do not change the current cell; therefore, the current cell may not always be visible.

To respond to vertical scrolling operations in code, use the TopIndexChange event. To respond to horizontal scrolling operations in code, use the LeftColChange event.

Row Tracking

If the RowTracking property is set to **True**, rows are automatically highlighted as the mouse is moved over them. When RowTracking is **True** and the user highlights a different row by moving the mouse, the ItemChanged event will not fire.

Clicking the Rightmost Column

The list always displays the leftmost column (the first visible column) in its entirety. The rightmost column, however, is usually clipped. The behavior of the last partially visible column when clicked by the user is controlled by the list's PartialRightColumn property.

The default value for the PartialRightColumn property is **True**. The rightmost column will be clipped if the control or split is not wide enough to accommodate the entire column.

If **False**, the rightmost column will not be clipped while other columns are visible. In this case, the rightmost column must be scrolled into view as the only visible column in the control or split.

Keyboard Navigation

By default, the user can navigate the list with the arrow keys, the Tab key, and the PgUp and PgDn keys. The following tables describes the behavior of the navigation keys.

Key	Description
UP/DOWN ARROWS	These keys move the current cell to adjacent rows.
Tab	The Tab key moves focus to the next control on the form, as determined by the tab order.
PgUp, PgDn	These keys scroll the list up or down an entire page at a time. Unlike the vertical scroll bar, the PgUp and PgDn keys change the current row by the number of visible rows in the list's display. When paging up, the current row becomes the first row in the display area. When paging down, the current row becomes the last row in the display area. The current column does not change.

Searching and Field Completion

The following sections describe how the C1List and C1Combo controls locate list items in response to keyboard input.

Searching in List for WinForms

The MatchEntry property determines how a C1List control performs searches based on user input:

- When set to None (the default), the control does not perform any incremental searches.
- When set to Standard, the search argument is limited to one character, and the control attempts to find a match for the character entered using the first letter of entries in the list. Repeatedly typing the same letter cycles through all of the entries in the list beginning with that letter.
- When set to Extended, the control searches for an entry matching all characters entered. The search is performed incrementally as characters are typed. This behavior is almost identical to that of a C1Combo control except that the search argument is cleared when a C1List control gains focus or when the user presses Backspace or hesitates for a few seconds. Use the MatchEntryTimeout property to specify the timeout value; the next characters entered by the user after hesitating for the specified time will begin a new search through the list.

Searching in C1Combo

Incremental searching is always enabled for C1Combo controls; with incremental searching C1Combo controls search for an entry matching all characters entered, performing the search incrementally as characters are typed. This behavior is almost identical to that of a C1List control with its MatchEntry property set to Extended except that the search argument is not cleared when the user presses Backspace or hesitates for a few seconds. If the ComboStyle property is set to DropdownCombo or SimpleCombo, you can use the MatchEntryTimeout property to specify the timeout value. After the user hesitates for the specified amount of time, the combo text entry box will clear and the next characters entered will begin a new search.

Field Completion (C1Combo Only)

The AutoCompletion property controls whether matching incremental search values are automatically copied to the text portion of a combo box during editing.

If **True**, when the user enters one or more characters that match a value in the combo box's list, the entire matching string is copied to the text portion of the control. The caret is positioned after the last character typed, and the remainder of the string is selected.

If **False** (the default), automatic completion does not occur, and the text portion of the control contains only the characters entered by the user.

> **Note:** Additionally, MatchCompare property can be used to enhance field completion, as it complements auto complete by enabling input string matching within a string in C1Combo.

Search Mismatch (C1Combo Only)

The Mismatch event is triggered whenever the user enters a value in the text portion of a combo box that is not found in the list portion. By default, Reposition is **False**, which resets the current row back to the top of the list portion when the event is fired.

If you set Reposition to **False**, and the combo cannot locate NewEntry, the current row remains unchanged when this event is fired.

> **Note:** This event is only fired when the LimitToList property is **True** and the ComboStyle property is set to DropdownCombo or SimpleCombo.

Selection and Movement

The following sections describe how users can select columns, move selected columns, and select rows. You can always restrict any or all of these operations at design time or in code.

Selecting Columns

If the AllowColSelect property is **True**, the user can select an individual column or a range of columns with the mouse. Nonadjacent column selections are not supported.

When the user points to the header of an unselected column, the mouse pointer changes to a down arrow to indicate that the column can be selected.

When the user clicks a column header, that column is selected and highlighted, and any columns or rows that were previously selected are deselected.

The user can select a range of columns using either of the following methods:

1. After selecting the first column in the range by clicking its header, the user can select the last column in the range by holding down the Ctrl or Shift key and clicking another column header. If necessary, the horizontal scroll bar can be used to bring additional columns into view.
2. Alternatively, the user can hold and drag the mouse pointer within the column headers to select multiple columns.

In order to manipulate the columns which have been selected at run-time, you will have to query the SelectedColumnCollection. This is a collection of all the C1DataColumn objects for the selected columns. For instance if you select columns 5 through 10, the SelectedColumnCollection will have six members, each a C1DataColumn object. This feature enables you to directly alter the display properties of the column. Using the Item property to access the C1DisplayColumn properties, the code to change the forecolor to red for the first column selected would be:

To write code in Visual Basic

Visual Basic

```
Dim dc as C1List.C1DataColumn
dc = Me.C1List1.SelectedCols(0)
Me.C1List1.Splits(0).DisplayColumns.Item(dc).Style.ForeColor =
System.Drawing.Color.Red
```

To write code in C#

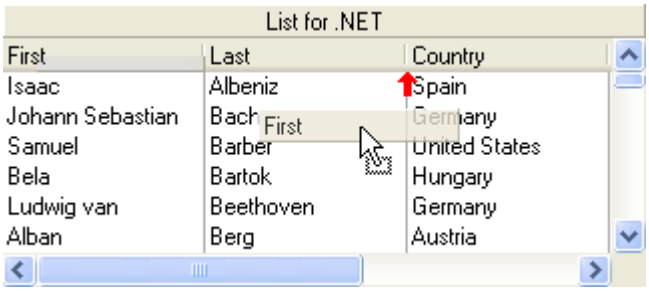
C#

```
C1List.C1DataColumn dc;
dc = this.c1List1.SelectedCols[0];
this.c1List1.Splits[0].DisplayColumns[dc].Style.ForeColor = System.Drawing.Color.Red;
```

You can prevent a column selection from occurring at run time by setting the ***Cancel*** argument to **True** in the list's SelectionChanging event.

Moving Columns

If the AllowColMove property is **True**, the user can move previously selected columns as a unit to a different location by pressing the mouse button within the header area of any selected column. The pointer will change to an arrow with a column header box on its tip, a small box at its lower right corner, and a position marker consisting of a red triangle will appear at the left edge of the column being pointed to and highlighted. The user specifies the desired location of the selected columns by dragging the column header, which changes positions as the mouse pointer crosses the columns.



The user completes the operation by releasing the mouse button, which moves the selected columns immediately to the right of the position marker.

If the user drags the marker to a position within the currently selected range, no movement occurs. Columns that are not selected cannot be moved interactively.

When a move occurs, the index in the Columns Collection is adjusted for all affected columns.

You can prevent interactive column movement from occurring at run time by setting Cancel to **True** in the ColMove event.

Moving Columns at Run Time

If the AllowColMove property is **True**, users can move columns at run-time. Since there is no order property for a C1DisplayColumn, the C1DisplayColumnCollection needs to be manipulated to move a column at run-time. The C1DisplayColumnCollection holds all of the columns in a split. So to move a column, the user needs to remove the DisplayColumn from the collection, and then replace the column in the new position. The common place collection methods of RemoveAt and Insert help accomplish this quite easily. The following code will transpose the first two columns in the default split:

To write code in Visual Basic

Visual Basic

```
Dim dc as C1List.C1DisplayColumn
dc = Me.C1List1.Splits(0).DisplayColumns.Item(1)
Me.C1List1.Splits(0).DisplayColumns.RemoveAt(1)
Me.C1List1.Splits(0).DisplayColumns.Insert(0, dc)
```

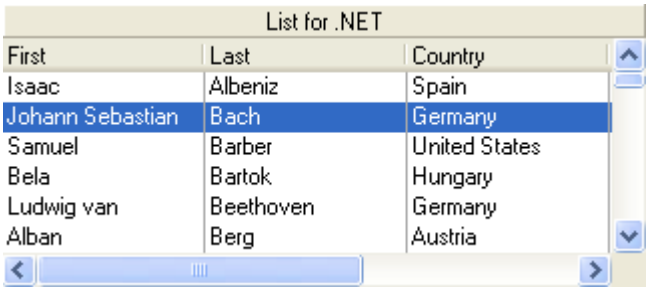
To write code in C#

C#

```
C1List.C1DisplayColumn dc;
dc = this.c1List1.Splits[0].DisplayColumns[1];
this.c1List1.Splits[0].DisplayColumns.RemoveAt(1);
this.c1List1.Splits[0].DisplayColumns.Insert(0, dc);
```

Selecting Rows (C1List Only)

If enabled, record selection behavior is determined by the SelectionMode property. By default, this property is set to One, and the user can select one row with the mouse. When the user clicks a row, that row is selected and highlighted, and any rows or columns that were previously selected are deselected. The newly selected row also becomes the current row.



However, unlike column selections, nonadjacent row selections are supported. If the user holds down the Ctrl key while making the selection, the current row does not change, and any previously selected rows remain selected (when the SelectionMode property is set to MultiExtended). This technique also enables the user to select multiple rows, one at a time. Since selected rows do not have to be adjacent, the user can also operate the vertical scroll bar to bring other rows into view if desired.

The user can also select a range of contiguous rows by clicking the first row in the range, then holding down the Shift key and clicking the last row in the range. If necessary, the vertical scroll bar can be used to bring additional rows into view.

- If SelectionMode is set to Extended, the default behavior is supported, but the user can also select records with the following key combinations: Shift + Up Arrow, Shift + Down Arrow, Shift + PgUp, and Shift + PgDn.
- If SelectionMode is set to None, then no row selection is allowed from a user's interaction, but you still can select rows from code.
- If SelectionMode is set to One, multiple selection is disabled but single selection is permitted. When the user clicks an unselected row, the current selection is cleared and the clicked row is selected and highlighted. However, the Ctrl and Shift keys are ignored, and the user can only select one row at a time.
- If SelectionMode is set to CheckBox, the user can select or unselect rows by clicking checkboxes.

Regardless of the value of SelectionMode, the user can deselect all rows by selecting columns.

You can prevent a row selection from occurring at run time by setting the ***Cancel*** argument to **True** in the list's SelectionChanging event.

Sizing and Splitting

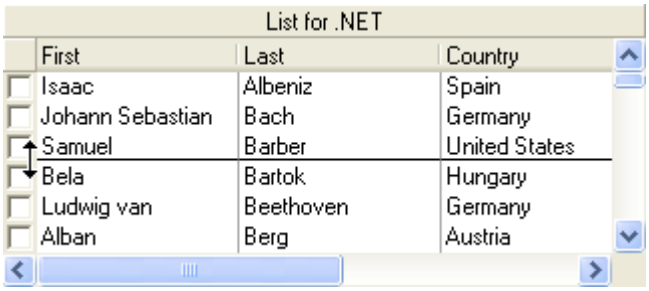
The following sections describe how users can resize rows, columns, and splits. You can always restrict any or all of these operations at design time or in code.

Sizing Rows

If the AllowRowSizing property is set to either RowSizingEnum.AllRows or RowSizingEnum.IndividualRows, the user can change the row height at run time.

> **Note:** This property only works when the SelectionMode property is set to CheckBox.

When the user points to a row divider between any pair of checkboxes, the pointer changes to a vertical double arrow, which the user can drag to resize the row.



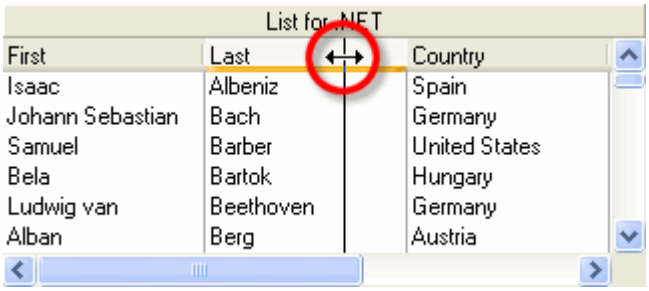
Dragging the pointer upward makes the rows smaller; dragging it downward makes the rows larger. If the property is set to AllRows, then all rows in the list will be resized to the same height; it is not possible to resize individual rows. If the property is set to IndividualRows, then each row can be sized independently.

The ItemHeight property of the list will be adjusted when the user completes the resize operation.

You can prevent row resizing from occurring at run time by setting the ***Cancel*** argument to **True** in the RowResize event. You can always change the ItemHeight of the list in code, even if AllowRowSizing is RowSizingEnum.None or you cancel the RowResize event.

Sizing Columns

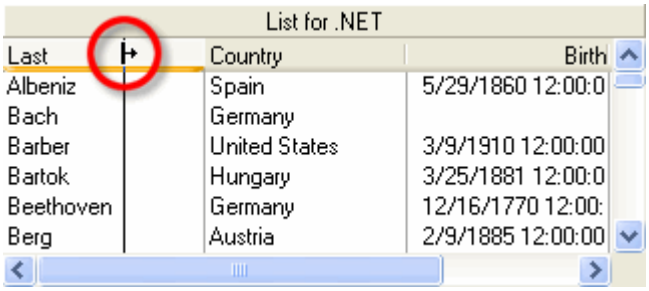
If the AllowSizing property is **True** for a column, then the user can adjust the width of the column individually at run time. When the user points to the divider at the right edge of a column's header, the pointer changes to a horizontal double arrow, which allows the user to resize the column in question.



Dragging the pointer to the left makes the column smaller; dragging it to the right makes the column larger. The column's Width property will be adjusted when the user completes the resize operation.

> **Note:** Columns can only be resized using the mouse when the column headers are shown (the ColumnHeaders property is **True**).

If the user drags the pointer all the way to the left, the column retains its original Width property setting, but its Visible property is set to **False**. To make the column visible again, the user can point to the right side of the divider of the column that preceded it. The pointer turns into a vertical bar with a right arrow.



Dragging the pointer to the right establishes a new column width and sets the column's Visible property back to **True**.

Another way to resize columns is to use the AutoSize method, which allows you to specify auto-sizing for a specific Column object in code. When a column is auto-sized, its width is adjusted to fit the longest visible field in that

column. Longer records that are not displayed in a visible row when AutoSize is invoked do not participate in the width calculations. Furthermore, if the specified column is either hidden or scrolled out of view, a trappable error occurs.

You can prevent column resizing from occurring at run time by setting Cancel to **True** in the ColResize event. You can always change the width of a column in code, even if AllowSizing is **False** for that column.

Additional User Interaction Features

List for WinForms provides additional data presentation and manipulation functionality that you can expose to your users at run time. For more information, please see the Context-sensitive Help with CellTips and Scroll Tracking and ScrollTips topics.

Data Binding

This section describes how to bind C1List or C1Combo to a data source and use unbound columns.

Binding C1List or C1Combo to a Data Source

With an amazing ease of use, **List for WinForms** can universally bind to any OLE DB data source. Requiring little or no code at all, the C1List control can create a fully-navigational database browser in mere seconds.

To associate a List control (C1List) or a Combo control (C1Combo) with an OLE DB data source, set the DataSource property of the list or combobox to the name of a DataSet on the same form. The DataSource property can be set both through code and through the Properties window. This is all that is required to make C1List fully aware of the database or DataTable in your application.

Once such a link exists, C1List and the DataSet automatically notify and respond to all operations performed by the other, simplifying your application development.

For additional information and examples, see List for WinForms Tutorials.

Unbound Columns

Normally, **List for WinForms** automatically displays data from bound database fields. However, you may need to augment the set of fields present in your layouts with columns derived from database fields, or columns which are unrelated (or only loosely related) to database information. For example, if your database contains a *Balance* field, you may instead want to display two columns, *Credit* and *Debit*, to show positive and negative numbers separately. Or, you may want to look up data in another database, or convert field data to some other form, such as mapping numeric codes to textual descriptions.

To accomplish such tasks you can use unbound columns. The term *unbound column* refers to a column that is part of a *bound list*, but is not tied directly to a database field.

Columns that do not have the DataField property set (that is, the DataField property is equal to an empty string), but do have the column Caption property set are considered unbound columns. The list will request data for these columns through the UnboundColumnFetch event.

Columns with their DataField property set will be bound if the DataField property is the same as one of the fields of the Data Source.

Columns with their DataField property set to a value that is not in the DataSet are ignored for the purposes of fetching data. Similarly, columns that have no value for both the DataField and Caption properties set are also ignored.

Creating Unbound Columns

The first step in using an unbound column is creating the column itself. This may be done in the designer by adding a column through the **C1List Designer**. In code, unbound columns may be added using the Insert method of the C1DataColumnCollection. The column must be given a name by setting its Caption property. In the designer, this is done using the **C1List Designer**. In code, the Caption property of the appropriate C1DataColumn object is set. C1DataColumn objects that are added to the C1DataColumnCollection cause a corresponding C1DisplayColumn to be added to the C1DisplayColumnCollection for all splits. The default visible property of the newly added C1DisplayColumn will be **True**.

If you attempt to insert an unbound column in code, you may need to use the HoldFields method to ensure that the column appears at the desired position within the list:

To write code in Visual Basic

```
Visual Basic
Dim Col As New C1List.C1DataColumn
Me.C1List1.Columns.Insert(0, Col)
Col.Caption = "Unbound"
Me.C1List1.HoldFields()
Me.C1List1.Rebind()
```

To write code in C#

```
C#
C1List.C1DataColumn Col = new C1List.C1DataColumn;
this.c1List1.Columns.Insert(0, Col);
Col.Caption = "Unbound";
this.c1List1.HoldFields();
this.c1List1.Rebind();
```

When the list needs to display the value of an unbound column, it fires the UnboundColumnFetch event. This event supplies the user with a row and column index as the means of identifying the list cell being requested. The ***Value*** argument to the event is of type Object, which by default is Null, but can be changed to any desired value, and will be used to fill the contents of the cell specified by the given bookmark and column index.

To write code in Visual Basic

```
Visual Basic
Private Sub C1List1_UnboundColumnFetch(ByVal sender As Object, ByVal e As
C1.Win.C1List.UnboundColumnFetchEventArgs) Handles C1List1.UnboundColumnFetch
```

To write code in C#

```
C#
private void c1List1_UnboundColumnFetch(object sender,
C1.Win.C1List.UnboundColumnFetchEventArgs e)
```

Implementing Multiple Unbound Columns

So far, our examples have demonstrated the UnboundColumnFetch event using only a single unbound column. Suppose you want to have more than one? Since the UnboundColumnFetch event is fired for each unbound column of each row, only one column value may be set at a time, and each column must be identified for the value to be properly determined. The second UnboundColumnFetch argument, ***Col***, is used to identify the column of the list for which the value is required.

To write code in Visual Basic

```
Visual Basic
' Will be used as the Copy.
Dim dtCopy As Data.DataTable

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e_ As System.EventArgs)
```

```
Handles MyBase.Load
    dtCopy = Me.DsRectangle1.Tables(0).Copy()
End Sub

Private Sub C1List1_UnboundColumnFetch(ByVal sender As System.Object, ByVal e As
C1.Win.C1List.UnboundColumnFetchEventArgs) Handles C1List1.UnboundColumnFetch
    Select Case Me.C1List1.Columns(e.Col).Caption
        Case "Area"
            ' Calculate the "Area" column of the list.
            e.Value = dtCopy.Rows(e.Row).Item("Length") *
dtCopy.Rows(e.Row).Item("Width")
        Case "Perimeter"

            ' Calculate the "Perimeter" column of the list.
            e.Value = 2 * dtCopy.Rows(e.Row).Item("Length") +
dtCopy.Rows(e.Row).Item("Width"))
    End Select
End Sub
```

To write code in C#

C#

```
// Will be used as the Copy.
Data.DataTable dtCopy;

private void Form1_Load(System.object sender, System.EventArgs e)
{
    dtCopy = this.DsRectangle1.Tables[0].Copy();
}

private void C1List1_UnboundColumnFetch(System.object sender,
C1.Win.C1List.UnboundColumnFetchEventArgs e)
{
    switch (this.c1List1.Columns[e.Col].Caption)
    {
        case "Area":

            // Calculate the "Area" column of the list.
            e.Value = dtCopy.Rows[e.Row]["Length"] * dtCopy.Rows[e.Row]["Width"];
            break;
        case "Perimeter":

            // Calculate the "Perimeter" column of the list.
            e.Value = 2 * (dtCopy.Rows[e.Row][ "Length"] + dtCopy.Rows[e.Row]
["Width"]);
            break;
    }
}
```

List for WinForms' AddItem Mode

When in **AddItem** mode, **List for WinForms** allows you to populate a list or combo box control (C1List or C1Combo) manually by using the AddItem, RemoveItem, InsertItem and ClearItems methods. In this mode, your application determines what data is contained within the list control.

To use **AddItem** mode, set the DataMode property of the control to **AddItem** at design-time, then use the AddItem method to populate the control.

The AddItem method allows you to populate individual rows within the list. The following code adds the names and street addresses to the list control:

To write code in Visual Basic

```
Visual Basic
Me.C1List1.AddItem "Smith;John;210 Elm St."
Me.C1List1.AddItem "Doe;Jane;100 Oak St."
```

To write code in C#

```
C#
this.c1List1.AddItem("Smith;John;210 Elm St.");
this.c1List1.AddItem("Doe;Jane;100 Oak St.");
```

Note: The default AddItem separator is ";".You can choose a custom separator by setting the AddItemSeparator property at design-time or from code.

The **RemoveItem** method allows you to remove rows from the list. The code below allows you to remove the selected row from the list control:

To write code in Visual Basic

```
Visual Basic
Me.C1List1.RemoveItem(Me.C1List1.SelectedIndex)
```

To write code in C#

```
C#
this.c1List1.RemoveItem(this.c1List1.SelectedIndex);
```

The **ClearItems** method removes all items from the list population. The code below removes the entire population from the list control:

To write code in Visual Basic

```
Visual Basic
Me.C1List1.ClearItems()
```

To write code in C#

```
C#
this.c1List1.ClearItems();
```

The **InsertItem** method will insert an item to a specified position. The code below will insert a second row:

To write code in Visual Basic

```
Visual Basic
Me.C1List1.InsertItem("Carl;Ziegler;51 Pine St.", 1)
```

To write code in C#

```
C#
this.c1List1.InsertItem("Carl;Ziegler;51 Pine St.", 1);
```

When to Use AddItem Mode

If you do not want to use a bound data source control to populate the list, but instead want to modify the list manually, use **AddItem** Mode.

How AddItem Mode Works

When **List for WinForms** runs in **AddItem** mode, it is not connected to a data control. Instead, your application must supply and maintain the data, while the list handles all user interaction and data display. In this mode, the functionality of the C1List control is similar in most ways to the standard listbox control.

When populating the list control, the first item starts at zero. Any event that uses a bookmark will use the zero-based index of the control.

Design-Time Support

The following sections describe how to use **List for WinForms**' design-time environment to configure the C1List control. Most of the following material also applies to the C1Combo control since it is a subset of C1List. **List for WinForms** provides visual editing to make creating the list simple. You can easily configure **List for WinForms** at design time by using one or more of the following visual editors:

Properties Window

You can access the C1List and C1Combo properties simply by right-clicking on the control and selecting **Properties** or by selecting the class from the drop-down box of the Properties window.

Tasks Menu

In Visual Studio 2005, the C1List and C1Combo controls include a smart tag which opens a tasks menu. The tasks menu provides the most commonly used properties in each control. You can invoke the tasks menu by clicking on the smart tag (▶) in the upper-right corner of each control. For more information on how to use the smart tags, see C1List Tasks Menu.

Context Menu

You can use the C1List and C1Combo controls' context menu for additional functionality at design time. See the Context Menu topic for details.

Collection Editors

List for WinForms provides the following collection editors:

- Split Collection Editor
- C1DisplayColumn Collection Editor
- ValueItem Collection Editor
- Style Collection Editor

The main part for each of the editor's application consists of a windows form which conveniently allows the user to edit the list.

Designer

Using the **C1List Designer** you can easily configure the list's behavior and appearance, and preview the appearance of the list. **C1List Designer** can be accessed by clicking the **ellipsis** button next to the Columns property in the Properties window.

Understanding the Object Model and Property Access

List for WinForms supports a rich object model that reflects the organization of its visual components. Therefore, in order to customize a list's appearance and behavior, you need to know how the Properties window and collection editors reflect the list's object model.

Split-Specific Properties

A *split* is similar to the split window features of products such as Microsoft Excel and Word. You can use splits to present your data in multiple vertical or horizontal panes. These panes, or splits, can display data in different colors and fonts. They can scroll as a unit or individually, and they can display different sets of columns or the same set. You can also use splits to prevent one or more columns or rows from scrolling. The image below, for example, includes a list with both a horizontal and a vertical split, creating four panes.

By default, a list contains a single split comprising all of its columns. Note that most of the split properties are not present in the main Properties window. For example, you cannot set the AlternatingRows property without opening up the **Splits Collection Editor** and modifying the **Split** object, because the value of this property can vary from split to split. The term *split-specific* is used to describe such properties, since they apply to individual splits rather than the list as a whole. For more information about accessing split-specific properties, see Accessing Split-Specific Properties.

Global Properties

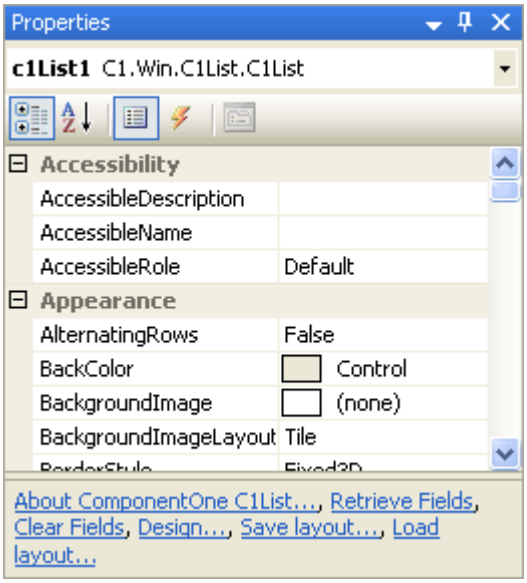
Conversely, the term *global* is used to describe properties that apply to the list as a whole, such as DataView and BorderStyle. Global properties are accessible through the Properties window, which is initially located in the lower right of the .NET IDE. The latter also shows *extender* properties specific to the Visual Studio environment, such as **Align** and **Tag**. For more information about accessing global properties, see Accessing Global List Properties.

Split-Specific Properties vs. Global Properties

The distinction between split-specific and global properties also extends to the two column objects which represent the columns of data within the list. Both of these objects govern a column's properties. The C1DataColumn object contains all of the column properties related to data access and formatting. The C1DisplayColumn object contains all of the column properties related to the column's visual display. The C1DataColumn properties are global column properties. These are properties that apply to all of the columns in the list, no matter their placement among the splits. For instance, when you add or remove a column, you would add or remove the associated C1DataColumn. On the other hand the C1DisplayColumn properties are split-specific properties. Setting one of these properties in one split doesn't mean that the properties are then set in all splits.

Accessing Global List Properties

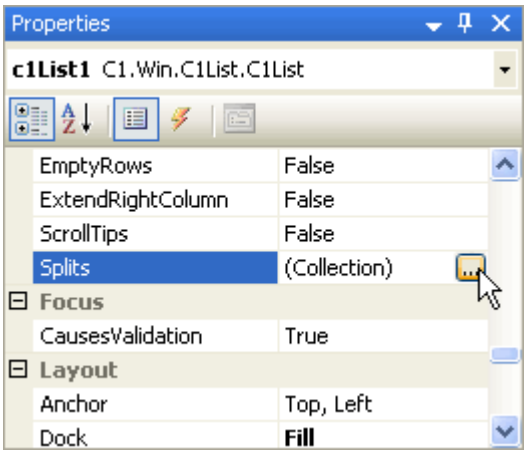
Properties which apply to the entire list object are considered global properties. Once set, these properties will remain set no matter what split-specific or column properties are set. You can access these properties through the Properties window. To open the Properties window, right-click on the C1List or C1Combo control and select **Properties** from the context menu. The Properties window will appear.



The Properties window enables easy access to all of the list's properties and allows you to set property values at design-time. The Properties window orders the properties either categorically or alphabetically, in the image above properties are ordered categorically. In order to allow the user access to objects and collections when ordered categorically, the Properties window supports a tree view structure where objects can be collapsed and expanded to hide and show constituent properties.

Accessing Split-Specific Properties

You can access split-specific properties through the Properties window. When in the C1List or C1Combo Properties window, navigate to the **Splits** property located in the Design node to access split properties.



You can access the **Split Collection Editor** by clicking the **ellipsis** button **(...)** next to the **Splits** property. This editor can be used to access all of the split-specific properties, as well as the C1DisplayColumnCollection properties for the current split. For more information, see Using the Split Collection Editor.

Accessing Column Properties

In the Properties window, *global* column properties, also known as C1DataColumn properties, are accessed through the C1DataColumnCollection object property. You can access the **C1List Designer** by clicking the **ellipsis** button **(...)** next to the Columns node in the Visual Studio Properties window. For more information, see Using the C1List Designer.

Each member of the SplitCollection in the Properties window also exposes a **DisplayColumns** property known as the

C1DisplayColumnCollection object. These C1DisplayColumn properties are split-specific properties. You can access the **C1DisplayColumn Collection Editor** by clicking the **ellipsis** button next to the Splits node in the Properties window and then clicking the **ellipsis** button next to the DisplayColumns node in the **Split Collection Editor**. For more information, see Using the C1DisplayColumn Collection Editor.

Using the Split Collection Editor

The SplitCollection is a collection of **Split** objects which provides access to most of the list's display properties and properties specific to a **Split**. Accessing these properties in code is done through the C1List object and is demonstrated by the following:

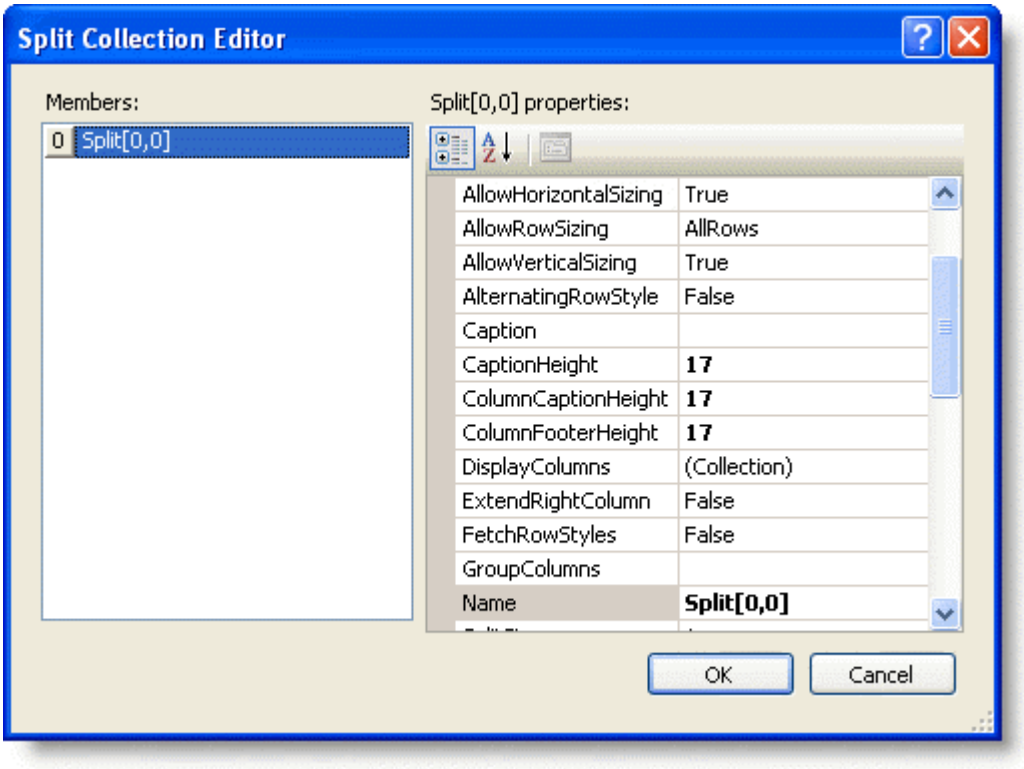
To write code in Visual Basic

```
Visual Basic
Me.C1List.Splits(0).AllowColMove = True
```

To write code in C#

```
C#
this.c1List.Splits[0].AllowColMove = true;
```

.NET contains useful collection editors which make the altering of a collection much easier. The SplitCollection is able to be modified at design-time through a .NET collection editor. The **Split Collection Editor** can be accessed by clicking the **ellipsis** button next to the Splits property in the Properties window.



Notice that the editor doesn't contain buttons to add and delete Splits. Even though you aren't able to use the collection editor to create and delete splits, you still are able to accomplish this at design-time. In the **C1List Designer** there are four commands: InsertHorizontalSplit, RemoveHorizontalSplit, InsertVerticalSplit, and RemoveVerticalSplit. By selecting one of these four options, you can quite simply add and remove splits in the list in design mode.

Split Collection Editor Properties

The following SplitCollection object properties are available in the **Split Collection Editor** through the Properties window:

Misc

Property	Description
AllowColMove	Gets or sets a value indicating the ability to move columns.
AllowColSelect	Gets or sets a value indicating the ability to select columns.
AllowFocus	Gets or sets a value indicating whether the split can receive focus.
AllowHorizontalSizing	Gets or sets a value indicating whether a user is allowed to resize horizontal splits.
AllowRowSizing	Gets or sets how interactive row resizing is performed.
AllowVerticalSizing	Gets or sets a value indicating whether a user is allowed to resize vertical splits.
AlternatingRowStyle	Gets or sets a value indicating whether the split uses the OddRowStyle for odd-numbered rows and EvenRowStyle for even-numbered rows.
Caption	Gets or sets the caption.
CaptionHeight	Gets or sets the height of the caption.
ColumnCaptionHeight	Gets or sets the height of the column captions.
ColumnFooterHeight	Gets or sets the height of column footers.
DisplayColumns	Gets a collection of C1DisplayColumn objects.
ExtendRightColumn	Gets or sets a value that determines how the last column will extend to fill the dead area of the split.
FetchRowStyles	Gets or sets a value indicating whether the FetchRowStyle event will be raised.
Name	Gets or sets the name of a split.
SplitSize	Gets or sets the size of a split.
SplitSizeMode	Gets or sets a value indicating how the **SplitSize** property is used to determine the actual size of a split.
SpringMode	Gets or sets a value that determines how columns will resize when the grid is resized.

Scrolling

Property	Description
HorizontalScrollGroup	Gets or sets the group which synchronizes horizontal scrolling between splits.
HScrollBar	Gets the HBar object that controls the appearance of the horizontal scrollbar.
VerticalScrollGroup	Gets or sets the group which synchronizes vertical scrolling between splits.
VScrollBar	Gets the VBar object that controls the appearance of the vertical scrollbar.

Styles

Property	Description
CaptionStyle	Gets or sets the Style object that controls the appearance of the caption area.
EvenRowStyle	Gets or sets the Style object that controls the appearance of an even-numbered row when using AlternatingRows.
FooterStyle	Gets or sets the Style object that controls the appearance of column footers.
HeadingStyle	Gets or sets the Style object that controls the appearance of the grids column headers.
HighLightRowStyle	Gets or sets the Style object that controls the current row/cell when the **MarqueeStyle** is set to **Highlight Row/Cell**.
OddRowStyle	Gets or sets the Style object that controls the appearance of an odd-numbered row when using AlternatingRows.
SelectedStyle	Gets or sets the Style object that controls the appearance of selected rows and columns.
Style	Gets or sets the root Style object for the Split.

Using the C1List Designer

The C1DataColumnCollection object is a collection of the column properties which relate to data, data accessing, or data formatting. These properties are contained under the **Column** tab. Accessing these properties in code is done through the Columns property under the C1List object, and is demonstrated by the following:

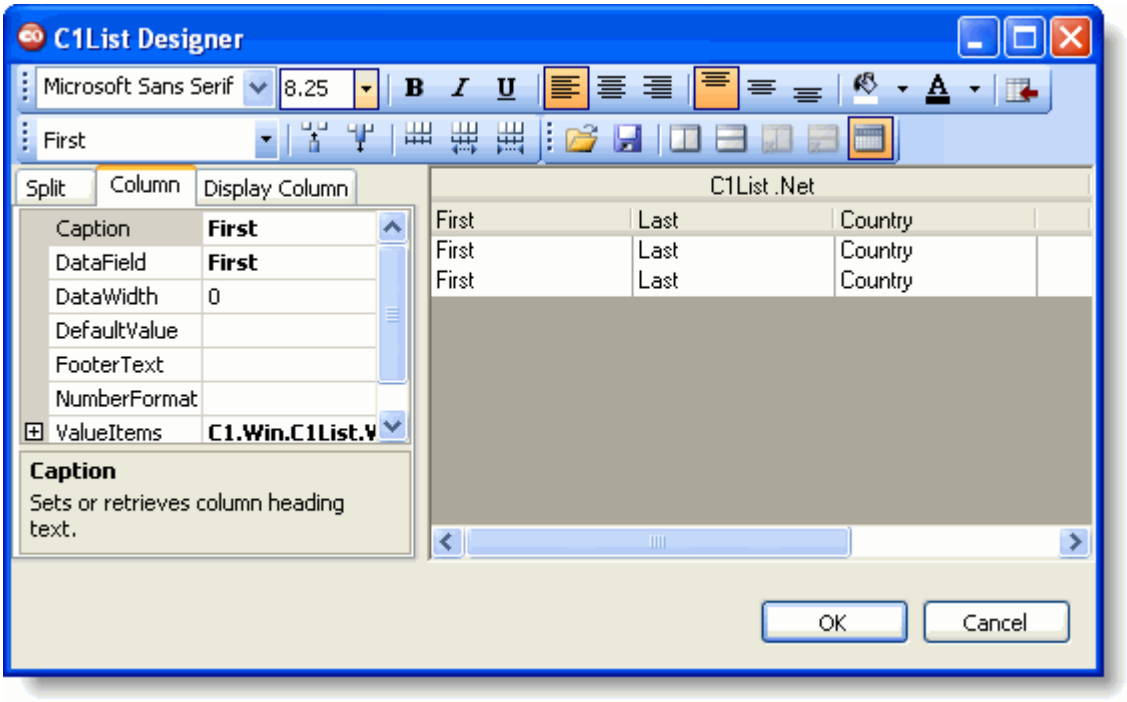
To write code in Visual Basic

```
Visual Basic
Me.C1List1.Columns(0).DataField = "CompanyName"
```

To write code in C#

```
C#
this.c1List1.Columns[0].DataField = "CompanyName";
```

Given the **List for WinForms**' object model, many of these properties would be tough to find and set efficiently. Luckily, .NET contains collection editors which help in categorizing and setting the C1List's collection properties. .NET's collection editor for the C1DataColumnCollection object enables the user to efficiently add, remove and alter C1DataColumn properties. The **C1List Designer** can be accessed by clicking the **ellipsis** button next to the Columns property in the Properties window.



The editor has two panes. The right pane contains the current columns in the list. By clicking on the **Add** or **Remove** buttons in the toolbar, the columns in the right pane can be created or deleted accordingly. The left pane contains the data-related properties for the specific column highlighted in the right pane.

SplitCollection and C1DisplayColumnCollection object properties are also accessible through the **C1List Designer**. The following sections list the available properties in the **C1List Designer**.

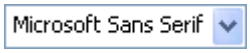
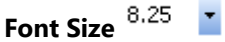
C1List Designer Toolbars

The **C1List Designer** contains three toolbars that can be used to customize the list's content and appearance. The three toolbars are a **Formatting** toolbar, a **Layout** toolbar, and a **Columns** toolbar.

Formatting Toolbar

The **Formatting** toolbar allows you to quickly customize the appearance of the column content, including text, alignment, and color.

The Formatting toolbar consists of the following elements:

Element	Description
Font 	Click the **Font** drop-down arrow to choose the font you want the column to use.
Font Size 	Click the **Font Size** drop-down arrow or enter a number to choose the font size you want the column to use.
Bold 	Select the **Bold** button to bold the text in the chosen column.
Italic 	Select the **Italic** button to italicize the text in the chosen column.

Underline	Select the **Underline** button to underline the text in the chosen column.
Align Left	Select **Align Left** to horizontally align the chosen column to the left of the cell.
Horizontally Center	Select **Horizontally Center** to horizontally align the content of the chosen column to the center of the cell.
Align Right	Select **Align Right** to horizontally align the content of the chosen column to the right of the cell.
Align Top	Select **Align Top** to vertically align the content of the chosen column to the top of the cell.
Vertically Center	Select **Vertically Center** to vertically align the content of the chosen column to the center of the cell.
Align Bottom	Select **Align Bottom** to vertically align the content of the chosen column to the bottom of the cell.
Set Background Color	Select **Set Background Color** to choose the background color of the cells in the chosen column.
Set Foreground Color	Select **Set Foreground Color** to choose the foreground color of the cells in the chosen column.
Apply to Header when Checked	Select **Apply to Header when Checked** to apply formatting changes to the header, and not the body cells, of the chosen column.

Layout Toolbar

The **Layout** toolbar allows you to quickly customize the list's layout allowing you to load and save layouts, insert and delete splits, and show and hide column headers.

The Layout toolbar consists of the following elements:

Element	Description
Load Layout	Select **Load Layout** to load a previously saved grid layout. Through saving and loading the layout you can restore the grid layout.
Save Layout	Select **Save Layout** to save the current grid layout to a file. Through saving and loading the layout you can restore the grid layout.
Insert Horizontal Split	Click **Insert Horizontal Split** to insert a horizontal split at the current selected place in the grid.
Insert Vertical Split	Click **Insert Vertical Split** to insert a vertical split at the current selected place in the grid.
Delete Horizontal	Click **Delete Horizontal** to delete the selected horizontal split.
Delete Vertical	Click **Delete Vertical** to delete the selected vertical split.

Element	Description
Column Headers 	Click **Column Headers** to view column headers in the grid, by default this is selected.

Columns Toolbar

The **Columns** toolbar allows you to quickly add and delete columns, and change column width.

The Columns toolbar consists of the following elements:

Element	Description
Column 	Select the column you wish to view or modify by choosing the column name from the drop-down list. You can also choose a column to modify by clicking the column in the preview window.
Insert Column 	Click **Insert Column** to insert a new column at the current location.
Remove Column 	Click **Remove Column** to delete the currently selected column.
Make all columns the same width 	Click **Make all columns the same width** to make all columns the same width.
Increase column width 	Click **Increase column width** to increase the selected column width.
Decrease column width 	Click **Decrease column width** to decrease the selected column width.

C1List Designer Split Tab Properties

The following SplitCollection object properties are available in the **C1List Designer** through the Properties window on the **Split** tab:

Property	Description
AllowColMove	Gets or sets a value indicating the ability to move columns.
AllowColSelect	Gets or sets a value indicating the ability to select columns.
AllowFocus	Gets or sets a value indicating whether the split can receive focus.
AllowHorizontalSizing	Gets or sets a value indicating whether a user is allowed to resize horizontal splits.
AllowRowSizing	Gets or sets how interactive row resizing is performed.
AllowVerticalSizing	Gets or sets a value indicating whether a user is allowed to resize vertical splits.
AlternatingRowStyle	Gets or sets a value indicating whether the split uses the OddRowStyle for odd-numbered rows and EvenRowStyle for even-numbered rows.
Caption	Gets or sets the caption.

Property	Description
CaptionHeight	Gets or sets the height of the caption.
ColumnCaptionHeight	Gets or sets the height of the column captions.
ColumnFooterHeight	Gets or sets the height of column footers.
DisplayColumns	Gets a collection of C1DisplayColumn objects.
EvenRowStyle	Gets or sets the Style object that controls the appearance of an even-numbered row when using AlternatingRows.
ExtendRightColumn	Gets or sets a value that determines how the last column will extend to fill the dead area of the split.
FetchRowStyles	Gets or sets a value indicating whether the FetchRowStyle event will be raised.
FooterStyle	Gets or sets the Style object that controls the appearance of column footers.
GroupColumns	Gets a collection of C1DisplayColumn objects.
HeadingStyle	Gets or sets the Style object that controls the appearance of the grids column headers.
HighlightRowStyle	Gets or sets the Style object that controls the current row/cell when the **MarqueeStyle** is set to **Highlight Row/Cell**.
HorizontalScrollGroup	Gets or sets the group which synchronizes horizontal scrolling between splits.
HScrollBar	Gets the HBar object that controls the appearance of the horizontal scrollbar.
Name	Gets or sets the name of a split.
OddRowStyle	Gets or sets the Style object that controls the appearance of an odd-numbered row when using AlternatingRows.
RecordSelectorStyle	Gets or sets the Style object that controls the appearance of the RecordSelectors.
SelectedStyle	Gets or sets the Style object that controls the appearance of selected rows and columns.
SplitSize	Gets or sets the size of a split.
SplitSizeMode	Gets or sets a value indicating how the **SplitSize** property is used to determine the actual size of a split.
SpringMode	Gets or sets a value that determines how columns will resize when the grid is resized.
Style	Gets or sets the root Style object for the Split.
VerticalScrollGroup	Gets or sets the group which synchronizes vertical scrolling between splits.
VScrollBar	Gets the VBar object that controls the appearance of the vertical scrollbar.

C1List Designer Column Tab Properties

The following C1DataColumnCollection object properties are available in the **C1List Designer** in the Properties window on the **Column** tab:

Property	Description
Caption	Gets or sets the text in the column header.
DataField	Gets or sets the database field name for a column.
DataWidth	Gets or sets the maximum number of characters which may be entered for cells in this column.

DefaultValue	Gets or sets the default value for a column when a new row is added by the grid.
FooterText	Gets or sets the text displayed in the column footer.
NumberFormat	Gets or sets the formatting string for a column.
ValueItems	Gets the ValueItems object for this column.

C1List Designer DisplayColumn Tab Properties

The following C1DisplayColumnCollection object properties are available in the **C1List Designer** in the Properties window on the **Display Column** tab:

Property	Description
AllowFocus	Gets or sets a value indicating the ability of a column to receive focus.
AllowSizing	Gets or sets a value indicating whether column resizing is allowed.
ButtonFooter	Gets or sets a value indicating whether a column footer will act like a button.
ButtonHeader	Gets or sets a value indicating whether a column header will act like a button.
ColumnDivider	Gets or sets the style of the border drawn between columns.
FetchStyle	Gets or sets a value indicating whether the **FetchCellStyle** event will be raised for a column.
FooterDivider	Gets or sets a value indicating whether to display the column divider in the footer area.
FooterStyle	Gets or sets the Style object that controls the appearance of column footers.
HeaderDivider	Gets or sets a value indicating whether to display the column divider in the header area.
HeadingStyle	Gets or sets the Style that controls the appearance of the column headers.
Height	Gets or sets the height of the column.
MinWidth	Gets or sets the minimum width a column can be resized to when in SpringMode.
Name	Gets the caption of the associated C1DataColumn objects.
OwnerDraw	Gets or sets a value indicating whether cells in this column are drawn by the user in the **OwnerDrawCell** event.
Style	Gets or sets the root Style for this column.
Visible	Gets or sets a value indicating the visibility of a column.
Width	Gets or sets the width of a column.

Using the C1DisplayColumn Collection Editor

The C1DisplayColumnCollection is a collection of the column properties which relate to display, color, font, and so on. These properties are contained under the Columns identifier under the SplitCollection. These properties are also split-specific, as each C1DisplayColumn property can have a different value in different splits. Accessing these properties in code is done through this SplitCollection and is demonstrated by the following:

To write code in Visual Basic

Visual Basic

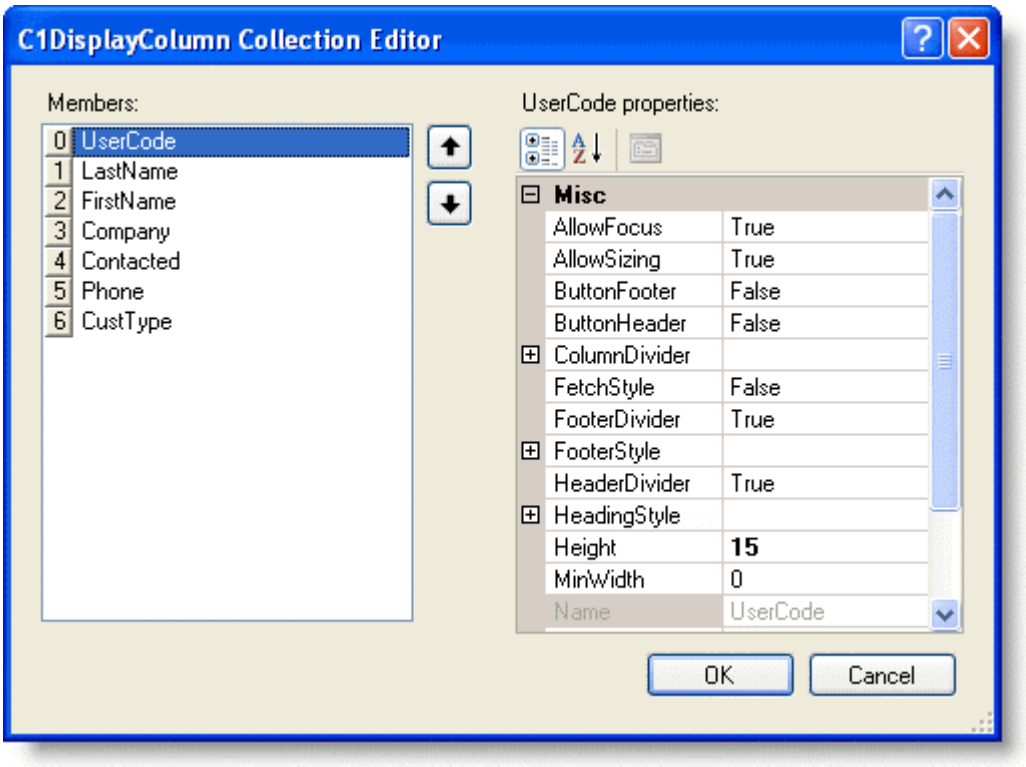
```
Me.C1List1.Splits(0).DisplayColumns(0).Width = 50
```

To write code in C#

C#

```
this.c1List1.Splits[0].DisplayColumns[0].Width = 50;
```

Given the **List for WinForms**' object model with its split-specific column properties and diverse collections, many of its properties might be tough to find and set efficiently. Luckily, .NET contains collection editors which help in categorizing and setting the C1List collection properties. The **Split Collection Editor** can be accessed by clicking the **ellipsis** button next to the Splits property of the list in the Properties window. In the **C1DisplayColumn Collection Editor**, click the **ellipsis** button next to the **DisplayColumns** property to bring up the editor.



The editor has two panes. The left pane contains the current columns in the list under the **Members** heading. Notice that the editor doesn't contain buttons to add and delete columns. Even though you aren't able to use the collection editor to create and delete columns, you still are able to accomplish this at design-time. The right pane contains the display-related properties for the specific column highlighted in the left pane.

Notice that there aren't any add or remove buttons in the **C1DisplayColumn Collection Editor**. Due to the fact that there can be multiple DisplayColumns for each split in the list, the addition or deletion of columns must occur in the **C1List Designer**. This ensures that a column is added to all splits, or removed from all splits.

C1DisplayColumn Collection Editor Properties

The following C1DisplayColumnCollection object properties are available in the **C1DisplayColumn Collection Editor** in the Properties window:

Property	Description
AllowFocus	Gets or sets a value indicating the ability of a column to receive focus.
AllowSizing	Gets or sets a value indicating whether column resizing is allowed.

Property	Description
ButtonFooter	Gets or sets a value indicating whether a column footer will act like a button.
ButtonHeader	Gets or sets a value indicating whether a column header will act like a button.
ColumnDivider	Gets or sets the style of the border drawn between columns.
FetchStyle	Gets or sets a value indicating whether the **FetchCellStyle** event will be raised for a column.
FooterDivider	Gets or sets a value indicating whether to display the column divider in the footer area.
FooterStyle	Gets or sets the Style object that controls the appearance of column footers.
HeaderDivider	Gets or sets a value indicating whether to display the column divider in the header area.
HeadingStyle	Gets or sets the Style that controls the appearance of the column headers.
Height	Gets or sets the height of the column.
MinWidth	Gets or sets the minimum width a column can be resized to when in SpringMode.
Name	Gets the caption of the associated C1DataColumn objects.
OwnerDraw	Gets or sets a value indicating whether cells in this column are drawn by the user in the **OwnerDrawCell** event.
Style	Gets or sets the root Style for this column.
Visible	Gets or sets a value indicating the visibility of a column.
Width	Gets or sets the width of a column.

Using the ValueItem Collection Editor

The ValueItemCollection is a collection of values and display values which allows for translated data within a column. This collection object can be accessed through the C1DataColumn.ValueItems.Values property. Accessing these properties in code is done through this collection, and is demonstrated by the following:

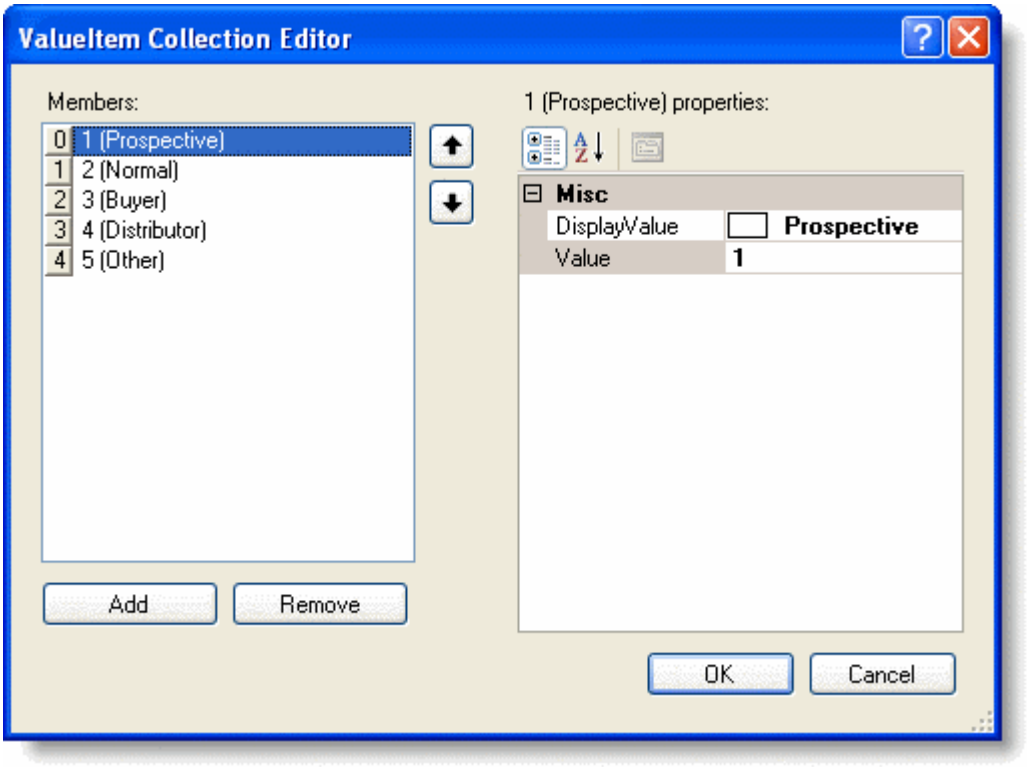
To write code in Visual Basic

```
Visual Basic
Me.C1List1.Columns(0).ValueItems.Values
```

To write code in C#

```
C#
this.c1List1.Columns[0].ValueItems.Values;
```

In order to make these properties more easily modifiable, there is a **ValueItem Collection Editor** which enables the user to add ValueItems, remove ValueItems, and alter their Value and DisplayValue properties. This editor is accessible through the Properties window. Click the **ellipsis** button next to the Columns property in the Properties window to bring up the **C1List Designer**. Expand the ValueItems node to expose the ValueItems collection items. Click the **ellipsis** button next to the Values property to bring up the **ValueItem Collection Editor**.



Using the Style Collection Editor

The Style collection is a collection of Microsoft Word-like styles which can associate certain sections for the list with a style. The Styles collection is located under the C1List object, and contains individual Style objects as its members. Accessing the individual Style objects and their properties in code is done through this collection, and is demonstrated by the following:

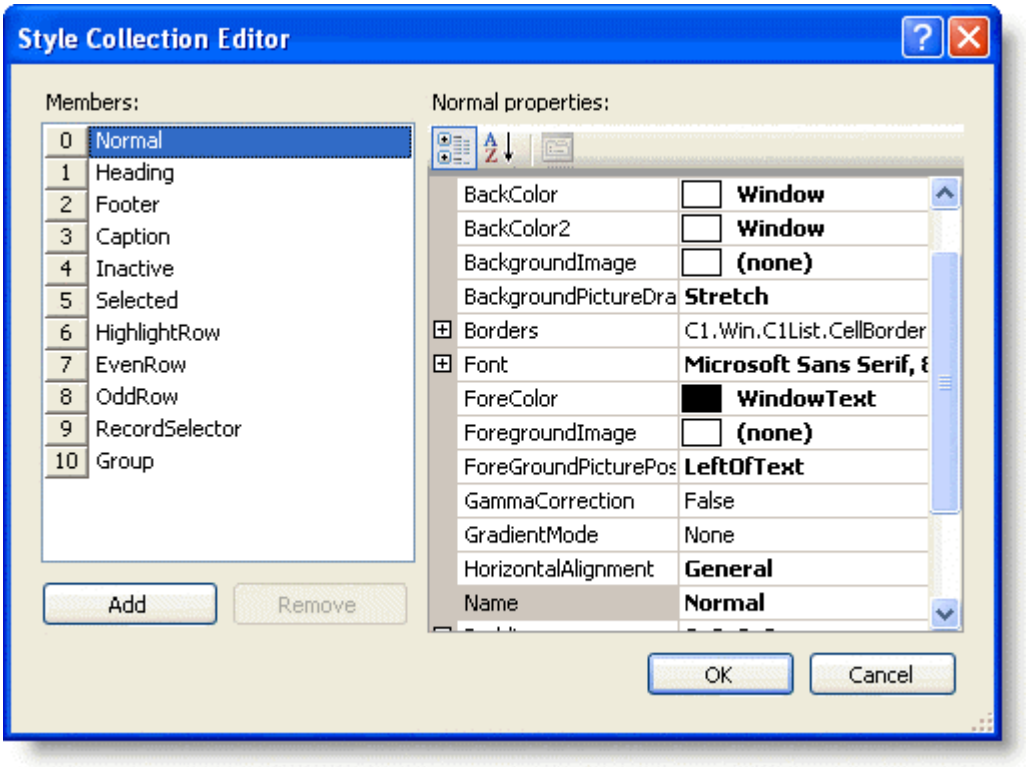
To write code in Visual Basic

```
Visual Basic
Me.C1List1.Styles("Normal").WrapText = False
```

To write code in C#

```
C#
this.c1List1.Styles["Normal"].WrapText = false;
```

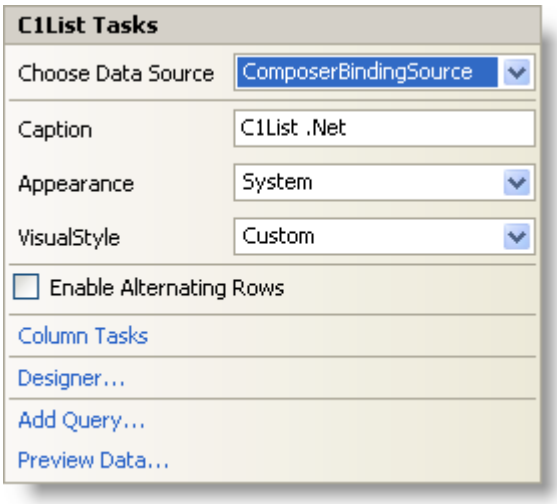
In order to make these properties more easily modifiable, the **Style Collection Editor** enables the user to add styles and modify the properties of existing styles. The **Style Collection Editor** is available in the Properties window. Clicking the **ellipsis** button next to the Styles node in the Properties window will bring up the editor.



C1List Tasks Menu

In the **C1List Tasks** menu, you can quickly and easily choose a data source, set the caption for the grid, change the appearance of the grid, and access the **C1List Designer**, as well as set the AlternatingRows property.

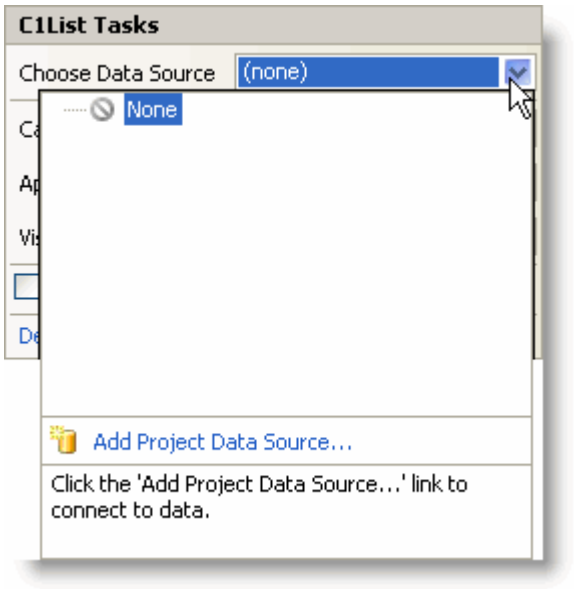
To access the **C1List Tasks** menu, click the smart tag (▶) in the upper right corner of the grid. This will open the **C1List Tasks** menu.



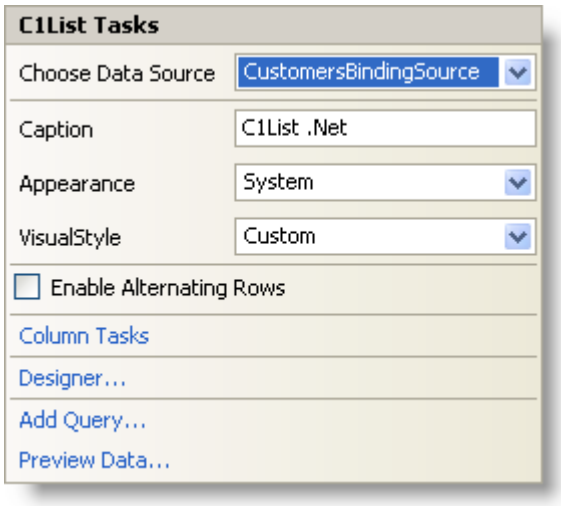
The **C1List Tasks** menu operates as follows:

- ### 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**.



After a data source is selected, three more options are added to the **C1List Tasks** menu: **Column Tasks**, **Add Query**, and **Preview Data**.

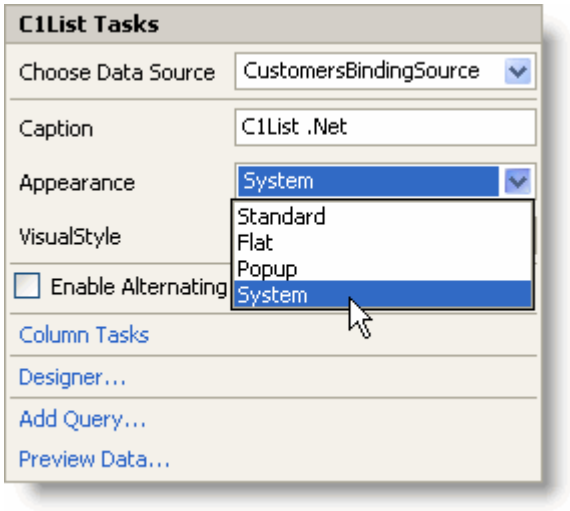


- ## Caption

  Entering a caption into the **Caption** box sets the Caption property for the grid.

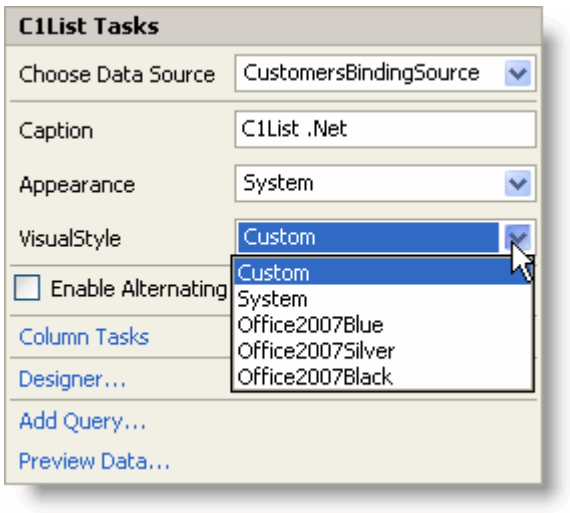
- ## Appearance

  Clicking the drop-down arrow in the **Appearance** box opens a list of different FlatStyle property options, such as **Standard**, **Flat**, **Popup**, and **System**. For more information on the different control appearance options, see Three-Dimensional vs. Flat Display.



- ## VisualStyle

Clicking the drop-down arrow in the **VisualStyle** box opens a list of different VisualStyle property options, such as **Custom**, **System**, **Office2007Blue**, **Office2007Silver**, and **Office2007Black**. For more information on the Visual Style options, see Customizing Visual Styles.



- ## Enable Alternating Rows

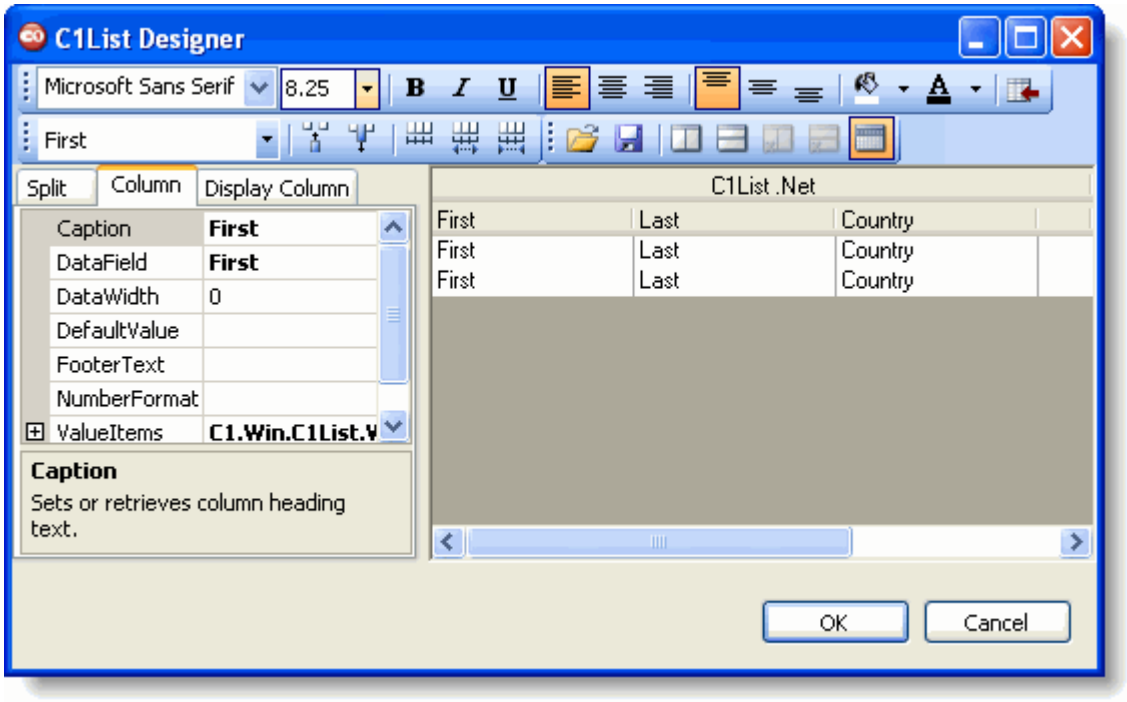
Selecting the **Enable Alternating Rows** check box sets the AlternatingRowsproperty to **True**, and allows adding new rows to the grid. The default is unchecked.

- ## Column Tasks (available only when bound to a data source)

Clicking **Column Tasks** opens the **Column Tasks** menu. For details on the **Column Tasks** menu, see Column Tasks Menu.

- ## Designer

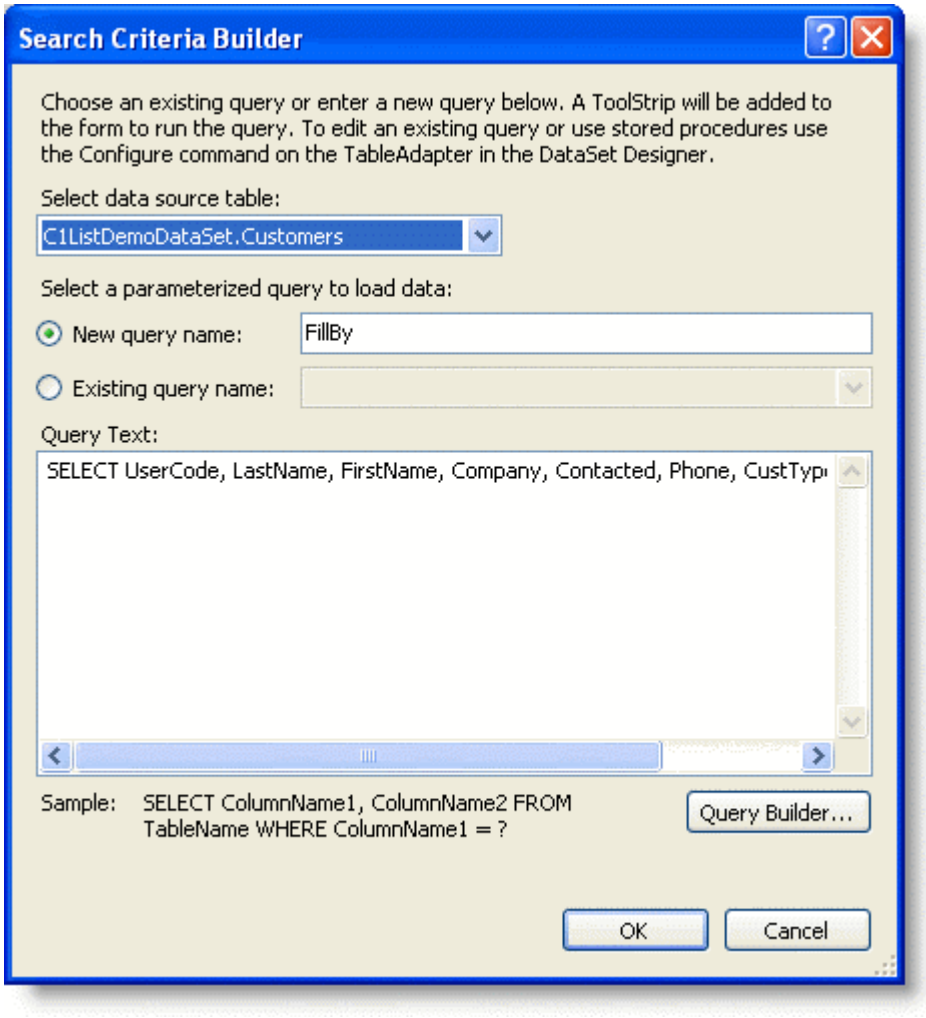
Clicking **Designer** opens the **C1List Designer**.



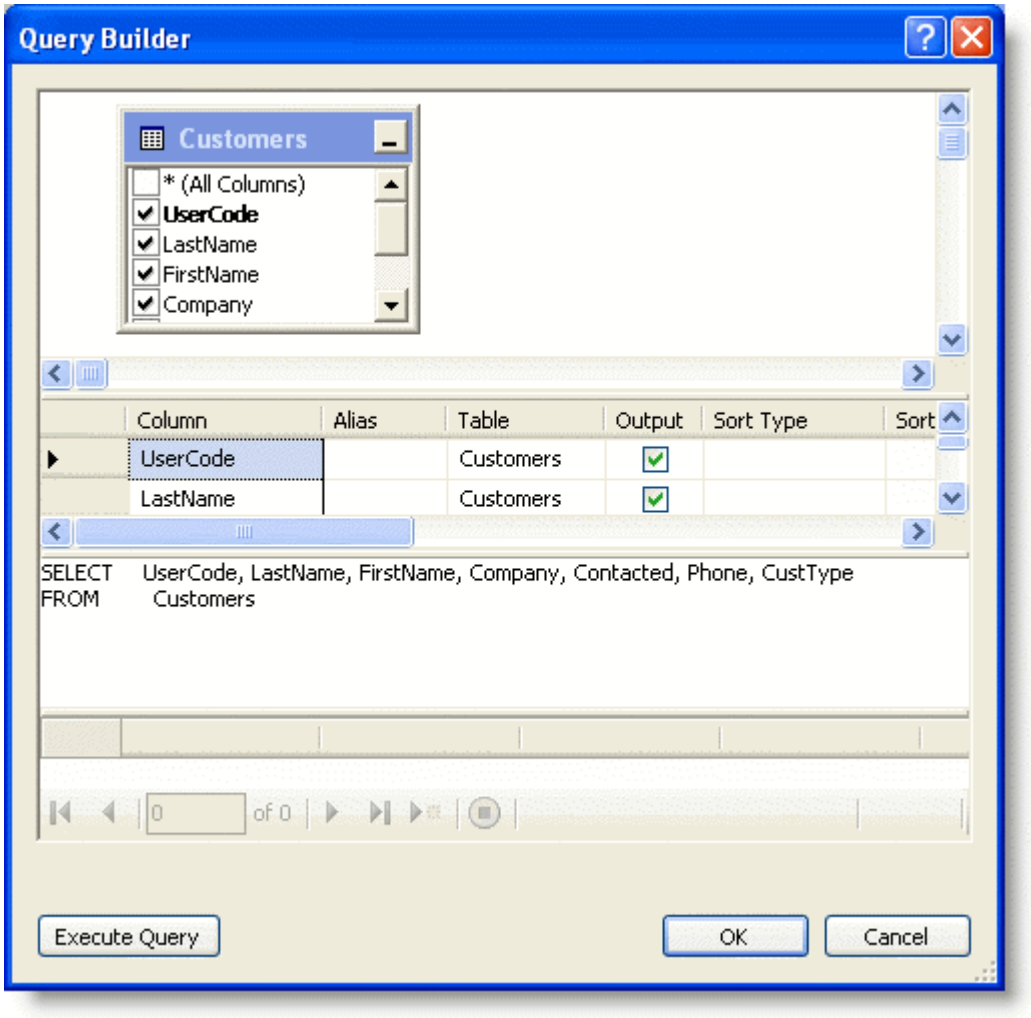
For more information on using the **C1List Designer**, see Using the C1List Designer.

- ## Add Query (available only when bound to a data source)

  Clicking **Add Query** opens the **Search Criteria Builder** dialog box, which allows you to create or modify a query.

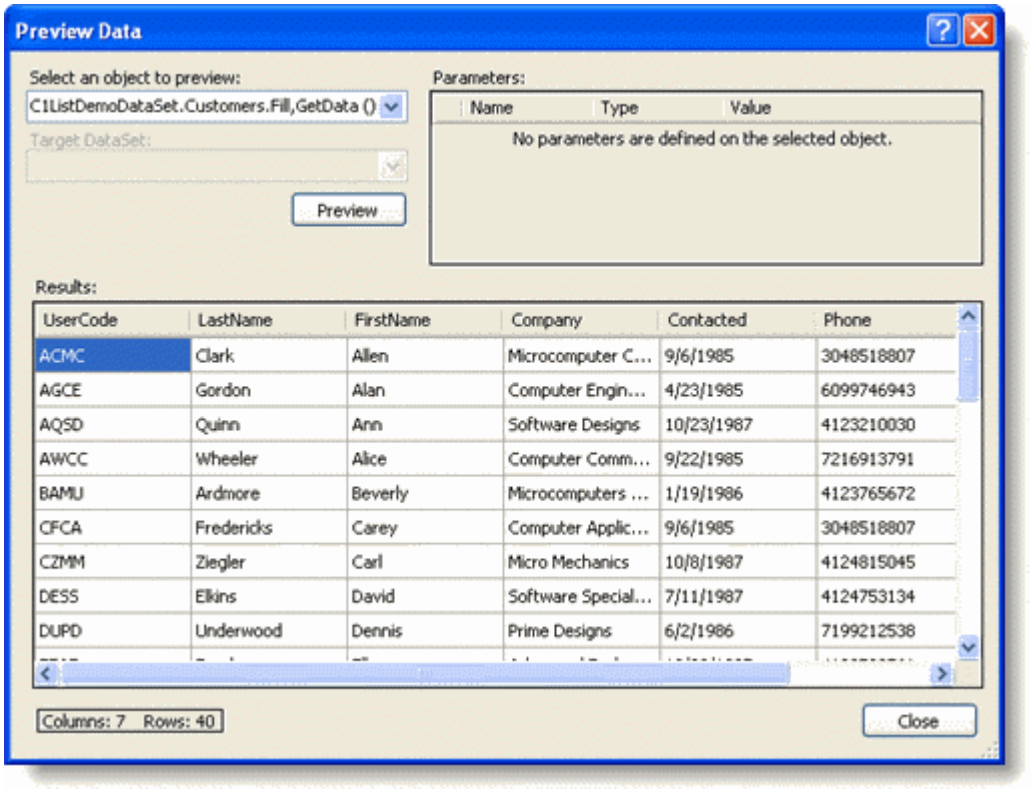


Instead of entering a query in the **Query** Text box, you can use the **Query Builder** to build a query by clicking on the **Query Builder** button.



- ## Preview Data (available only when bound to a data source)

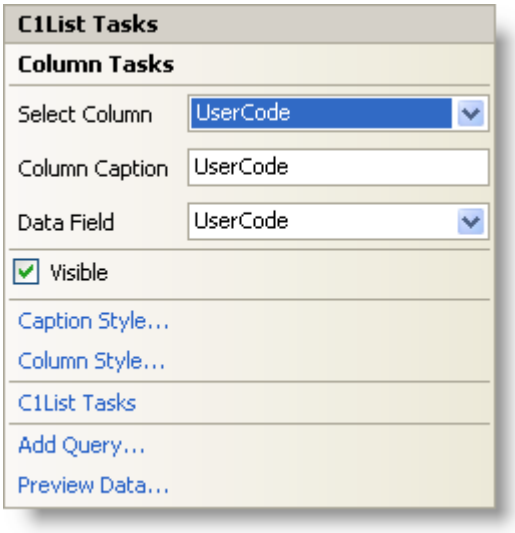
  Clicking **Preview Data** opens the **Preview Data** dialog box, where you can preview the data in the DataSet.



Column Tasks Menu

The **Column Tasks** menu allows you to set the column caption, data field, caption style, and column style, as well as set the Visible property.

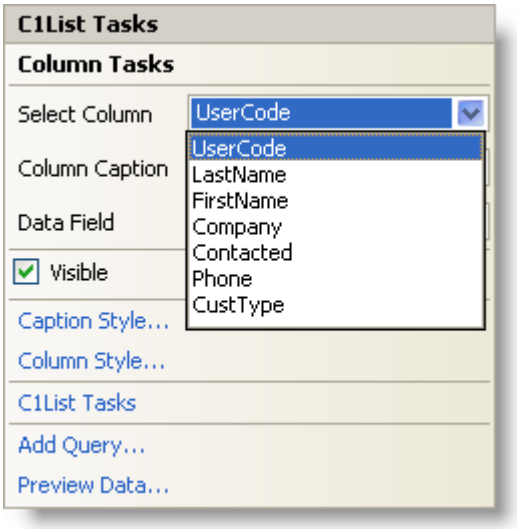
The **Column Tasks** menu can only be accessed when the grid is bound to a data source. To access the **Column Tasks** menu, either click on a column in the grid or select **Column Tasks** from the **C1List Tasks** menu.



The **Column Tasks** menu operates as follows:

- **Select Column**

  Clicking the drop-down arrow in the **Select Column** box opens a list of available columns in the grid. If you clicked on a column in the grid to open the tasks menu, that column will be the selected column.

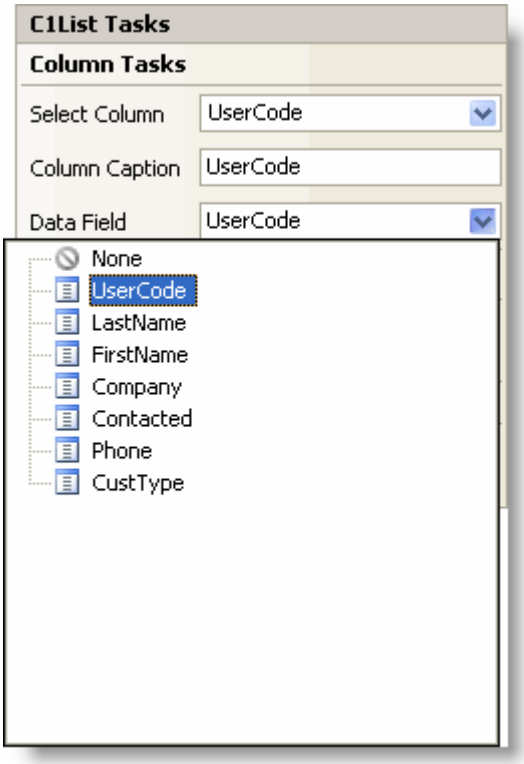


- **Column Caption**

  Entering a caption into the **Column Caption** box set the Caption property for the column.

- **Data Field**

  Clicking the drop-down arrow in the **Data Field** box opens a list of available fields in the data source.

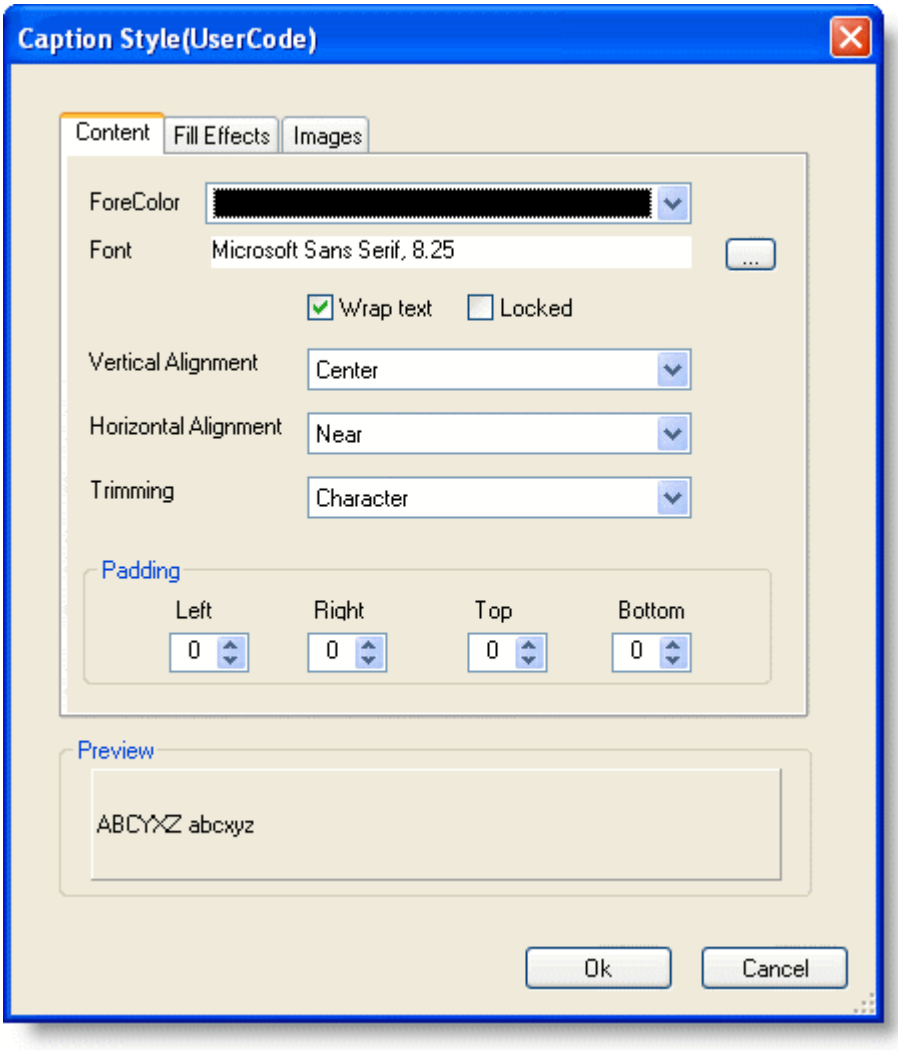


- **Visible**

  Selecting the **Visible** check box sets the Visible property to **True** for the selected column. The default is checked.

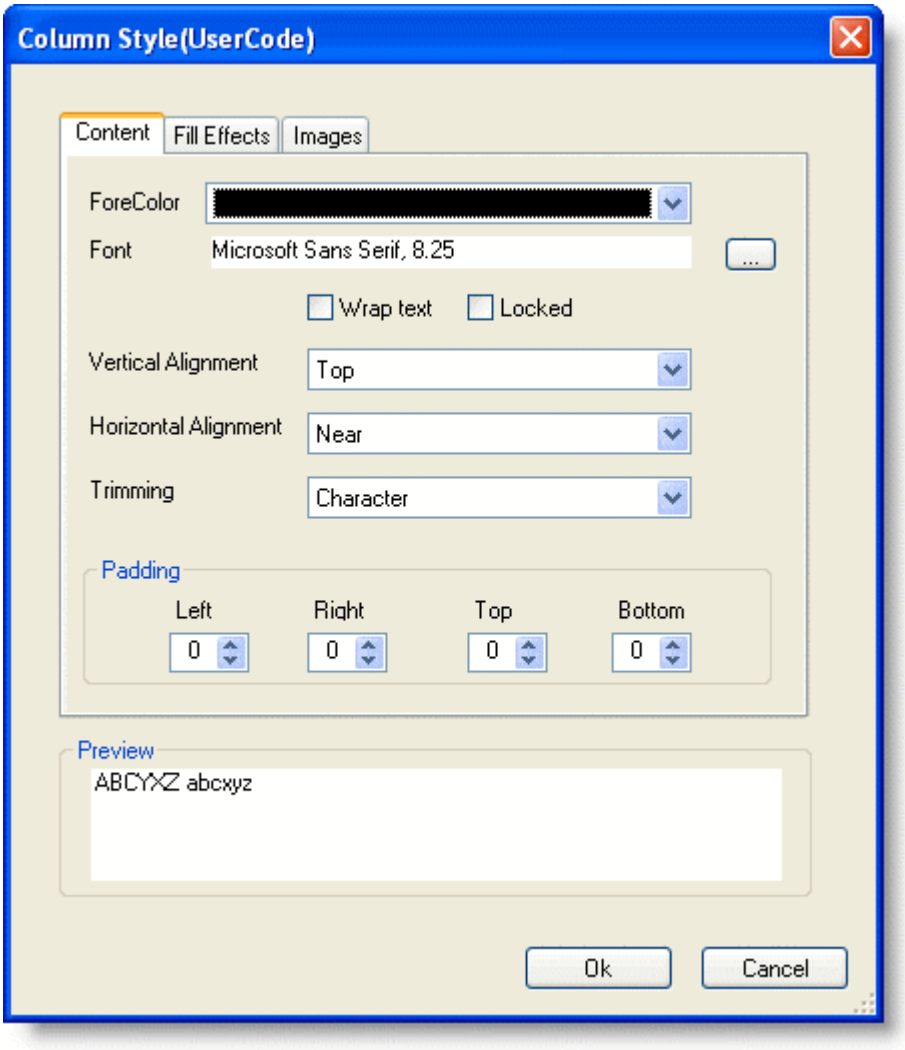
- **Caption Style**

  Clicking **Caption Style** opens the **Caption Style** editor for the selected column, which allows you to specify the properties for the caption , including style, fill effects, and images.



- **Column Style**

  Clicking **Column Style** opens the **Column Style** editor for the selected column, which allows you to specify properties for the column, including style, fill effects, and images.



- **C1List Tasks**

  Clicking **C1List Tasks** returns you to the **C1List Tasks** menu. For details on the **C1List Tasks** menu, see C1List Tasks Menu.

- **Add Query**

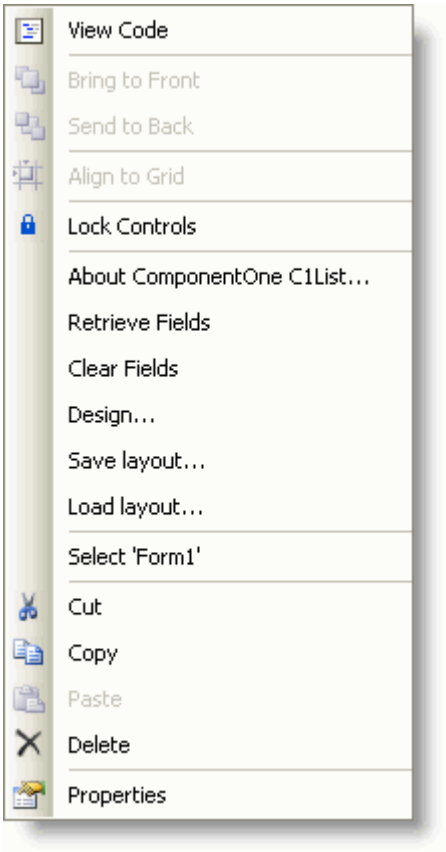
  Clicking **Add Query** opens the **Search Criteria Builder** dialog box, which allows you to create or modify a query.

- **Preview Data**

  Clicking **Preview Data** opens the **Preview Data** dialog box, where you can preview the data in the DataSet.

Context Menu

Right-click anywhere on the list to display the C1List context menu, which is a context menu that Visual Studio provides for all .NET controls, although the C1List context menu has a few extra features.



The context menu commands operate as follows:

- **View Code**

  This command displays the list's code window, which enables you to view and edit the list's event handling code.

- **Bring To Front, Send To Back**

  These commands control the z-order of the list relative to the other objects on the Visual Studio form. **Bring To Front** places the list in front of other objects; **Send To Back** places it behind other objects.

- **Align to Grid**

  This command automatically aligns the outer edges of the list control to the design-time list lines on the form.

- **About ComponentOne C1List**

  This command displays the list's **About** box. This is helpful if you need to know the build number of the list.

- **Retrieve Fields/Clear Fields**

  These commands initiate the methods **RetrieveFields** and ClearFields of the list. RetrieveFields goes back to the data source and retrieves all of the formatting information and base data for the column. **ClearFields** clears out any existing column formatting.

- **Design**

  This command opens the **C1List Designer**.

- **Save Layout/Load Layout**

  These commands allow you to save the current layout of C1List as an XML file or load a previously saved XML layout file.

- **Cut, Copy, Paste, Delete**

  These commands are identical to those on the Visual Studio Edit menu. **Cut** (Ctrl+X) moves the list from the Visual Studio form to the Clipboard. **Copy** (Ctrl+C) moves a copy of the list to the Clipboard while leaving the list on the form intact. **Paste** (Ctrl+V) copies the list from the Clipboard to the form. **Delete** (the Del key) removes the list but does not move it to the Clipboard. You can undo the Delete command by selecting **Undo** (Ctrl+Z) from the Visual Studio Edit menu.

Customizing the List's Appearance

This chapter explains how to configure the non-interactive elements of **List for WinForms** display, such as visual styles, captions, headers, footers, record selectors, and dividing lines.

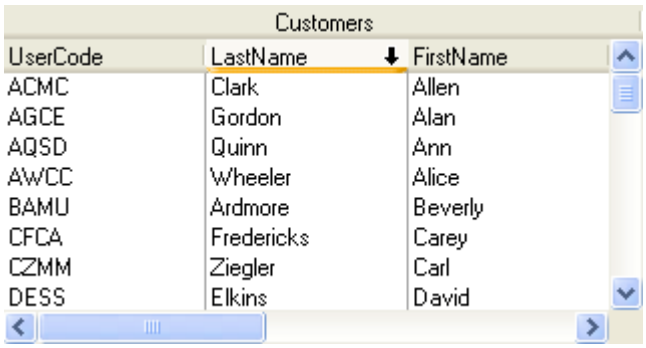
Customizing Visual Styles

List for WinForms supports Visual Styles that mimic the styles available in Office2007. Customizing Visual Styles simple; for example, you can set the list's VisualStyle from the **C1List Tasks** menu, the Properties window, or in code. By default the list's VisualStyle is set to Custom, a standard appearance that does not use Visual Styles and renders the control using only the set styles and properties.

The following Visual Styles are available in **List for WinForms**:

- ## Custom VisualStyle

  When VisualStyle is set to Custom, the list looks like this:

  The Custom Visual Style renders the control using only the set styles and properties. This is the default setting.

- ## System VisualStyle

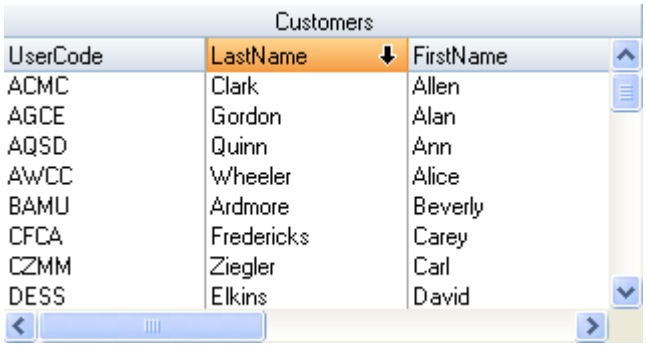
  When VisualStyle is set to System, the list looks like this:

  The System Visual Style renders the control with an appearance based on the current system settings.

- ## Office2007Blue VisualStyle

  When VisualStyle is set to Office2007Blue, the list looks like this:



The Office2007Blue Visual Style renders the control with an appearance based on the Office 2007 Blue color scheme.

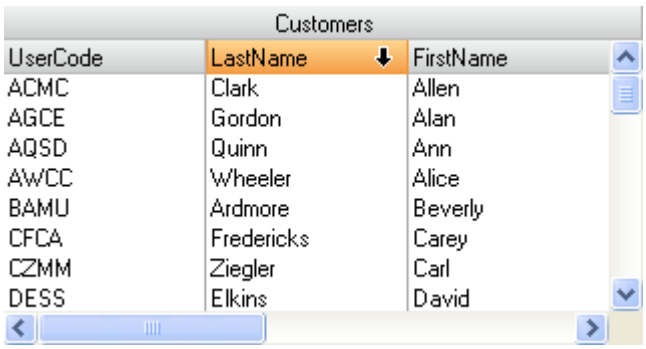
- ## Office2007Silver VisualStyle

When VisualStyle is set to Office2007Silver, the list looks like this:

The Office2007Silver Visual Style renders the control with an appearance based on the Office 2007 Silver color scheme.

- ## Office2007Black VisualStyle

When VisualStyle is set to Office2007Black, the list looks like this:



The Office2007Black Visual Style renders the control with an appearance based on the Office 2007 Black color scheme.

Captions, Headers, and Footers

You can affix a title to a list, column, or split by setting the Caption property of the appropriate object, for example:

To write code in Visual Basic

Visual Basic

```
Me.C1List1.Caption = "List Caption"
Me.C1List1.Columns(0).Caption = "Column 0 Caption"
Me.C1List1.Splits(0).Caption = "Split 0 Caption"
```

To write code in C#

C#

```
this.c1List1.Caption = "List Caption";
this.c1List1.Columns[0].Caption = "Column 0 Caption";
this.c1List1.Splits[0].Caption = "Split 0 Caption";
```

Column and List Captions

For C1DataColumn objects, the Caption property specifies the text that appears in each column's header area.

If you are using C1List bound to a DataSet, the column captions are set automatically at run time.

You can also set column captions using the **C1List Designer** or by manipulating the C1DataColumnCollection in code.

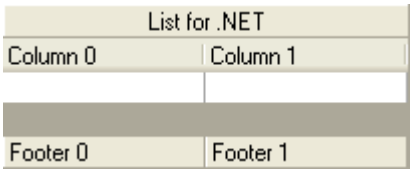
The Caption property also applies to the C1List control itself, which lets you provide a descriptive header for the entire list.

The default value of CaptionHeight property for C1List is **-1**, which automatically sets the caption height according to the font size. To hide the Caption from the control, set CaptionHeight property to **0** at design time or in code.

By default, **C1List** displays headings for each column, even if you never set the Caption property of an individual column explicitly. However, you can hide all column headings by setting the ColumnHeaders property to **False**.

Column Footers

Just as the ColumnHeaders property controls the display of column captions, the ColumnFooters property controls the display of the column footer row. Column footers, which are similar in appearance to column headers, are always displayed at the bottom of the list, even if it is underpopulated.



For each C1DataColumn object, the FooterText property determines the text that is displayed within the footer row. You can set the footer text using the **C1List Designer** or by manipulating the C1DataColumnCollection collection in code, as in the following example:

To write code in Visual Basic

```
Visual Basic
Me.C1List1.ColumnFooters = True
Me.C1List1.Columns(0).FooterText = "Footer 0"
Me.C1List1.Columns(1).FooterText = "Footer 1"
```

To write code in C#

```
C#
this.c1List1.ColumnFooters = true;
this.c1List1.Columns[0].FooterText = "Footer 0";
this.c1List1.Columns[1].FooterText = "Footer 1";
```

Unlike the Caption property, the FooterText property is not set automatically from a bound data source, so you will have to set it yourself.

Multiple-Line Headers and Footers

The split specific property ColumnCaptionHeight property controls the height of the column headers. By default, it is based upon the font setting of the HeadingStyle property. If you need to display more than one line of text in a column header, you can increase the ColumnCaptionHeight property to accommodate additional lines, as in the following example:

To write code in Visual Basic

```
Visual Basic
Me.C1List1.Splits(0).ColumnCaptionHeight = Me.C1List1.Splits(0).ColumnCaptionHeight *
2
Me.C1List1.Columns(0).Caption = "First line" + vbNewLine + "Second line"
```

To write code in C#

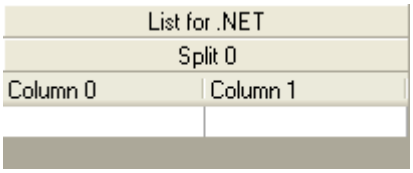
```
C#
this.c1List1.Splits[0].ColumnCaptonHeight = this.c1List1.Splits[0].ColumnCaptonHeight
* 2;
this.c1List1.Columns[0].Caption = "First line \n" + "Second line";
```

Note the use of the constant to specify a line break within the caption text. After this code executes, the first column's caption will contain two lines of text, and the second column's caption will be centered vertically.

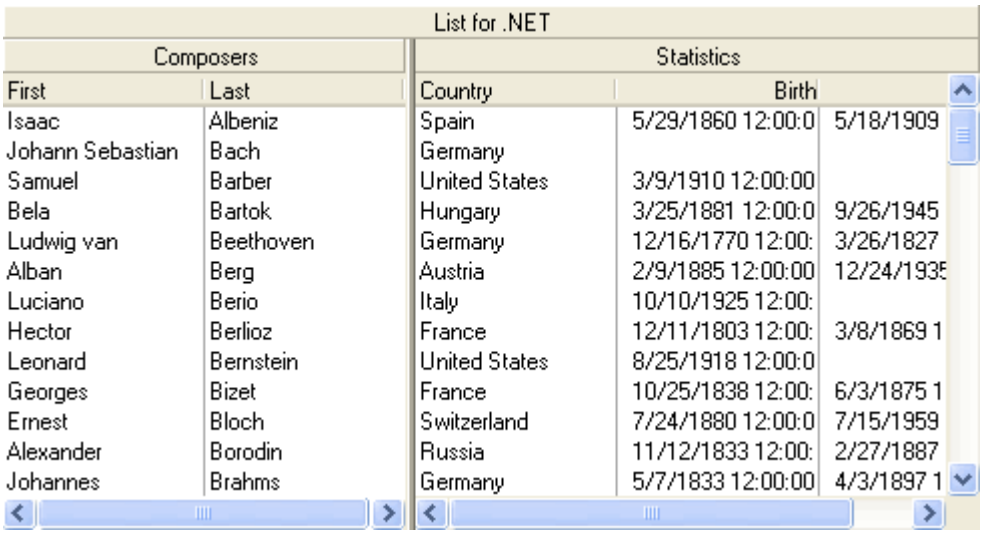
Similarly, you can set the ColumnFooterHeight property to control the height of column footers, and use the vbCr constant to specify a line break when setting the FooterText property of a column.

Split Captions

Split objects can also have their own captions. For a list with one split, a split caption can serve as a second list caption.



However, split captions are best used in lists with at least two splits, as they are ideal for categorizing groups of columns for end users.



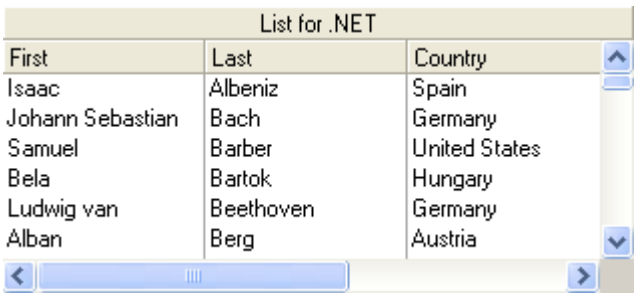
Three-Dimensional vs. Flat Display

List for WinForms supports a standard, "flat" control appearance, the more attractive three-dimensional appearance used by many controls, and a third that combines the flat appearance with the 3D. By default, the list's FlatStyle property is set to System so that the three-dimensional look is used. However, this property only controls whether three-dimensional effects are used to draw the list's border, caption bars, column headings and footings, and the record selector column. It does not affect the list's data cells or row and column dividers.

The following appearance settings are available in **List for WinForms**:

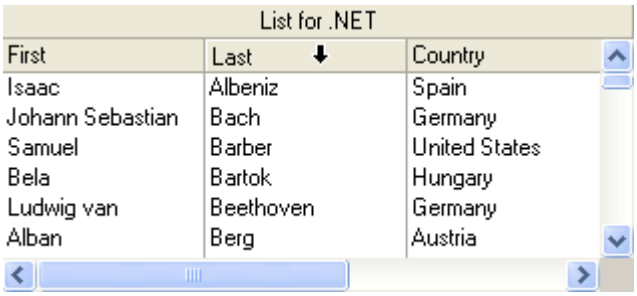
Standard Appearance

When FlatStyle is set to Standard, the list looks like this:



PopUp Appearance

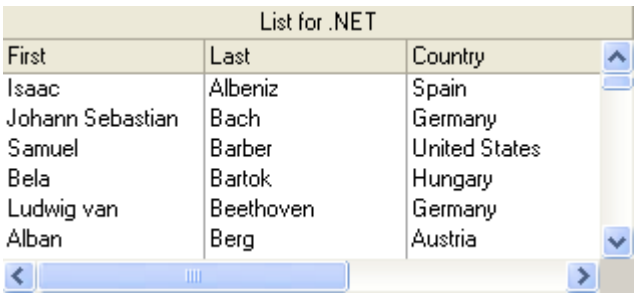
When FlatStyle is set to PopUp, the list looks like this:



The initial list has the same appearance as when FlatStyle is set to Flat (see below). As the mouse moves over any control element, the appearance of that element takes on a 3D look.

Flat Appearance

When FlatStyle is set to Flat, the list looks like this:



Three-Dimensional Appearance

To achieve a three-dimensional appearance for the entire list, including its interior, set the following properties either in the designer or in code:

1. In the Properties window, set the RowDivider style property to Inset. Or, in code:

   To write code in Visual Basic

   ```
   Visual Basic
   Me.C1List1.RowDivider.Style = C1.Win.C1List.LineStyleEnum.Inset
   ```

   To write code in C#

   ```
   C#
   this.c1List1.RowDivider.Style = C1.Win.C1List.LineStyleEnum.Inset;
   ```

2. In the **Splits Collection Editor**, set the Style property to Inset for all ColumnDivider style objects for each split. Or, in code:

   To write code in Visual Basic

   ```
   Visual Basic
   Dim C As C1.Win.C1List.C1DisplayColumn
   ```

```
For Each C In Me.C1List1.Splits(0).DisplayColumns
 C.ColumnDivider.Style = C1.Win.C1List.LineStyleEnum.Inset
Next C
```

To write code in C#

```
C#
C1.Win.C1List.C1DisplayColumn C;
foreach (C in this.c1List1.Splits[0].DisplayColumns)
{
 C.ColumnDivider.Style = C1.Win.C1List.LineStyleEnum.Inset;
}
```

3. In the Properties window, set the **Style.BackColor** property to **Medium Aquamarine**. Or, in code:

   To write code in Visual Basic

```
Visual Basic
Me.C1List1.Style.BackColor = System.Drawing.Color.MediumAquamarine
```

   To write code in C#

```
C#
this.c1List1.Style.BackColor = System.Drawing.Color.MediumAquamarine;
```

The list will look like this:

Note that changing the Style property of the RowDivider object to Inset consumes an extra vertical pixel in each data row, resulting in fewer visible rows.

You can experiment to achieve different three-dimensional effects using other color combinations and divider styles, as explained in the Borders and Dividing Lines section.

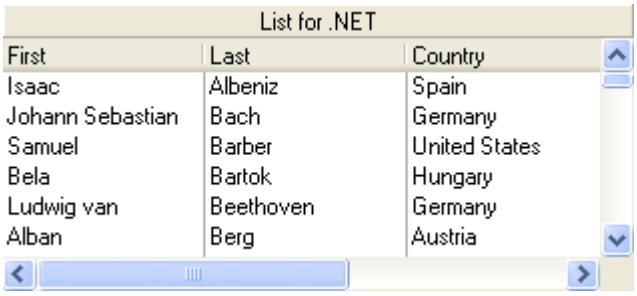
Borders and Dividing Lines

The RowDivider and ColumnDivider properties enable you to choose different horizontal and vertical lines and set their colors. They return an underlying GridLines object which has two properties. These two properties, Style and Color define how the list's cell borders will look. The allowable values for the Style property are as follows:

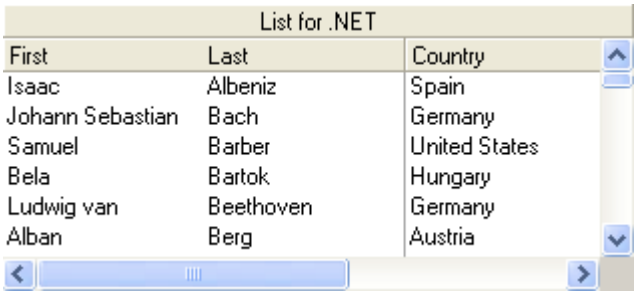
Setting	Description
Double	A double line.
Inset	A line with a three-dimensional inset appearance.

Setting	Description
None	No line.
Raised	A line with a three-dimensional raised appearance.
Single	A single line.

For example, the default setting of the style property of RowDivider to LineStyleEnum.None eliminates the dividing lines between rows and enables you to view slightly more data in the available area.

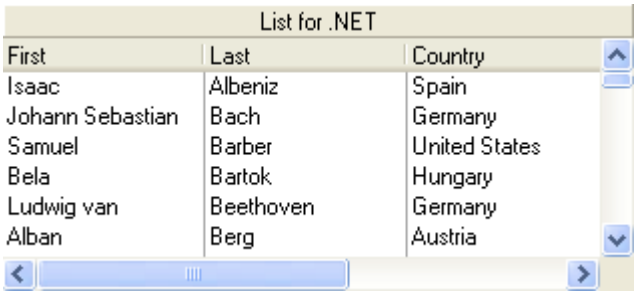


For ColumnDivider properties, you can set the Style property toNone and the HeaderDivider property to **False**. This enables you to visually group related columns, as shown in the following figure.

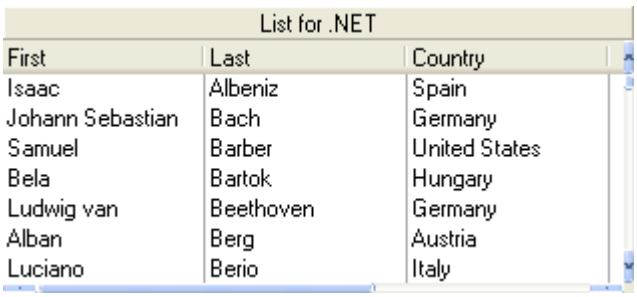


Customizing the Scrollbars

In some applications, you may want to change the thickness of the scroll bars on the list control. By setting the HScrollBar and VScrollBar properties, you can modify the thickness of each scroll bar individually in code.



Scrollbars with default values.



Scrollbars with a height and width value of 5 px.

Scrollbars with a height and width value of 25 px.

As in the following code, you can change the HScrollBar and VScrollBar property values, to create most any scroll bar thickness variation.

To write code in Visual Basic

```
Visual Basic
Me.C1List1.VScrollBar.Width = 40
Me.C1List1.HscrollBar.Height = 50
```

To write code in C#

```
C#
this.c1List1.VScrollBar.Width = 40;
this.c1List1.HscrollBar.Height = 50;
```

Unpopulated Regions

Depending upon the number of rows and columns in the data source, a portion of the list's interior may not contain data cells. However, you can eliminate these "dead zones" using the ExtendRightColumn and EmptyRows properties. Change the color of the dead areas by using the DeadAreaBackColor property.

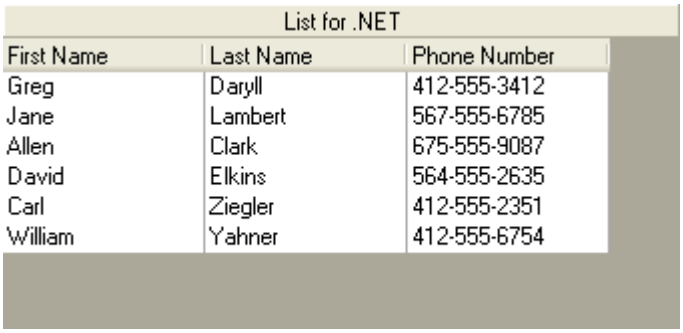
The Rightmost Column

As the list scrolls horizontally until the last column is totally visible, there is usually a blank area between the last column and the right border of the list.

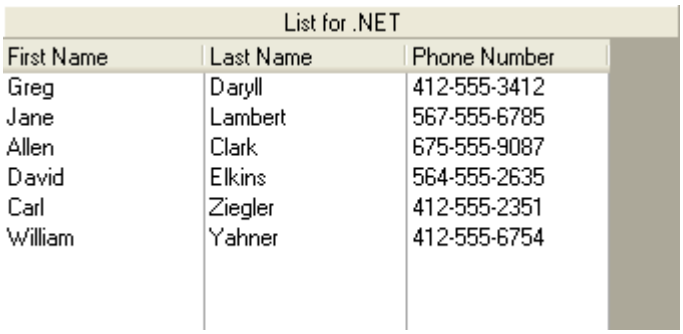
The color of this blank area depends on the setting of the DeadAreaBackColor property. You can eliminate this blank area with the ExtendRightColumn property. The default value of this property is **False**, but if you set it to **True**, then the last column will extend its width to the right edge of the list.

Unused Data Rows

If the data source contains fewer rows than the list can display, the list will display empty rows below the last usable data row.

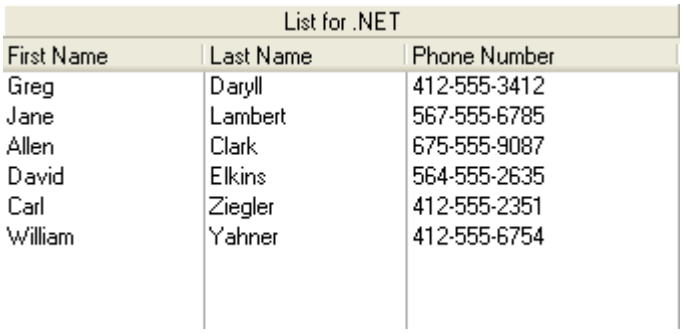


The color of this blank area depends on the setting of the DeadAreaBackColor property. You can eliminate this blank area with the EmptyRows property. The default value of this property is **False**, but if you set it to **True**, then the list will display empty rows below the last usable data row.



Note that the empty rows cannot receive focus.

You can set both the EmptyRows and ExtendRightColumn properties to **True** to ensure that no blank areas appear within the interior of the list.



Row Height and Word Wrap

Row height and word wrap describe how to adjust the height of all **List for WinForms** rows and enable word wrapping in cells.

Adjusting the Height of All List for WinForms Rows

You can configure the row height interactively at design time by placing the list in its visual editing mode or by changing the list's ItemHeight property in the Properties window. At run time, the user can adjust the row height interactively if AllowRowSizing is set to AllRows or IndividualRows. For more information, see Run-Time Interaction.

The ItemHeight property is expressed in units of the container's coordinate system. However, a setting of 0 causes the list to readjust its display so that each row occupies a single line of text in the current font. Therefore, you can use the following code to adjust the row height to display exactly three lines of text:

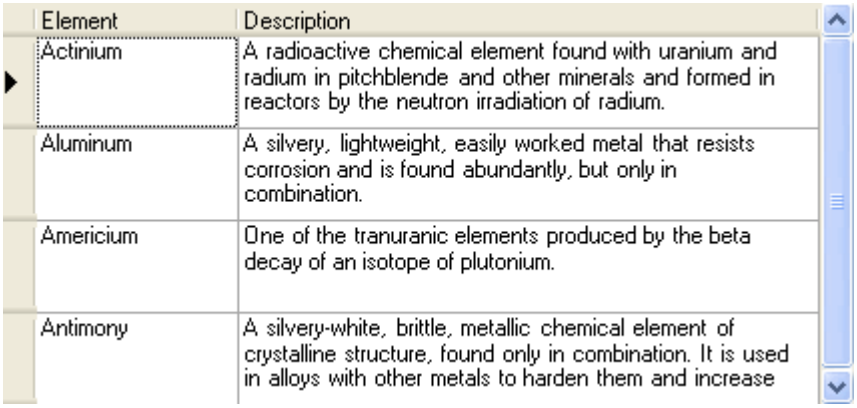
To write code in Visual Basic

```
Visual Basic
Me.C1List1.ItemHeight = 3 * Me.C1List1.ItemHeight
```

To write code in C#

```
C#
this.c1List1.ItemHeight = 3 * this.c1List1.ItemHeight;
```

This technique is particularly effective when displaying multiple-line memo fields, as in this example:



Note that the *Description* column's Style object must have its WrapText property set to **True**; otherwise, the memo field display will be truncated after the first line.

Enabling Word Wrap in Cells

By default, a list cell displays a single line of text, truncated at the cell's right boundary. You can display multiple lines of text in a cell by increasing the list's ItemHeight property and setting the WrapText property of the desired column's Style object to **True**. If WrapText is **True** (the default is **False**), a line break occurs before words that would otherwise be partially displayed in a cell. The cell contents will continue to display on the next line, assuming that the list's row height accommodates multiple lines.

You can use the following loop to enable word wrap for all list columns:

To write code in Visual Basic

```
Visual Basic
```

```
Dim C As C1.Win.C1List.C1DisplayColumn
For Each C In Me.C1List1.Splits(0).DisplayColumns
    C.Style.WrapText = True
Next C
```

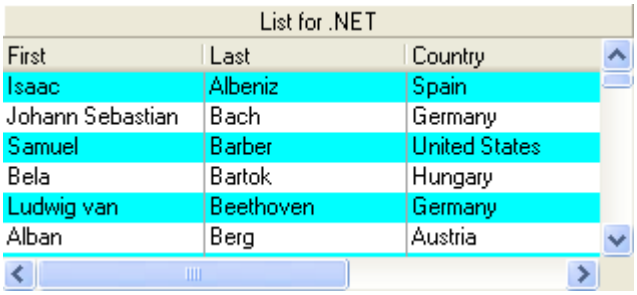
To write code in C#

C#

```
C1.Win.C1List.C1DisplayColumn C;
foreach (C in this.c1List1.Splits[0].DisplayColumns)
{
    C.Style.WrapText = true;
}
```

Alternating Row Colors

You can often improve the readability of the display by alternating the background colors of adjacent rows. When you set the AlternatingRows property to **True**, the list displays odd-numbered rows (the first displayed row is 1) using the built-in style *OddRow*, and even-numbered rows using the built-in style *EvenRow*.



Horizontal and Vertical Alignment

Use the HorizontalAlignment property of the column's Style object to control the horizontal placement of cell text within a column. The allowable values for this property are as follows:

- General
- Near
- Center
- Far
- Justify

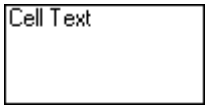
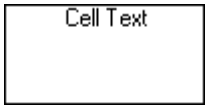
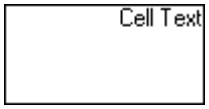
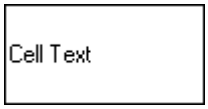
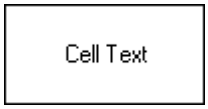
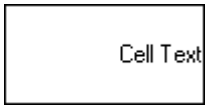
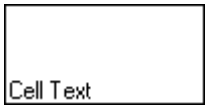
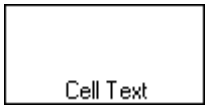
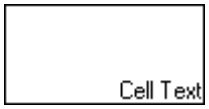
The setting General, which is the default for data cells, indicates that the alignment should be based upon the underlying data type. For example, strings are left-aligned, while numbers are right-aligned.

Use the VerticalAlignment member of the Style object to control the vertical placement of text within the cells of a list, split, or column. The allowable values for this property are as follows:

- Top
- Center
- Bottom

For data cells, the default value is Top. For static list elements such as caption bars, column headers, and column footers, the default value is Center. See the section Named Style Defaults to learn how the default values are derived.

The following grid depicts all possible combinations of the HorizontalAlignment and VerticalAlignment properties:

	AlignHorzEnum.Near	AlignHorzEnum.Center	AlignHorzEnum.Far
AlignVertEnum.Top			
AlignVertEnum.Center			
AlignVertEnum.Bottom			

The General and Justify settings have been omitted because the General setting aligns text as Near and numeric values as Far. The Justify setting aligns text with respect to the cells boundries, but in this case appears exactly like the Near setting.

The HorizontalAlignment and VerticalAlignment properties are tightly integrated with the concept of styles. For more information, see How to Use Styles.

Data Presentation Techniques

The following topics explain how to display cell data in a variety of textual and graphical formats.

Text Formatting

In many cases, the raw numeric data that **List for WinForms** receives from its data source is not suitable for end-user display. For example, date fields may need to be converted to a specific international format; currency fields may contain too many insignificant digits after the decimal point. Therefore, **List for WinForms** provides a method with which you can alter the format of numerical fields, the NumberFormat property.

In addition, for situations where universal formatting of the database information is inadequate, **List for WinForms** provides an event, FormatText, that enables your application to override the default formatting on a per-column basis. By writing a simple handler for this event, you can customize the display of your column data in a myriad of ways.

Numeric Field Formatting

List for WinForms supports a variety of data formatting options through the C1DataColumn object's NumberFormat property. NumberFormat reconfigures the data format that is handed to the list from the database. It can alter most types of numeric values for a particular column. For example, to display all date values within a column according to the form 26-Apr-01, you would use the **Medium Date** setting:

To write code in Visual Basic

```
Visual Basic
Me.C1List1.Columns("HireDate").NumberFormat = "Medium Date"
```

To write code in C#

```
C#
this.c1List1.Columns["HireDate"].NumberFormat = "Medium Date";
```

Note that if you change the NumberFormat property of a column at run time, you do not need to refresh the display, as **List for WinForms** handles this automatically.

For numeric data, the following predefined options are available:

Formatting Option	Description
Standard	Display number with thousands separator, at least one digit to the left and two digits to the right of the decimal separator.
General Number	Display number as is, with no thousand separators.
Currency	Display number with thousand separator, if appropriate; display two digits to the right of the decimal separator. Note that output is based on system local settings.
Percent	Display number multiplied by 100 with a percent sign (%) appended to the right; always display two digits to the right of the decimal separator.

Formatting Option	Description
Fixed	Display at least one digit to the left and two digits to the right of the decimal separator.
Scientific	Use standard scientific notation.
Yes/No	Display No if number is 0; otherwise, display Yes.
True/False	Display False if number is 0; otherwise, display True.
On/Off	Display Off if number is 0; otherwise, display On.
0%	Display number multiplied by 100, then rounded to the nearest integer, with a percent sign (%) appended to the right.
0.00%	Same as Percent.

For date and time data, the following predefined options are available:

Formatting Option	Description
General Date	Display a date and/or time. For real numbers, display a date and time (for example, 4/3/93 05:34 PM); if there is no fractional part, display only a date (for example, 4/3/93); if there is no integer part, display only a time (for example, 05:34 PM). Date display is determined by your system settings.
Long Date	Display a date using your system's long date format.
Medium Date	Display a date using the medium date format appropriate for the language version of Visual Studio.
Short Date	Display a date using your system's short date format.
Long Time	Display a time using your system's long time format: includes hours, minutes, seconds.
Medium Time	Display a time in 12-hour format using hours and minutes and the AM/PM designator.
Short Time	Display a time using the 24-hour format (for example, 17:45).

Formatting with a Custom Event Handler

On occasion, you may find that your current formatting options do not suit your particular needs. Furthermore, you may be restricted in the type of formatting that you can use, or you may need a custom formatting option. In these cases, the FormatText event option can be specified for the NumberFormat property. Choosing this option for a column will cause the FormatText event to fire each time data is about to be displayed in that column. The event allows you to reformat, translate, indent, or do anything you want to the data just prior to display:

To write code in Visual Basic

Visual Basic
`Private Sub C1List1_FormatText(ByVal sender As Object, ByVal e As`

```
C1.Win.C1List.FormatTextEventArgs) Handles C1List1.FormatText
```

To write code in C#

C#

```
private void C1List1_FormatText(object sender, C1.Win.C1List.FormatTextEventArgs e)
```

ColIndex, a member of the FormatTextEventArgs object, is the column number of the list to be reformatted. The Value member contains the current value of the data and serves as a placeholder for the formatted display value. For example, suppose the first column contains numeric values from 1 to 30, and you wish to display the data as Roman numerals:

To write code in Visual Basic

Visual Basic

```
Private Sub C1List1_FormatText(ByVal sender As Object, ByVal e As
C1.Win.C1List.FormatTextEventArgs) Handles C1List1.FormatText

 Dim result As String
    If e.ColIndex = 0 Then

        ' Determine how many X's.
        While e.Value >= 10
            result = result & "X"
            e.Value = e.Value - 10
        End While

        ' Append "digits" 1-9.
        Select Case e.Value
            Case 1
                result = result & "I"
            Case 2
                result = result & "II"
            Case 3
                result = result & "III"
            Case 4
                result = result & "IV"
            Case 5
                result = result & "V"
            Case 6
                result = result & "VI"
            Case 7
                result = result & "VII"
            Case 8
                result = result & "VIII"
            Case 9
                result = result & "IX"
        End Select

        ' Change the actual format.
        e.Value = result
    End If
```

```
End Sub
```

To write code in C#

C#

```
private void C1List1_FormatText( object sender, C1.Win.C1List.FormatTextEventArgs e)

{
    string  result;
    if ( e.ColIndex == 0 )
    {

        // Determine how many X's.
        while ( e.Value >= 10)
        {
            result = result + "X";
            e.Value = e.Value - 10;
        }

        // Append "digits" 1-9.
        switch (e.Value)
        {
            case 1;
                result = result + "I";
                break;
            case 2;
                result = result + "II";
                break;
            case 3;
                result = result + "III";
                break;
            case 4;
                result = result + "IV";
                break;
            case 5;
                result = result + "V";
                break;
            case 6;
                result = result + "VI";
                break;
            case 7;
                result = result + "VII";
                break;
            case 8;
                result = result + "VIII";
                break;
            case 9;
                result = result + "IX";
                break;
        }
```

```
        // Change the actual format.
        e.Value = result;
    }
}
```

Since the **FormatText** event has fewer restrictions than other formatting techniques, you can always use it to gain full control over the textual content of any value displayed in the list.

Automatic Data Translation with ValueItems

While you can always use the FormatText event to map data values to descriptive display values, **List for WinForms** provides a way to translate data automatically without code. With the ValueItemCollection collection object, you can specify alternate text or even pictures to be displayed in place of the underlying data values.

This feature is ideally suited for displaying numeric codes or cryptic abbreviations in a form that makes sense to end users. For example, country codes can be rendered as proper names or even as pictures of their respective flags. The numbers 0, 1, and 2 may be displayed as Yes, No, and Maybe. Note that either the actual values (0, 1, 2) or the translated values (Yes, No, Maybe) may be displayed as radio buttons in a cell or in a drop-down combo box.

What are ValueItems?

The ValueItemCollection object contains a collection and properties which define the association between an underlying data value and its visual representation within the list. The ValueItemCollection contains a collection of zero or more ValueItem objects. Each ValueItem supports two main properties: Value, the underlying data value, and DisplayValue, its visual representation. Both properties are of type Object.

Each C1DataColumn object owns one ValueItemCollection object, which is initially empty.

In code, you can manipulate the collection of ValueItem pairs as you would any other **List for WinForms** or Visual Studio collection. ValueItems can be added, removed, or manipulated through the ValueItemCollection object.

At design time, a **ValueItem Collection Editor** is available through the C1List Designer. For more information, see Using the ValueItem Collection Editor.

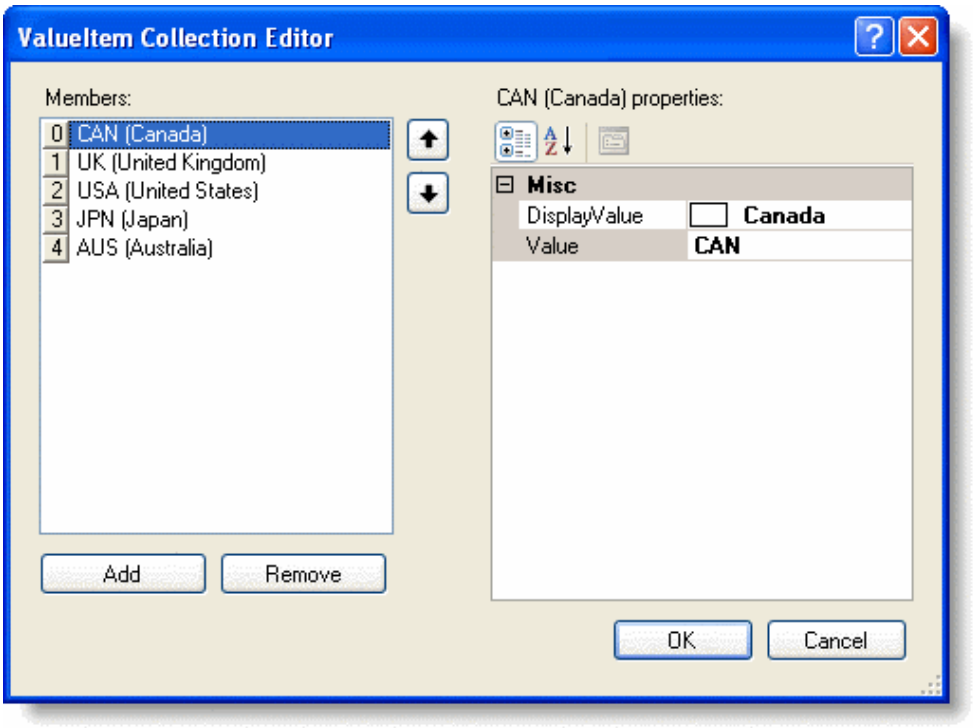
Specifying Text-to-Text Translations

Consider the following example in which the *Country* field is represented by a short character code:

List for .NET

Company	Country
Maple Leaf Systems	CAN
Her Majesty's Software	UK
Software Mart	USA
Far East Distributors	JPN
Outback Software, Inc.	AUS
Northwest Purchasing Agents, Inc.	USA

To display the character codes as proper names, you can use the column's ValueItemCollection object to specify automatic data translations. At design time, this is done with .NET's **ValueItem Collection Editor**



Altering the ValueItemCollection object through the collection editor enables you to specify data translations on a per-column basis in a simple window. To construct a list of data translations for an individual column, do the following:

1. 1 Open up the **C1List Designer** by clicking on the **ellipsis** button **( )** next to the **Columns** collection in the Properties window
2. 2 Select the column whose contents you would like translated in the left pane. In the right pane expand the ValueItems Node. Click the **ellipsis** button next to the Values Node to bring up the **ValueItem Collection Editor**
3. 3 Click the **Add** button in the left pane to add ValueItem objects. In the right pane specify a Value and DisplayValue for each ValueItem. When entering the Value text, disregard the **ellipsis** button. This is used for entering a bitmap as a DisplayValue
4. 4 After the ValueItem objects are configured correctly, select **OK** and return to the **C1List Designer**. In the right pane under the **ValueItems** node, set the Translate property to **True**
5. 5 Select **OK** or **Apply** to commit the changes

When the program is run, *Country* field values that match an item in the Value column appear as the corresponding DisplayValue entry. For example, CAN becomes Canada, UK becomes United Kingdom, and so forth

List for .NET	
Company	Country
Maple Leaf Systems	Canada
Her Majesty's Software	United Kingdom
Software Mart	United States
Far East Distributors	Japan
Outback Software, Inc.	Australia
Northwest Purchasing Agents, Inc.	United States

Note that the underlying database is not affected; only the presentation of the data value is different. The same effect can be achieved in code as follows:

To write code in Visual Basic

Visual Basic

```
Dim Item As C1.Win.C1List.ValueItem

With Me.C1List1.Columns("Country").ValueItems.Values
    Item = New C1.Win.C1List.ValueItem()
    Item.Value = "CAN"
```

```
    Item.DisplayValue = "Canada"
    .Add(Item)

    Item = New C1.Win.C1List.ValueItem()
    Item.Value = "UK"
    Item.DisplayValue = "United Kingdom"
    .Add(Item)

    Item = New C1.Win.C1List.ValueItem()
    Item.Value = "USA"
    Item.DisplayValue = "United States"
    .Add(Item)

    Item = New C1.Win.C1List.ValueItem()
    Item.Value = "JPN"
    Item.DisplayValue = "Japan"
    .Add(Item)

    Item = New C1.Win.C1List.ValueItem()
    Item.Value = "AUS"
    Item.DisplayValue = "Australia"
    .Add(Item)

    Me.C1List1.Columns("Country").ValueItems.Translate = True
End With
```

To write code in C#

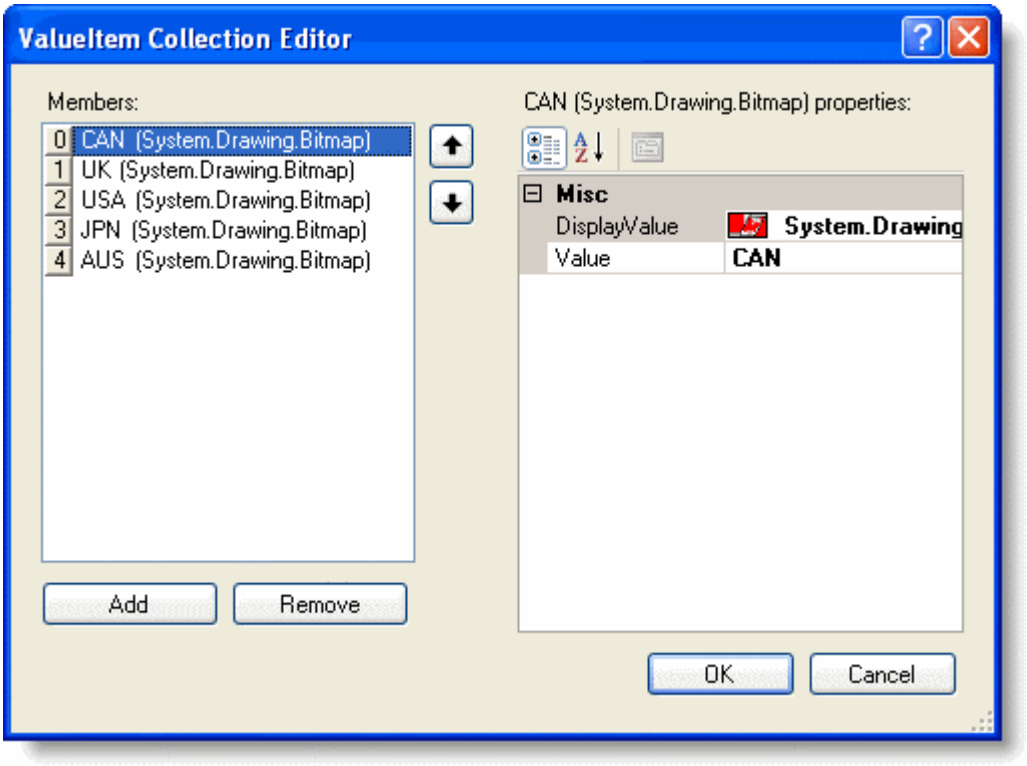
C#

```
C1.Win.C1List.ValueItem Item;

C1.Win.C1List.ValueItem Item = new C1.Win.C1List.ValueItem();
C1.Win.C1List.ValueItemCollection values = this.c1List1.Columns[0].ValueItems.Values;
Item.Value = "CAN";
Item.DisplayValue = "Canada";
values.Add(Item);

Item = new C1.Win.C1List.ValueItem();
Item.Value = "UK";
Item.DisplayValue = "United Kingdom";
values.Add(Item);

Item = new C1.Win.C1List.ValueItem();
Item.Value = "USA";
Item.DisplayValue = "United States";
values.Add(Item);

Item = new C1.Win.C1List.ValueItem();
Item.Value = "JPN";
Item.DisplayValue = "Japan";
values.Add(Item);

Item = new C1.Win.C1List.ValueItem();
Item.Value = "AUS";
```

```
Item.DisplayValue = "Australia";
values.Add(Item);

this.c1List1.Columns["Country"].ValueItems.Translate = true;
```

Specifying Text-to-Picture Translations

The same techniques used to specify text-to-text translations can also be used for text-to-picture translations. Within the **ValueItems Collection Editor**, instead of typing a string into the DisplayValue column, you can use the **ellipsis** button to select a bitmap to be used for data translations. To delete your bitmap selection, simply delete the text in the DisplayValue property box and either select another bitmap or type in text.



Note that the Translate property for the ValueItems object must be set to **True**. Depending upon the height of the bitmaps, you may need to increase the value of the ItemHeight property in the Properties window. If you do so, you may also want to change the VerticalAlignment member of the list's Style property to Center. This ensures that the bitmaps (as well as textual data in other columns) are centered vertically within list cells instead of being anchored at the top.

When the program is run, Country field values that match an item in the Value column appear as the corresponding DisplayValue picture.

As with textual translations, the underlying database is not affected; only the presentation of the data value is different. The same effect can be achieved in code as follows:

To write code in Visual Basic

Visual Basic

```
Dim Item As C1.Win.C1List.ValueItem

With Me.C1List1.Columns("Country").ValueItems.Values
    Item = new C1.Win.C1List.ValueItem()
    Item.Value = "CAN"
    Item.DisplayValue = System.Drawing.Image.FromFile("canada.bmp")
    .Add(Item)

    Item = new C1.Win.C1List.ValueItem()
    Item.Value = "UK"
    Item.DisplayValue = System.Drawing.Image.FromFile("uk.bmp")
    .Add(Item)

    Item = new C1.Win.C1List.ValueItem()
    Item.Value = "USA"
    Item.DisplayValue = System.Drawing.Image.FromFile("usa.bmp")
    .Add(Item)

    Item = new C1.Win.C1List.ValueItem()
    Item.Value = "JPN"
    Item.DisplayValue = System.Drawing.Image.FromFile("japan.bmp")
    .Add(Item)

    Item = new C1.Win.C1List.ValueItem()
    Item.Value = "AUS"
    Item.DisplayValue = System.Drawing.Image.FromFile("australia.bmp")
    .Add(Item)

    Me.C1List1.Columns("Country").ValueItems.Translate = True
End With
```

To write code in C#

C#

```
C1.Win.C1List.ValueItem Item;

C1.Win.C1List.ValueItem Item = new C1.Win.C1List.ValueItem();
C1.Win.C1List.ValueItemCollection values = this.c1List1.Columns[0].ValueItems.Values;
Item.Value = "CAN";
Item.DisplayValue = System.Drawing.Image.FromFile("canada.bmp");
values.Add(Item);
Item = new C1.Win.C1List.ValueItem();
Item.Value = "UK";
Item.DisplayValue = System.Drawing.Image.FromFile("uk.bmp");
values.Add(Item);

Item = new C1.Win.C1List.ValueItem();
Item.Value = "USA";
Item.DisplayValue = System.Drawing.Image.FromFile("usa.bmp");
```

```
values.Add(Item);

Item = new C1.Win.C1List.ValueItem();
Item.Value = "JPN";
Item.DisplayValue = System.Drawing.Image.FromFile("japan.bmp");
values.Add(Item);
Item = new C1.Win.C1List.ValueItem();
Item.Value = "AUS";
Item.DisplayValue = System.Drawing.Image.FromFile("australia.bmp");
values.Add(Item);

this.c1List1.Columns["Country"].ValueItems.Translate = true;
```

Displaying Both Text and Pictures in a Cell

Once you have configured the ValueItemCollection object to perform text-to-picture translations for a column, you can cause both the Value string and the DisplayValue bitmap to appear within the same cell by selecting the AnnotatePicture property under the ValueItems node in the **C1List Designer**. Or, in code:

To write code in Visual Basic

Visual Basic

```
Me.C1List1.Columns("Country").ValueItems.AnnotatePicture = True
```

To write code in C#

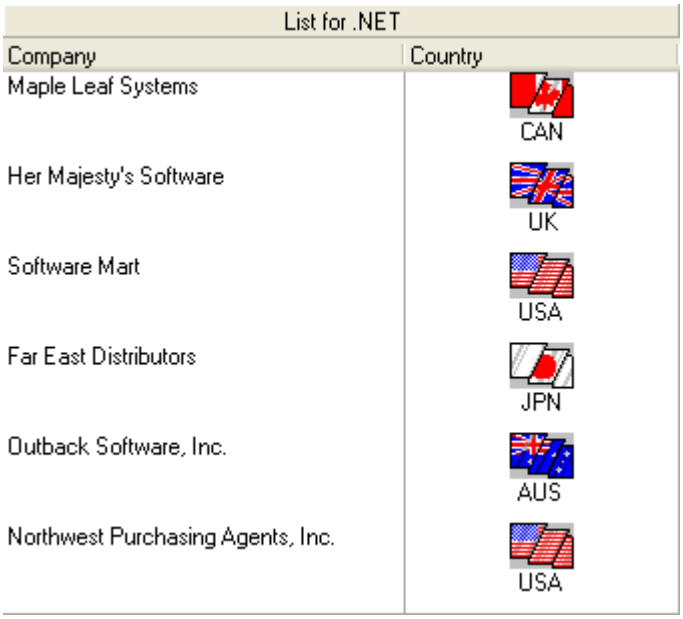
C#

```
this.c1List1.Columns["Country"].ValueItems.AnnotatePicture = true;
```

The horizontal placement of the bitmap with respect to the cell text is determined by the HorizontalAlignment and ForegroundPicturePosition properties of the column's Style object. The enumeration objects for these two properties are the AlignHorzEnum and ForegroundPicturePositionEnum objects respectively. In the following example, HorizontalAlignment is set to General. Since the *Country* column represents a string field, the cell text is left-aligned. However, since the ForegroundPicturePosition property is set to the default value of Near, the bitmap is placed at the left edge of the cell, and the cell text is left-aligned in the remaining space.

However, if you change the ForegroundPicturePosition property to Far, then the cell text is left-aligned as usual, but the bitmap is right-aligned.

To place the cell text below the bitmap while centering both items, set the HorizontalAlignment property to Center and the ForegroundPicturePosition property to TopOfText.



Note: For an illustration of all possible combinations of the HorizontalAlignment and ForegroundPicturePosition properties, see Displaying Foreground Pictures.

When editing, the editor uses all space available in the text portion of the cell. When the Presentation property of the ValueItemCollection object is set to one of the combo box options, the bitmap will not change until editing is completed.

Note that in the preceding examples, the text is displayed as it is stored in the database without formatting. But what if you want to display both a picture *and* formatted text? Since the ValueItem object can only accommodate one translation, you cannot accomplish this with ValueItems alone. However, you can use the FormatText event to translate the text, and then use the ValueItemCollection object to associate the **translated text** (not the underlying data value) with a picture:

To write code in Visual Basic

```
Visual Basic
Me.C1List1.Columns("Country").NumberFormat = "FormatText Event"
```

To write code in C#

```
C#
this.c1List1.Columns["Country"].NumberFormat = "FormatText Event";
```

In this example, the NumberFormat property is set to a special value that causes the FormatText event to fire:

To write code in Visual Basic

Visual Basic

```
Private Sub C1List1_FormatText(ByVal sender As Object, ByVal e As
C1.Win.C1List.FormatTextEventArgs) Handles C1List1.FormatText

    Select Case e.Value
        Case "CAN"
            e.Value = "Canada"
        Case "UK"
            e.Value = "United Kingdom"
        Case "USA"
            e.Value = "United States"
        Case "JPN"
            e.Value = "Japan"
        Case "AUS"
            e.Value = "Australia"
    End Select

End Sub
```

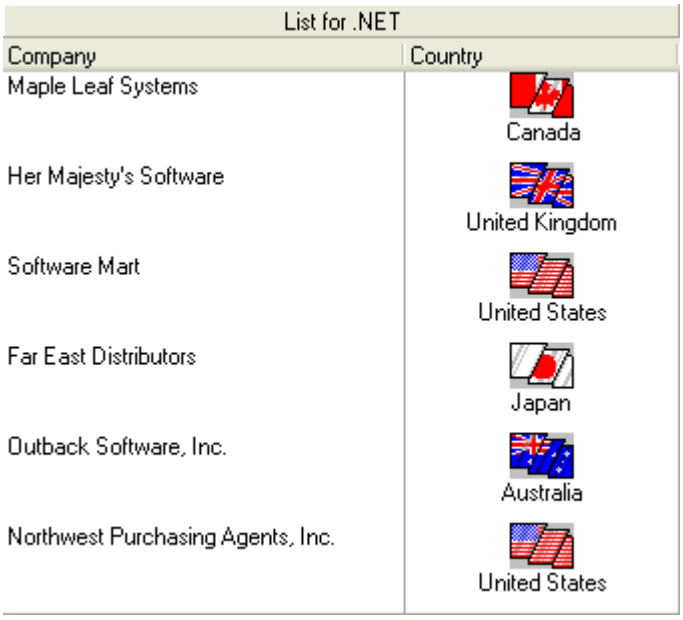
To write code in C#

C#

```
private void C1List1_FormatText(object sender, C1.Win.C1List.FormatTextEventArgs e)

{
    switch (e.Value)
    {
        case "CAN":
            e.Value = "Canada";
            break;
        case "UK":
            e.Value = "United Kingdom";
            break;
        case "USA":
            e.Value = "United States";
            break;
        case "JPN":
            e.Value = "Japan";
            break;
        case "AUS":
            e.Value = "Australia";
            break;
    }

}
```

Since the FormatText event now translates the country codes stored in the database into actual names for display, the Value property of each ValueItem in the ValueItemCollection object must be changed accordingly.

Assuming that the HorizontalAlignment and ForegroundPicturePosition properties are set as in the previous example, the end result is that the underlying data is displayed as both descriptive text and a picture.

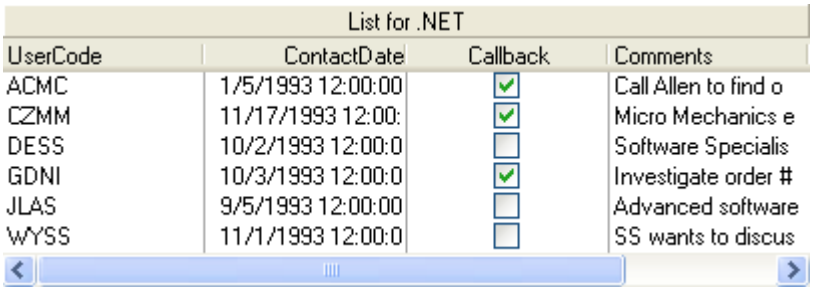


Note: DisplayValue pictures are ideally suited for translating status codes or other fields where the number of allowable values is relatively small. If you need a more generalized picture display mechanism than the ValueItemCollection object offers, you can use the ForegroundPicturePosition property in conjunction with the FetchCellStyle event to display arbitrary pictures on a per-cell basis. For more information, see Displaying Foreground Pictures.

Displaying Boolean Values as Check Boxes

You can also use the ValueItemCollection object to represent Boolean values as in-cell check boxes. You can achieve the effect of a working Boolean checkbox without defining any ValueItem objects just set the Presentation property to CheckBox.

Note that you do not need to select the Translate check box to enable automatic data translation. The following figure shows a typical check box display.



Note: If you want to use different check box bitmaps, you can still define a two-state collection of ValueItem objects through the Value property of the C1DataColumn. Set the Presentation property to Normal, and set the Translate property to **True**.

Displaying Allowable Values as Radio Buttons

If the number of allowable values for a column is relatively small, you may want to consider a radio button

presentation. At design time, go to the **C1List Designer**, then to the **ValueItems** node for the column and set the Presentation property to RadioButton. Or, in code:

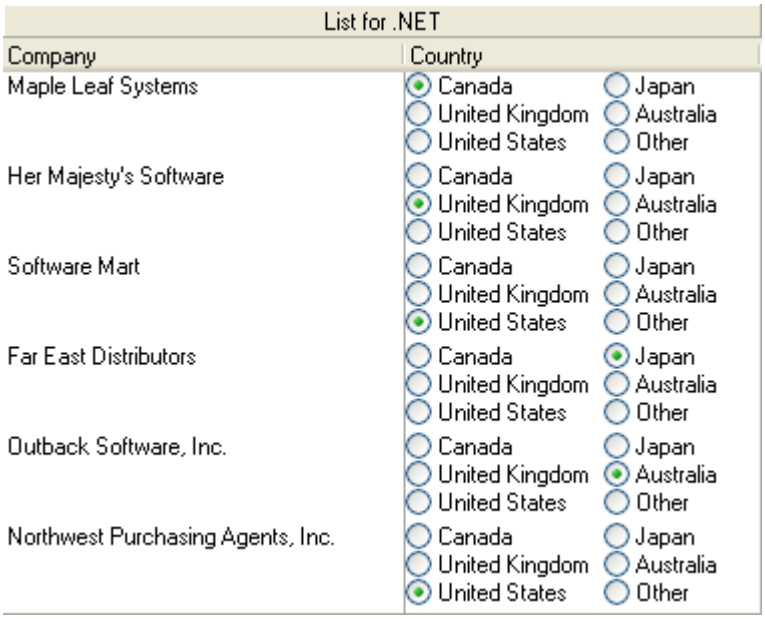
To write code in Visual Basic

```
Visual Basic
Me.C1List1.Columns("Country").ValueItems.Presentation =
C1.Win.C1List.PresentationEnum.RadioButton
```

To write code in C#

```
C#
this.c1List1.Columns["Country"].ValueItems.Presentation =
C1.Win.C1List.PresentationEnum.RadioButton;
```

If necessary, adjust the Width property of the column style and the ItemHeight property of the list to accommodate all of the items.



For a given cell, if the underlying data does not match any of the available values, none of the radio buttons will be selected for that cell. However, you can provide a default ValueItem object that will be selected instead.

In this example, the last ValueItem has an empty Value property, but the DisplayValue is set to "Other". This means that when *Country* fields that do not match any of the items are displayed, their value in the list will be displayed as Other.

To write code in Visual Basic

```
Visual Basic
Me.C1List1.Columns("Country").ValueItems.DefaultItem = 5
```

To write code in C#

```
C#
this.c1List1.Columns["Country"].ValueItems.DefaultItem = 5;
```

In this example, 5 is the zero-based index of the default item.

Context-Sensitive Help with CellTips

In many Windows applications, when the user points to a toolbar button and leaves the mouse at rest for a short time, a ToolTip window appears with the name of the associated command. You can provide similar context-sensitive help for your users with the CellTips property of **List for WinForms**.

The CellTips property determines whether the list displays a pop-up text window when the cursor is idle. By default, this property is set to NoCellTips, and cell tips are not displayed.

The setting Anchored aligns the cell tip window with either the left or right edge of the cell. The left edge is favored, but the right edge will be used if necessary in order to display as much text as possible.

The setting Floating displays the cell tip window below the cursor, if possible.

If you do not provide a handler for the FetchCellTips event, and the cursor is over a list cell, the default behavior is to display a text box containing the cell's contents (up to 256 characters). This enables the user to peruse the contents of a cell even if it is not big enough to be displayed in its entirety. You can also program the FetchCellTips event to override the default cell text display in order to provide users with context-sensitive help.

A common application of the FetchCellTips event is to display the contents of an invisible column that provides additional information about the row being pointed to, as in the following example:

To write code in Visual Basic

Visual Basic

```
' General Declarations.
Dim DescCol As C1.Win.C1List.C1DataColumn

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
    ' Set the column to be displayed as a CellTip.
    Set DescCol = Me.C1List1.Columns("Country")
End Sub
```

```
Private Sub C1List1_FetchCellTips(ByVal sender As System.Object, ByVal e As
C1.Win.C1List.FetchCellTipsEventArgs) Handles C1List1.FetchCellTips
    ' Display the column.
    e.CellTip = DescCol.CellText(e.Row)
End Sub
```

To write code in C#

C#

```
// General Declarations.
C1.Win.C1List.C1DataColumn DescCol;

private void Form1_Load( System.object sender,  System.EventArgs e)
{
    // Set the column to be displayed as a CellTip.
    DescCol = this.c1List1.Columns["Country"];
}

private void c1List1_FetchCellTips( System.object sender,
C1.Win.C1List.FetchCellTipsEventArgs e)
{
    // Display the column.
    e.CellTip = DescCol.CellText(e.Row);
}
```

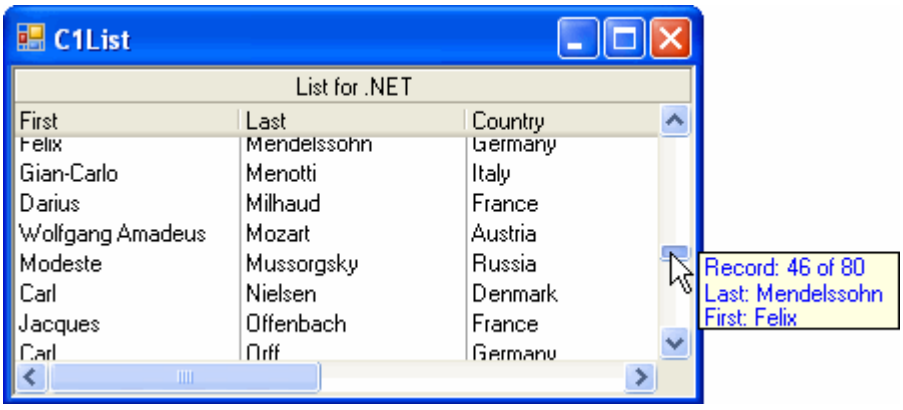
You can use the CellTipsDelay property to control the amount of time that must elapse before the cell tip window is displayed.

You can use the CellTipsWidth property to control the width of the cell tip window.

Scroll Tracking and ScrollTips

If the ScrollTrack property is set to **True**, moving the scrollbar thumb causes vertical scrolling of the list's display. By default, this property is **False**, and no scrolling occurs until the thumb is released.

If the ScrollTips property is set to **True**, moving the scrollbar thumb causes the FetchScrollTips event to fire. You can use this event to track the position of the scroll bar on a record by record basis. You can also use this event to present the user with useful information relating to the current record or recordset. When used in tandem, the ScrollTrack and ScrollTips properties provide users with visual feedback when scrolling through large DataSets



Multiple Line Data Display

Normally, a record is displayed in a single row in the list. If the list is not wide enough to display all of the columns in the record, a horizontal scroll bar automatically appears to enable users to scroll columns in and out of view. For discussion purposes, we shall distinguish between the following:

- A **line** in a list is a single *physical row* of cells displayed in the list. Do not confuse this with a line of text inside a list cell; depending upon the settings of the ItemHeight and Style.WrapText properties, data in a list cell may be displayed in multiple lines of text.
- A **row** in a list is used to display a single record. A row may contain multiple lines or multiple *physical rows*.

Setting the DataView property of the list to MultipleLines will display every field of data in the dataset in the available list area. Ultimately, if your dataset contains more fields than can fit in the list area, then a single record will span multiple lines. This feature enables the end user to view simultaneously all of the columns (fields) of a record within the width of the list without scrolling horizontally, as in the following figure:

If the resulting column layout is not to your liking, you can adjust it either in the designer or in code by changing the widths and orders of the columns. When changing the width of a column, the list will only increase the size of the column at the expense of the other columns in the line. Unlike previous versions of the list, the columns won't wrap to another line if a column is resized.

To change the order of the columns while in Multiple Line view, click and drag the column header to the new position. A red arrow should indicate where you are about to place the column. After you have dropped the column, the list will reposition the columns accordingly.

> **Note:** At design time, if the HScrollBar and VScrollBar style property is set to Automatic, and the DataView property is set to MultipleLines, a vertical scroll bar appears even though no data is displayed. This is done so that you can take the width of the scroll bar into account when adjusting columns at design time.

Implications of Multiple-Line Mode

Existing row-related properties, methods, and events fit well with the earlier definitions of records, rows, and lines (with two exceptions to be described later). For example:

- The VisibleRows property returns the number of visible rows or records displayed on the list not the number of visible lines. If a row spans 2 lines, and the VisibleRows property is 5, then there are 10 visible lines displayed on the list.
- The RowTop method accepts a row number argument ranging from 0 to VisibleRows - 1. If a row spans 2 lines, then RowTop(2) returns the position of the top of the third displayed row (that is, the fifth displayed line).
- The RowResize event will be fired whenever a row is resized by the user at run time. In fact, at the record selector column, only row divider boundaries are displayed; thus, the user can only resize rows, not lines.

Other row-related properties, methods, and events can be interpreted similarly. There are two exceptions:

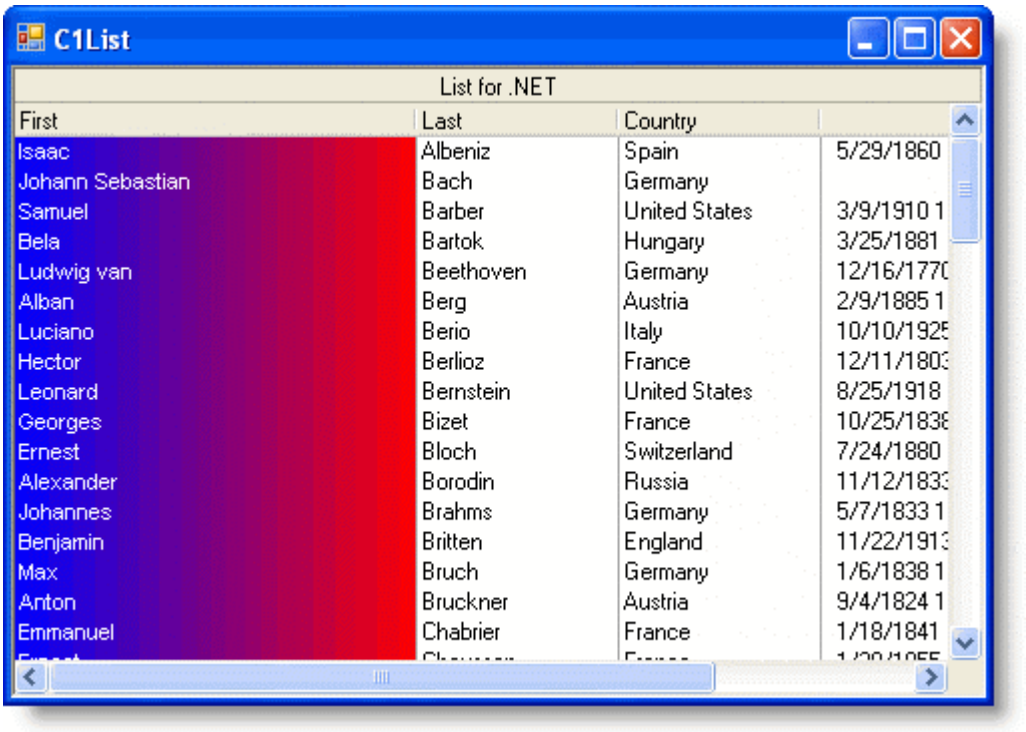
1. The first is the ItemHeight property. The ItemHeight property returns the height of a cell or a line, **not** the

height of a row. Changing this property would break users' existing code, so we decided to keep this property the same.

2. The second is more of a limitation than an exception. Currently the dividers between rows and lines are the same. When you change the RowDivider object's Style property, all dividers between rows and lines change to the same style. That is, you cannot have different dividers for rows and for lines.

Owner-Drawn Cells

For cases where you need to perform complex per-cell customizations, you can draw directly to the list's device context by writing a handler for the OwnerDrawCell event. This event is fired as needed to display the contents of cells that have their OwnerDraw property set to **True**.



The following steps implement the example depicted here:

1. Bind C1List to the **Composer** DataSet. For more information, see Tutorial 1 - Binding C1List to a DataSet.
2. Add the following handler for the **Form_Load** event:

To write code in Visual Basic

Visual Basic

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

    ' This line of code added by Visual Studio 2005.
    Me.ComposerTableAdapter.Fill(Me.DsComposer.Composer)

    ' OwnerDraw the first column.
    Me.C1List1.Splits(0).DisplayColumns(0).OwnerDraw = True
    Me.C1List1.Splits(0).DisplayColumns(0).Width = 2 *
Me.C1List1.Splits(0).DisplayColumns(0).Width
End Sub
```

To write code in C#

C#

```
private void Form1_Load(System.object sender, System.EventArgs e)
{

    // This line of code added by Visual Studio 2005.
    this.ComposerTableAdapter.Fill(this.DsComposer.Composer);

    // OwnerDraw the first column.
    this.c1List1.Splits[0].DisplayColumns[0].OwnerDraw = true;
    this.c1List1.Splits[0].DisplayColumns[0].Width = 2 *
this.c1List1.Splits[0].DisplayColumns[0].Width;
}
```

The **Form_Load** event fills the list and creates the first column for use in the OwnerDrawCell event.

3. Finally, implement the OwnerDrawCell event as follows:

   To write code in Visual Basic

Visual Basic

```
Private Sub C1List1_OwnerDrawCell(ByVal sender As Object, ByVal e As
C1.Win.C1List.OwnerDrawCellEventArgs) Handles C1List1.OwnerDrawCell
    If e.Col = 0 Then

        ' Create a gradient brush, blue to red.
        Dim pt1, pt2 As Point
        Dim lineGrdBrush As System.Drawing.Drawing2D.LinearGradientBrush
        Dim ft As Font
        Dim br As Brush
        pt1 = New Point(e.CellRect.X, e.CellRect.Y)
        pt2 = New Point(e.CellRect.Right, e.CellRect.Y)
        lineGrdBrush = New System.Drawing.Drawing2D.LinearGradientBrush(pt1,
pt2, Color.Blue, Color.Red)

        ' Fill the cell rectangle with the gradient.
        e.Graphics.FillRectangle(lineGrdBrush, e.CellRect)

        ' Draw string.
        ft = New Font("Arial", 8.25)
        br = New SolidBrush(Color.White)
        e.Graphics.DrawString(e.Text, ft, br, e.CellRect.X, e.CellRect.Y)
        lineGrdBrush.Dispose()
        ft.Dispose()
        br.Dispose()

        ' Let the list know that we handled the event.
        e.Handled = True
    End If
End Sub
```

To write code in C#

C#

```
private void C1List1_OwnerDrawCell( object sender,
C1.Win.C1List.OwnerDrawCellEventArgs e)
{
    if ( e.Col == 0 )
    {
        // Create a gradient brush, blue to red.
        pt1, pt2 Point;
       System.Drawing.Drawing2D.LinearGradientBrush lineGrdBrush;
        Font ft;
        Brush br;
        pt1 = new Point(e.CellRect.X, e.CellRect.Y);
        pt2 = new Point(e.CellRect.Right, e.CellRect.Y);
        lineGrdBrush = new System.Drawing.Drawing2D.LinearGradientBrush(pt1,
pt2, Color.Blue, Color.Red);

        // Fill the cell rectangle with the gradient.
        e.Graphics.FillRectangle(lineGrdBrush, e.CellRect);

        // Draw string.
        ft = new Font("Arial", 8.25);
        br = new SolidBrush(Color.White);
        e.Graphics.DrawString(e.Text, ft, br, e.CellRect.X, e.CellRect.Y);
        lineGrdBrush.Dispose();
        ft.Dispose();
        br.Dispose();

        // Let the list know that we handled the event.
        e.Handled = true;
     }
}
```

Note: If you set the *e.Handled* argument to **True**, the list will **not** fill in the cell's background, nor will it display cell text or graphics. Therefore, you are responsible for filling in the **entire cell**, even if there is no data to display.

How to Use Splits

In **List for WinForms**, a *split* is similar to the split window features of products such as Microsoft Excel and Word. You can use splits to present your data in multiple horizontal or vertical panes. The list pictured below, for example, includes two splits, a horizontal and a vertical split, to create four distinct panes.

These panes, or splits, can display data in different colors and fonts. They can scroll as a unit or individually, and they can display different sets of columns or the same set. You can also use splits to prevent one or more columns or a set of rows from scrolling. Unlike other list products, fixed (nonscrolling) columns or rows in **List for WinForms** do not have to be at the left edge of the list, but can be at the right edge or anywhere in the middle. You can even have multiple groups of fixed columns or rows within a list. Splits open up an endless variety of possibilities for presenting data to users of your applications.

Whenever you use **List for WinForms**, you are always using a split. A list always contains at least one horizontal split, and the default values for the split properties are set so that you can ignore splits until you want to use them. Therefore, you can skip this chapter if you do not need to create and manipulate more than one split within a list.

You create and manipulate splits by working with **Split** objects and the SplitCollection object. Since an individual column can be visible in one split but hidden in another, each **Split** object maintains its own collection of columns, known as C1DisplayColumnCollection. This gives you complete control over the appearance of each split and the columns they contain.

Referencing Splits and Their Properties

A C1List object initially contains a single horizontal split. If additional splits are created, you can determine or set the current split (that is, the split that has received focus) using the list's **Split** property:

To write code in Visual Basic

Visual Basic

```
' Read the zero-based index of the current split.
Variable = Me.C1List1.Split

' Set focus to the split with an index equal to Variable.
Me.C1List1.Split = Variable
```

To write code in C#

C#

```
// Read the zero-based index of the current split.
Variable = this.c1List1.Split;

// Set focus to the split with an index equal to Variable.
this.c1List1.Split = Variable;
```

Each split in a list is a different view of the same data source, and behaves just like an independent list. If you create additional splits without customizing any of the split properties, all splits will be identical and each will behave very much like the original list with one split.

Note that some properties are supported by both the C1List and **Split** objects. Three rules of thumb apply to properties that are common to a list and its splits:

1. When you set or get the property of a **Split** object, you are accessing a specific split, and other splits in the same list are not affected.
2. When you get the property of a C1List object, you are accessing the same property within the current split.
3. When you set the property of a C1List object, you are setting the same property within all splits.

To understand how these rules work in code, consider a list with two horizontal splits, and assume that the current split index is 1.

Split Properties Common to List for WinForms

The following properties, which are supported by both **Split** and C1List objects, adhere to the rules described in the preceding section:

Property	Description
AllowColMove	Gets or sets a value indicating the ability to move columns.
AllowColSelect	Gets or sets a value indicating the ability to select columns.
AllowRowSizing	Gets or sets how interactive row resizing is performed.
Caption	Gets or sets the caption.
CaptionHeight	Gets or sets the height of the caption.
CaptionStyle	Gets or sets the Style object that controls the appearance of the caption area.
ColumnCaptionHeight	Gets or sets the height of the column captions.
ColumnFooterHeight	Gets or sets the height of column footers.
EvenRowStyle	Gets or sets the Style object that controls the appearance of an even-numbered row when using AlternatingRows.
ExtendRightColumn	Gets or sets a value that determines how the last column will extend to fill the dead area of the split.
FetchRowStyles	Gets or sets a value indicating whether the FetchRowStyle event will be raised.
TopIndex	Returns or sets a value containing the bookmark for the first visible row in a list or split.
FooterStyle	Gets or sets the Style object that controls the appearance of column footers.
HeadingStyle	Gets or sets the Style object that controls the appearance of the grids column headers.
HighLightRowStyle	Gets or sets the Style object that controls the current row/cell when the **MarqueeStyle** is set to **Highlight Row/Cell**.

Property	Description
HScrollBar	Gets the HBar object that controls the appearance of the horizontal scrollbar.
LeftCol	Gets or sets the left most visible column for a split.
OddRowStyle	Gets or sets the Style object that controls the appearance of an odd-numbered row when using AlternatingRows.
SelectedStyle	Gets or sets the Style object that controls the appearance of selected rows and columns.
SpringMode	Gets or sets a value that determines how columns will resize when the grid is resized.
Style	Gets or sets the root Style object for the Split.
VScrollBar	Gets the VBar object that controls the appearance of the vertical scrollbar.

Note: The Caption property is not included in this list, even though it is supported by both objects. Since lists and splits maintain separate caption bars, setting the Caption property of the list does not apply the same string to each split caption.

Split-Only Properties Not Supported by List for WinForms

The following properties are supported by **Split** objects but not by C1List. Therefore, to apply a value to the entire list, you need to explicitly set the value for each split individually.

Property	Description
AllowFocus	Gets or sets a value indicating whether the split can recive focus.
AllowHorizontalSizing	Gets or sets a value indicating whether a user is allowed to resize horizontal splits.
AllowVerticalSizing	Gets or sets a value indicating whether a user is allowed to resize vertical splits.
AlternatingRowStyle	Gets or sets a value indicating whether the split uses the OddRowStyle for odd-numbered rows and EvenRowStyle for even-numbered rows.
DisplayColumns	Gets a collection of C1DisplayColumn objects.
HorizontalScrollGroup	Gets or sets the group which synchronizes horizontal scrolling between splits.
SplitSize	Gets or sets the size of a split.
SplitSizeMode	Gets or sets a value indicating how the **SplitSize** property is used to determine the actual size of a split.
VerticalScrollGroup	Gets or sets the group which synchronizes vertical scrolling between splits.

Split Matrix Notation

When the list contains both horizontal and vertical splits, it is said to be organized in a two-dimensional split matrix. Reference and access to the properties of the split objects in the matrix is accomplished with a two-dimensional matrix notation. The index for a particular split in a split matrix is the split row, then the column delimited by a comma. For instance, accessing the second vertical split (column) in the third horizontal split (row) would look like the following:

To write code in Visual Basic

Visual Basic

```
Me.C1List1.Splits.Item(3, 2).Style.ForeColor = System.Drawing.Color.Blue
```

To write code in C#

```
C#
this.c1List1.Splits[3, 2].Style.ForeColor = System.Drawing.Color.Blue;
```

Note: Notice that the Item property is used in the previous example. When accessing a split through split matrix notation, the item property must be explicitly specified. When accessing a split in a list with a one-dimensional structure, the Item property is taken as implicit and can be omitted.

For instance, accessing the second split in a list with only horizontal splits would look like the following:

To write code in Visual Basic

```
Visual Basic
Me.C1List1.Splits(2).Style.ForeColor = System.Drawing.Color.Blue
```

To write code in C#

```
C#
this.c1List1.Splits[2].Style.ForeColor = System.Drawing.Color.Blue;
```

Creating and Removing Splits

In code, you must create and remove splits using the RemoveHorizontalSplit, InsertHorizontalSplit, RemoveVerticalSplit, and InsertVerticalSplit methods. Each method takes a zero-based split index:

To write code in Visual Basic

```
Visual Basic
' Create a split with index 7.
Me.C1List1.InsertVerticalSplit(7)

' Remove the split with index 5.
Me.C1List1.RemoveVerticalSplit(5)
```

To write code in C#

```
C#
// Create a split with index 7.
this.c1List1.InsertVerticalSplit[7];

// Remove the split with index 5.
this.c1List1.RemoveVerticalSplit[5];
```

You can determine the number of splits in a list using the SplitCollection **Count** property:

To write code in Visual Basic

Visual Basic

```
' Set variable equal to the number of splits in C1List1.
variable = Me.C1List1.Splits.Count
```

To write code in C#

C#

```
// Set variable equal to the number of splits in C1List1.
variable = this.c1List1.Splits.Count;
```

You can iterate through all splits using the **Count** property, for example:

To write code in Visual Basic

Visual Basic

```
For n = 0 To Me.C1List1.Splits.Count - 1
    Debug.WriteLine (Me.C1List1.Splits(n).Caption)
Next n
```

To write code in C#

C#

```
for ( n = 0; n <= this.C1List1.Splits.Count   1; i++)
{
    Console.WriteLine (this.C1List1.Splits[n].Caption);
}
```

Of course, a more efficient way to code this would be to use a For Each...Next loop:

To write code in Visual Basic

Visual Basic

```
Dim S As C1List.Split
For Each S In Me.C1List1.Splits
    Debug.WriteLine (S.Caption)
Next S
```

To write code in C#

C#

```
C1List.Split S;
foreach (S in this.C1List1.Splits)
{
    Console.WriteLine (S.Caption);
}
```

The new **Split** object will inherit all of its properties from the last object in the collection.

Working with Columns in Splits

Each split in a C1List control maintains its own collection of columns. The C1DisplayColumnCollection object provides access to both split-specific display properties for columns inside a split. The split-specific properties of the C1DisplayColumnCollection allow for tremendous flexibility in controlling the look and behavior of individual splits. The list is connected to a single data source, so the splits just present different views of the same data. Therefore, the C1DisplayColumnCollection in each split contains the same number of columns and the columns are bound to the same data fields.

However, the values of other C1DisplayColumn object properties, such as Visible, may vary from split to split. These properties are said to be split-specific. For example, a column created in code is not visible by default. Thus, the *LastName* column created in the preceding example is invisible in all splits. The following code makes it visible in the second split:

To write code in Visual Basic

```
Visual Basic
Me.C1List1.Splits(1).DisplayColumns("LastName").Visible = True
```

To write code in C#

```
C#
this.c1List1.Splits[1].DisplayColumns["LastName"].Visible = true;
```

Since Visible is a split-specific property, the *LastName* column remains invisible in other splits.

Sizing and Scaling Splits

List for WinForms gives you full control over the size and scaling of individual splits. You can configure a split to occupy an exact width or height, hold a fixed number of columns or rows, or adjust its size proportionally in relation to other splits. If you are just starting out with **List for WinForms**, you can use splits in a variety of ways without having to master all of the details.

At run time, the actual size of a Split object depends upon its **SplitSize** and **SplitSizeMode** properties. The **SplitSizeMode** property specifies the unit of measurement; the **SplitSize** property specifies the number of units. **List for WinForms** supports three different sizing modes for splits, as determined by the setting of the **SplitSizeMode** property:

Sizing Mode	Description
Scalable	The **SplitSize** indicates the relative size of the split with respect to other scalable splits.
Exact	The **SplitSize** indicates the size of the split in pixels.
NumberofColumns	The **SplitSize** indicates the number of columns displayed in the split.

A scalable split uses the value of its **SplitSize** property to determine the percentage of space the split will occupy. For any scalable split, the percentage is determined by dividing its **SplitSize** value by the sum of the **SplitSize** values of all other scalable splits. Thus, you can consider the **SplitSize** property of a scalable split to be the numerator of a fraction, the denominator of which is the sum of the scalable split sizes. Scalable splits compete for the space remaining after nonscalable splits have been allocated. By default, all splits are scalable, so they compete for the entire list display region. **SplitSizeMode** is always Scalable when a list contains only one split.

An exact split uses the value of its **SplitSize** property as its fixed width in container coordinates. Exact splits will be truncated if they will not fit within the horizontal list boundaries. This mode is not applicable when a list contains only

one split.

A fixed-column split uses the **SplitSize** property to indicate the exact number of columns that should always be displayed within the split. These splits automatically reconfigure the entire list if the size of the displayed columns changes (either by code or user interaction), or if columns in the split are scrolled horizontally so that the widths of the displayed columns are different. This mode is primarily used to create fixed columns that do not scroll horizontally. However, it can be used for a variety of other purposes as well. This mode is not applicable when a list contains only one split.

Note that when there is only one split (the list's default behavior), the split spans the entire width of the list, the **SplitSizeMode** property is always Scalable, and the **SplitSize** property is always 1. Setting either of these properties has no effect when there is only one split. If there are multiple splits, and you then remove all but one, the **SplitSizeMode** and **SplitSize** properties of the remaining split automatically revert to 0 and 1, respectively.

By default, the **SplitSizeMode** property for a newly created split is **Scalable**, and the **SplitSize** property is set to **1**. For example, if you create two additional splits with the following code:

To write code in Visual Basic

Visual Basic

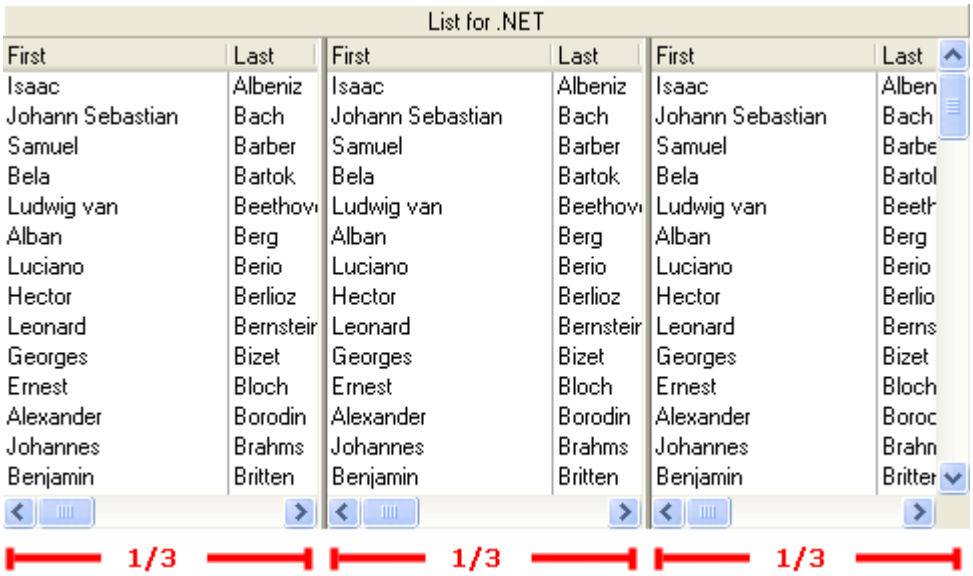
```
' Create a Split on the left.
Me.C1List1.InsertHorizontalSplit(0)

' Create another Split on the left.
Me.C1List1.InsertHorizontalSplit(0)
```

To write code in C#

C#

```
// Create a Split on the left.
this.c1List1.InsertHorizontalSplit(0);

// Create another Split on the left.
this.c1List1.InsertHorizontalSplit(0);
```

The resulting list display will look like this:



Notice that each split occupies 1/3 of the total list space. This is because there are three scalable splits, and each split

has a **SplitSize** of 1. If you change the sizes of the splits to 1, 2, and 3, respectively:

To write code in Visual Basic

Visual Basic

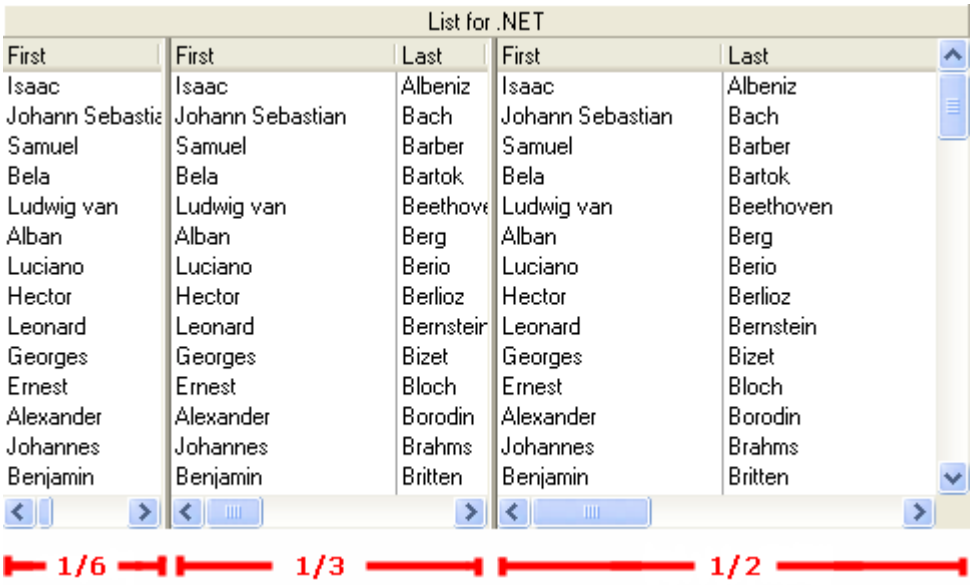
```
' Change relative size to 1.
Me.C1List1.Splits(0).SplitSize = 1

' Change relative size to 2.
Me.C1List1.Splits(1).SplitSize = 2

' Change relative size to 3.
Me.C1List1.Splits(2).SplitSize = 3
```

To write code in C#

C#

```
// Change relative size to 1.
this.c1List1.Splits[0].SplitSize = 1

// Change relative size to 2.
this.c1List1.Splits[1].SplitSize = 2

// Change relative size to 3.
this.c1List1.Splits[2].SplitSize = 3
```

The resulting list display will look like this:



Notice the sum of the split sizes (1+2+3) is 6, so the size of each split is a fraction with the numerator being the value of its **SplitSize** property and a denominator of 6.

When a split's **SplitSizeMode** is set to **Exact**, that split receives space before the other splits. This behavior is somewhat more complex, but understanding how scalable splits work is helpful. For example, assume that splits are set in the following way:

To write code in Visual Basic

Visual Basic

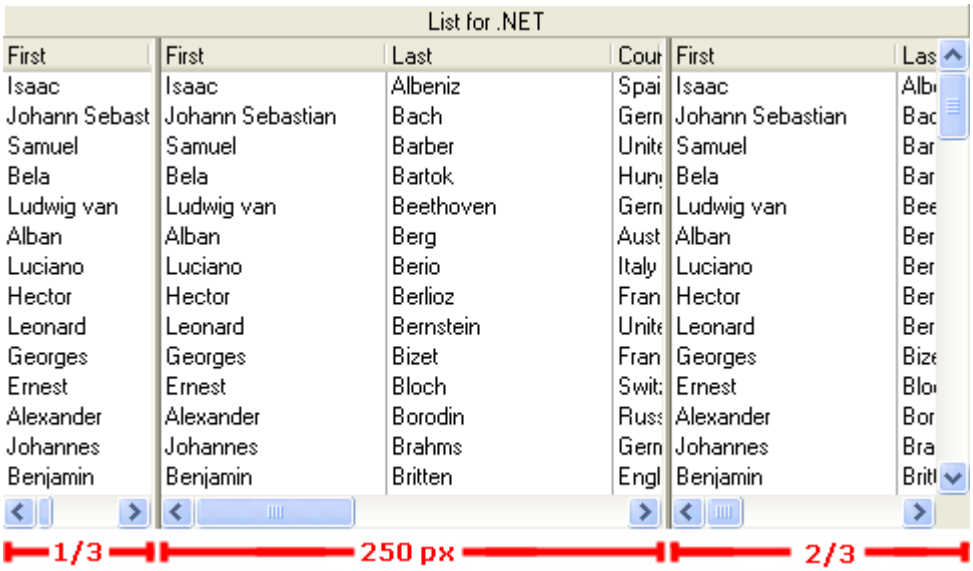
```
Me.C1List1.Splits(0).SplitSizeMode = C1.Win.C1List.SizeModeEnum.Scalable
Me.C1List1.Splits(0).SplitSize = 1
Me.C1List1.Splits(1).SplitSizeMode = C1.Win.C1List.SizeModeEnum.Exact
Me.C1List1.Splits(1).SplitSize = 250
Me.C1List1.Splits(2).SplitSizeMode = C1.Win.C1List.SizeModeEnum.Scalable
Me.C1List1.Splits(2).SplitSize = 2
```

To write code in C#

C#

```
this.c1List1.Splits[0].SplitSizeMode = C1.Win.C1List.SizeModeEnum.Scalable;
this.c1List1.Splits[0].SplitSize = 1;
this.c1List1.Splits[1].SplitSizeMode = C1.Win.C1List.SizeModeEnum.Exact;
this.c1List1.Splits[1].SplitSize = 250;
this.c1List1.Splits[2].SplitSizeMode = C1.Win.C1List.SizeModeEnum.Scalable;
this.c1List1.Splits[2].SplitSize = 2;
```

After configuring the splits in this way, the resulting list display will look like this:



The fixed-size split in the middle (Split1) is configured to exactly 250 pixels, and the remaining splits compete for the space remaining in the list. Since the remaining splits are both scalable splits, they divide the remaining space among themselves according to the percentages calculated using their **SplitSize** property values. So, the leftmost split occupies 1/3 of the **remaining** space, and the rightmost split occupies 2/3.

Splits with **SplitSizeMode** set to **NumberofColumns** behave almost identically to exact splits, except their size is determined by the width of an integral number of columns. The width, however, is dynamic, so resizing the columns or scrolling so that different columns are in view will cause the entire list to reconfigure itself.

Avoid creating a list with no scalable splits. Although **List for WinForms** handles this situation, it is difficult to work with a list configured in this way. For example, if no splits are scalable, all splits will have an exact size, which may not fill the entire horizontal width of the list. If the total width of the splits is too short, **List for WinForms** displays a "null-zone" where there are no splits. If the total width of the splits is wider than the list, then **List for WinForms** will show only the separator lines for the splits that cannot be shown.

Creating and Resizing Splits through User Interaction

You can always create and resize splits in code. However, you can also let your users create and resize splits interactively by setting the AllowHorizontalSplit or AllowVerticalSplit property of a split to **True**. By default, both properties are set to **False**, and users are prevented from creating and resizing splits.

A typical list with these properties set to **False** is shown in the following figure. Notice that there is no split box at the left edge of the horizontal scroll bar or at the top of the vertical scroll bar.

If you set the split's AllowVerticalSplit property to **True**:

To write code in Visual Basic

```
Visual Basic
Me.C1List1.AllowVerticalSplit = True
```

To write code in C#

```
C#
this.c1List1.AllowVerticalSplit = true;
```

A split box will appear at the top edge of the vertical scroll bar, and the user will be able to create new splits at run time.

If you set the split's C1ListBase.AllowHorizontalSplitproperty to **True**:

To write code in Visual Basic

```
Visual Basic
Me.C1List1.AllowHorizontalSplit = True
```

To write code in C#

```
C#
```

```
this.c1List1.AllowHorizontalSplit = true;
```

A split box will appear at the left edge of the horizontal scroll bar, and the user will be able to create new splits at run time.

The new split will inherit its properties from the original split. The **SplitSizeMode** properties of both splits will be automatically set to 0 - Scalable, regardless of the **SplitSizeMode** of the original split. The **SplitSize** properties of both splits will be set to the correct ratio of the splits' sizes. The values of the **SplitSize** properties may end up being rather large. This is because **List for WinForms** needs to choose the least common denominator for the total split size, and the user may drag the pointer to an arbitrary position.

If the user points to the split box between the two splits, the pointer will turn into a double vertical bar with horizontal arrows. The user can drag this pointer to the left or right to adjust the relative sizes of the splits.

To summarize:

- You can always create or resize splits in code, but the AllowHorizontalSplit and AllowVerticalSplit properties control whether users can create or resize splits interactively at run time.
- The user can resize the relative sizes of two splits only if both splits AllowSizing properties are **True**. When the user completes a resize operation, the total size of the two splits remains unchanged, but the **SplitSizeMode** properties of both splits will automatically be set to Scalable regardless of their previous settings. The **SplitSize** properties of the two splits will be set to reflect the ratio of their new sizes.

Vertical Scrolling and Split Groups

By default, the list has only one horizontal split, with split index 0, and its HScrollBar and VScrollBar style properties are set to ScrollBarStyleEnum.Automatic. That is, the horizontal or vertical scroll bar will appear as necessary depending upon the column widths and the number of data rows available. The default split's **HorizontalScrollGroup** and **VerticalScrollGroup** properties are set to **1**. Splits having the same scrollgroup property setting will scroll vertically or horizontally together. When a new split is created, it will inherit both the state of the scroll bars and the Scroll Group properties from the parent split. If all of the splits belonging to the same **HorizontalScrollGroup** or **VerticalScrollGroup** have their HScrollBar and VScrollBar style properties set to Automatic, then C1List will display the vertical scroll bar or horizontal scroll bar only at the rightmost or bottomost split of the scroll group. Manipulating this single scroll bar will cause all splits in the same scroll group to scroll simultaneously.

For example, if you create two additional splits with the following code:

To write code in Visual Basic

Visual Basic

```
'Create a Split on the left.
Me.C1List1.InsertHorizontalSplit(0)
' Create another Split on the left.
Me.C1List1.InsertHorizontalSplit(0)
```

To write code in C#

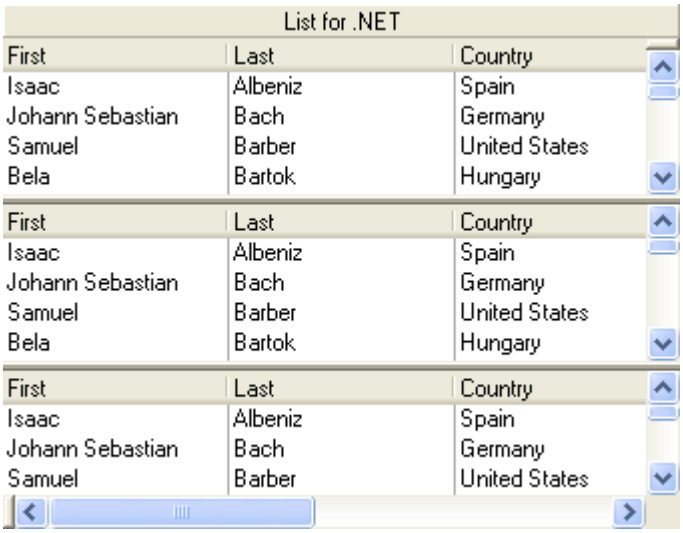
C#

```
// Create a Split on the left.
this.c1List1.InsertHorizontalSplit[0];
// Create another Split on the left.
this.c1List1.InsertHorizontalSplit[0];
```

The resulting list display will look like this:

All three splits will have the same HScrollBar and VScrollBar settings and a **VerticalScrollGroup** setting of 1. However, only one vertical scroll bar will be displayed, within the rightmost split. When the user operates this scroll bar, all three splits will scroll simultaneously.

Vertical splits react in the same manner. After adding two vertical splits to the list, all of the splits have the same **HorizontalScrollGroup** value of 1. Thus there is only one horizontal scroll bar at the bottom of the list, and if this scroll bar is scrolled, all three splits will scroll simultaneously.



You can change one of the scroll group properties of the splits to create split groups that scroll independently. In the preceding example, setting the **HorizontalScrollGroup** property of the middle split to 2 creates a new scroll group:

To write code in Visual Basic

Visual Basic

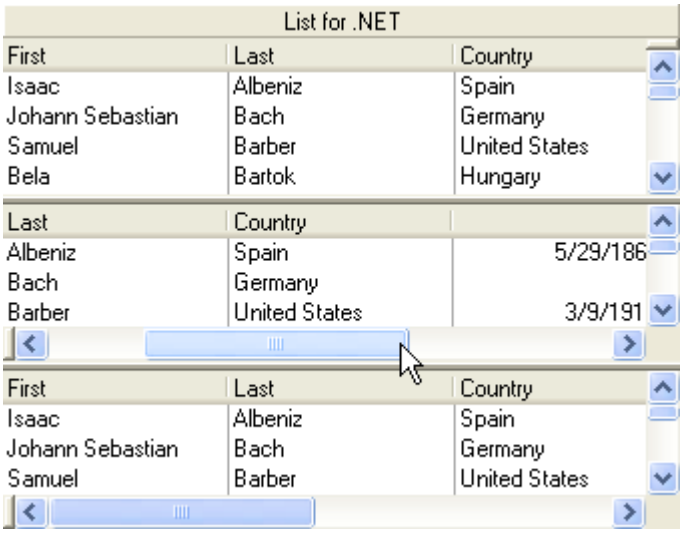
```
Me.C1List1.Splits.Item(0, 1).HorizontalScrollGroup = 2
```

To write code in C#

C#

```
this.c1List1.Splits.Item[0, 1].HorizontalScrollGroup = 2;
```

After this statement executes, scrolling the middle split will not disturb the others, as shown in the following figure:



Note that the middle split now contains a horizontal scroll bar. This scroll bar operates only on the middle split, since it is the only one with its **HorizontalScrollGroup** property equal to **2**. The horizontal scroll bar in the bottom most split now controls the bottom and top splits only. It no longer affects the middle split.

A common application of this feature is to create two independent split groups so that users can compare field values from different records by scrolling each split to view a different set of rows.

Horizontal Scrolling and Fixed Columns

Scrolling is independent for each split. Often, you need to prevent one or more columns from scrolling horizontally or vertically so that they will always be in view. **List for WinForms** provides you with an easy way to keep any number of columns from scrolling at any location within the list (even in the middle!) by setting a few split properties.

As an example, assume that you have a list with three horizontal splits. The following code will "fix" columns 0 and 1 in the middle split:

To write code in Visual Basic

Visual Basic

```
' Hide all columns in Splits(1) except for columns 0 and 1.
Dim Cols As C1.Win.C1List.C1DisplayColumnCollection
Dim C As C1.Win.C1List.C1DisplayColumn
Cols = Me.C1List1.Splits(1).DisplayColumns
For Each C In Cols
    C.Visible = False
Next C
Cols(0).Visible = True
Cols(1).Visible = True

' Configure Splits(1) to display exactly two columns, and disable resizing.
Me.C1List1.Splits(1).SplitSizeMode = C1.Win.C1List.SizeModeEnum.NumberOfColumns
Me.C1List1.Splits(1).SplitSize = 2
```

To write code in C#

C#

```
// Hide all columns in Splits[1] except for columns 0 and 1.
C1.Win.C1List.C1DisplayColumnCollection Cols;
C1.Win.C1List.C1DisplayColumn C;
Cols = this.C1List1.Splits[1].DisplayColumns;
foreach (C in  Cols)
{
    C.Visible = false;
}
Cols[0].Visible = true;
Cols[1].Visible = true;

// Configure Splits[1] to display exactly two columns, and disable resizing
this.C1List1.Splits[1].SplitSizeMode = C1.Win.C1List.SizeModeEnum.NumberOfColumns;
this.C1List1.Splits[1].SplitSize = 2;
```

Usually, if you keep columns 0 and 1 from scrolling in one split, you will want to make them invisible in the other splits:

To write code in Visual Basic

Visual Basic

```
' Make columns 0 and 1 invisible in splits 0 and 2.
Dim Cols As C1.Win.C1List.C1DisplayColumnCollection
Cols = Me.C1List1.Splits(0).DisplayColumns
Cols(0).Visible = False
Cols(1).Visible = False
Cols = Me.C1List1.Splits(2).DisplayColumns
Cols(0).Visible = False
Cols(1).Visible = False
```

To write code in C#

C#

```
// Make columns 0 and 1 invisible in splits 0 and 2.
C1.Win.C1List.C1DisplayColumnCollection Cols;
Cols = this.c1List1.Splits[0].DisplayColumns;
Cols[0].Visible = false;
Cols[1].Visible = false;
Cols = this.c1List1.Splits[2].DisplayColumns;
Cols[0].Visible = false;
Cols[1].Visible = false;
```

List for WinForms Tutorials

Twenty-one tutorials are presented in this section. The tutorials assume that you are familiar with programming in Visual Studio, know what a DataSet is, and know how to use bound controls in general. The tutorials provide step-by-step instructions and no prior knowledge of **List for WinForms** is needed. By following the steps outlined in this chapter, you will be able to create projects demonstrating a variety ofList for WinForms features, and get a good sense of what List for WinForms can do and how to do it.

The tutorials use an Access database, **C1NWind.mdb**. The database files **C1NWind.mdb** and the tutorial projects are in the **Tutorials** subdirectory of the **List for WinForms** installation directory.

We encourage you to run the tutorial projects in Visual Studio, examine the code, and experiment with your own modifications. This is the best and quickest way to realize the full potential of List for WinForms. You will find that List for WinForms is very easy to use, and it enables you to create powerful database applications.

The tutorials assume that the database file **C1NWind.mdb** is in the **\common** directory that was added, by default, during the **WinForms Edition** installation (**Documents\ComponentOne Samples\Common**), and refer to it by filename instead of the full pathname for the sake of brevity.

> **Note:** Depending on where you store the projects and database files, you may need to change the location of the **C1NWind.mdb** reference in the DataSet. For more information, see Modifying the DataSource.

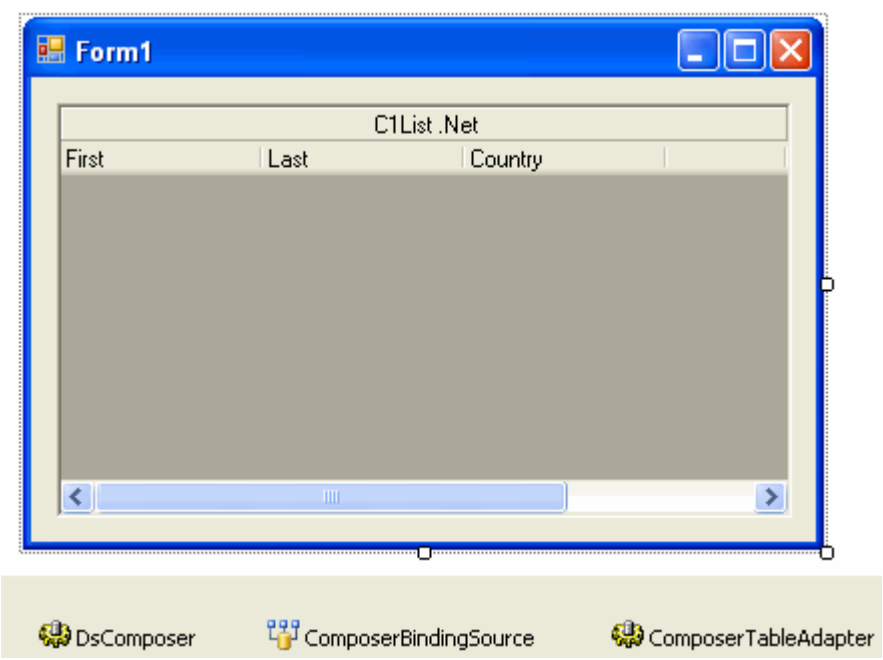
Tutorial 1 - Binding C1List to a DataSet

In this tutorial, you will learn how to bind C1List to a DataSet and create a fully functional database browser. You will also learn about the basic properties of the C1List control. You will then be able to run the program and observe the run-time features of the list.

1. Create a new .NET project.
2. Double-click the **C1List** icon in Visual Studio's Toolbox to add it to the Form.

   > **Note: See Adding ComponentOne Components to a Project for information on adding a component to the Toolbox.**

3. Go to the **DataSource** property and select **Add Project Data Source** from the drop-down. In the adapter's **Data Source Configuration Wizard**, either select a connection to **C1NWind.mdb** or create a new connection to this database. On the **Choose your database objects** page of the wizard, select all fields in the **Composer** table and type "DsComposer" into the **DataSet name** box, and then finish out the wizard.
   At this point, your .NET design window and form should look like the following:



4. Visual Studio adds the following code to the **Form_Load** event:

To write code in Visual Basic

```
Visual Basic
Me.ComposerTableAdapter.Fill(Me.DsComposer.Composer)
```

To write code in C#

```
C#
this.ComposerTableAdapter.Fill(this.DsComposer.Composer);
```

Run the program and observe the following:

- C1List retrieves the database schema information from the DataSet and automatically configures itself to display all of the fields contained in the database table. Note that the field names are used as the default column headings.
- C1List automatically communicates with the DataSet. Any actions taken on the DataSet will be reflected in the list.
- You have created a fully functional database browser by only writing a one line of simple code.

Refer to Run-Time Interaction and try the instructions for navigating, editing, and configuring the list at run time.

To end the program, close the window or press the stop button on the .NET toolbar.

This concludes the tutorial.

Tutorial 2 - Binding C1Combo to a DataSet

In this tutorial, you will learn how to bind **C1Combo** to a DataSet. You will also learn about the basic properties of the C1Combo control. You will then be able to run the program and observe the run-time features of the combo.

1. Create a new .NET 2.0 project.

2. Double-click the **C1Combo** icon in Visual Studio's Toolbox to add it to the Form. **Note:** See Adding the C1List Components to a Project for information on adding a component to the Toolbox.
3. Go to the **DataSource** property and select **Add Project Data Source** from the drop-down. In the adapter's **Data Source Configuration Wizard**, either select a connection to **C1NWind.mdb** or create a new connection to this database. On the Choose your database objects page of the wizard, select all fields in the **Composer** table and type "DataSet1" into the **DataSet name** box, and then finish out the wizard.
   At this point your .NET design window and form should look like the following:

4. Visual Studio 2005 adds the following code to the **Form_Load** event:

To write code in Visual Basic

```
Visual Basic
Me.ComposerTableAdapter.Fill(Me.DataSet1.Composer)
```

To write code in C#

```
C#
this.ComposerTableAdapter.Fill(this.dataSet1.Composer);
```

Run the program and observe the following:

Note: When C1Combo has the focus, press F4 to open the drop-down box.

To end the program, close the window or press the stop button on the toolbar.

This concludes the tutorial.

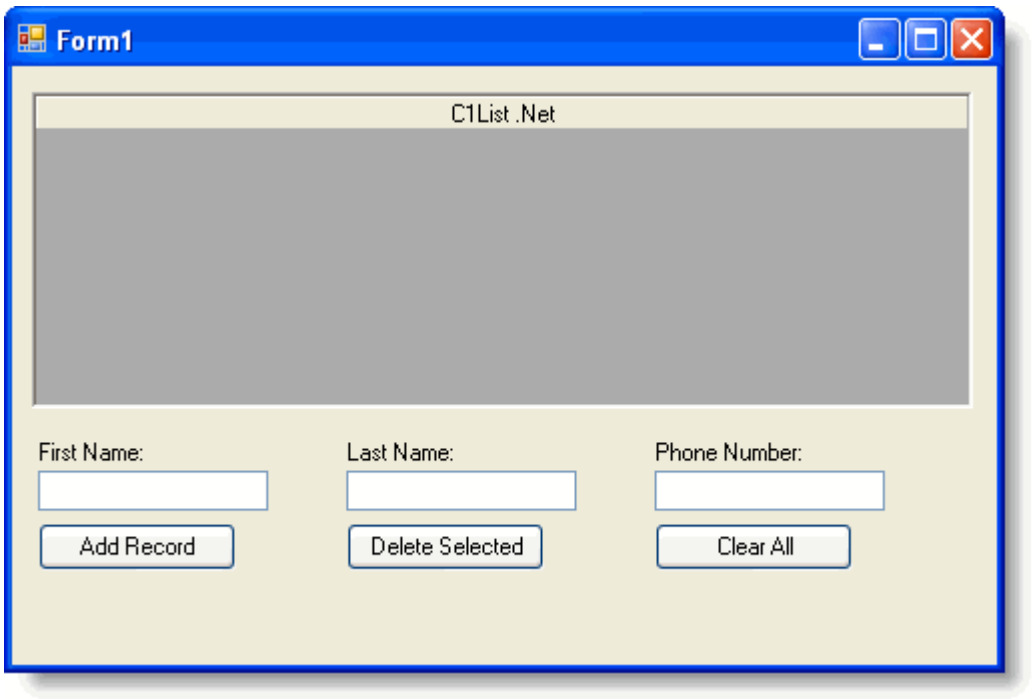
Tutorial 3 AddItem Mode

This tutorial demonstrates how you can manually add information to a list using the AddItem Mode without having to bind C1List to a DataSet.

1. Create a new .NET project and add:

   - One C1List control (C1List1)
   - Three text boxes (TextBox1, 2, 3)
   - Three labels and three buttons (Button1, 2, 3)

   The form should look like the following:



2. Set the C1ListBase.DataMode property for the C1List control to **AddItem**.
3. In the **Form_Load** event, add the following code:

   To write code in Visual Basic

```
Visual Basic
With Me.C1List1
   .AddItemTitles("First Name; LastName; Phone Number")
   .AddItem("Greg;Daryll;412-657-3412")
   .AddItem("Jane;Lambert;567-123-6785")
   .AddItem("Allen;Clark;675-345-9087")
   .AddItem("David;Elkins;564-345-2635")
   .AddItem("Carl;Ziegler;412-678-2351")
   .AddItem("William;Yahner;412-980-6754")
End With
```

   To write code in C#

```
C#
this.c1List1.AddItemTitles("First Name; LastName; Phone Number");
this.c1List1.AddItem("Greg;Daryll;412-657-3412");
this.c1List1.AddItem("Jane;Lambert;567-123-6785");
```

```
this.c1List1.AddItem("Allen;Clark;675-345-9087");
this.c1List1.AddItem("David;Elkins;564-345-2635");
this.c1List1.AddItem("Carl;Ziegler;412-678-2351");
this.c1List1.AddItem("William;Yahner;412-980-6754");
```

4. Add the following code to the **Button1_Click** event to allow the user to add a record:

To write code in Visual Basic

Visual Basic

```
Me.C1List1.AddItem(Me.TextBox1.Text + ";" + Me.TextBox2.Text + ";" +
Me.TextBox3.Text)
```

To write code in C#

C#

```
this.c1List1.AddItem(this.TextBox1.Text + ";" + this.TextBox2.Text + ";" +
this.TextBox3.Text);
```

5. Add the following code to the **Button2_Click** event to allow the user to delete a record:

To write code in Visual Basic

Visual Basic

```
Try
    Me.C1List1.RemoveItem(Me.C1List1.SelectedIndex)
Catch
    MessageBox.Show("Select an item from the list first.", "List")
End Try
```

To write code in C#

C#

```
try
{
    this.c1List1.RemoveItem(this.C1List1.SelectedIndex);
}
catch
{
    MessageBox.Show("Select an item from the list first.", "List");
}
```

6. Add the following to the **Button3_Click** event to allow the user to clear the list:

To write code in Visual Basic

Visual Basic

```
Me.C1List1.ClearItems()
```

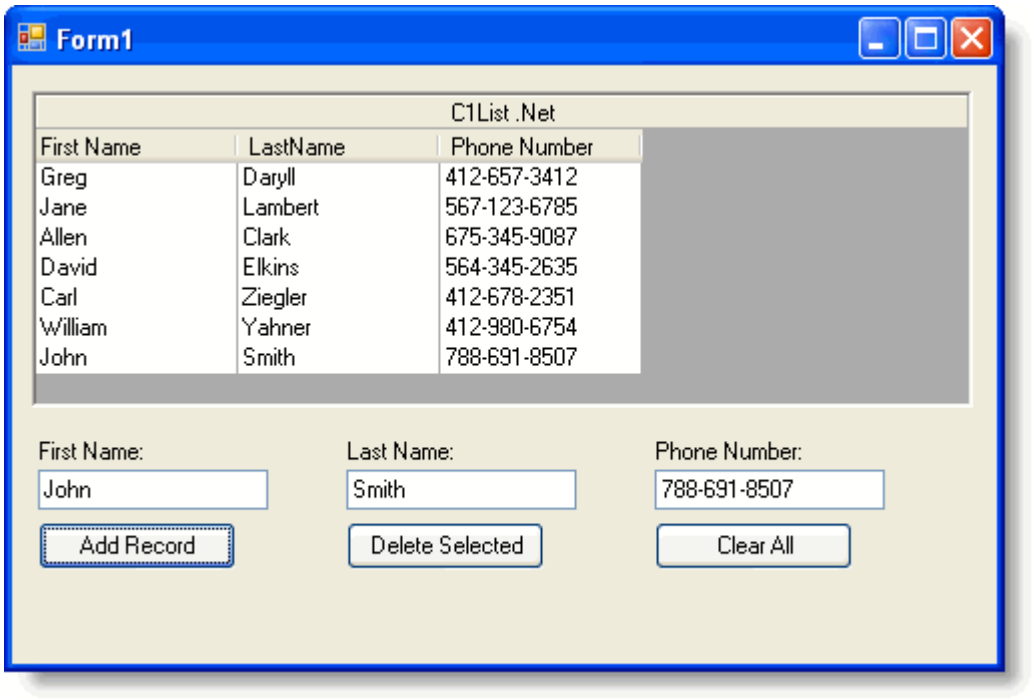
To write code in C#

C#

```
this.c1List1.ClearItems();
```

Run the program and observe the following:

- When you enter information into the text boxes and click the **Add Record** button, the information you entered is added to the list.



- Selecting a row and clicking **Delete Selected** will delete the whole row, while clicking **Clear All** will delete the entire list.

> **Note: For faster data processing, you can use the C1ListBase.AddItem method.**

For example, replace the code in the **Form_Load** event with the following code:

To write code in Visual Basic

Visual Basic

```
With Me.C1List1
   .AddItemTitles("First Name; LastName; Phone Number")
   .SuspendBinding()
   .AddItem("Greg;Daryll;412-657-3412")
   .AddItem("Jane;Lambert;567-123-6785")
   .AddItem("Allen;Clark;675-345-9087")
   .AddItem("David;Elkins;564-345-2635")
   .AddItem("Carl;Ziegler;412-678-2351")
   .AddItem("William;Yahner;412-980-6754")
   .ResumeBinding()
End With
```

To write code in C#

C#

```
this.C1List1.AddItemTitles("First Name; LastName; Phone Number");
this.C1List1.SuspendBinding();
this.C1List1.AddItem("Greg;Daryll;412-657-3412");
this.C1List1.AddItem("Jane;Lambert;567-123-6785");
this.C1List1.AddItem("Allen;Clark;675-345-9087");
this.C1List1.AddItem("David;Elkins;564-345-2635");
this.C1List1.AddItem("Carl;Ziegler;412-678-2351");
this.C1List1.AddItem("William;Yahner;412-980-6754");
this.C1List1.ResumeBinding();
```

In this tutorial only a small amount of data was used, so you may not notice a difference in the processing time. For larger amounts of data, use the C1ListBase.AddItem method for a much faster processing time.

This concludes the tutorial.

Tutorial 4 - Displaying Master-Detail Relationships

This tutorial demonstrates how to link multiple C1List and C1Combo controls using the Change event to trigger related actions. This technique is particularly useful for displaying master-detail relationships.

1. Create a new .NET 2.0 project.
2. Place a C1Combo control (**dbcComp**), and a C1List control (**dblOpus**) on the form (**Form1**) as shown in this figure.

3. Go to the **DataSource** property for **dbcComp** and select **Add Project Data Source** from the drop-down. In the adapter's **Data Source Configuration Wizard**, either select a connection to **C1NWind.mdb** or create a new connection to this database. On the **Choose your database objects** page of the wizard, select the *Last* field in the **Composer** table and type "DataSet1" into the **DataSet name** box, and then finish out the wizard.
4. Go to the **DataSource** property for **dblOpus** and select **Add Project Data Source** from the drop-down. In the adapter's **Data Source Configuration Wizard**, either select a connection to C1NWind.mdb or create a new connection to this database. On the Choose your database objects page of the wizard, select all fields in the **Opus** table and type "DataSet2" into the **DataSet name** box, and then finish out the wizard.
5. Set the DisplayMember property of **dbcComp** to **None**.
6. In the **Form_Load** event, add the following code:

To write code in Visual Basic

```
Visual Basic
 ' Visual Studio adds these two lines of code to load the data.
 Me.ComposerTableAdapter.Fill(Me.DataSet1.Composer)
 Me.OpusTableAdapter.Fill(Me.DataSet2.Opus)
 Me.dbcComp_RowChange(Nothing, e.Empty)
```

To write code in C#

```
C#
  // Visual Studio adds these two lines of code to load the data.
 this.ComposerTableAdapter.Fill(this.DataSet1.Composer);
 this.OpusTableAdapater.Fill(this.DataSet2.Opus);
 this.dbcComp.RowChange += new EventHandler(this.dbcComp_RowChange);
```

7. Add the following code to the RowChange event of **dbcComp**:

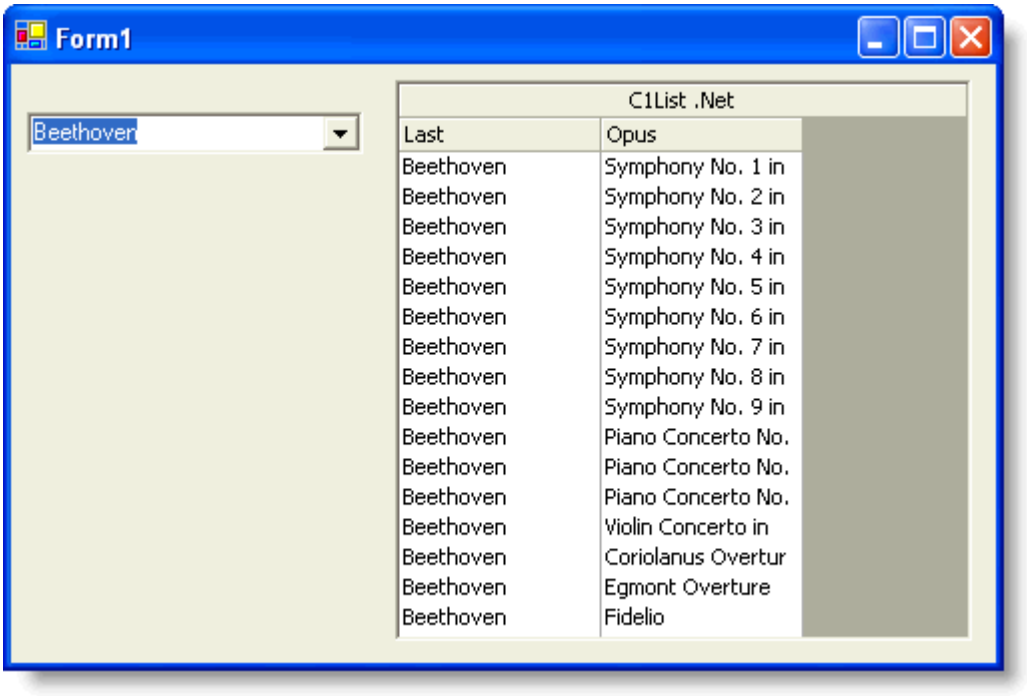
To write code in Visual Basic

```
Visual Basic
 Private Sub dbcComp_RowChange(ByVal sender As Object, ByVal e As
System.EventArgs) Handles dbcComp.RowChange
     Dim dr As DataRow()
     dr = Me.DataSet2.Opus.Select("Last ='" & Me.dbcComp.Text + "'")
     Dim tb As New DataSet2.OpusDataTable()
     tb = New DataSet2.OpusDataTable()
     Dim item As DataRow
     For Each item In dr
         tb.ImportRow(item)
     Next item
     Me.dblOpus.DataSource = tb
 End Sub
```

To write code in C#

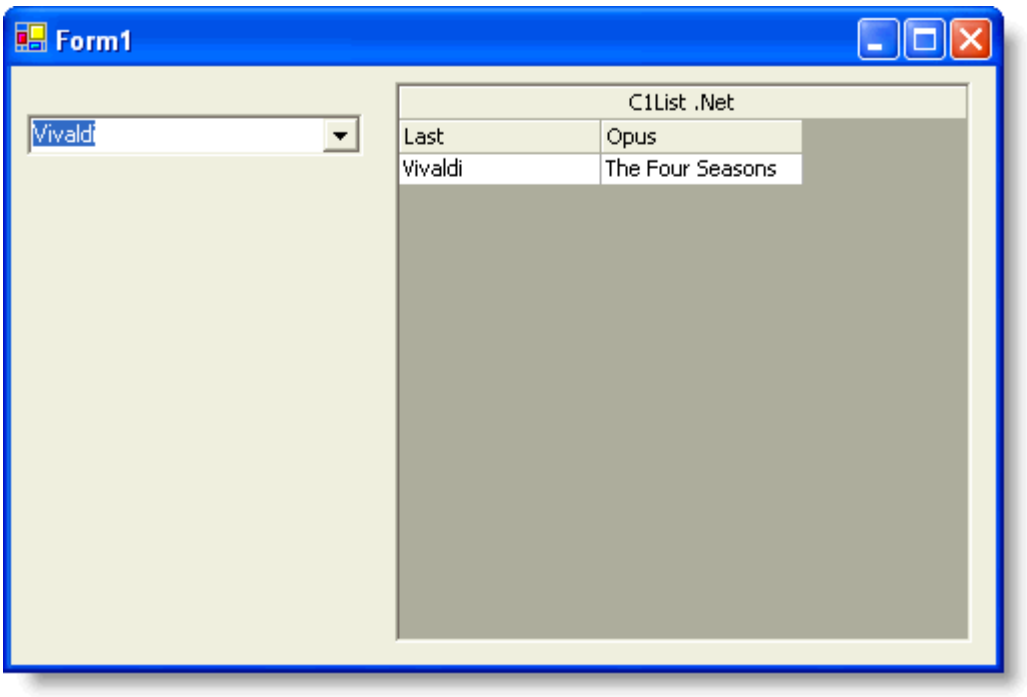
```
C#
 private void dbcComp_RowChange(object sender, System.EventArgs e)
 {
     DataRow[] dr = this.dataSet2.Opus.Select("Last ='" + this.dbcComp.Text +
"'");
     DataSet2.OpusDataTable tb = new DataSet2.OpusDataTable();
     foreach (DataRow item in dr)
     {
         tb.ImportRow(item);
     }
     this.dblOpus.DataSource = tb;
 }
```

Run the program and observe the following:



Change the current record position of **dbcComp** by selecting a different composer. Observe that **dblOpus** (the detail list) displays a new set of compositions each time the row changes in C1Combo1 (the master list).

Note: When C1Combo has the focus, press F4 to open the drop-down box.



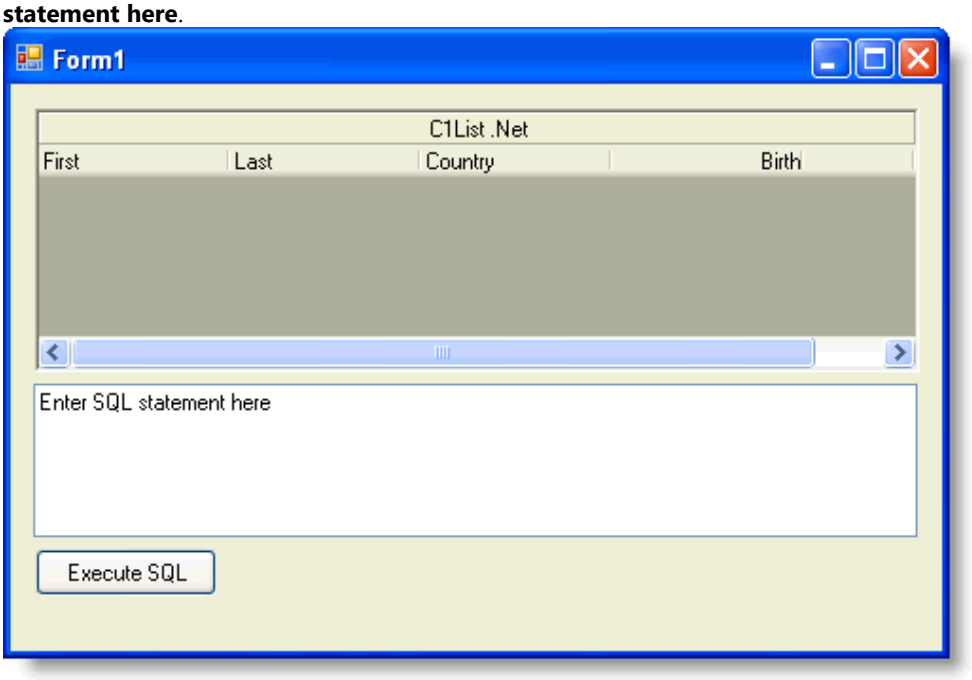
To end the program, press the **End** button on the toolbar. This concludes the tutorial.

Tutorial 5 - Using C1List with SQL Query Results

In this tutorial, you will learn how to use C1List to display the results of ad-hoc SQL queries. In addition, it will also outline how to set up a connection to a DataSet at run-time. Note that in order for the list to automatically respond to field layout

changes, you must not have defined any column properties at design time. If a layout is already defined, use the list's **Clear Fields** context menu command to remove it. This will cause the list to configure itself automatically at run time.

1. Follow steps 1 through 4 of Tutorial 1 - Binding C1List to a DataSet to create a project with a C1List control bound to a Data Set.
2. Set the DataMember property of the list to **Composer**.
3. Place a C1List control, a Command button and a Text box on the form as shown in the figure. Rename the text property of the command button to read **Execute SQL** and set the **Text** property of the TextBox to **Enter SQL statement here**.



4. Add the following code to the **Button1_Click** event:

To write code in Visual Basic

Visual Basic

```
 Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
     Dim ds As New DataSet()
     Dim sqlStr As String
     Dim da As OleDb.OleDbDataAdapter
     sqlStr = Me.TextBox1.Text.Trim()
     da = New OleDb.OleDbDataAdapter(sqlStr, Me.ComposerTableAdapter.Connection)
     Try
         da.Fill(ds, "TestSQL")
         Me.C1List1.DataSource = Nothing
         Me.C1List1.ClearFields()
         Me.C1List1.DataSource = ds.Tables("TestSQL")
     Catch x As Exception
         MsgBox("Invalid SQL statement.")
     End Try
 End Sub
```

To write code in C#

C#

```
 private void Button1_Click( System.object sender, System.EventArgs e)
```

```
{
    DataSet ds = new DataSet();
    string  sqlStr;
    OleDb.OleDbDataAdapter da;
    sqlStr = this.TextBox1.Text.Trim();
    da = new OleDb.OleDbDataAdapter(sqlStr, this.ComposerTableAdapter.Connection);
    try
    {
        da.Fill(ds, "TestSQL");
        this.c1List1.DataSource = null;
        this.c1List1.ClearFields();
        this.c1List1.DataSource = ds.Tables("TestSQL");
    }
    catch( Exception x)
    {
        MsgBox("Invalid SQL statement.");
    }
}
```

Run the program and observe the following:

As in Tutorial 1 - Binding C1List to a DataSet, C1List retrieves the database schema information from the DataSet and automatically configures itself to display the data for all fields in the database table. Note that the field names are used as the default column headings.

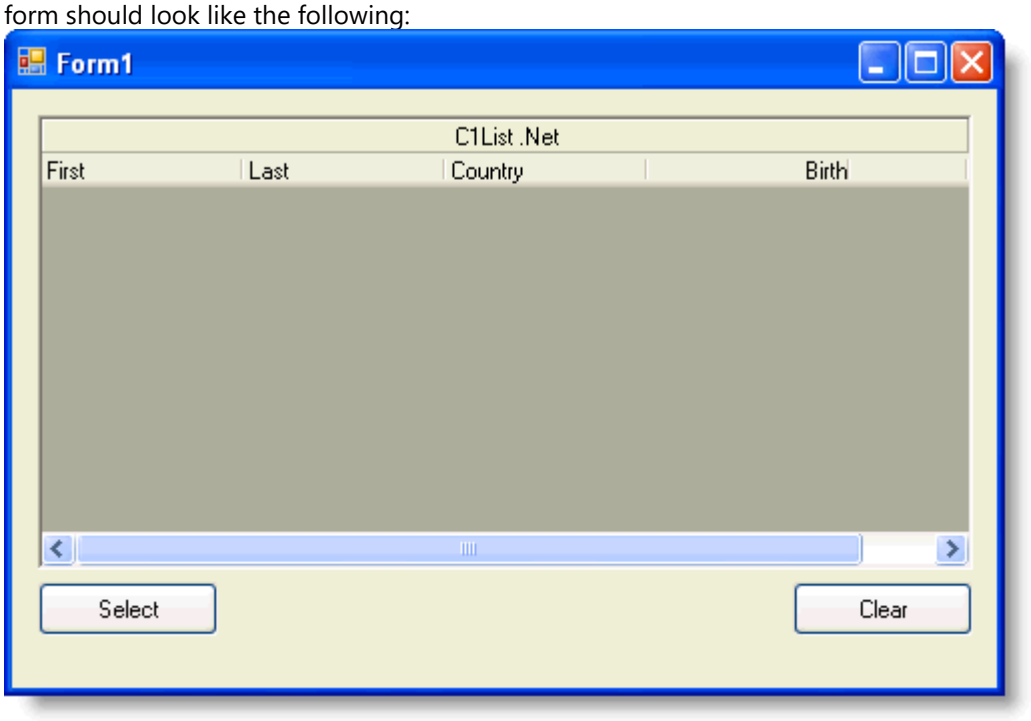
1. In the TextBox control, enter the following SQL statement: `Select * from Composer`
   then press the **Execute SQL** command button. The above SQL statement displays all fields from the **Customer** table and is equivalent to the default display.
2. In the TextBox control, enter the following SQL statement: `Select Country from Composer`
   then click the **Execute SQL** command button. The list responds by displaying only one column for the Country field.
3. In the TextBox control, enter the following SQL statement: `Select Last, Country from Composer`
   then click the **Execute SQL** command button. This is similar to the previous SQL statement except that two columns (Last and Country) are now displayed.
4. In the TextBox control, enter the following SQL statement: `Select Count(*) from Composer`
   then click the **Execute SQL** command button. The above SQL statement uses an aggregate function, **Count(*):SQL**, to return the total number of records in the Customer table. Even though the SQL result is not a set of records, the list faithfully responds by displaying the number of records in a single column. By default, *Expr1000* is used as the column heading, indicating that the display is the result of an expression.
5. In the TextBox control, enter the following SQL statement: `Select UCase(Last) as ULAST, UCase(First) AS UFIRST from Composer`
   then click the **Execute SQL** command button. The above SQL statement produces two calculated columns which display the Last and First fields in upper case. The list also displays the (assigned) calculated column names, ULAST and UFIRST, as the column headings.
6. In the TextBox control, enter the following SQL statement: `SELECT * FROM Composer WHERE First = "Edward"`
   then click the **Execute SQL** command button. The above SQL statement displays only records with *First* equal to **Edward**.
7. `In the TextBox control, enter the following SQL statement: SELECT * FROM Composer ORDER BY Last`
   then press the **Execute SQL** command button. The above SQL statement displays records in alphabetical order according to the Last field.

This concludes the tutorial.

Tutorial 6 - Selecting Multiple Rows Using Bookmarks

In this tutorial, you will learn how to select and highlight records that satisfy specified criteria. A group of similar items is generally implemented as a collection in C1List. When manipulating a group of items in C1List, use techniques similar to those described here.

1. Follow steps 1 through 4 of the Tutorial 1 - Binding C1List to a DataSet to create a project with a C1List control bound to a Data Set.
2. Set Button1's **Text** property to **Select** and set Button2's **Text** property to **Clear**. The .NET design window and form should look like the following:



3. We can easily select and deselect rows in C1List by manipulating the SelectedRowCollection. To select rows, place the following code in the **Button1_Click** event:

To write code in Visual Basic

Visual Basic

```
Dim i As Long
For i = 0 To Me.DsComposer.Composer.Rows.Count - 1
    If Me.DsComposer.Composer.Rows(i).Item("Country") = "Germany" Then
</code><br/> Me.C1List1.SelectedIndices.Add(i)
    End If
Next i
```

To write code in C#

C#

```
long i;
for ( i = 0; i <= this.DsComposer.Composer.Rows.Count - 1; i++)
{
    if ( this.DsComposer.Composer.Rows[i] ["Country"] == "Germany" )
    {
 </code><br/>this.c1List1.SelectedIndices.Add(i);
```

```
    }
}
```

4. 4To deselect rows, place the following code in the **Button2_Click** event:

To write code in Visual Basic

```
Visual Basic
Me.C1List1.ClearSelected()
```

To write code in C#

```
C#
this.c1List1.ClearSelected();
```

Run the program and observe the following:

- C1List1 retrieves the database schema information from the DataSet and automatically configures itself to display all of the fields in the database table. This is again similar to the behavior of the list in Tutorial 1 - Binding C1List to a DataSet.
- Click the **Select** command button and observe that all records with the *Country* field equal to **Germany** will be highlighted.
- To deselect the highlighted records, click the **Clear** command button. Alternatively, clicking anywhere on a list cell will also clear the selected rows.

This concludes the tutorial.

Tutorial 7 - Displaying Unbound Columns in a Bound List

In this tutorial, you will learn how to use the UnboundColumnFetch event to display two fields (First and Last) together in one column.

1. Follow steps 1 through 4 of Tutorial 1 - Binding C1List to a DataSet to create a project with a C1List control bound to a Data Set.
2. Declare a new global DataTable object in Form1:

To write code in Visual Basic

```
Visual Basic
Dim dtCopy As New DataTable()
```

To write code in C#

```
C#
DataTable dtCopy = new DataTable();
```

3. Add the following code to the **Form_Load** event:

To write code in Visual Basic

```
Visual Basic
' Visual Studio 2005 adds this line of code to load the data.
Me.ComposerTableAdapater.Fill(Me.DsComposer.Composer)
```

```
dtCopy = Me.DsComposer1.Tables(0).Copy()
Me.C1List1.Splits(0).DisplayColumns.Item ("Last").Visible = False
Me.C1List1.Splits(0).DisplayColumns.Item ("First").Visible = False
```

To write code in C#

C#

```
// Visual Studio 2005 adds this line of code to load the data.
this.ComposerTableAdapter.Fill(this.DsComposer.Composer);

dtCopy = this.DsComposer1.Tables[0].Copy();
this.c1List1.Splits[0].DisplayColumns["Last"].Visible = false;
this.c1List1.Splits[0].DisplayColumns["First"].Visible = false;
```

Note: The first line fills the dataset, the second line makes a copy of this DataSet, which we will use later to populate the unbound column. The last two lines of code hide the columns with field names *First* and *Last*.

4. To create an unbound column open up the **C1List Designer** by clicking on the **ellipsis** button next to the Columns property in the Properties window. Next click the **Add** button in the left pane to create column, which is added at the bottom of the list. You can maneuver where you would like the column to appear by using dragging the column to where you would like it to appear. Move the new column to the very first column and set its Caption property to **Name** in the left pane. Notice that we have a value in the Caption field, but no value in the Data field. This is how C1List knows it is an unbound column. C1List now knows to fire the UnboundColumnFetch event.
5. Open the **Split Collection Editor** by clicking on the **ellipsis** button next to the Splits property in the Properties window. Open the **C1DisplayColumn Collection Editor** by clicking on the **ellipsis** button next to the **DisplayColumns** property. In this editor, find the unbound column that we just created, and set its Visible property equal to **True**. Now our unbound column is visible to the end-user and not just the C1List control.
6. Add the following code to the UnboundColumnFetch event:

To write code in Visual Basic

Visual Basic

```
 Private Sub C1List1_UnboundColumnFetch(ByVal sender As Object, ByVal e As
C1.Win.C1List.UnboundColumnFetchEventArgs) Handles C1List1.UnboundColumnFetch
     If (e.Col = 0 And e.Row < dtCopy.Rows.Count) Then
         e.Value = dtCopy.Rows(e.Row).Item("First") + " " +
dtCopy.Rows(e.Row).Item("Last")"
     End If
 End Sub
```

To write code in C#

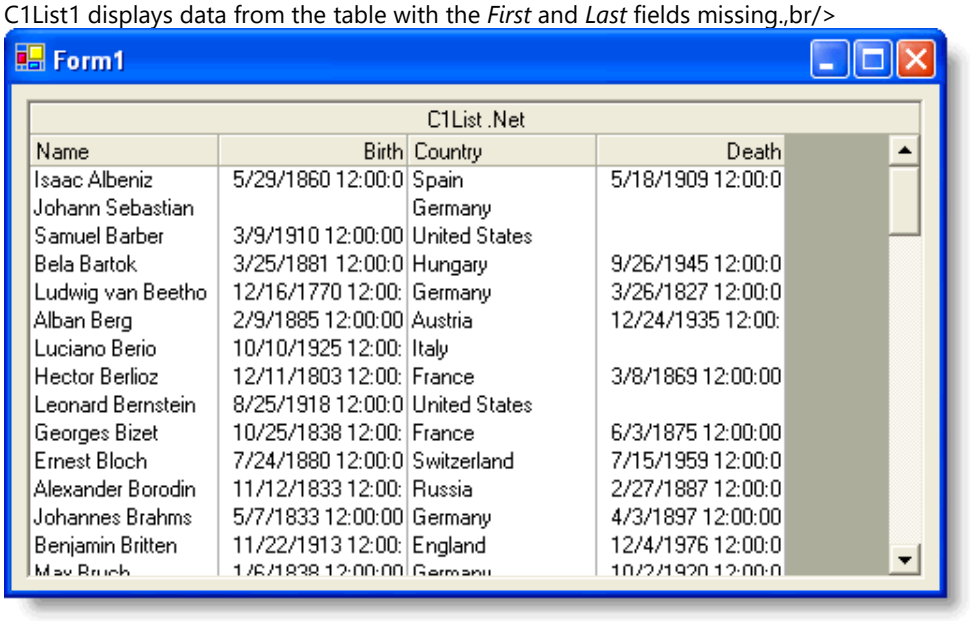
C#

```
 private void C1List1_UnboundColumnFetch( object sender,
C1.Win.C1List.UnboundColumnFetchEventArgs e)
 {
     if (e.Col == 0 && e.Row < dtCopy.Rows.Count)
     {
         e.Value = dtCopy.Rows[e.Row].["First"] + " " + dtCopy.Rows[e.Row].["Last"]";
     }
 }
```

Note: This code uses dtCopy to gather values, which are set to e.Value, to place them in the unbound column.

Run the program and observe the following:

- C1List1 displays data from the table with the *First* and *Last* fields missing.,br/>



- The first column displays the combined *First* and *Last* fields as defined in the UnboundColumnFetch event.

This concludes the tutorial.

Tutorial 8 Displaying Translated Data

In this tutorial, you will learn how to use the Translate and DisplayValue properties to assign text to numerical values for a more descriptive list.

1. Create a new .NET 2.0 project.
2. Double-click the C1List icon in Visual Studio's Toolbox to add it to the Form.

   > **Note: See Adding the C1List Components to a Project for information on adding a component to the Toolbox.**

3. Go to the **DataSource** property and select **Add Project Data Source** from the drop-down. In the adapter's **Data Source Configuration Wizard**, either select a connection to **C1NWind.mdb** or create a new connection to this database. On the **Choose Your Database Objects** page of the wizard, select all fields in the **Contacts** table and type "DsContacts" into the **DataSet name** box, and then finish out the wizard.
4. Double click on **DsContacts.xsd** in the Solution Explorer window to edit it in the Designer. Right click on the **Contacts** table and choose **Configure** from the context menu.
5. Modify the SQL string in the **Table Adapter Configuration Wizard** to:

   ```
   SELECT Contacts.UserCode, Contacts.ContactDate, Contacts.Callback,
   Contacts.Comments, Contacts.ContactType, Customers.LastName,
   Customers.FirstName, Customers.Company, Customers.Contacted, Customers.Phone,
   Customers.CustType FROM (Contacts INNER JOIN Customers ON Contacts.UserCode =
   Customers.UserCode)
   ```

6. The **Contacts** table is now joined with the **Customers** table. Click **Finish** to exit the wizard.
7. Return to Design view and if prompted to replace existing column layout, click **Yes**.

   > **Note: If all of the columns are not showing up in C1List, select the DataSource again from the drop-down menu.**

8. Visual Studio 2005 adds the following code to the **Form_Load** event:

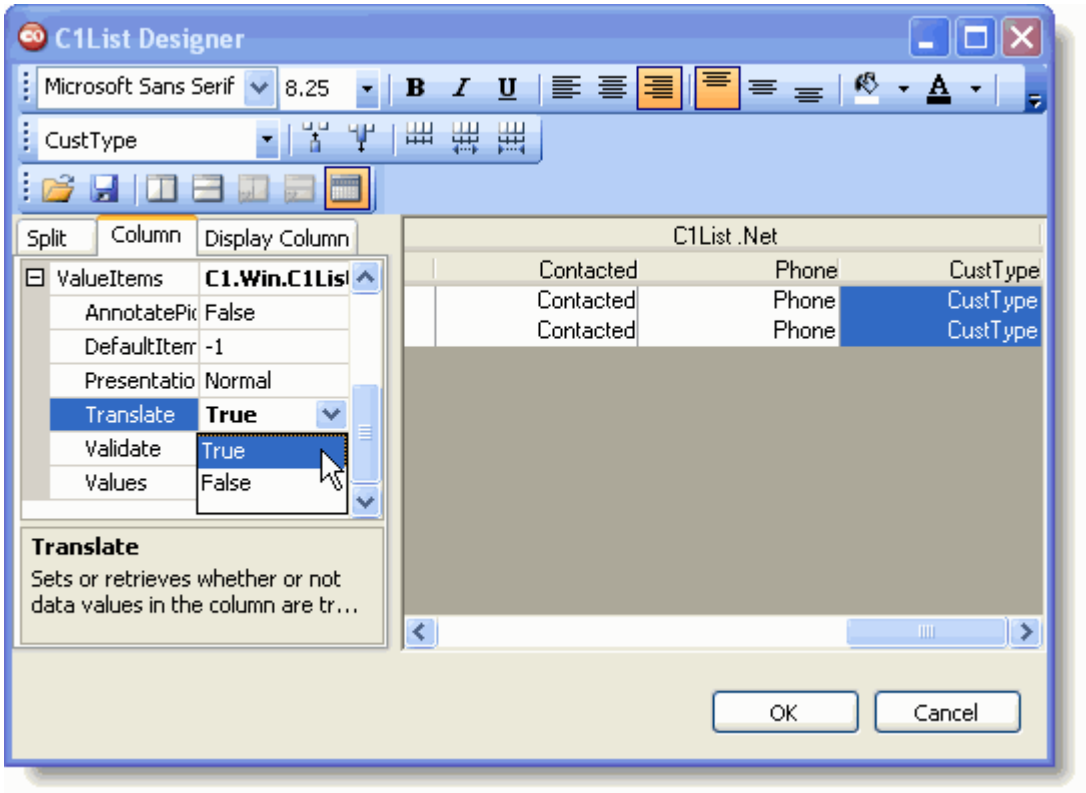
To write code in Visual Basic

```
Visual Basic
Me.ContactsTableAdapater.Fill(Me.DsContacts.Contacts)
```

To write code in C#

```
C#
this.ContactsTableAdapater.Fill(this.DsContacts.Contacts);
```

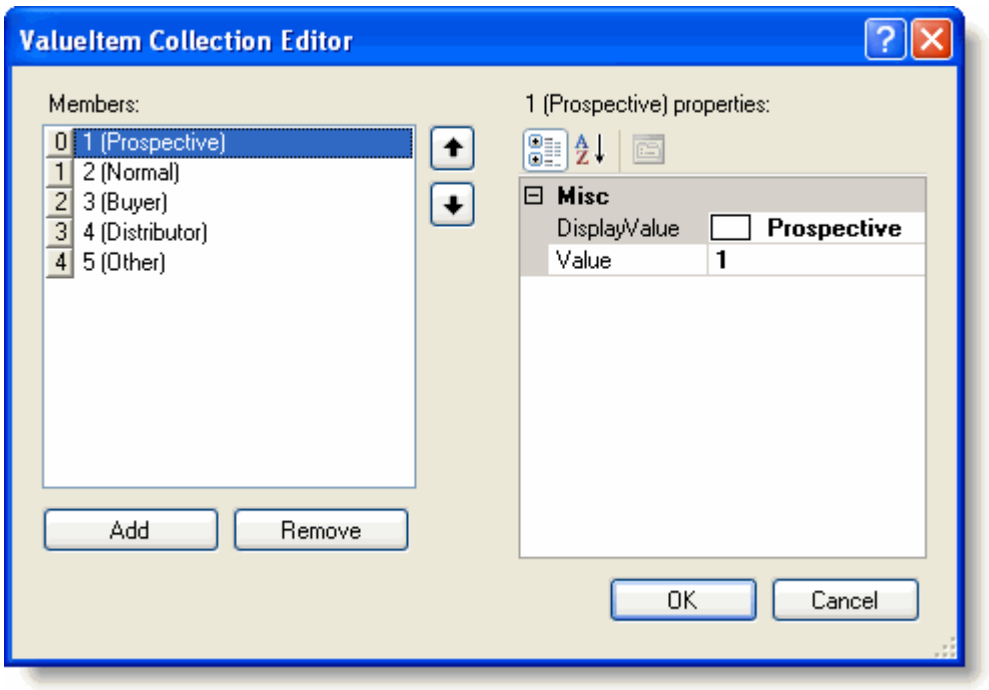
9. Click the **ellipsis** button next to the Columns property of C1List1 in the Properties window. This will bring up the **C1List Designer**. Select **CustType** in the right pane, and in the left pane expand the ValueItems property. Set the Translate property to **True**.



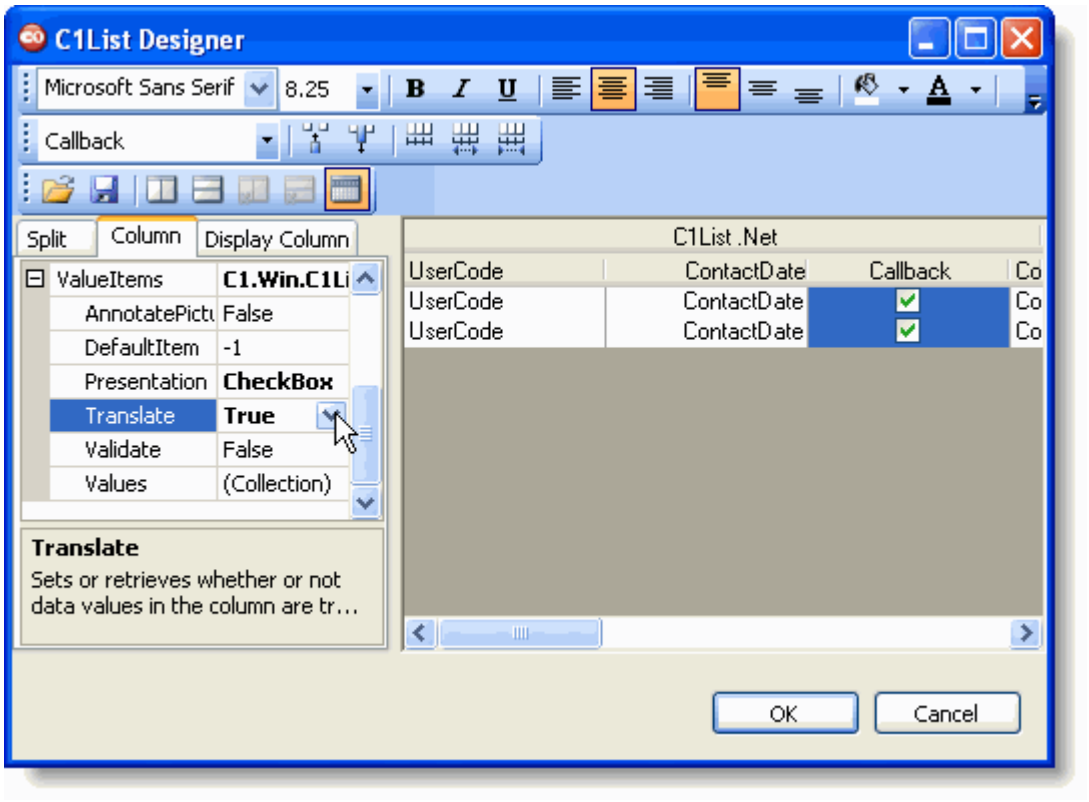
10. Click the **ellipsis** button next to the Values property and the **ValueItem Collection Editor** appears. Click the **Add** button and the first ValueItem member appears in the left pane. In the right pane enter **Prospective** in the **DisplayValue** box and enter **1** in the **Value** box.
11. Add four more members with the following properties to the **Members** list:

Member	DisplayValue	Value
1	Normal	2
2	Buyer	3
3	Distributor	4
4	Other	5

The **ValueItem Collection Editor** should look like the following:

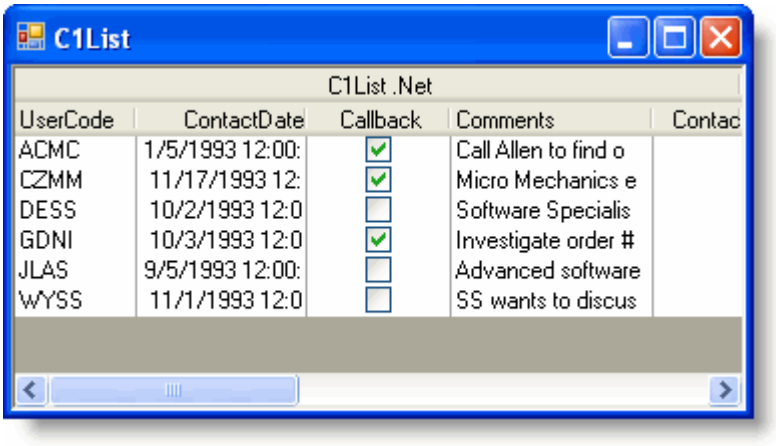


12. Click **OK** to return to the **C1List Designer**. Select *Callback* in the right pane, and in the left pane expand the ValueItems property. Set the Presentation property to **CheckBox** and the Translate property to **True**.



Run the program and observe the following:

- In the *CustType* column C1List1 displays the DisplayValues that you entered in the **ValueItem Collection Editor**. Setting the Translate property to **True** displays the DisplayValues instead of the data values entered for the Value property.



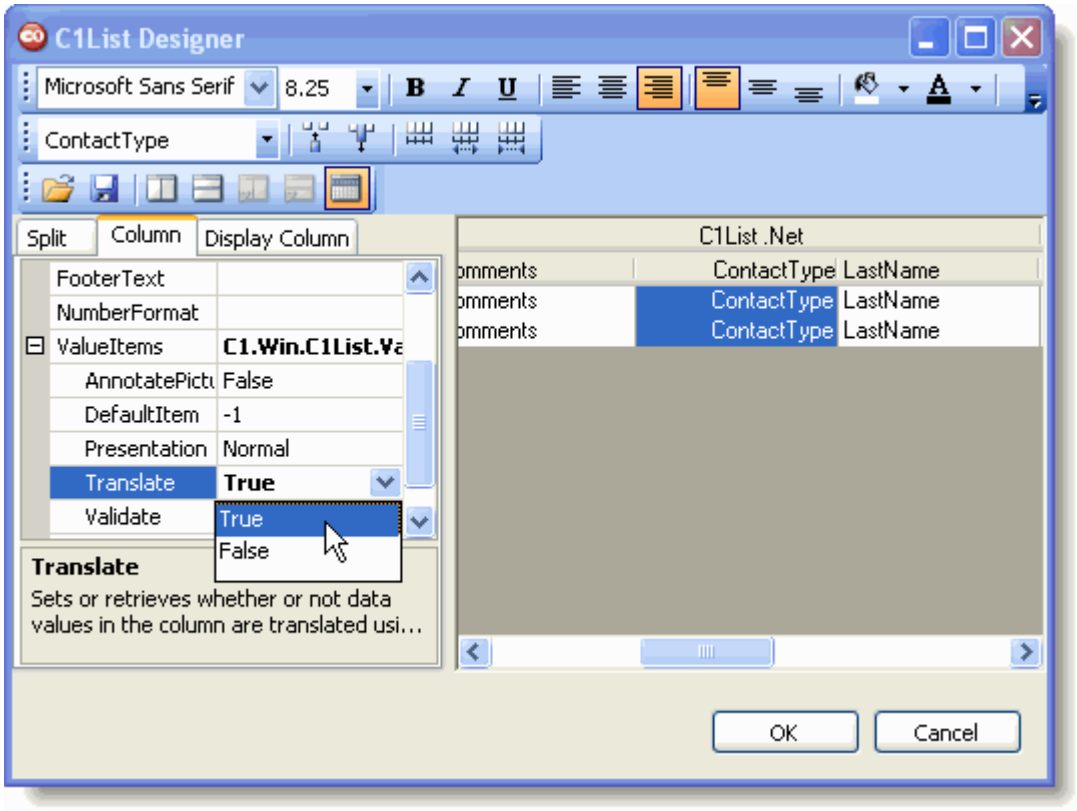
- In the *Callback* column C1List1 automatically places a checkmark next to the customers wanting a call back instead of **True** or **False**.

This concludes the tutorial.

Tutorial 9 Displaying Translated Data Using Images

In this tutorial, you will learn how to use the Translate and DisplayValue properties to display images in the column.

1. Follow steps 1 through 6 of Tutorial 8 - Displaying Translated Data to create a project with a C1List control bound to a Data Set.
2. Click the **ellipsis** button next to the Columns property of C1List1 in the Properties window. This will bring up the **C1List Designer**. Select *ContactType* in the right pane, and in the left pane expand the ValueItems property. Set the Translate property to **True**.



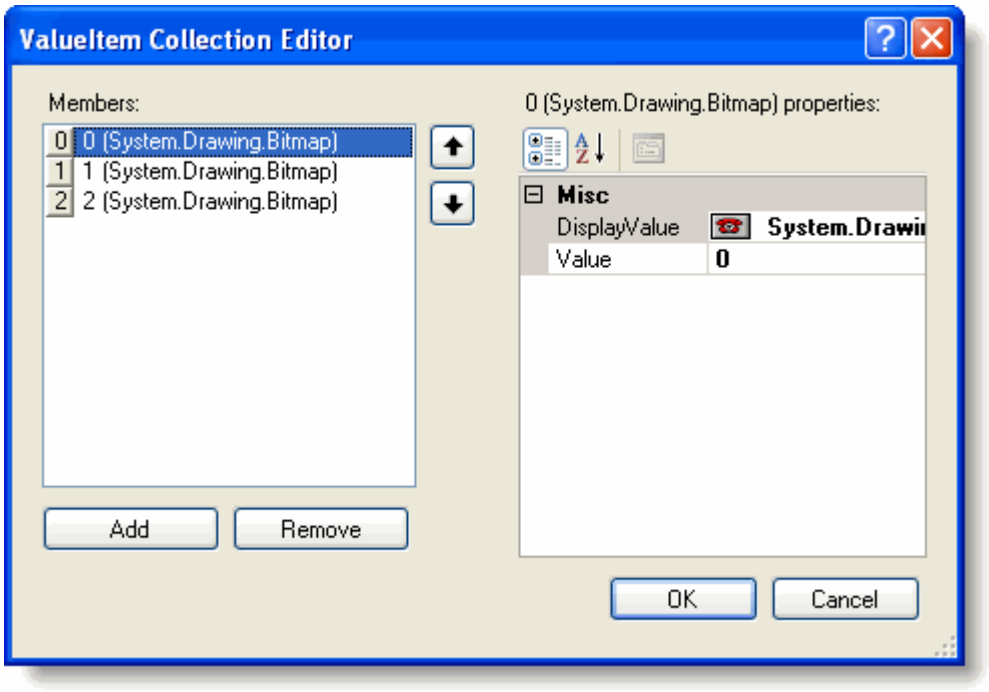
3. Click the **ellipsis** button next to the Values property and the **ValueItem Collection Editor** appears. Click the

Add button and the first ValueItem member appears in the left pane. In the right pane enter **0** in the **Value** box. For the **DisplayValue**, click the **ellipsis** button and find the phone.bmp file that came with this tutorial. Click **Open** and a phone icon appears next to the **DisplayValue**.

4. Add two more members with the following properties to the **Members** list:

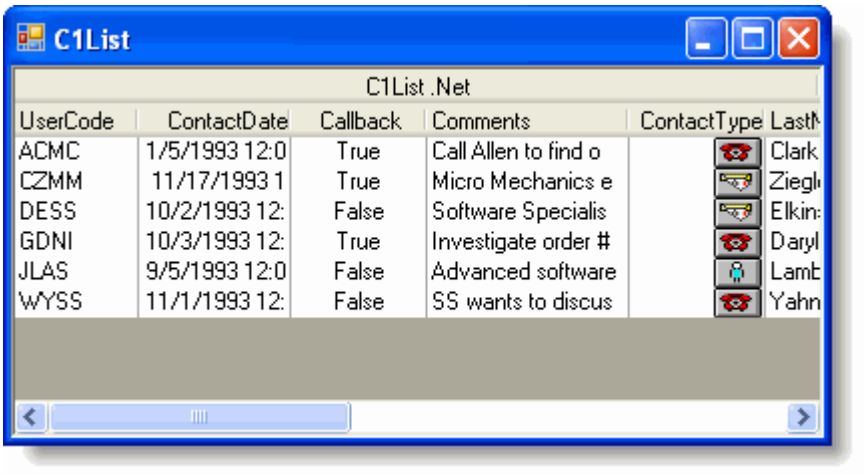
Member	DisplayValue	Value
1	mail.bmp	1
2	person.bmp	2

The ValueItem collection should look like the following:



Run the program and observe the following:

In the *ContactType* column C1List1 displays the icons that you entered as DisplayValues in the **ValueItem Collection Editor**. Setting the Translate property to **True** displays the DisplayValues instead of the data values entered for the Value property.

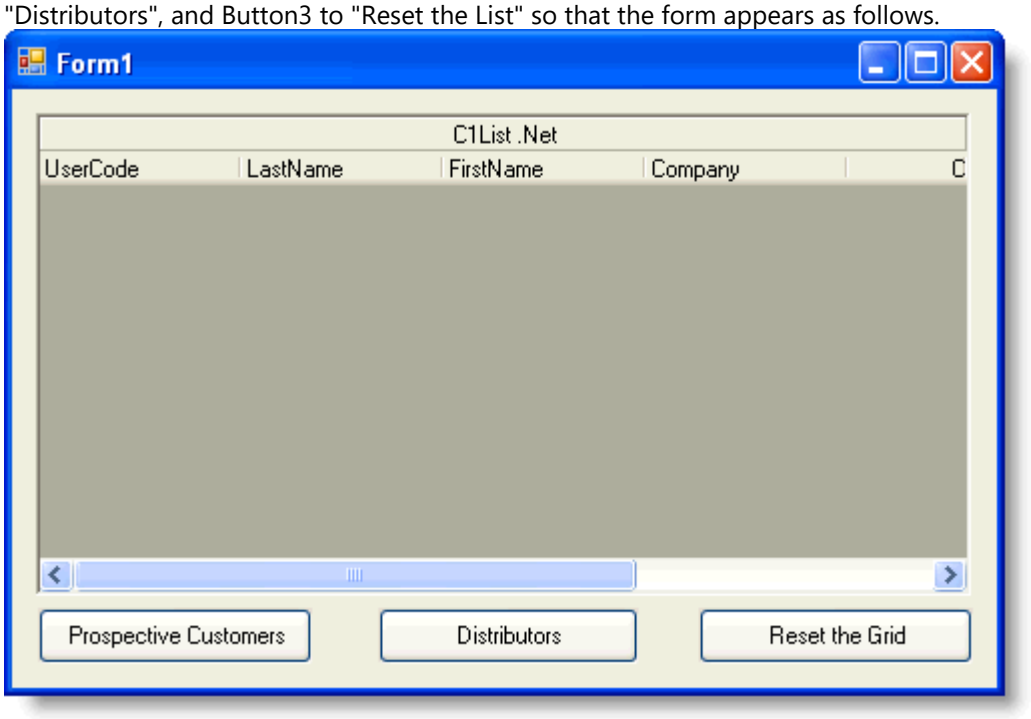


This concludes the tutorial.

Tutorial 10 - Using Row Styles to Highlight Related Data

In this tutorial, you will learn how to change the list's display to highlight rows by creating row styles depending upon a value in the list. C1List uses the FetchRowStyle event to create style characteristics and apply them to rows dynamically.

1. Follow steps 1 through 6 of Tutorial 8 - Displaying Translated Data, setting SELECT * FROM CUSTOMERS as the SQL statement, to create a project with a C1List control bound to a Data Set.
2. Add thee buttons to the form. Change the caption of Button1 to "Prospective Customers", Button2 to "Distributors", and Button3 to "Reset the List" so that the form appears as follows.



3. Add the following declarations to the General section of Form1:

To write code in Visual Basic

```
Visual Basic
Dim bFlag1, bFlag2 As Boolean
```

To write code in C#

```
C#
bool bFlag1, bFlag2;
```

4. Enter the following code in the **Button1_Click** event:

To write code in Visual Basic

```
Visual Basic
Me.C1List1.FetchRowStyles = True
bFlag1 = True
Me.C1List1.Refresh()
```

To write code in C#

```
C#
this.c1List1.FetchRowStyles = true;
bFlag1 = true;
this.c1List1.Refresh();
```

5. Enter the following code in the **Button2_Click** event:

To write code in Visual Basic

```
Visual Basic
Me.C1List1.FetchRowStyles = True
bFlag2 = True
Me.C1List1.Refresh()
```

To write code in C#

```
C#
this.c1List1.FetchRowStyles = true;
bFlag2 = true;
this.c1List1.Refresh();
```

6. Enter the following code in the **Button3_Click** event:

To write code in Visual Basic

```
Visual Basic
Me.C1List1.FetchRowStyles = True
bFlag1 = False
bFlag2 = False
Me.C1List1.Refresh()
```

To write code in C#

```
C#
this.c1List1.FetchRowStyles = true;
bFlag1 = false;
bFlag2 = false;
this.c1List1.Refresh();
```

7. Enter the following code in the FetchRowStyle event. This code interacts with the setting of the FetchRowStyles property in the click event. When the FetchRowStyles is set to **True**, the list fires FetchRowStyle event when it needs to repaint the cells. Thus the row style is applied according to the value of the bflag flag integer:

To write code in Visual Basic

```
Visual Basic

Private Sub C1List1_FetchRowStyle(ByVal sender As Object, ByVal e As
C1.Win.C1List.FetchRowStyleEventArgs) Handles C1List1.FetchRowStyle
    If (bFlag1 And Me.C1List1.Columns("CustType").CellValue(e.Row) = 1) Then
 Dim fntFont As New Font(e.CellStyle.Font.Name, e.CellStyle.Font.Size,
FontStyle.Bold)
```

```
 e.CellStyle.Font = fntFont
 e.CellStyle.ForeColor = Color.Blue
    End If
    If (bFlag2 And Me.C1List1.Columns("CustType").CellValue(e.Row) = 4) Then
 e.CellStyle.ForeColor = Color.White
 e.CellStyle.BackColor = Color.Red
    End If
End Sub
```

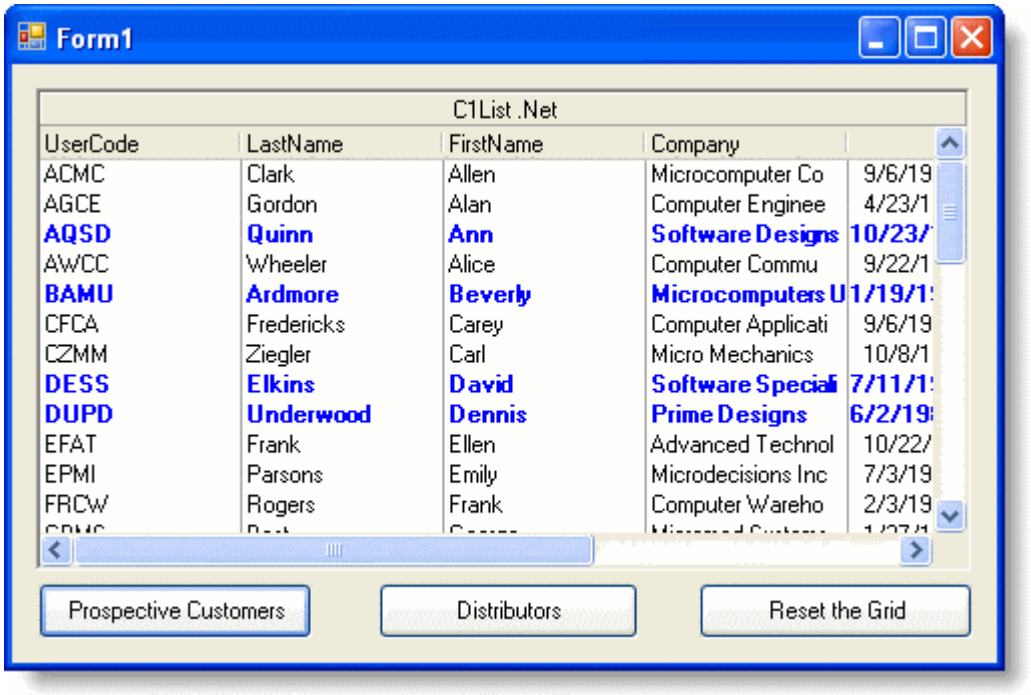
To write code in C#

C#

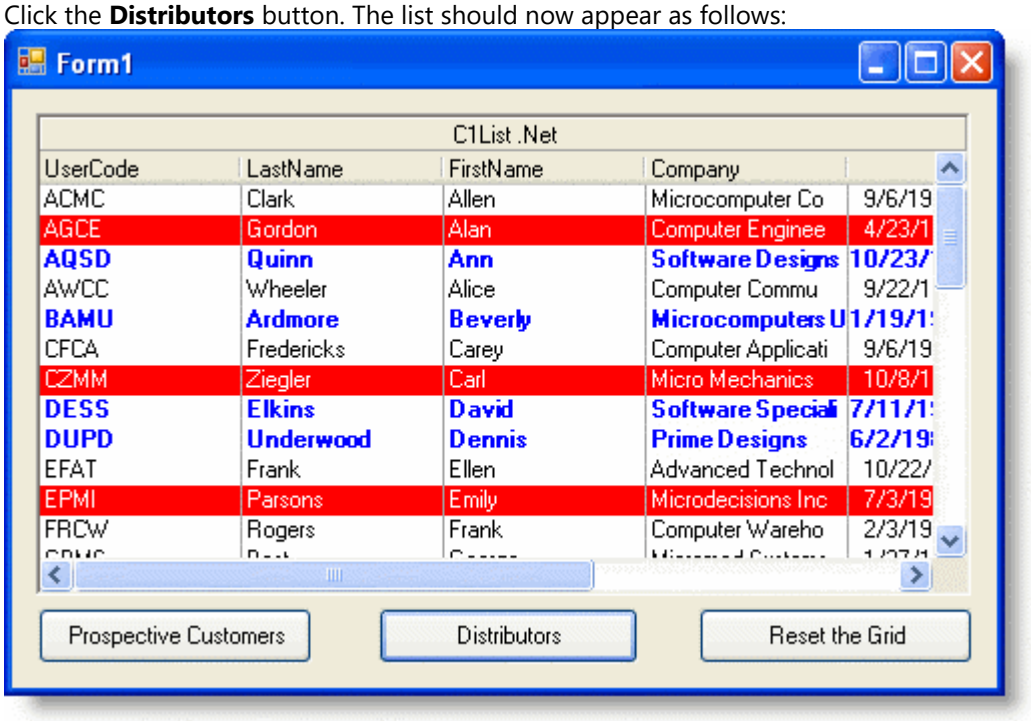
```
private void C1List1_FetchRowStyle( object sender,
C1.Win.C1List.FetchRowStyleEventArgs e)
{
    if (bFlag1 && this.c1List1.Columns["CustType"].CellValue(e.Row) == 1)
    {
 Font fntFont = new Font(e.CellStyle.Font.Name, e.CellStyle.Font.Size,
FontStyle.Bold);
 e.CellStyle.Font = fntFont;
 e.CellStyle.ForeColor = Color.Blue;
    }
    if (bFlag2 && this.c1List1.Columns["CustType"].CellValue(e.Row) == 4)
    {
 e.CellStyle.ForeColor = Color.White;
 e.CellStyle.BackColor = Color.Red;
    }
}
```

Run the program and observe the following:

- Click the **Prospective Customers** button. The list should appear as follows:



- Click the **Distributors** button. The list should now appear as follows:



- Finally, click the **Reset the List** button. The list should clear itself of the styles.

This concludes the tutorial.

Tutorial 11 - Displaying Rows in Alternating Colors

In this tutorial, you will learn how to use the AlternatingRows property and built-in styles to apply alternating colors to list rows to improve their readability.

1. Follow steps 1 through 6 of Tutorial 8 - Displaying Translated Data to create a project with a C1List bound to a Data Set.

2. In the **Form_Load** event add the following code:

To write code in Visual Basic

```
Visual Basic
Me.C1List1.AlternatingRows = True
```

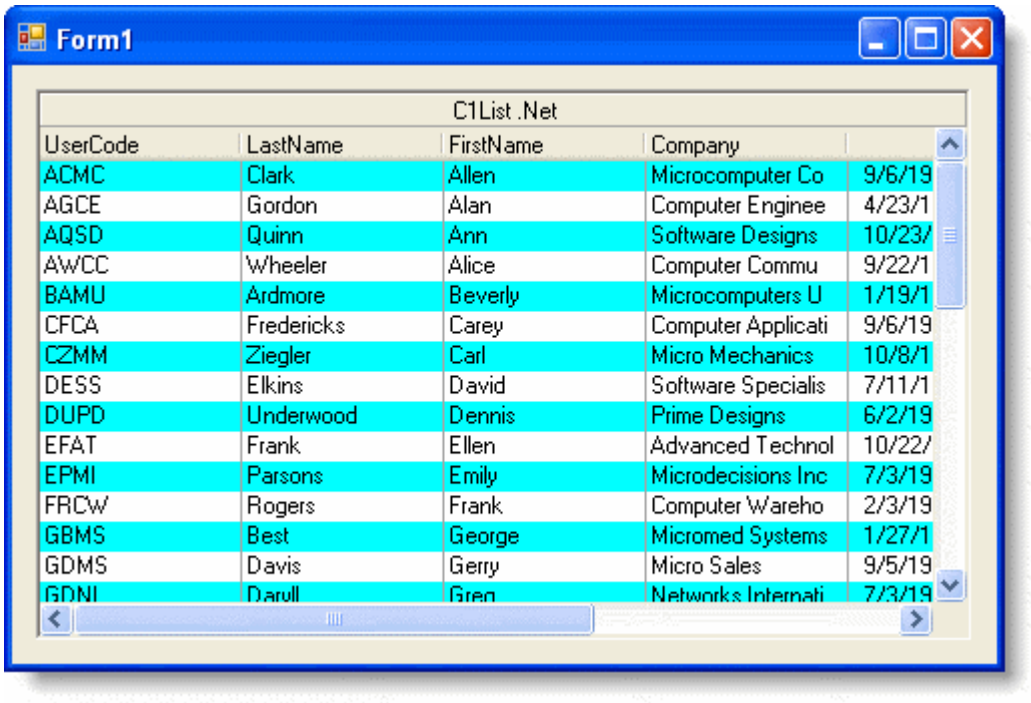
To write code in C#

```
C#
this.c1List1.AlternatingRows = true;
```

The list has default settings for both the EvenRow and OddRow styles. We will use the default settings first and then change the settings for the EvenRowStyle.

Run the program and observe the following:

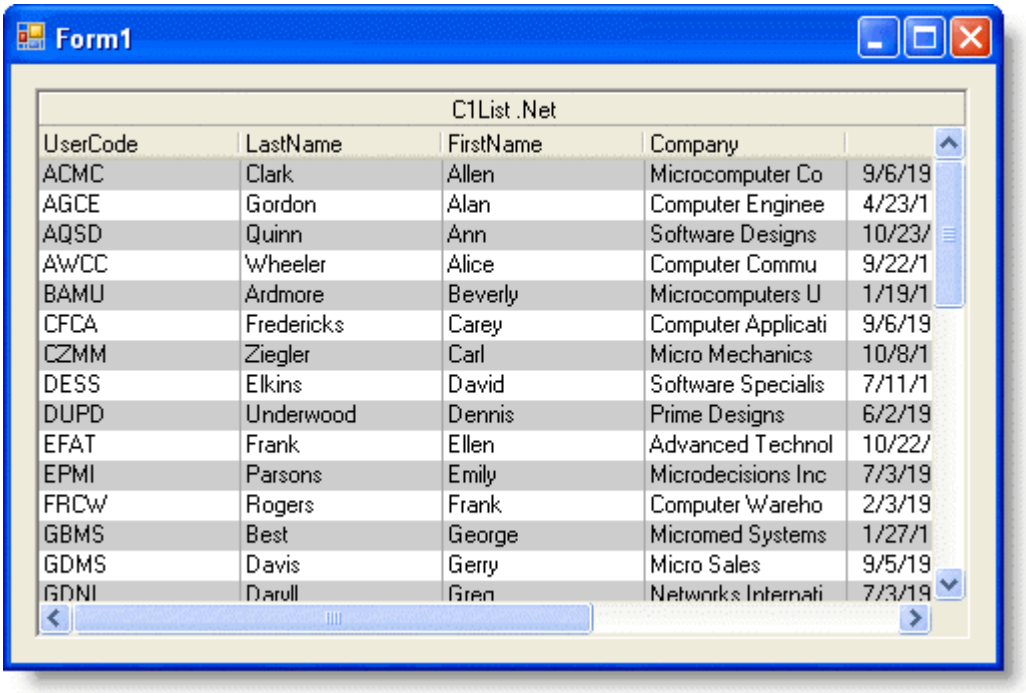
C1List1 displays even-numbered rows with a light cyan background.



3. 3Click the **ellipsis** button next to the Styles property in the Properties window. This will bring up the **Style Collection Editor**. Select the EvenRowStyle in the left pane, and in the right pane change its backcolor to **LightGray**. Click **OK** and close the editor.

Run the program and observe the following:

C1List1 displays data as in the previous figure, except that even-numbered rows now have a light gray background.



This concludes the tutorial.

Tutorial 12 - Creating Fixed, Nonscrolling Columns in List

Often, you would like to prevent one or more columns from scrolling horizontally or vertically so that they will always be in view. The SplitCollection of **List for WinForms** provides a generalized mechanism for defining groups of adjacent columns, and can be used to implement any number of fixed, nonscrolling columns or rows. In this tutorial, you will learn how to write code to create a list with two horizontal splits, and then "fix" a pair of columns in the leftmost split.

1. Follow steps 1 through 4 of Tutorial 1 - Binding C1List to a DataSet to create a project with a C1List bound to a Data Set.
2. In the **Form_Load** event, add the following code to create an additional split and to fix columns 0 and 1 in the leftmost split:

   To write code in Visual Basic

   Visual Basic

```
' Hide all the columns from the left split except col 0 and 1.
Dim i As Integer
For i = 2 To Me.C1List1.Columns.Count - 1
    Me.C1List1.Splits(0).DisplayColumns(i).Visible = False
Next i

' Configure the left split to display 2 columns exactly.
Me.C1List1.Splits(0).SplitSizeMode = C1.Win.C1List.SizeModeEnum.NumberOfColumns
Me.C1List1.Splits(0).SplitSize = 2
Me.C1List1.Splits(0).AllowHorizontalSizing = False

' Make col 0 and 1 invisible in the right split.
Me.C1List1.Splits(1).DisplayColumns(0).Visible = False
Me.C1List1.Splits(1).DisplayColumns(1).Visible = False
```

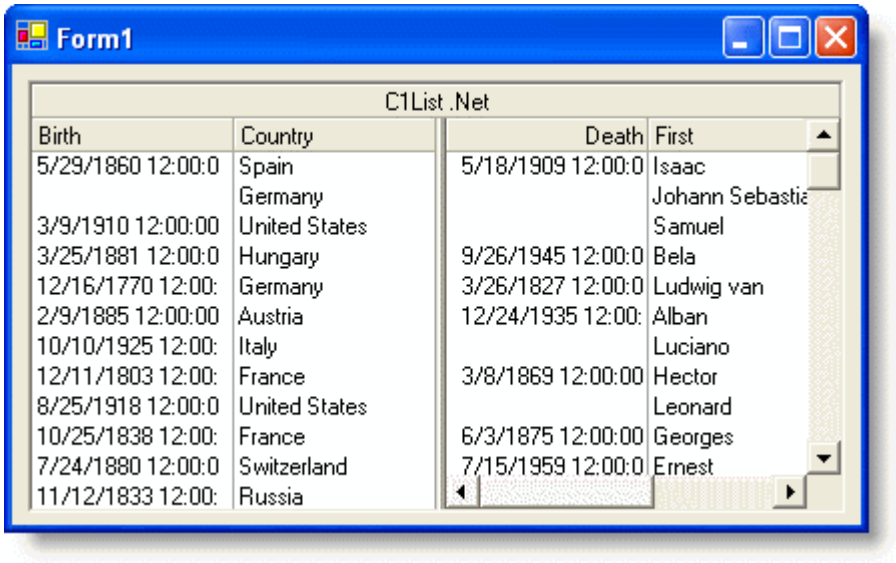
To write code in C#

C#

```
// Hide all the columns from the left split except col 0 and 1.
int i;
for ( i = 2; i<= this.C1List1.Columns.Count - 1; i++)
{
    this.c1List1.Splits[0].DisplayColumns[i].Visible = false;
}

// Configure the left split to display 2 columns exactly.
this.c1List1.Splits[0].SplitSizeMode =
C1.Win.C1List.SizeModeEnum.NumberOfColumns;
this.c1List1.Splits[0].SplitSize = 2;
this.c1List1.Splits[0].AllowHorizontalSizing = false;

// Make col 0 and 1 invisible in the right split.
this.c1List1.Splits[1].DisplayColumns[0].Visible = false;
this.c1List1.Splits[1].DisplayColumns[1].Visible = false;
```

Run the program and observe the following:

The two columns (*Birth* and *Country*) in the leftmost split are fixed and cannot be scrolled. In fact, there is no horizontal scroll bar present under the left split. A horizontal scroll bar appears under the rightmost split, allowing users to scroll the columns in this split.



You can use splits to create fixed, nonscrolling columns anywhere within the list even in the middle. You can also use splits to present different views of your data. For example, you can create splits which scroll independently (in the vertical direction) so that users may compare records at the beginning of the database with those at the end. For more information, see How to Use Splits.

Note: There is another way to fix columns. Use the FixColumn method for the DisplayColumn object.

To use the FixColumn method to fix columns, follow the steps below:

3. Add another C1List control (C1List2) to the form right below C1List1. Add two buttons with the text "Fix Columns" and "Unfix Columns", respectively.
4. Under the **Button_Click1** event for **Fix Columns**, add the following code:

To write code in Visual Basic

```
Visual Basic
' Make the country and birth columns fixed.
Me.C1List2.Splits(0).DisplayColumns("Birth").FixColumn(True)
Me.C1List2.Splits(0).DisplayColumns("Country").FixColumn(True, 1)
```

To write code in C#

```
C#
// Make the country and birth columns fixed.
this.c1List2.Splits[0].DisplayColumns["Birth"].FixColumn(true);
this.c1List2.Splits[0].DisplayColumns["Country"].FixColumn(true, 1);
```

5. Under the **Button_Click2** event for **Unfix Columns**, add the following code:

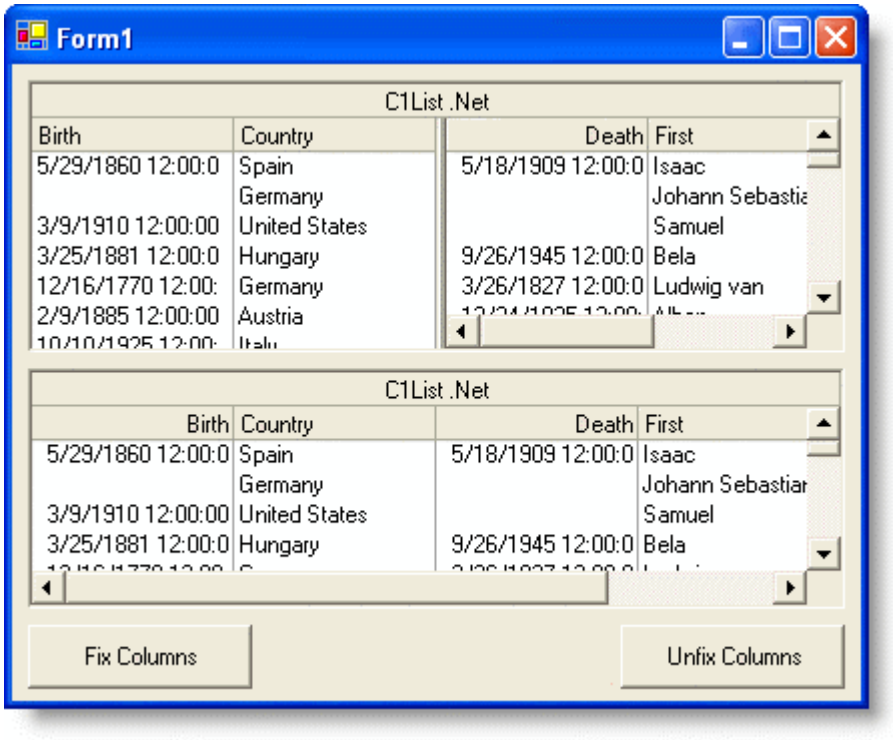
To write code in Visual Basic

```
Visual Basic
' Make the country and birth columns unfixed.
Me.C1List2.Splits(0).DisplayColumns("Country").FixColumn(False, 1)
Me.C1List2.Splits(0).DisplayColumns("Birth").FixColumn(False, 0)
```

To write code in C#

```
C#
// Make the country and birth columns unfixed.
this.c1List2.Splits[0].DisplayColumns["Country"].FixColumn(false, 1);
this.c1List2.Splits[0].DisplayColumns["Birth"].FixColumn(false, 0);
```

6. Run the program and click **Fix Columns**. The first two columns are fixed and you cannot scroll them. Click **Unfix Columns**. The first two columns are not fixed. You may fix any column this way.



This concludes the tutorial.

Tutorial 13 CheckBox Selection

In this tutorial you will learn to use the **SelectionMode** property to allow users to select multiple items in a list using a checkbox.

1. Follow steps 1 through 6 of Tutorial 8 - Displaying Translated Data to create a project with a C1List bound to a Data Set.
2. In the **Form_Load** event, add the following code:

To write code in Visual Basic

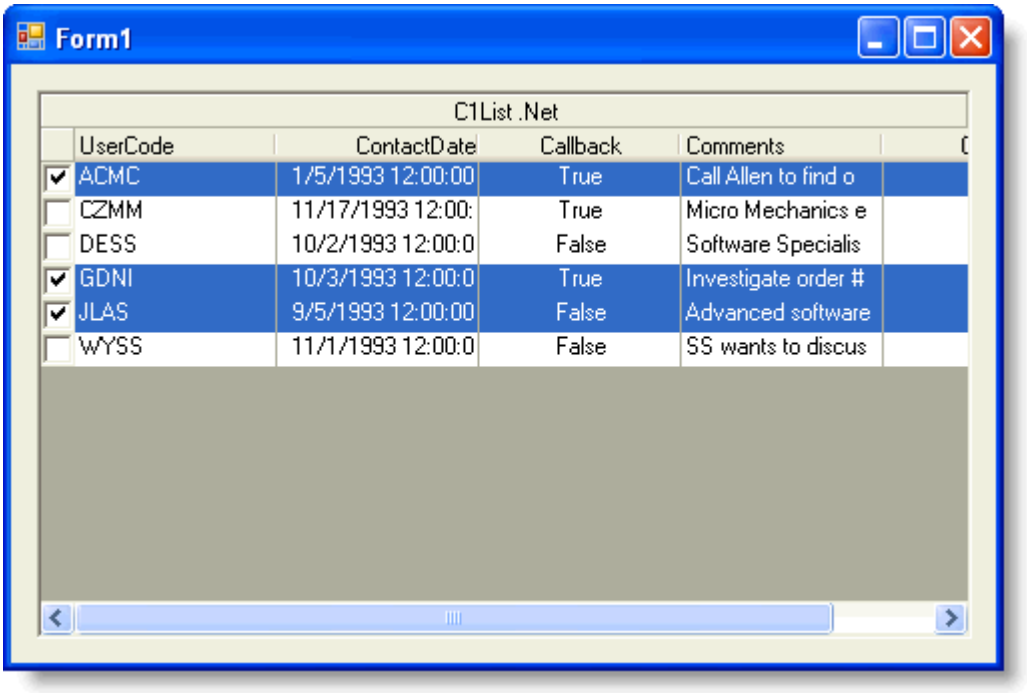
```
Visual Basic
' CheckBox selection.
Me.C1List1.SelectionMode = C1.Win.C1List.SelectionModeEnum.CheckBox
```

To write code in C#

```
C#
// CheckBox selection.
this.c1List1.SelectionMode = C1.Win.C1List.SelectionModeEnum.CheckBox;
```

Run the program and observe the following:

You can place a checkmark in multiple boxes to select multiple rows:



This concludes the tutorial.

Tutorial 14 Displaying Bitmaps

In this tutorial, you will learn how to use Style objects to display arbitrary bitmaps within list cells. In addition to font, color, and alignment settings, Style objects support both background and foreground pictures. Background pictures can be centered, tiled, or stretched to fill the entire cell. Foreground pictures can be positioned relative to the cell text, and can also be displayed using a transparent color.

1. Create a new .NET project.
2. Place a C1List control (C1List1), a command button (Button1), and a TextBox control (TextBox1) on the form (Form1).
3. Set the DataMode property of C1List1 to **AddItem**.
4. In the **Form_Load** event, add the following code to find the Flag directory that came with this tutorial:

To write code in Visual Basic

Visual Basic

```
' Get the image directory.
Dim dir As String
dir = Environment.CurrentDirectory
dir = dir.Substring(0, dir.LastIndexOf("\"))
dir = dir & "\Flags"
Me.TextBox1.Text = dir
```

To write code in C#

C#

```
// Get the image directory.
string dir;
dir = Environment.CurrentDirectory;
```

```
dir = dir.Substring(0, dir.LastIndexOf("\"));
dir = dir + "\Flags";
this.TextBox1.Text = dir;
```

5. To handle the **Button1_Click** event, add the following code:

To write code in Visual Basic

Visual Basic

```
' Add the title.
Me.C1List1.AddItemTitles("File Name; Picture")

' Extend the second column width.
Me.C1List1.ExtendRightColumn = True

' Enlarge the row height.
Me.C1List1.ItemHeight = 50

' Add the items for the list.
Dim dir, str, fileName As String
dir = Me.TextBox1.Text.Trim()
Dim strCol As String()
strCol = System.IO.Directory.GetFiles(dir, "*.bmp")
For Each str In strCol
    fileName = str.Substring(str.LastIndexOf("\") + 1)
    Me.C1List1.AddItem(fileName & ";")
Next str

' Fire the FecthCellStyle event.
Me.C1List1.Splits(0).DisplayColumns(1).FetchStyle = True
```

To write code in C#

C#

```
// Add the title.
this.c1List1.AddItemTitles("File Name; Picture");

// Extend the second column width.
this.c1List1.ExtendRightColumn = true;

// Enlarge the row height.
this.c1List1.ItemHeight = 50;

// Add the items for the list.
string dir, str, fileName;
dir = this.TextBox1.Text.Trim();
string[]  strCol;
strCol = System.IO.Directory.GetFiles(dir, "*.bmp");
foreach ( str in strCol)
{
    fileName = str.Substring(str.LastIndexOf("\") + 1);
    this.c1List1.AddItem(fileName + ";");
```

```
}

// Fire the FecthCellStyle event.
this.c1List1.Splits[0].DisplayColumns[1].FetchStyle = true;
```

6. In the FetchCellStyle event, add the following code:

To write code in Visual Basic

Visual Basic

```
Private Sub C1List1_FetchCellStyle(ByVal sender As Object, ByVal e As
C1.Win.C1List.FetchCellStyleEventArgs) Handles C1List1.FetchCellStyle
    If e.Col = 1 Then

 ' Get the image name.
 Dim file As String
 file = Me.C1List1.Splits(0).DisplayColumns(0).DataColumn.CellText(e.Row)
 file = Me.TextBox1.Text.Trim() & "\" & file
 e.CellStyle.ForeGroundPicturePosition =
C1.Win.C1List.ForeGroundPicturePositionEnum.PictureOnly
 e.CellStyle.ForegroundImage = Image.FromFile(file)
    End If
End Sub
```

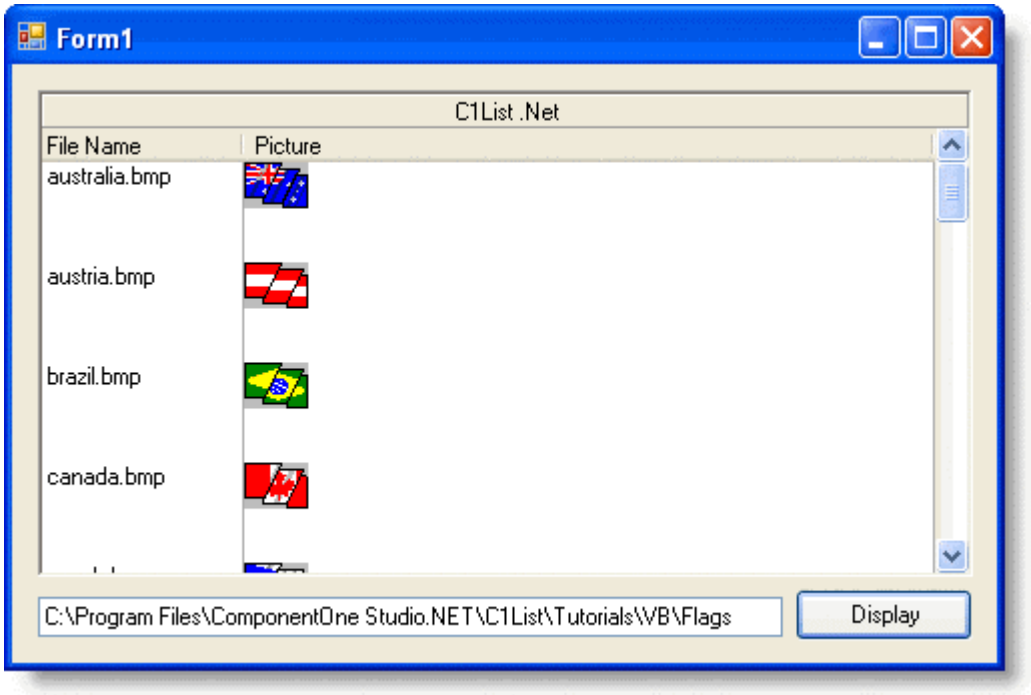
To write code in C#

C#

```
private void C1List1_FetchCellStyle( object sender,
C1.Win.C1List.FetchCellStyleEventArgs e)
{
    if ( e.Col == 1 )
    {

 // Get the image name.
 string  file;
 file = this.c1List1.Splits[0].DisplayColumns[0].DataColumn.CellText(e.Row);
 file = this.TextBox1.Text.Trim() + "\" + file;
 e.CellStyle.ForeGroundPicturePosition =
C1.Win.C1List.ForeGroundPicturePositionEnum.PictureOnly;
 e.CellStyle.ForegroundImage = Image.FromFile(file);
    }
}
```

Run the program and observe the following:

- Press the **Display** button. The list will display pictures in the first column and their filenames in the second.



- You have created a fully functional bitmap browser with just a few lines of code.

For more information, see Applying Pictures to List Elements.

This concludes the tutorial.

Tutorial 15 - OwnerDraw List Cells

In this tutorial you will learn to format your list with the OwnerDrawCell event.

1. Follow steps 1 through 6 of Tutorial 8 - Displaying Translated Data, setting the SQL statement to SELECT * FROM CUSTOMER, to create a project with a C1List bound to a Data Set.
2. Define the following variables at the Form level:

   To write code in Visual Basic

   ```
   Visual Basic
   Dim bh1, bh2 As Brush
   Dim ft As Font
   ```

   To write code in C#

   ```
   C#
   bh1, bh2 Brush;
   ft Font;
   ```

3. In the **Form_Load** event, add the following code:

   To write code in Visual Basic

   ```
   Visual Basic
   ' Fill the data.
   Me.CustomerTableAdapter.Fill(Me.DsCustomer.Customer)
   ```

```
' Owner draw every column.
Dim i As Integer
For i = 0 To Me.C1List1.Splits(0).DisplayColumns.Count - 1
    Me.C1List1.Splits(0).DisplayColumns(i).OwnerDraw = True
Next i

' Set the row height.
Me.C1List1.ItemHeight = 25

' Set the brushes and font.
bh1 = New SolidBrush(Color.Pink)
bh2 = New SolidBrush(Color.Yellow)
Dim ff As FontFamily
ff = New FontFamily("Arial")
ft = New Font(ff, 14, FontStyle.Regular, GraphicsUnit.Pixel)
```

To write code in C#

C#

```
// Fill the data.
this.CustomerTableAdapter.Fill(this.DsCustomer.Customer);

// Owner draw every column.
int i;
for ( i = 0; i <= this.c1List1.Splits[1].DisplayColumns.Count - 1; i++)
{
    this.c1List1.Splits[0].DisplayColumns[i] OwnerDraw = true;
}

// Set the row height.
this.c1List1.ItemHeight = 25;

// Set the brushes and font.
bh1 = new SolidBrush(Color.Pink);
bh2 = new SolidBrush(Color.Yellow);
FontFamily ff;
ff = new FontFamily("Arial");
ft = new Font(ff, 14, FontStyle.Regular, GraphicsUnit.Pixel);
```

4. Add the following code to the OwnerDrawCell event:

To write code in Visual Basic

Visual Basic

```
 Private Sub C1List1_OwnerDrawCell(ByVal sender As Object, ByVal e As
C1.Win.C1List.OwnerDrawCellEventArgs) Handles C1List1.OwnerDrawCell

    ' Draw alternative background.
    If (e.Row Mod 2 = 0) Then
        e.Graphics.FillRectangle(Me.bh1, e.CellRect)
    Else
```

```
            e.Graphics.FillRectangle(Me.bh2, e.CellRect)
        End If
        e.Graphics.DrawString(e.Text, Me.ft, New SolidBrush(Color.Black),
e.CellRect.Left, e.CellRect.Top)
        e.Handled = True
    End Sub
```

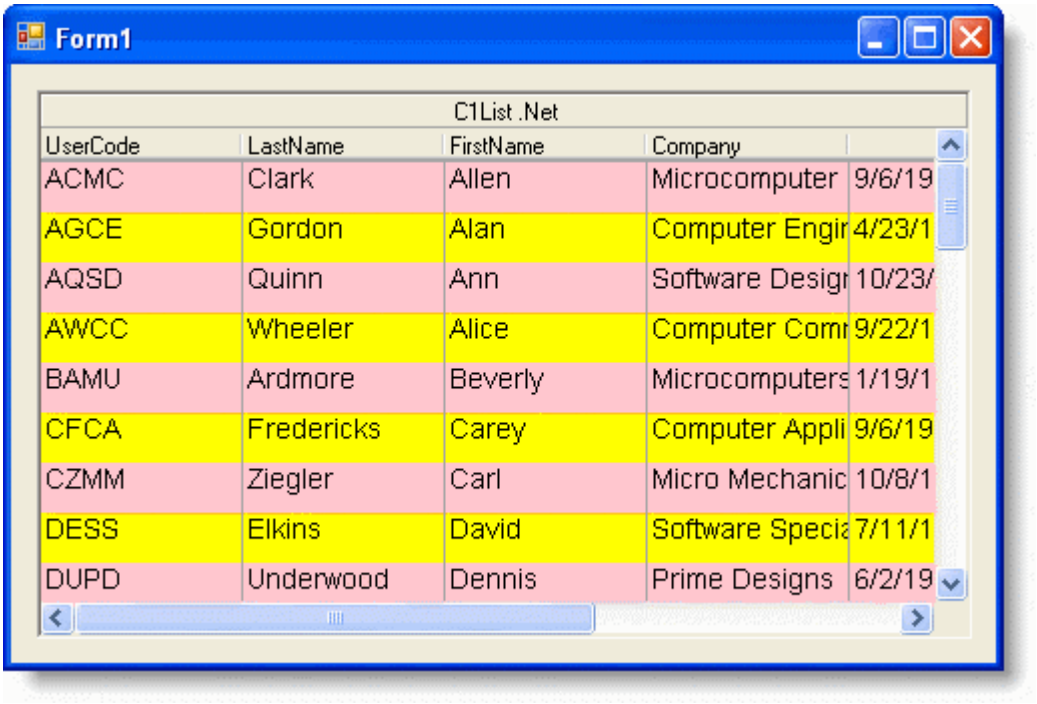
To write code in C#

C#

```
 private void C1List1_OwnerDrawCell(object sender,
C1.Win.C1List.OwnerDrawCellEventArgs e)
 {
     // Draw alternative background.
     if ((e.Row % 2 == 0) )
     {
         e.Graphics.FillRectangle(this.bh1, e.CellRect);
     }
     else
     {
         e.Graphics.FillRectangle(this.bh2, e.CellRect);
     }
     e.Graphics.DrawString(e.Text, this.ft, new SolidBrush(Color.Black),
e.CellRect.Left, e.CellRect.Top);
     e.Handled = true;
 }
```

Run the program and observe the following:

C1List1 displays the data using the font and color changes specified in step 4.

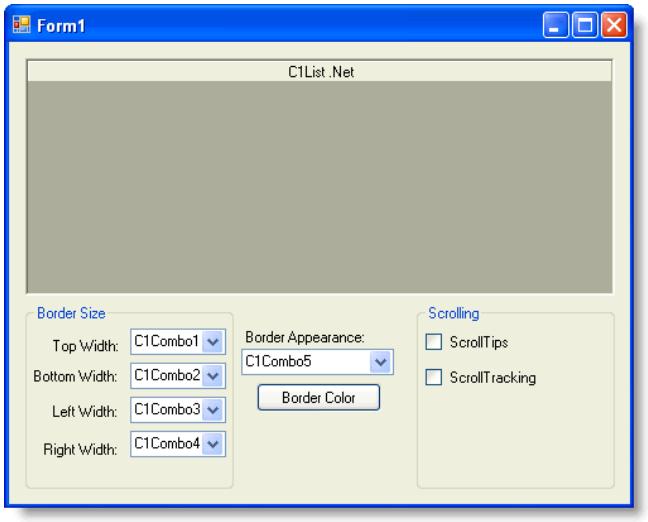


This concludes the tutorial.

Tutorial 16 Borders, Scroll Tracking and Scroll Tips

In this tutorial, you will learn how to use the ScrollTips property and the FetchScrollTips event to provide a tip box as the user scrolls through a list. You will also learn to create borders and colored borders for each item.

1. Create a new .NET project.
2. Double-click the **C1List** icon in Visual Studio's Toolbox to add it to the Form.**Note:** See Adding the C1List Components to a Project for information on adding a component to the Toolbox.
3. Add the following items to the form and situate them like they appear in the following figure:
   - C1Combo boxes (C1Combo1 - 5)
   - Two frames and set their text property to **Border Size** and **Scrolling**
   - Four Labels (Label1 - 5) and set their **Text** properties to **Top Width**, **Bottom Width**, **Left Width**, **Right Width**, and **Border Appearance** respectively
   - A Button (Button1) and set its Text property to **Border Color**
   - Two Checkboxes and set their text properties to **ScrollTips** and **ScrollTracking**



4. Add a Color Dialog to the form (ColorDialog1).
5. Go to the **DataSource** property for C1List and select **Add Project Data Source** from the drop-down. In the adapter's Data Source Configuration Wizard, either select a connection to C1NWind.mdb or create a new connection to this database. On the Choose your database objects page of the wizard, select all fields in the **Customer** table and type "DsCustomer" into the **DataSet name** box, and then finish out the wizard.
6. Visual Studio 2005 adds the following code to the **Form_Load** event:

To write code in Visual Basic

```
Visual Basic
Me.CustomerTableAdapter.Fill(Me.DsCustomer.Customer)
```

To write code in C#

```
C#
this.CustomerTableAdapter.Fill(this.DsCustomer.Customer);
```

7. In the general section of Form1 add the following declarations:

To write code in Visual Basic

```
Visual Basic
Dim tb As DataTable
Dim borderColor As Color
Dim bLeft, bTop, bRight, bBottom As Integer
Dim bType As C1.Win.C1List.BorderTypeEnum
```

To write code in C#

```
C#
DataTable tb;
Color borderColor;
int bLeft, bTop, bRight, bBottom;
C1.Win.C1List.BorderTypeEnum bType;
```

8. In the **Form_Load** event, add the following code:

To write code in Visual Basic

```
Visual Basic
tb = Me.DsCustomer.Tables(0).Copy()
Me.C1List1.ItemHeight = 40
```

```
' Fill each combo.
Me.CheckBox2.Checked = True
FillCombo(C1Combo1)
FillCombo(C1Combo2)
FillCombo(C1Combo3)
FillCombo(C1Combo4)
FillCombo5()

' Initialize the border size.
bBottom = 1
bTop = 1
bLeft = 1
bRight = 1
bType = C1.Win.C1List.BorderTypeEnum.None
```

To write code in C#

C#

```
tb = this.DsCustomer.Tables[0].Copy();
this.c1List1.ItemHeight = 40;

// Fill each combo.
this.checkBox2.Checked = true;
FillCombo(C1Combo1);
FillCombo(C1Combo2);
FillCombo(C1Combo3);
FillCombo(C1Combo4);
FillCombo5();

// Initialize the border size.
bBottom = 1;
bTop = 1;
bLeft = 1;
bRight = 1;
bType = C1.Win.C1List.BorderTypeEnum.None;
```

9. Now add the functions which will fill the C1Combo boxes:

To write code in Visual Basic

Visual Basic

```
' Fill each combo with numbers from 1 to 10.
Private Sub FillCombo(ByRef combo As C1.Win.C1List.C1Combo)
    Dim i As Integer
    combo.DataMode = C1.Win.C1List.DataModeEnum.AddItem
    For i = 1 To 10
 combo.AddItem(i.ToString())
    Next
    combo.Text = 1
End Sub

Private Sub FillCombo5()
    With Me.C1Combo5
 .DataMode = C1.Win.C1List.DataModeEnum.AddItem
 .AddItem("Fillet")
 .AddItem("Flat")
 .AddItem("Groove")
 .AddItem("Inset")
 .AddItem("InsetBevel")
 .AddItem("None")
 .AddItem("Raised")
 .AddItem("RaisedBevel")
 .SelectedIndex = 5
    End With
End Sub
```

To write code in C#

C#

```
// Fill each combo with numbers from 1 to 10.
private void FillCombo(C1.Win.C1List.C1Combo combo)
{
    int i;
    combo.DataMode = C1.Win.C1List.DataModeEnum.AddItem;
    for ( i =1 ; i <= 10 ; i++)
    {
 combo.AddItem(i.ToString());
    }
    combo.Text = 1;
}
```

```
private void FillCombo5()
{
    this.C1Combo5..DataMode = C1.Win.C1List.DataModeEnum.AddItem;
    this.C1Combo5.AddItem("Fillet");
    this.C1Combo5.AddItem("Flat");
    this.C1Combo5.AddItem("Groove");
    this.C1Combo5.AddItem("Inset");
    this.C1Combo5.AddIte("InsetBevel");
    this.C1Combo5.AddItem("None");
    this.C1Combo5.AddItem("Raised");
    this.C1Combo5.AddItem("RaisedBevel");
    this.C1Combo5.SelectedIndex = 5;
}
```

10. Create a handler for the **Button1_Click** event which sets the color of the border using the color dialog box:

To write code in Visual Basic

Visual Basic

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    Dim result As DialogResult
    result = Me.ColorDialog1.ShowDialog()
    If result = DialogResult.OK Then
 Me.borderColor = Me.ColorDialog1.Color
 Me.Button1.BackColor = Me.borderColor
    End If
    UpdateBorder()
End Sub
```

To write code in C#

C#

```
private void Button1_Click( System.object sender,
 System.EventArgs e)
{
    DialogResult result;
    result = this.ColorDialog1.ShowDialog();
    if ( result == DialogResult.OK )
    {
 this.borderColor = this.ColorDialog1.Color;
 this.Button1.BackColor = this.borderColor;
    }
    UpdateBorder();
}
```

11. Include the function which updates the borders:

To write code in Visual Basic

Visual Basic

```
Private Sub UpdateBorder()
    If Me.C1List1.Splits(0).DisplayColumns(Me.C1List1.Col) Is Nothing Then
 Exit Sub
    End If

    With Me.C1List1.Splits(0).DisplayColumns(Me.C1List1.Col).Style.Borders
 .Color = Me.borderColor
 .BorderType = Me.bType
 .Bottom = Me.bBottom
 .Left = Me.bLeft
 .Right = Me.bRight
 .Top = Me.bTop
    End With
End Sub
```

To write code in C#

C#

```
private void UpdateBorder()
{
    if ( this.c1List1.Splits[0].DisplayColumns[this.c1List1.Col] == null )
    {
 return;
    }

    CellBorders borders = this.c1List1.Splits[0].DisplayColumns [this.c1List1.Col].Style.Borders;
    borders.Color = this.borderColor;
    borders.BorderType = this.bType;
    borders.Bottom = this.bBottom;
    borders.Left = this.bLeft;
    borders.Right = this.bRight;
    borders.Top = this.bTop;
```

```
}
```

12. Enter code which handles changes in the C1Combo box values:

To write code in Visual Basic

Visual Basic

```
Private Sub C1Combo1_RowChange(ByVal sender As Object, ByVal e As System.EventArgs) Handles C1Combo1.RowChange
    Me.bTop = Me.C1Combo1.SelectedIndex + 1
    Me.UpdateBorder()
End Sub

Private Sub C1Combo2_RowChange(ByVal sender As Object, ByVal e As System.EventArgs) Handles C1Combo2.RowChange
    Me.bBottom = Me.C1Combo2.SelectedIndex + 1
    Me.UpdateBorder()
End Sub

Private Sub C1Combo3_RowChange(ByVal sender As Object, ByVal e As System.EventArgs)
Handles C1Combo3.RowChange
    Me.bLeft = Me.C1Combo3.SelectedIndex + 1
    Me.UpdateBorder()
End Sub

Private Sub C1Combo4_RowChange(ByVal sender As Object, ByVal e As System.EventArgs) Handles C1Combo4.RowChange
    Me.bRight = Me.C1Combo4.SelectedIndex + 1
    Me.UpdateBorder()
End Sub

Private Sub C1Combo5_RowChange(ByVal sender As Object, ByVal e As System.EventArgs) Handles C1Combo5.RowChange
    Select Case Me.C1Combo5.Text
 Case "Fillet"
     Me.bType = C1.Win.C1List.BorderTypeEnum.Fillet
 Case "Flat"
     Me.bType = C1.Win.C1List.BorderTypeEnum.Flat
 Case "Groove"
     Me.bType = C1.Win.C1List.BorderTypeEnum.Groove
 Case "Inset"
     Me.bType = C1.Win.C1List.BorderTypeEnum.Inset
 Case "InsetBevel"
     Me.bType = C1.Win.C1List.BorderTypeEnum.InsetBevel
 Case "None"
     Me.bType = C1.Win.C1List.BorderTypeEnum.None
 Case "Raised"
     Me.bType = C1.Win.C1List.BorderTypeEnum.Raised
 Case "RaisedBevel"
     Me.bType = C1.Win.C1List.BorderTypeEnum.RaisedBevel
    End Select
    Me.UpdateBorder()
End Sub
```

To write code in C#

C#

```
private void C1Combo1_RowChange( object sender, System.EventArgs e)
{
    this.bTop = this.C1Combo1.SelectedIndex + 1;
    this.UpdateBorder();
}

private void C1Combo2_RowChange( object sender, System.EventArgs e)
{
    this.bBottom = this.C1Combo2.SelectedIndex + 1;
    this.UpdateBorder();
}

private void C1Combo3_RowChange( object sender, System.EventArgs e)
{
    this.bLeft = this.C1Combo3.SelectedIndex + 1;
    this.UpdateBorder();
}

private void C1Combo4_RowChange( object sender, System.EventArgs e)
{
    this.bRight = this.C1Combo4.SelectedIndex + 1;
    this.UpdateBorder();
}

private void C1Combo5_RowChange( object sender, System.EventArgs e)
{
    switch (this.C1Combo5.Text)    {
 case "Fillet":
     this.bType = C1.Win.C1List.BorderTypeEnum.Fillet;
```

```
     break;
 case "Flat":
     this.bType = C1.Win.C1List.BorderTypeEnum.Flat;
     break;
 case "Groove":
     this.bType = C1.Win.C1List.BorderTypeEnum.Groove;
     break;
 case "Inset":
     this.bType = C1.Win.C1List.BorderTypeEnum.Inset;
     break;
 case "InsetBevel":
     this.bType = C1.Win.C1List.BorderTypeEnum.InsetBevel;
     break;
 case "None":
     this.bType = C1.Win.C1List.BorderTypeEnum.None;
     break;
 case "Raised":
     this.bType = C1.Win.C1List.BorderTypeEnum.Raised;
     break;
 case
"RaisedBevel":
     this.bType = C1.Win.C1List.BorderTypeEnum.RaisedBevel;
     break;
    }
    this.UpdateBorder();
}
```

13. Finally, include the code which handles the checkboxes and the C1ListBase.FetchScrollTips event which sets the ToolTip box that displays when the user is scrolling:

To write code in Visual Basic

Visual Basic

```
Private Sub CheckBox1_Click(ByVal sender As Object,
ByVal e As System.EventArgs) Handles CheckBox1.Click
    Me.C1List1.ScrollTips = Me.CheckBox1.Checked
End Sub
Private Sub CheckBox2_Click(ByVal sender As Object,
ByVal e As System.EventArgs) Handles CheckBox2.Click
    Me.C1List1.ScrollTrack = Me.CheckBox2.Checked
End Sub
Private Sub C1List1_FetchScrollTips(ByVal sender As Object,
ByVal e As C1.Win.C1List.FetchScrollTipsEventArgs)
Handles C1List1.FetchScrollTips
    ' Set the ScrollTip depending on which scroll bar was moved.
    Select Case e.ScrollBar
 Case C1.Win.C1List.ScrollBarEnum.Horizontal
     e.ScrollTip = Me.C1List1.Columns(e.ColIndex).Caption
 Case C1.Win.C1List.ScrollBarEnum.Vertical
     e.ScrollTip = "Record: " & CStr(e.Row + 1) & " of " &
CStr(Me.C1List1.ListCount) & vbCrLf & "Company: " &
Me.tb.Rows(e.Row).Item("Company") & vbCrLf &
"User Code: " & Me.tb.Rows(e.Row).Item("UserCode")
    End Select
    e.TipStyle.ForeColor = Color.Blue
End Sub
```

To write code in C#

C#

```
private void CheckBox1_Click( object sender,
 System.EventArgs e)
{
    this.c1List1.ScrollTips = this.CheckBox1.Checked;
}

private void CheckBox2_Click( object sender, System.EventArgs e)
{
    this.c1List1.ScrollTrack = this.CheckBox2.Checked;
}

private void C1List1_FetchScrollTips( object sender, C1.Win.C1List.FetchScrollTipsEventArgs e)
{
    // Set the ScrollTip depending on which scroll bar was moved.
    switch (e.ScrollBar)
    {
 case C1.Win.C1List.ScrollBarEnum.Horizontal;
     e.ScrollTip = this.c1List1.Columns[e.ColIndex].Caption;
     break;
 case C1.Win.C1List.ScrollBarEnum.Vertical;
     e.ScrollTip = "Record: " + (e.Row + 1) + " of " + this.c1List1.ListCount_ +
+ "\n" + "\nCompany: " + this.tb.Rows[e.Row].["Company"] + "\n" +
"User Code: " + this.tb.Rows[e.Row].["UserCode"];
```

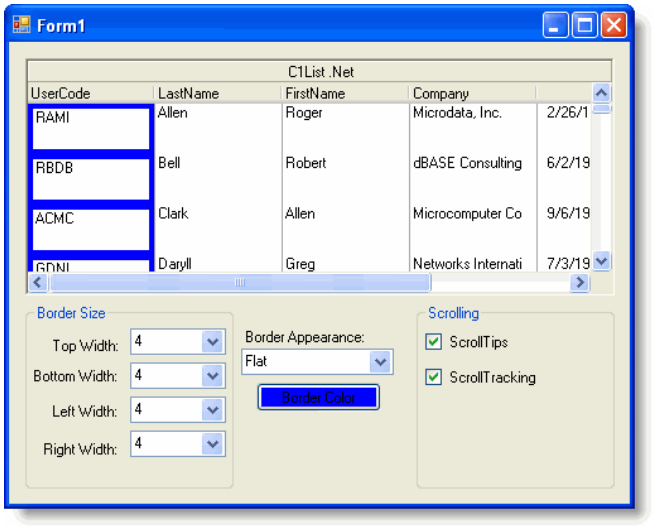
```
    break;
   }
   e.TipStyle.ForeColor = Color.Blue;
}
```

Run the program and observe the following:

- C1List1 displays the data specified.
- Setting ScrollTrack to **True** lets you see the data as it is being scrolled.
- Setting C1ListBase.ScrollTips to **True** shows a ToolTip box with column information while the user is scrolling.

- By manipulating the C1Combo boxes, and the Color Dialog, you can create a border around a column's cells and set them to a System color.

Note: When C1Combo has the focus, press F4 to open the drop-down box.

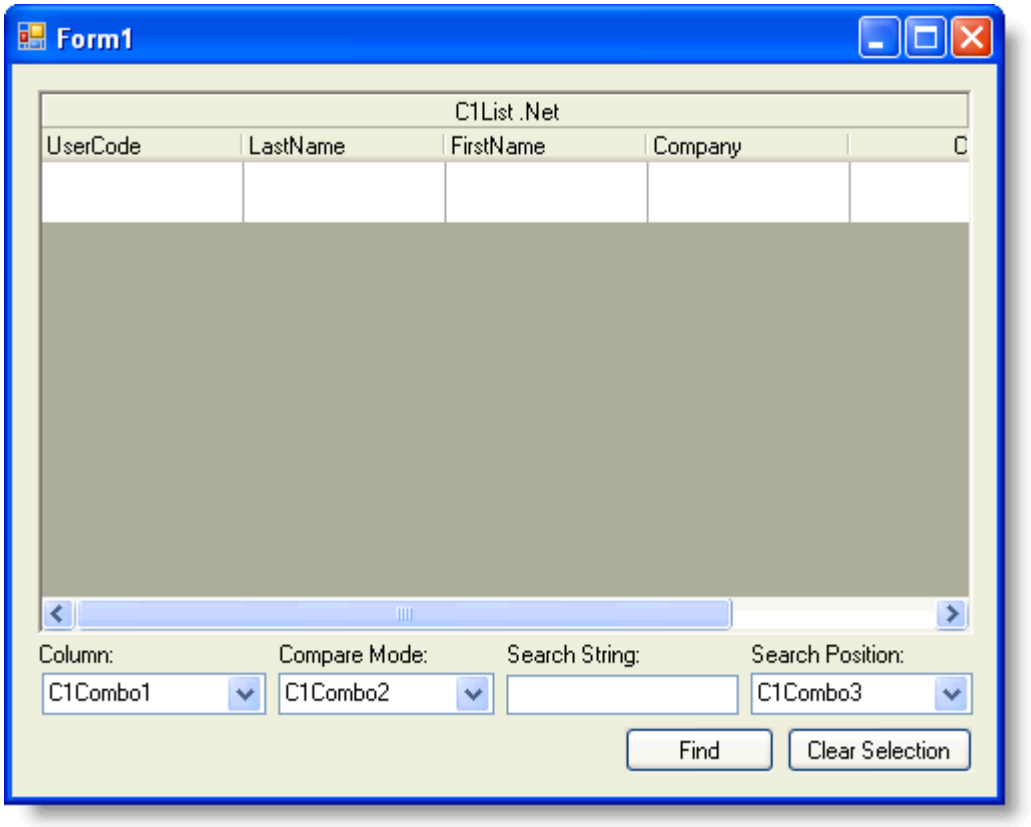


This concludes the tutorial.

Tutorial 17 Conduct Searching using the Find Method

In this tutorial you will utilize the Find method in C1List. The Find method allows you to perform custom searches within the control.

1. Create a new .NET 2.0 project.
2. Double-click the **C1List** icon in Visual Studio's Toolbox to add it to the Form. **Note:** See Adding the C1List Components to a Project for information on adding a component to the Toolbox.
3. From the Visual Studio Toolbox, place the following controls on the form as shown in the illustration below:
   - Three C1Combo boxes (C1Combo1, 2 and 3)
   - A text box (TextBox1)
   - Two command buttons (Button1, 2)
   - Four labels



4. Go to the **DataSource** property for C1List and select **Add Project Data Source** from the drop-down. In the adapter's **Data Source Configuration Wizard**, either select a connection to **C1NWind.mdb** or create a new connection to this database. On the Choose your database objects page of the wizard, select all fields in the **Customers** table and type "DsCustomers" into the **DataSet name** box, and then finish out the wizard.
5. 5. Define some variables at the Form level:

To write code in Visual Basic

Visual Basic

```
Dim matchCompare As C1.Win.C1List.MatchCompareEnum
Dim fromStart As Boolean
```

To write code in C#

C#

```
C1.Win.C1List.MatchCompareEnum matchCompare;
bool fromStart;
```

To connect to the datasource and fill the three **C1Combo** boxes with values at run time, add the following code to the **Form_Load** event:

To write code in Visual Basic

Visual Basic

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    ' The following line is added by Visual Studio 2005.
    Me.CustomersTableAdapter.Fill(Me.DsCustomers.Customers)
```

```
    ' Fill Combo1.
    With Me.C1Combo1
 .DataMode = C1.Win.C1List.DataModeEnum.AddItem
 .AddItem("Company")
 .AddItem("Contacted")
 .AddItem("CustType")
 .AddItem("FirstName")
 .AddItem("LastName")
 .AddItem("Phone")
 .AddItem("UserCode")
 .SelectedIndex = 0
    End With

    ' Fill Combo2.
    With Me.C1Combo2
 .DataMode = C1.Win.C1List.DataModeEnum.AddItem
 .AddItem("Partial Include")
 .AddItem("Equal")
 .AddItem("Less Than")
 .AddItem("Greater Than")
 .SelectedIndex = 0
    End With

    ' Fill Combo3.
    With Me.C1Combo3
 .DataMode = C1.Win.C1List.DataModeEnum.AddItem
 .AddItem("Start From Beginning")
 .AddItem("Start After Current Row")
 .SelectedIndex = 0
    End With

    Me.TextBox1.Text = ""
End Sub
```

To write code in C#

C#

```
private void Form1_Load( System.object sender, System.EventArgs e)
{
    // The following line is added by Visual Studio 2005.
    this.CustomersTableAdapter.Fill(this.DsCustomers.Customers);

    // Fill Combo1.
    this.c1Combo1.DataMode = C1.Win.C1List.DataModeEnum.AddItem;
    this.c1Combo1.AddItem("Company");
    this.c1Combo1.AddItem("Contacted");
    this.c1Combo1.AddItem("CustType");
    this.c1Combo1.AddItem("FirstName");
    this.c1Combo1.AddItem("LastName");
    this.c1Combo1.AddItem("Phone");
```

```
    this.c1Combo1.AddItem("UserCode");
    this.c1Combo1.SelectedIndex = 0;

    // Fill Combo2.
    this.c1Combo2.DataMode = C1.Win.C1List.DataModeEnum.AddItem;
    this.c1Combo2.AddItem("Partial Include");
    this.c1Combo2.AddItem("Equal");


    this.c1Combo2.AddItem("Less Than");
    this.c1Combo2.AddItem("Greater Than");
    this.c1Combo2.SelectedIndex = 0;

    // Fill Combo3.
    this.c1Combo3.DataMode = C1.Win.C1List.DataModeEnum.AddItem;
    this.c1Combo3.AddItem("Start From Beginning");
    this.c1Combo3.AddItem("Start After Current Row");
    this.c1Combo3.SelectedIndex = 0;

    this.TextBox1.Text = "";
}
```

6. To handle the **Button1_Click** event, add the following code:

   To write code in Visual Basic

Visual Basic

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    ' If there is no search string, do nothing.
    If Me.TextBox1.Text.Trim() = "" Then
Exit Sub
    End If
    Select Case Me.C1Combo2.Text
 Case "Partial Include"
     matchCompare = C1.Win.C1List.MatchCompareEnum.PartiallyEqual
 Case "Equal"
     matchCompare = C1.Win.C1List.MatchCompareEnum.Equal
 Case "Less Than"
     matchCompare = C1.Win.C1List.MatchCompareEnum.LessThan
 Case "Greater Than"
     matchCompare = C1.Win.C1List.MatchCompareEnum.GreaterThan
    End Select

    Select Case Me.C1Combo3.Text
 Case "Start From Beginning"
     Me.fromStart = True
 Case "Start After Current Row"
     Me.fromStart = False
    End Select
```

```
    Dim found As Integer
    If Me.fromStart Then
 found = Me.C1List1.Find(Me.TextBox1.Text.Trim(), Me.matchCompare, True, 0,
Me.C1Combo1.Text)
    Else
 found = Me.C1List1.Find(Me.TextBox1.Text.Trim(), Me.matchCompare, False,
Me.C1List1.Bookmark, Me.C1Combo1.Text)
    End If

    If found >= 0 Then
 Me.C1List1.SelectedIndex = found
    Else
 MessageBox.Show("No further record is found", "List")
    End If
End Sub
```

To write code in C#

C#

```
private void Button1_Click( System.object sender, System.EventArgs e)
{
    // If there is no search string, do nothing.
    if ( this.TextBox1.Text.Trim() == "" )
    {
 return;
    }

    switch (this.c1Combo2.Text)
    {
 case "Partial Include":
     matchCompare = C1.Win.C1List.MatchCompareEnum.PartiallyEqual;
     break;
 case "Equal":
     matchCompare = C1.Win.C1List.MatchCompareEnum.Equal;
     break;
 case "Less Than":
     matchCompare = C1.Win.C1List.MatchCompareEnum.LessThan;
     break;
 case "Greater Than":
     matchCompare = C1.Win.C1List.MatchCompareEnum.GreaterThan;
 break;
    }

    switch (this.C1Combo3.Text)
    {
 case "Start From Beginning":
     this.fromStart = true;
     break;
 case "Start After Current Row":
     this.fromStart = false;
     break;
```

```
    }

    int found;
    if ( this.fromStart )
    {
 found = this.C1List1.Find(this.TextBox1.Text.Trim(), this.matchCompare, true,
0, this.C1Combo1.Text);
    }
    else
    {
 found = this.c1List1.Find(this.TextBox1.Text.Trim(),
 this.matchCompare, false, this.C1List1.Bookmark, this.C1Combo1.Text);
    }
    if ( found >= 0 )
    {
 this.c1List1.SelectedIndex = found;
    }
    else
    {
 MessageBox.Show("No further record is found", "List");
    }
}
```

7. Finally, add the code below to clear the selected rows:

To write code in Visual Basic

Visual Basic

```
Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
    Me.C1List1.ClearSelected()
End Sub
```

To write code in C#

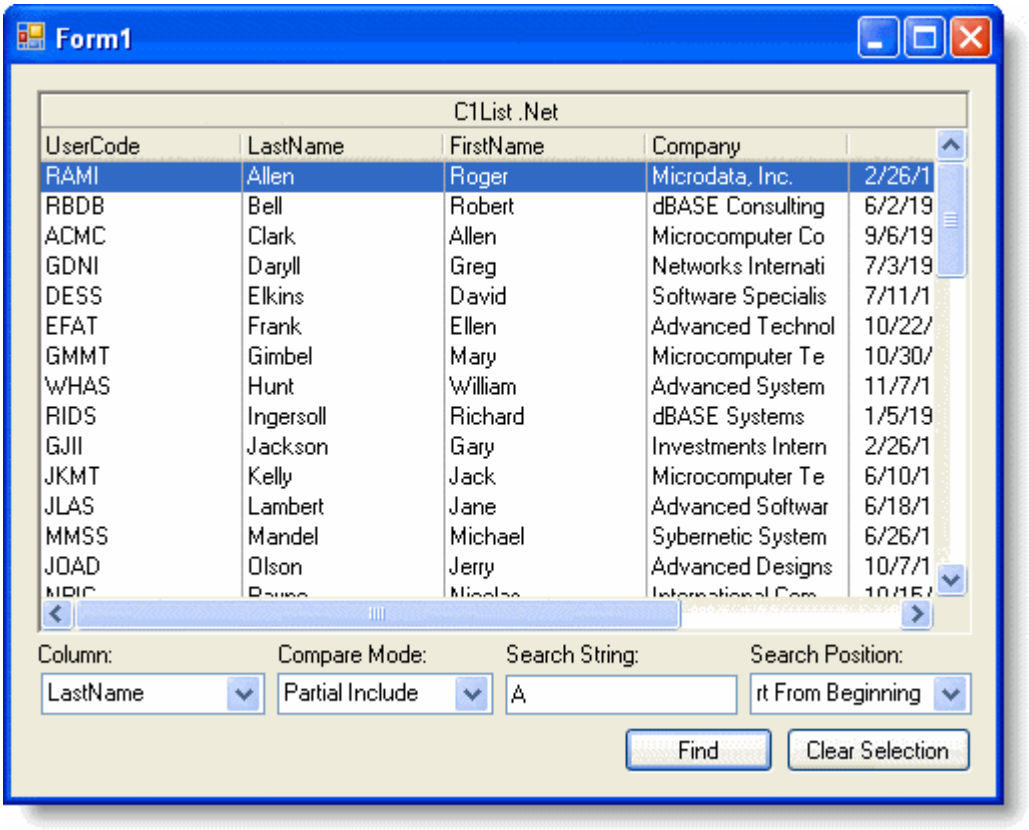
C#

```
private void Button2_Click( System.object sender, System.EventArgs e)
{
    this.c1List1.ClearSelected();
}
```

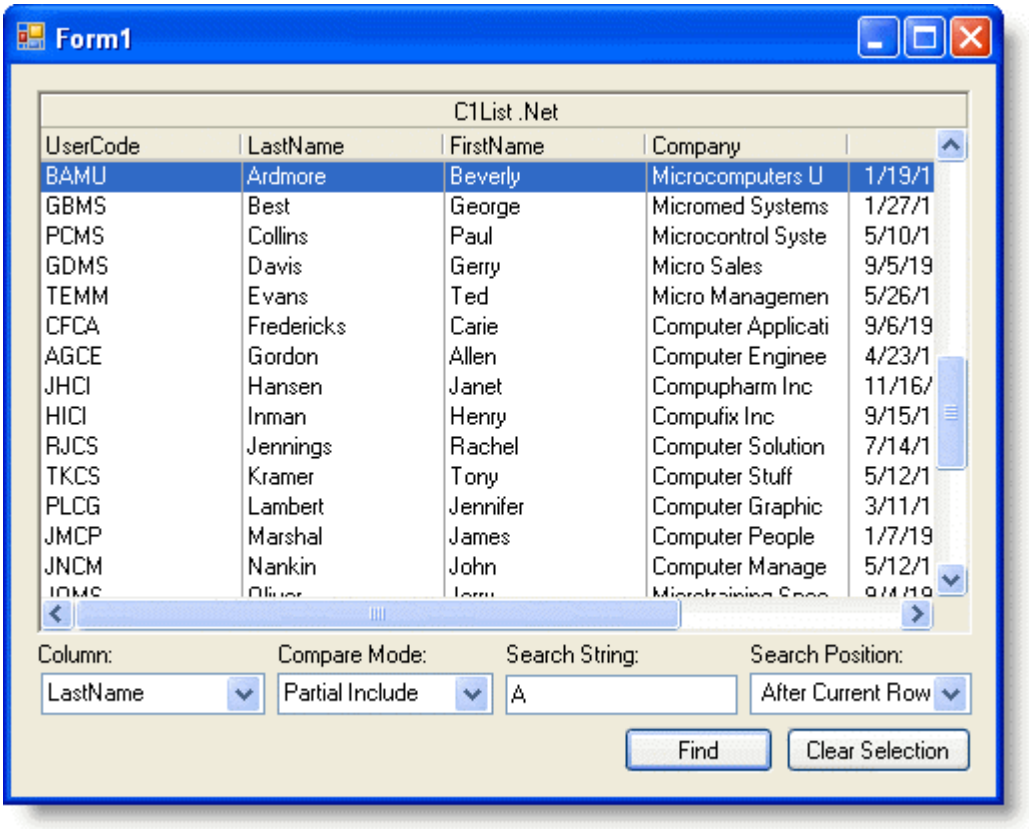
Run the program and observe the following:

Set the **Column** box to **LastName** and the **Compare Mode** box to **Partial Include**. Then place a letter in the **Search String** box and press the **Find** button. Notice how the first item in the *LastName* column beginning with the letter you entered is found.

Note: When C1Combo has the focus, press F4 to open the drop-down box.



Next, set the **Search Pos** box to **Start After Current Row** and press the **Find** button. Notice how the next item in the *LastName* column beginning with the letter you entered is found.



The Find method will also let you search numeric strings. Set the **Column** box to **Contacted** and the **Compare Mode** box to **Partial Include**. Then place a date from the *Contacted* column in the **Search String** box and press the **Find**

button. Notice how the first item in the *Contacted* column with the date you entered is found.

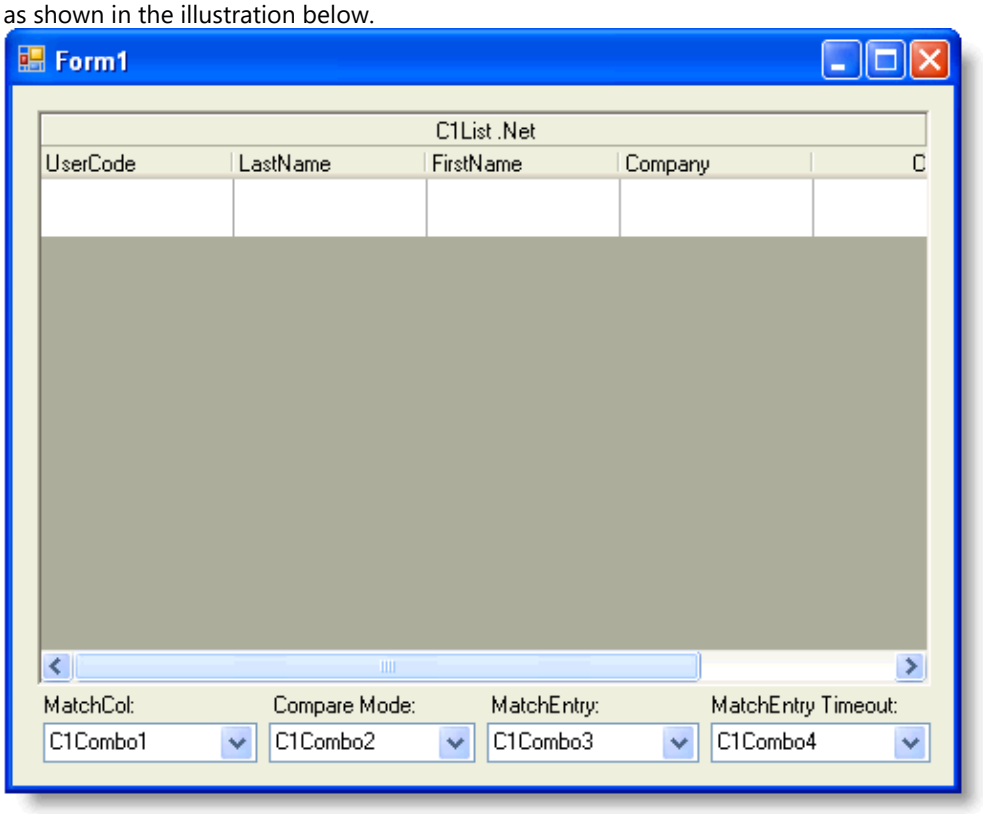
This concludes the tutorial.

Tutorial 18 Search Using MatchEntry

In this tutorial, you will learn how to search a record from a list using the MatchEntry, MatchCol, MatchCompare and MatchEntryTimeout properties.

1. Create a new .NET project.
2. Double-click the **C1List** icon in Visual Studio's Toolbox to add it to the Form.

   Note: See Adding the C1List Components to a Project for information on adding a component to the Toolbox.
3. Based on the following illustration, place four **C1Combo** boxes (C1Combo1, 2, 3 and 4), and four labels on the form as shown in the illustration below.



4. Go to the **DataSource** property for C1List and select **Add Project Data Source** from the drop-down. In the adapter's Data Source Configuration Wizard, either select a connection to C1NWind.mdb or create a new connection to this database. On the Choose your database objects page of the wizard, select all fields in the **Customers** table and type "DsCustomers" into the **DataSet name** box, and then finish out the wizard.
5. In the **Form_Load** event, add the following code to fill the dataset and the combo boxes:

   To write code in Visual Basic

Visual Basic

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
    ' The following line is added by Visual Studio 2005.
    Me.CustomersTableAdapter.Fill(Me.DsCustomers.Customers)
    ' Fill Combo1.
    With Me.C1Combo1
        .DataMode = C1.Win.C1List.DataModeEnum.AddItem
```

```
        .AddItem("Current Mouse Position")
        .AddItem("DisplayMember")
        .AddItem("Current Selected Col")
        .SelectedIndex = 0
    End With

    ' Fill Combo2.
    With Me.C1Combo2
        .DataMode = C1.Win.C1List.DataModeEnum.AddItem
        .AddItem("Partial Include")
        .AddItem("Equal")
        .AddItem("Less Than")
        .AddItem("Greater Than")
        .SelectedIndex = 0
    End With

    ' Fill Combo3.
    With Me.C1Combo3
        .DataMode = C1.Win.C1List.DataModeEnum.AddItem
        .AddItem("Standard")
        .AddItem("Extended")
        .SelectedIndex = 0
    End With

    ' Fill Combo4.
    With Me.C1Combo4
        .DataMode = C1.Win.C1List.DataModeEnum.AddItem
        .AddItem("2000")
        .AddItem("1000")
        .AddItem("3000")
        .SelectedIndex = 0
    End With

    ' Start.
    With Me.C1List1
        .MatchCol = C1.Win.C1List.MatchColEnum.CurrentMousePos
        .MatchCompare = C1.Win.C1List.MatchCompareEnum.PartiallyEqual
        .MatchEntry = C1.Win.C1List.MatchEntryEnum.Standard
        .MatchEntryTimeout = 2000
    End With
End Sub
```

To write code in C#

C#

```
private void Form1_Load( System.object sender, System.EventArgs e)
{
    // The following line is added by Visual Studio 2005.
    this.OleDbDataAdapter1.Fill(this.DsCustomers1);

    // Fill Combo1.
    this.c1Combo1.DataMode = C1.Win.C1List.DataModeEnum.AddItem;
    this.c1Combo1.AddItem("Current Mouse Position");
    this.c1Combo1.AddItem("DisplayMember");
```

```
    this.c1Combo1.AddItem("Current Selected Col");
    this.c1Combo1.SelectedIndex = 0;

    // Fill Combo2.
    this.c1Combo2.DataMode = C1.Win.C1List.DataModeEnum.AddItem;
    this.c1Combo2.AddItem("Partial Include");
    this.c1Combo2.AddItem("Equal");
    this.c1Combo2.AddItem("Less Than");
    this.c1Combo2.AddItem("Greater Than");
    this.c1Combo2.SelectedIndex = 0;

    // Fill Combo3.
    this.c1Combo3.DataMode = C1.Win.C1List.DataModeEnum.AddItem;
    this.c1Combo3.AddItem("Standard");
    this.c1Combo3.AddItem("Extended");
    this.c1Combo3.SelectedIndex = 0;

    // Fill Combo4.
    this.c1Combo4.DataMode = C1.Win.C1List.DataModeEnum.AddItem;
    this.c1Combo4.AddItem("2000");
    this.c1Combo4.AddItem("1000");
    this.c1Combo4.AddItem("3000");
    this.c1Combo4.SelectedIndex = 0;

    // Start.
    this.c1List1.MatchCol = C1.Win.C1List.MatchColEnum.CurrentMousePos;
    this.c1List1.MatchCompare = C1.Win.C1List.MatchCompareEnum.PartiallyEqual;
    this.c1List1.MatchEntry = C1.Win.C1List.MatchEntryEnum.Standard;
    this.c1List1.MatchEntryTimeout = 2000;
}
```

6. Add the following code for the **Change** event of each of the combo boxes:

To write code in Visual Basic

Visual Basic

```
Private Sub C1Combo1_Change(ByVal sender As Object, ByVal e As System.EventArgs)
Handles C1Combo1.Change
    Select Case Me.C1Combo1.Text
        Case "Current Mouse Position"
            Me.C1List1.MatchCol = C1.Win.C1List.MatchColEnum.CurrentMousePos
        Case "ListField"
            Me.C1List1.MatchCol = C1.Win.C1List.MatchColEnum.DisplayMember
        Case "Current Selected Col"
            Me.C1List1.MatchCol = C1.Win.C1List.MatchColEnum.CurrentSelectedCol
    End Select
End Sub

Private Sub C1Combo2_Change(ByVal sender As Object, ByVal e As System.EventArgs)
Handles C1Combo2.Change
    Select Case Me.C1Combo2.Text
        Case "Partial Include"
            Me.C1List1.MatchCompare = C1.Win.C1List.MatchCompareEnum.PartiallyEqual
        Case "Equal"
```

```
            Me.C1List1.MatchCompare = C1.Win.C1List.MatchCompareEnum.Equal
        Case "Less Than"
            Me.C1List1.MatchCompare = C1.Win.C1List.MatchCompareEnum.LessThan
        Case "Greater Than"
            Me.C1List1.MatchCompare = C1.Win.C1List.MatchCompareEnum.GreaterThan
    End Select
End Sub

Private Sub C1Combo3_Change(ByVal sender As Object, ByVal e As System.EventArgs)
Handles C1Combo3.Change
    Select Case Me.C1Combo3.Text
        Case "Standard"
            Me.C1List1.MatchEntry = C1.Win.C1List.MatchEntryEnum.Standard
        Case "Extended"
            Me.C1List1.MatchEntry = C1.Win.C1List.MatchEntryEnum.Extended
    End Select
End Sub

Private Sub C1Combo4_Change(ByVal sender As Object, ByVal e As System.EventArgs)
Handles C1Combo4.Change
    Select Case Me.C1Combo4.Text
        Case "2000"
            Me.C1List1.MatchEntryTimeout = 2000
        Case "1000"
            Me.C1List1.MatchEntryTimeout = 1000
        Case "3000"
            Me.C1List1.MatchEntryTimeout = 3000
    End Select
End Sub
```

To write code in C#

C#

```
private void C1Combo1_Change( object sender, System.EventArgs e)
{
    switch (this.C1Combo1.Text)
    {
        case "Current Mouse Position":
            this.c1List1.MatchCol = C1.Win.C1List.MatchColEnum.CurrentMousePos;
            break;
        case "ListField":
            this.c1List1.MatchCol = C1.Win.C1List.MatchColEnum.DisplayMember;
            break;
        case "Current Selected Col":
            this.c1List1.MatchCol = C1.Win.C1List.MatchColEnum.CurrentSelectedCol;
            break;
    }
}

private void C1Combo2_Change( object sender,  System.EventArgs e)
{
    switch (this.C1Combo2.Text)
    {
        case "Partial Include":
```

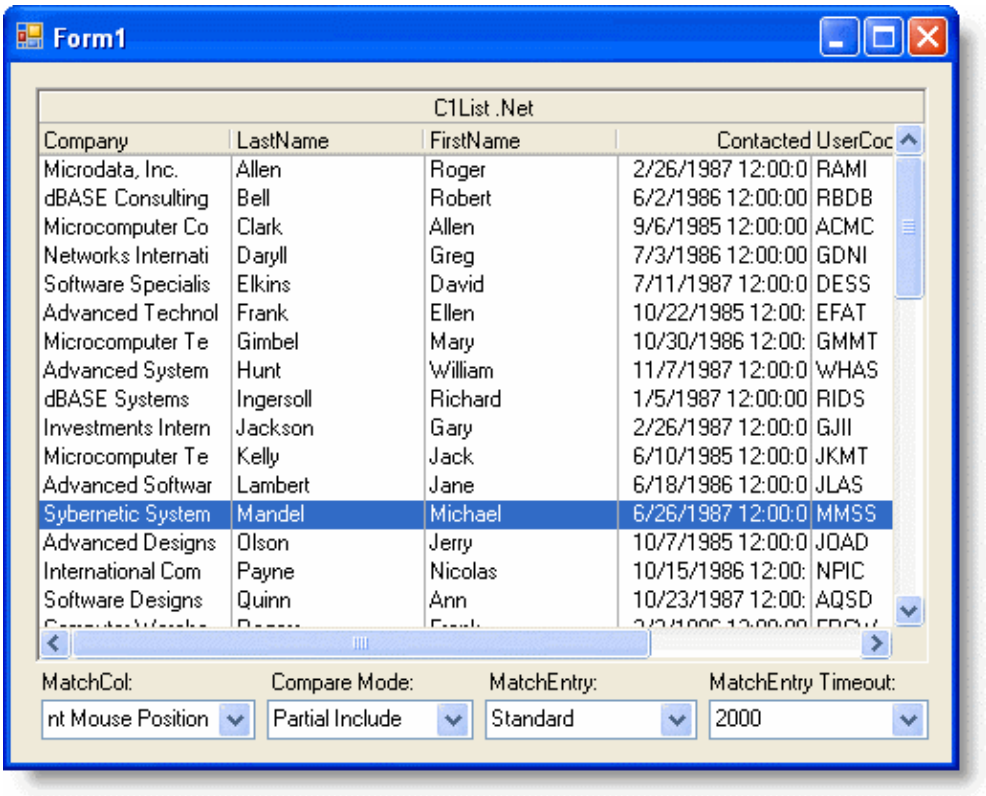
```
            this.c1List1.MatchCompare =
C1.Win.C1List.MatchCompareEnum.PartiallyEqual;
            break;
        case "Equal":
            this.c1List1.MatchCompare = C1.Win.C1List.MatchCompareEnum.Equal;
break;
        case "Less Than":
            this.c1List1.MatchCompare = C1.Win.C1List.MatchCompareEnum.LessThan;
            break;
        case "Greater Than":
            this.c1List1.MatchCompare = C1.Win.C1List.MatchCompareEnum.GreaterThan;
            break;
    }
}

private void C1Combo3_Change( object sender, System.EventArgs e)
{
    switch (this.C1Combo3.Text)
    {
        case "Standard";
            this.c1List1.MatchEntry = C1.Win.C1List.MatchEntryEnum.Standard;
            break;
        case "Extended";
            this.c1List1.MatchEntry = C1.Win.C1List.MatchEntryEnum.Extended;
            break;
    }
}

private void C1Combo4_Change( object sender, System.EventArgs e)
{
    switch (this.C1Combo4.Text)
    {
        case "2000";
            this.c1List1.MatchEntryTimeout = 2000;
            break;
        case "1000";
            this.c1List1.MatchEntryTimeout = 1000;
            break;
        case "3000";
            this.c1List1.MatchEntryTimeout = 3000;
            break;
    }
}
```

Run the program and observe the following:

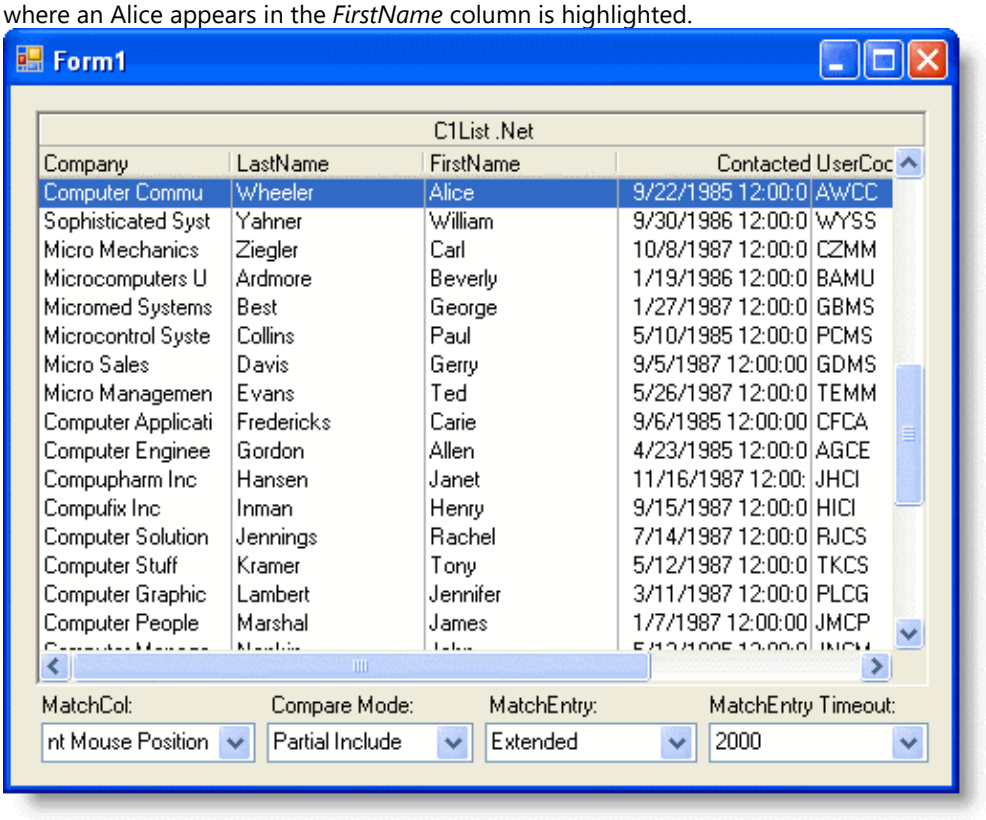
- Set the **MatchCol** box to **Current Mouse position**.

  Note: When C1Combo has the focus, press F4 to open the drop-down box.

- Place your mouse in the *Company* column, select a row and type the letter S. The next row where an S appears in the Company name is highlighted. If you type an S again, the next row where an S appears in the Company name after that becomes highlighted and so on.



- Set the **MatchEntry** box to **Extended**. Place your mouse in the *FirstName* column, and type "Alice". The next row where an Alice appears in the *FirstName* column is highlighted.



This concludes the tutorial.

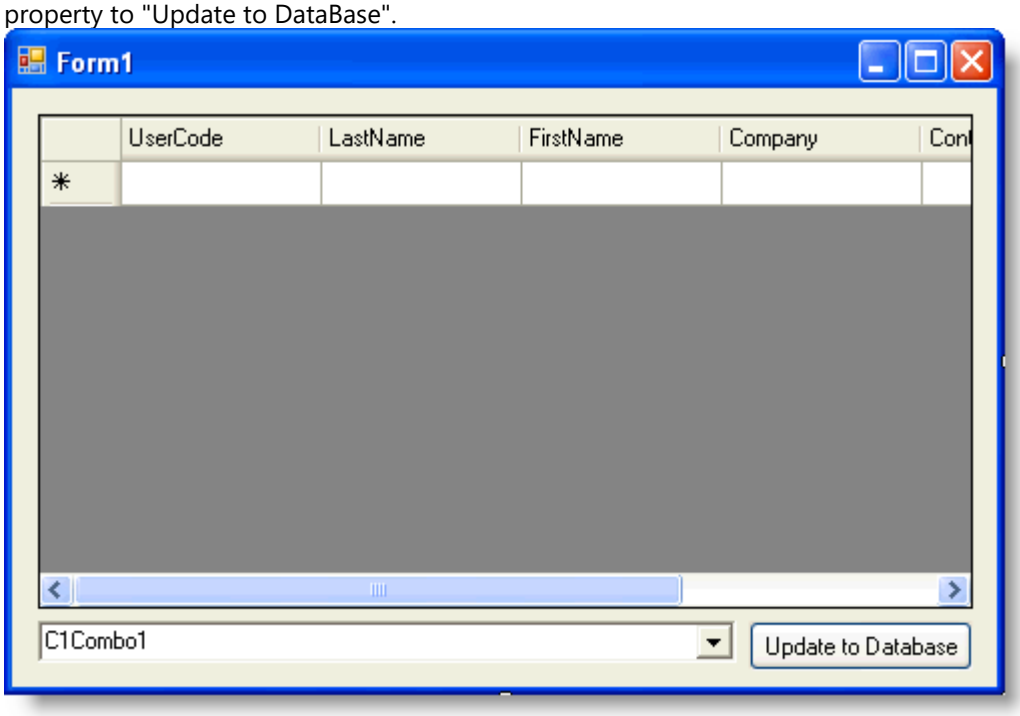
Tutorial 19 Using the LookUp Feature

In this tutorial, you will learn how to use the LookUp feature to change the value of a C1Combo box.

In the database C1NWind.mdb there are two tables: **CustType** and **Customers**. **CustType** has two fields: *TypeDesc* and *TypeId*. **Customers** has a field called *CustType* corresponding to the *TypeId* in the **CustType** table. In other words, the *CustType* field in the **Customers** table is the foreign key to the **CustType** table. When users want to navigate or update the **Customers** table, they usually want more meaningful words than Ids. In our case, they want the *TypeDesc* instead of the *TypeId*. This is where the LookUp feature comes into play.

1. Create a new .NET project.
2. Double-click the **C1List** icon in Visual Studio's Toolbox to add it to the Form.

   Note: See Adding the C1List Components to a Project for information on adding a component to the Toolbox.
3. Add a **C1Combo** control, a DataGrid object, and a button control to the form. Change the button's **Text** property to "Update to DataBase".



4. Go to the **DataSource** property for C1List and select **Add Project Data Source** from the drop-down. In the adapter's Data Source Configuration Wizard, either select a connection to C1NWind.mdb or create a new connection to this database. On the Choose your database objects page of the wizard, select all fields in the **Customers** table and all fields in the **CustType** table, and type "DataSet1" into the **DataSet name** box, and then finish out the wizard.
5. Select **DataGrid1** and set its **DataSource** property to **DataSet1.Customers** in the Properties panel. Select **C1Combo1** and set its DataSource property to **DataSet1.CustType**. Set its ValueMember property to **TypeId**.
6. Add the following code in the **Form_Load** event:

   To write code in Visual Basic

Visual Basic

```
' The following two lines are added by Visual Studio 2005.
Me.CustomersTableAdapter.Fill(Me.DataSet1.Customers)
Me.CustTypeTableAdapter.Fill(Me.DataSet1.CustType)

' Add a relation.
Dim parentCol, childCol As DataColumn
Dim relCustType As DataRelation
```

```
parentCol = DataSet1.Tables("CustType").Columns("TypeId")
childCol = DataSet1.Tables("Customers").Columns("CustType")
relCustType = New DataRelation("CustomersType", parentCol, childCol)
DataSet1.Relations.Add(relCustType)

' Bind the SelectedValue property.
Me.C1Combo1.DataBindings.Add("SelectedValue", DataSet1.Tables("Customers"),
"CustType")
```

To write code in C#

C#

```
// The following two lines are added by Visual Studio 2005.
this.CustomersTableAdapter.Fill(this.DataSet1.Customers);
this.CustTypeTableAdapter.Fill(this.DataSet1.CustType);

// Add a relation.
DataColumn parentCol, childCol;
DataRelation relCustType;
parentCol = DataSet1.Tables["CustType"].Columns["TypeId"];
childCol = DataSet1.Tables["Customers"].Columns["CustType"];
relCustType = new DataRelation("CustomersType", parentCol, childCol);
DataSet1.Relations.Add(relCustType);

// Bind the SelectedValue property.
this.c1Combo1.DataBindings.Add("SelectedValue", DataSet1.Tables["Customers"],
"CustType");
```

7. Add the following code to the **Button1_Click** event:

To write code in Visual Basic

Visual Basic

```
Me.CustomersTableAdapter.Connection.Open()
Dim ds As DataSet
ds = DataSet1.GetChanges(DataRowState.Modified)

If (Not ds Is Nothing) Then
    Try
        Me.CustomersTableAdapter.Update(ds)
        Me.CustTypeTableAdapter.Update(ds)
    Catch eU As Exception
        MessageBox.Show(eU.Message)
    End Try
End If

Me.CustomersTableAdapter.Connection.Close()
```

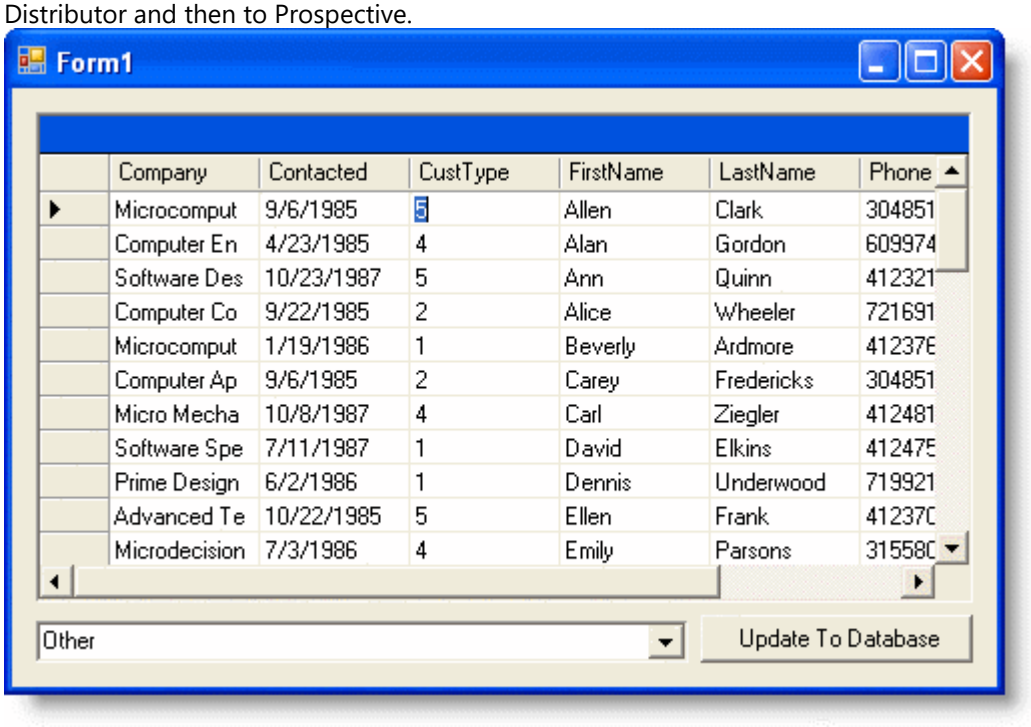
To write code in C#

C#

```
this.CustomersTableAdapter.Connection.Open();
DataSet ds;
ds = DataSet1.GetChanges(DataRowState.Modified);

if (ds != null)
{
    try
    {
        this.CustomersTableAdapter.Update(ds);
        this.CustTypeTableAdapter.Update(ds);
    }
    catch (Exception eU)
    {
        MessageBox.Show(eU.Message);
    }
}

this.CustomersTableAdapter.Connection.Close();
```

Run the program and observe the following:

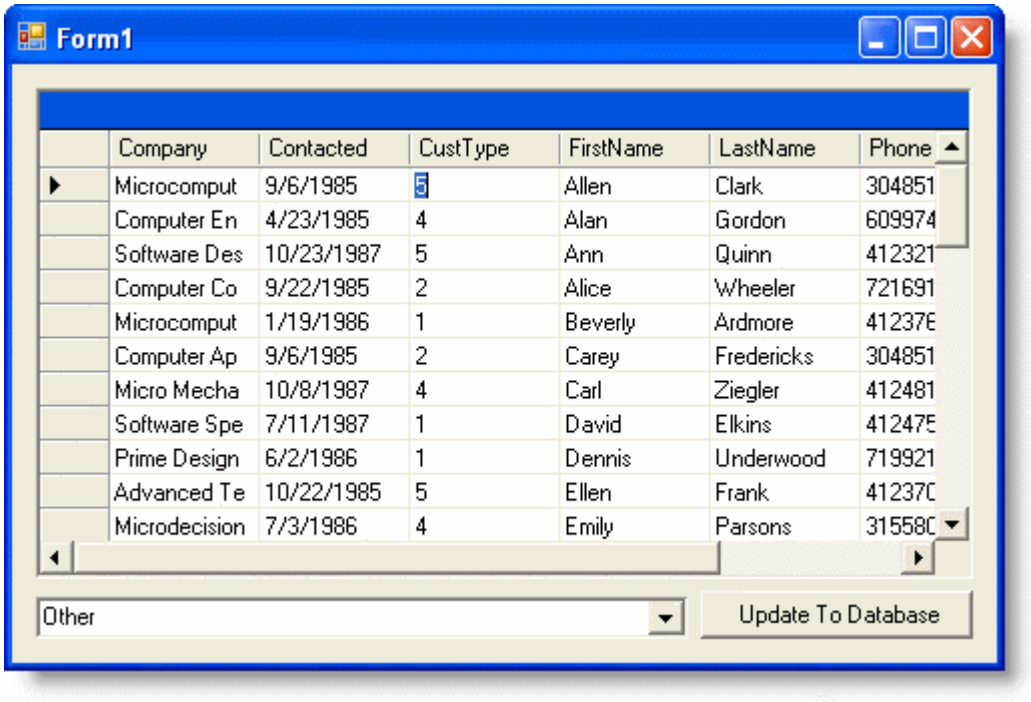
- When you click the second row and then third row, the C1Combo box value will change from Normal to Distributor and then to Prospective.



- Go to the first row with the C1Combo box value changed to Normal and click the arrow button from the combo box.

> **Note**: When C1Combo has the focus, press F4 to open the drop-down box.

- Choose **Other** and move the DataGrid to second row. Notice that the value in the *CustType* field in the first row changed from 2 to 5. You can change the CustType value for others row using the same steps. Note that the changed value is not in the database yet (only changed to the dataset now). If you want to update the changed value to database, click the **Update To Database** button, and the value will be updated to the database.

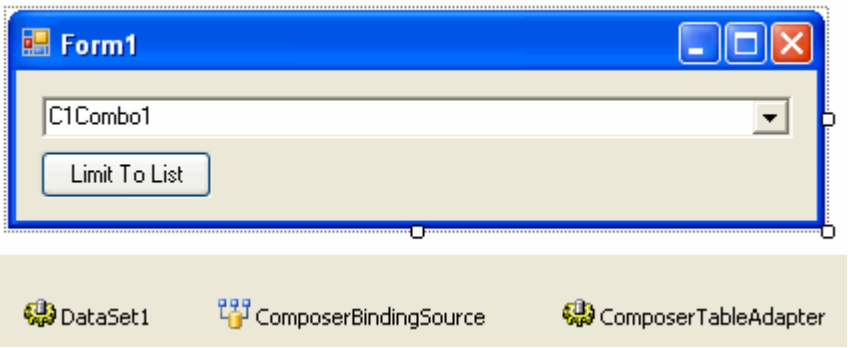


This concludes the tutorial.

Tutorial 20 - Using the LimitToList Feature

In this tutorial, you will learn how to use the **LimitToList** feature to prevent users from entering an item which does not appear in the combo box list.

1. Follow steps 1 through 4 of Tutorial 2 - Binding C1Combo to a DataSet to create a project with a C1Combo control bound to a Data Set.
2. Select the C1Combo control and set its DisplayMember property to **First**.
3. Add one button with the text "Limit To List" and one label with the text as it appears in the label below:



4. Add the following code to the **Form_Load** event:

To write code in Visual Basic

Visual Basic

```
' The following line is added by Visual Studio 2005.
Me.ComposerTableAdapter.Fill(Me.DataSet1.Composer)

' Add some settings.
With Me.C1Combo1
```

```
    .SelectedIndex = 0
    .DropdownWidth = 500
    .MaxDropDownItems = 10
End With
```

To write code in C#

```
C#
// The following line is added by Visual Studio 2005.
this.ComposerTableAdapter.Fill(this.DataSet1.Composer);

// Add some settings.
this.c1Combo1.SelectedIndex = 0;
this.c1Combo1.DropdownWidth = 500;
this.c1Combo1.MaxDropDownItems = 10;
```

5. Add the following code to the **Button1_Click** event:

To write code in Visual Basic

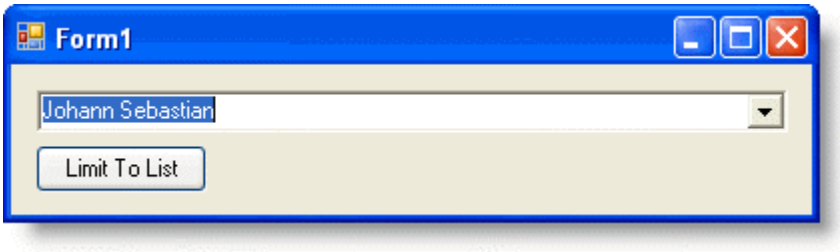
```
Visual Basic
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click

    ' Prevent MisMatch.
    With Me.C1Combo1
        .LimitToList = True
        .AutoCompletion = True
        .SuperBack = True
    End With
End Sub
```

To write code in C#

```
C#
private void Button1_Click( System.object sender, System.EventArgs e)
{
    // Prevent MisMatch.
    this.c1Combo1.LimitToList = true;
    this.c1Combo1.AutoCompletion = true;
    this.c1Combo1.SuperBack = true;
}
```

Run the program and observe the following:

Click the **Limit To List** button. You cannot change the text to something that does not appear in the combo box list.



This concludes the tutorial.

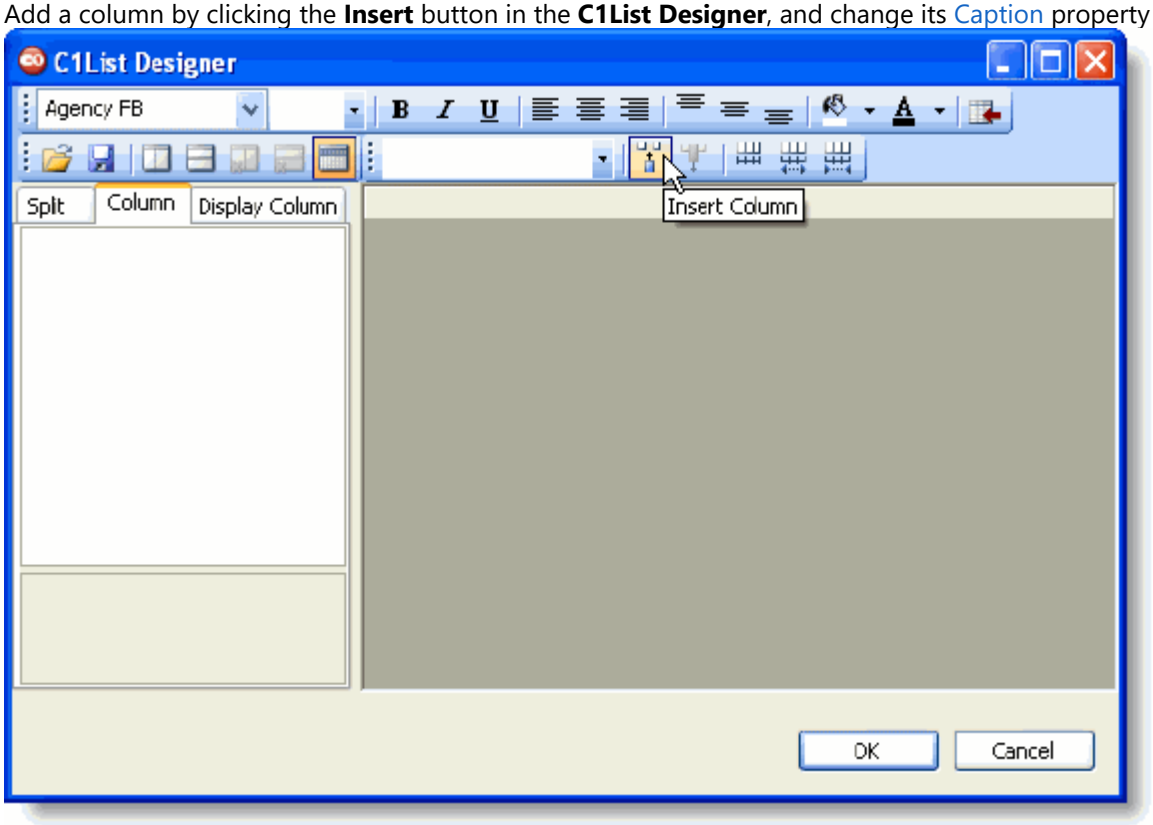
Tutorial 21 Design-Time Support for C1Combo's AddItem Mode

Design-time support for the **AddItem** mode of C1List is now available. The following steps demonstrate how to add items and set a layout at design time.

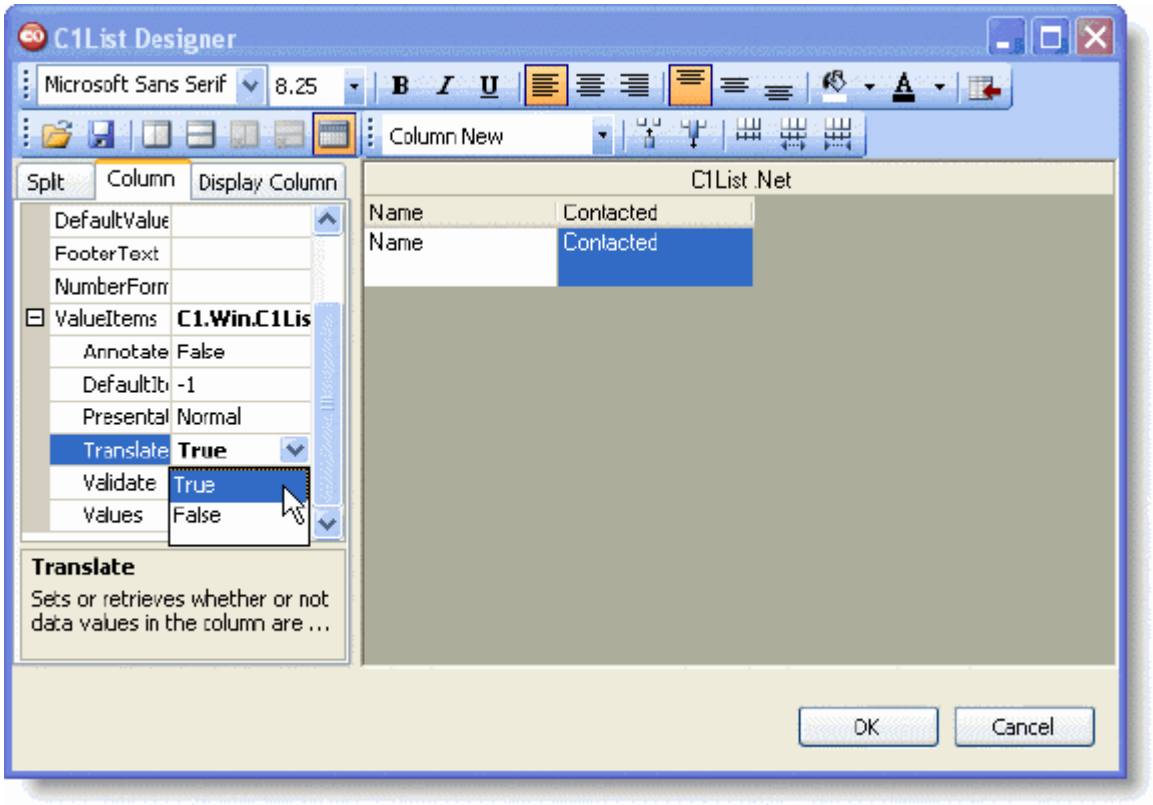
1. Create a new project.
2. Double-click the **C1Combo** icon in Visual Studio's Toolbox to add it to the Form.

   > **Note**: See Adding the C1List Components to a Project for information on adding a component to the Toolbox.

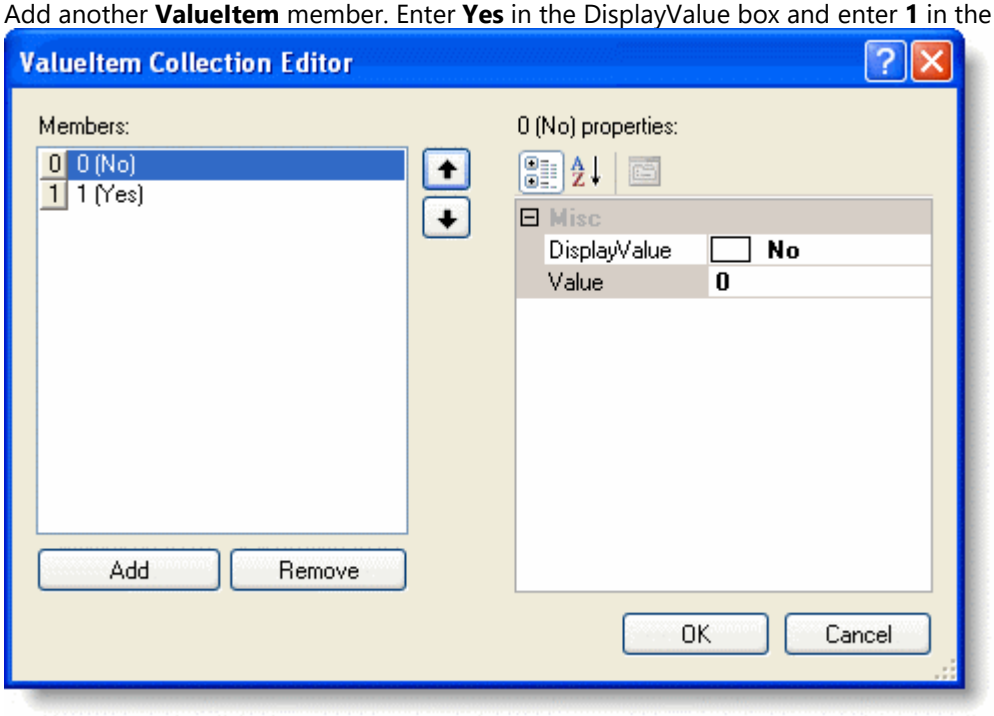
3. Select the **DataMode** property in the Properties window and change it to **AddItem**.
4. Open the **C1List Designer** by clicking the **ellipsis** button next to the Columns property in the Properties window.
5. Add a column by clicking the **Insert** button in the **C1List Designer**, and change its Caption property to **Name**.



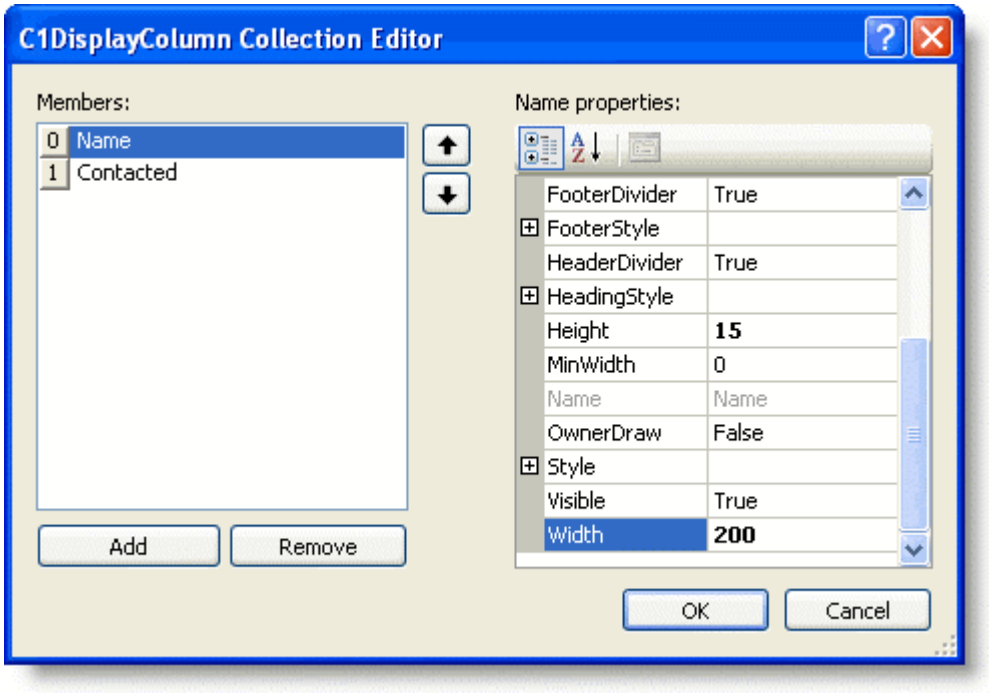
6. Add another column and change its Caption property to **Contacted**.
7. Expand the Contacted column's ValueItems property by clicking the ValueItems node, and set the Translate property to **True**.



8. With ValueItems still expanded, click the **ellipsis** button next to the Values property, and the **ValueItem Collection Editor** appears.
9. Click the **Add** button and the first **ValueItem** member appears in the left pane. In the right pane enter **No** in the DisplayValue box and enter **0** in the **Value** box.
10. Add another **ValueItem** member. Enter **Yes** in the DisplayValue box and enter **1** in the **Value** box.



11. Open the **Split Collection Editor** by clicking the **ellipsis** button next to the Splits property in the Properties window.
12. In the right pane, click the **ellipsis** button next to the **DisplayColumns** property. The **DisplayColumns Collection Editor** appears.
13. Select the *Name* column: set its Visible property to **True** and its Width to 200.



14. Select the **Contacted** column and set its Visible property to **True**.
15. Click **OK** to close the editor windows.
16. Add the following code to the **Form_Load** event:

To write code in Visual Basic

Visual Basic

```
With Me.C1Combo1
    .HoldFields()
    .AddItem("Greg Daryll;0")
    .AddItem("Jane Lambert;1")
    .AddItem("Allen Clark;1")
    .AddItem("David Elkins;1")
    .AddItem("Carl Ziegler;0")
    .AddItem("William Yahner;1")
End With
```

To write code in C#

C#

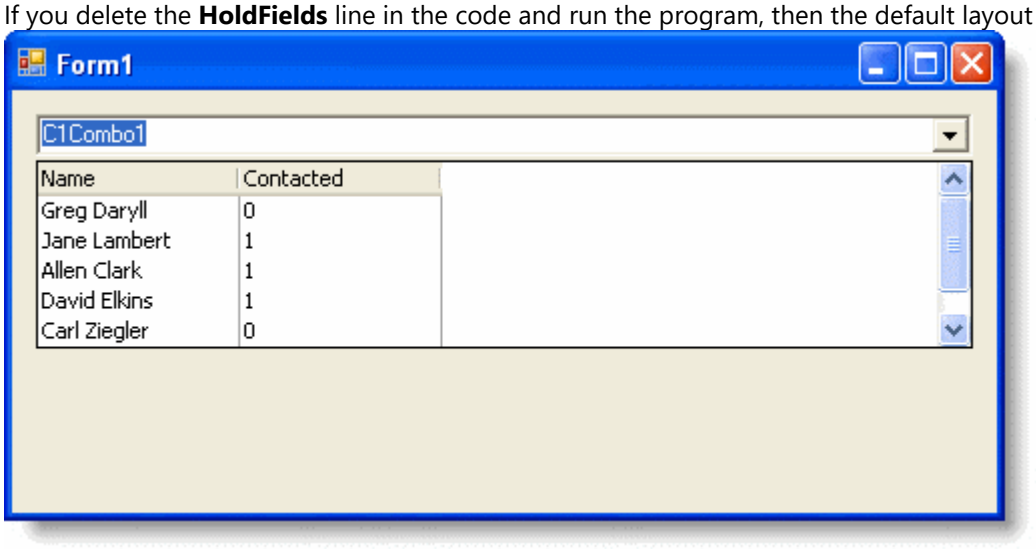
```
this.c1Combo1.HoldFields();
this.c1Combo1.AddItem("Greg Daryll;0");
this.c1Combo1.AddItem("Jane Lambert;1");
this.c1Combo1.AddItem("Allen Clark;1");
this.c1Combo1.AddItem("David Elkins;1");
this.c1Combo1.AddItem("Carl Ziegler;0");
this.c1Combo1.AddItem("William Yahner;1");
```

Run the program and observe the following:

- The design-time layout is used. The width for the first column is 200, and anything with a value of 0 is changed

to **False**, or **No**. Anything with a value of 1 is changed to **True**, or **Yes**.

- If you use the up and down arrow keys to navigate through the list, the item currently selected appears in the combo box.
- If you delete the **HoldFields** line in the code and run the program, then the default layout is used.



Note: For faster data processing, use the AddItem method.

For example, replace the code entered in the **Form_Load** event with the following code:

To write code in Visual Basic

Visual Basic

```
With Me.C1Combo1
    .HoldFields()
    .SuspendBinding()
    .AddItem ("Greg Daryll;0")
    .AddItem ("Jane Lambert;1")
    .AddItem ("Allen Clark;1")
    .AddItem ("David Elkins;1")
    .AddItem ("Carl Ziegler;0")
    .AddItem ("William Yahner;1")
    .ResumeBinding()
    .Rebind()
End With
```

To write code in C#

C#

```
this.c1Combo1.HoldFields();
this.c1Combo1.SuspendBinding();
this.c1Combo1.AddItem ("Greg Daryll;0");
this.c1Combo1.AddItem ("Jane Lambert;1");
this.c1Combo1.AddItem ("Allen Clark;1");
this.c1Combo1.AddItem ("David Elkins;1");
this.c1Combo1.AddItem ("Carl Ziegler;0");
this.c1Combo1.AddItem ("William Yahner;1");
this.c1Combo1.ResumeBinding();
```

```
this.c1Combo1.Rebind();
```

In this tutorial only a small amount of data was used, so you may not notice a difference in the processing time. For larger amounts of data, use the AddItem method for a much faster processing time.

This concludes the tutorial.

Tutorial 22 Exporting C1List to Excel

The following steps demonstrate exporting the data from C1List to an Excel file:

1. Create a new WinForms application and add the following references by right-clicking **References** in the Solution Explorer and selecting **Add Reference** from the list:
   - C1.Win.C1List.dll
   - C1.C1Excel.dll
2. Add a C1List control to the Form by performing a drag-and-drop operation.
3. Locate the C1Excel control in your Toolbox and double-click to add the control to the application. The control will be added to the component tray below the form.
4. Locate the WinForms **Button** and **Checkbox** controls in the Toolbox and add them to the form below the C1List control. Select the **Button** control to show the available Properties in the Properties window.
5. Locate the **Text** property in the list and enter "Export" in the **Text** textbox.
6. Select the **Checkbox** control on your Form to show the available Properties in the Properties window.
7. Locate the **Text** property in the list and enter "Export Headers" in the **Text** textbox.
8. Double-click the C1List control to add a **FormLoad()** event to the code. The event will resemble the following code:

   To write code in Visual Basic

   Visual Basic
   ```
   Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
   System.EventArgs) Handles MyBase.Load
   ```

   To write code in C#

   C#
   ```
   private void Form1_Load(object sender, System.EventArgs e)
   ```
9. While still on the **Form1.cs** page, add the following statement to bind the C1List control to a Data Source:

   To write code in Visual Basic

   Visual Basic
   ```
   C1List1.DataSource = getTable()
   ```

   To write code in C#

   C#
   ```
   C1List1.DataSource = getTable();
   ```
10. Add the following function directly after the binding statement to create the DataTable to which the List is bound:

    To write code in Visual Basic

Visual Basic

```
Public Function getTable() As DataTable
    Dim dTable As New DataTable
    dTable.Columns.Add(New DataColumn("FName"))
    dTable.Columns.Add(New DataColumn("LName"))
    dTable.Columns.Add(New DataColumn("Date of Birth", GetType(DateTime)))
    dTable.Columns.Add(New DataColumn("Hire Date", GetType(DateTime)))

    Dim dr As DataRow = dTable.NewRow
    dr.ItemArray = New String() {"John", "Smith", Now.AddYears(-55),
Now.AddYears(-25)}
    dTable.Rows.Add(dr)

    dr = dTable.NewRow
    dr.ItemArray = New String() {"Dave", "Richardson", Now.AddYears(-32),
Now.AddYears(-5)}
    dTable.Rows.Add(dr)

    dr = dTable.NewRow
    dr.ItemArray = New String() {"Tom", "Franklin", Now.AddYears(-16),
Now.AddYears(-43), Now.AddYears(-16)}
    dTable.Rows.Add(dr)

    dr = dTable.NewRow
    dr.ItemArray = New String() {"Jason", "Lee", Now.AddYears(-30),
Now.AddYears(-5)}
    dTable.Rows.Add(dr)

    dr = dTable.NewRow
    dr.ItemArray = New String() {"Henry", "Mckinley", Now.AddYears(-19),
Now.AddYears(-1)}
    dTable.Rows.Add(dr)

    Return dTable
End Function
```

To write code in C#

C#

```
{
 DataTable dTable = new DataTable();
 dTable.Columns.Add(new DataColumn("FName"));
 dTable.Columns.Add(new DataColumn("LName"));
 dTable.Columns.Add(new DataColumn("Date of Birth", typeof(DateTime)));
 dTable.Columns.Add(new DataColumn("Hire Date", typeof(DateTime)));

 DataRow dr = dTable.NewRow();
 dr.ItemArray = new object[] {"John", "Smith", DateTime.Now.AddYears(-55),
DateTime.Now.AddYears(-25) };
    dTable.Rows.Add(dr);
```

```
 dr = dTable.NewRow();
 dr.ItemArray = new object[] { "Dave", "Richardson", DateTime.Now.AddYears(-32),
DateTime.Now.AddYears(-5) };
    dTable.Rows.Add(dr);

 dr = dTable.NewRow();
 dr.ItemArray = new object[] { "Tom", "Franklin", DateTime.Now.AddYears(-43),
DateTime.Now.AddYears(-16) };
 dTable.Rows.Add(dr);

 dr = dTable.NewRow();
 dr.ItemArray = new object[] { "Jason", "Lee", DateTime.Now.AddYears(-30),
DateTime.Now.AddYears(-5) };
 dTable.Rows.Add(dr);

 dr = dTable.NewRow();
 dr.ItemArray = new object[] { "Henry", "Mckinley", DateTime.Now.AddYears(-19),
DateTime.Now.AddYears(-1) };
 dTable.Rows.Add(dr);

    return dTable;
}
```

11. Add the following code below the code creating the Data Table to handle exporting the C1List content to an Excel file:

To write code in Visual Basic

Visual Basic

```
Public Sub ExportData(ByVal exportHeaderCondition As Boolean)

    Dim excelSheet As C1.C1Excel.XLSheet = C1XLBook1.Sheets(0)

    If excelSheet IsNot Nothing Then

        Dim excelRow As Integer = 0

        If exportHeaderCondition = True Then
            For i As Integer = 0 To C1List1.Columns.Count - 1
                excelSheet(excelRow, i).Value = C1List1.Columns(i).Caption
            Next
            excelRow += 1
        End If

        For row As Integer = 0 To C1List1.ListCount - 1
            For col As Integer = 0 To C1List1.Columns.Count - 1
                Dim cellValue As Object = C1List1.GetDisplayText(row, col)
                excelSheet(excelRow, col).Value = cellValue
            Next
            excelRow += 1
        Next
    End If
```

```
    C1XLBook1.Save("C:\GeneratedReport.xls")
    Process.Start("C:\GeneratedReport.xls")
End Sub
```

To write code in C#

C#

```
public void ExportData(bool exportHeaderCondition)
{
    C1.C1Excel.XLSheet excelSheet = C1XLBook1.Sheets[0];
    if (excelSheet != null)
    {
        int excelRow = 0;
        if ((exportHeaderCondition == true))
        {
            for (int i = 0; (i <= (C1List1.Columns.Count - 1)); i++)
            {
                excelSheet[excelRow, i].Value = C1List1.Columns[i].Caption;
            }
            excelRow++;

            for (int row = 0; (row <= (C1List1.ListCount - 1)); row++)
            {
                for (int col = 0; (col <= (C1List1.Columns.Count - 1)); col++)
                {
                    object cellValue = C1List1.GetDisplayText(row, col);
                    excelSheet[excelRow, col].Value = cellValue;
                }
                excelRow++;
            }
        }
        C1XLBook1.Save("C:\\GeneratedReport.xls");
        Process.Start("C:\\GeneratedReport.xls");
    }
```

12. Next, you will add the **Button_Click()** event to allow the Export button to export the list data. Add the following code to handle the **Button_Click()** event:

To write code in Visual Basic

Visual Basic

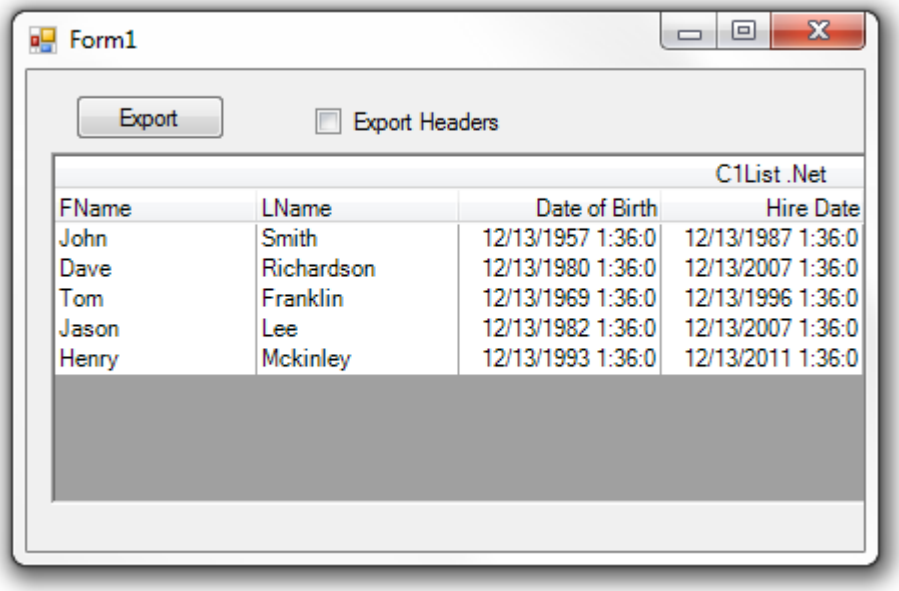
```
Private Sub exportButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles exportButton.Click

 If exportHeaderCheckBox.Checked Then
    ExportData(True)
 Else
    ExportData(False)
 End If

End Sub
```

To write code in C#

C#

```
private void exportButton_Click(object sender, System.EventArgs e)
{
    if (exportHeaderCheckBox.Checked)
    {
        ExportData(true);
    }
    else
    {
        ExportData(false);
    }
}
```

13. Press F5 to run your application. The application should appear as follows:



Note that you can export the Headers along with the rest of the data from the List by checking the Export Headers **Checkbox** .

How to Use Styles

List for WinForms uses a style model similar to that of Microsoft Word and Excel to simplify the task of customizing a list's appearance. A Style object is a combination of font, color, picture, and formatting information comprising the following properties:

Property	Description
Alpha	Gets or sets the alpha component when the style is rendered.
BackColor	Gets or sets the background color associated with a Style.
BackColor2	Gets or sets the background color associated with a Style.
BackgroundImage	Gets or sets the background image associated with a Style.
BackgroundPictureDrawMode	Gets or sets the rendering method for a BackgroundImage.
Borders	Gets the CellBorders associated with this Style.
Font	Gets or sets the Font associated with a Style.
ForeColor	Gets or sets the foreground color associated with a Style.
ForegroundImage	Gets or sets the foreground image associated with a style.
ForeGroundPicturePosition	Gets or sets the position that the **ForegroundImage** is rendered.
GammaCorrection	Gets or sets a value indicating whether gamma correction is enabled when a linear gradient style is rendered.
GradientMode	Specifies the direction of a linear gradient.
HorizontalAlignment	Gets or sets the horizontal text alignment.
Name	Gets or sets the name of the Style.
Padding	Gets or sets the spacing between cell content and its edges.
Trimming	Gets or sets the trim characters for a string that does not completely fit into a layout shape.
VerticalAlignment	Gets or sets the vertical text alignment.
WrapText	Gets or sets a value indicating whether text is word-wrapped when it does not fit into a layout shape.

Built-in Named Styles

When a list is first created, it has a collection of built-in named styles that control various aspects of its display. For example, the Heading style determines the attributes used to display column headers. In the designer, you can change the appearance of the list as a whole by modifying the built-in named styles in the **Style Collection Editor**. In code, the GridStyleCollection object provides access to the same set of named styles. Initially, all lists contain ten built-in styles, which control the display of the following list elements:

Style	Description
Normal	Data cells in unselected, unhighlighted rows
Heading	Column headers

Style	Description
Footer	Column footers
Caption	List and split caption bars
Selected	Data cells in selected rows
HighlightRow	Data cells in highlighted rows
EvenRow	Data cells in even numbered rows
OddRow	Data cells in odd numbered rows
RecordSelector	Data in the record selector column
Group	Group columns in list grouping area

A *selected row* is one whose bookmark has been added to the SelectedRowCollection, either in code or through user interaction. The term *highlighted row* refers to the current row when the ForeColor property is set to **HighlightText**.

The EvenRow and OddRow styles are used only when the AlternatingRows property is set to **True**.

Named Style Defaults

As in Microsoft Word, a Style object in **List for WinForms** can inherit its characteristics from another style, referred to as the parent style. For a newly created list, the Normal style is the parent (or grandparent) of all named styles. Its default properties are as follows:

Style Property	Default Value
Alpha	255
BackColor	System.Drawing.Color.White
BackColor2	System.Drawing.Color.White
BackgroundImage	None
BackgroundPictureDrawMode	BackgroundPictureDrawModeEnum.Stretch
Font	Microsoft Sans Serif, 8.25pt
ForeColor	System.Drawing.Color.Black
ForegroundImage	None
ForeGroundPicturePosition	ForegroundPicturePositionEnum.LeftOfText
GammaCorrection	False
GradientMode	GradientModeEnum.None
HorizontalAlignment	AlignHorzEnum.General
Name	Normal
Padding	0, 0, 0, 0
Trimming	Character
VerticalAlignment	AlignVertEnum.Top
WrapText	False

The Heading and Footing styles are defined similarly. Each inherits from Normal, and each overrides the following

properties:

Style Property	Default Value
BackColor	System.Drawing.SystemColors.Control
ForeColor	System.Drawing.Color.Black
VerticalAlignment	AlignVertEnum.Center

The Heading style overrides one additional property that the Footer style does not:

Style Property	Default Value
WrapText	True

The Selected style also inherits from Normal and overrides two color properties:

Style Property	Default Value
BackColor	System.Drawing.SystemColors.Highlight
ForeColor	System.Drawing.SystemColors.HighlightText

The same is true of the HighlightRow style, which uses the inverse of the color settings for the default Normal style:

Style Property	Default Value
BackColor	System.Drawing.SystemColors.Highlight
ForeColor	System.Drawing.SystemColors.HighlightText

The EvenRow and OddRow styles inherit from Normal, but only EvenRow overrides any properties:

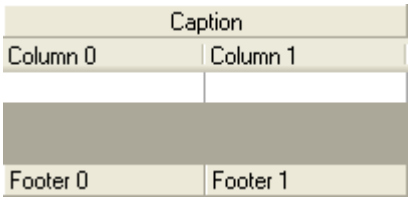
Style Property	Default Value
BackColor	System.Drawing.Color.Aqua

The only styles that do not inherit directly from Normal are Caption and RecordSelector, which inherit from Heading. The reason that list and split captions are justified by default is that the Caption style specializes the following property:

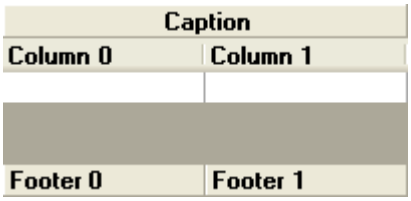
Style Property	Default Value
HorizontalAlignment	AlignHorzEnum.*Justify*

Named Style Inheritance

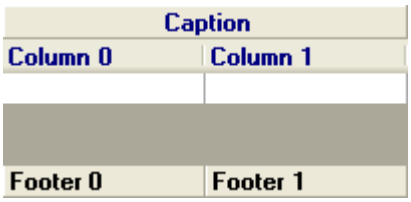
To see for yourself how named style inheritance works, place a list on a form and set the Caption property of the list and its default columns. Set the FooterText property of the default columns and set the ColumnFooters property of the list to **True**. The list should look something like this:



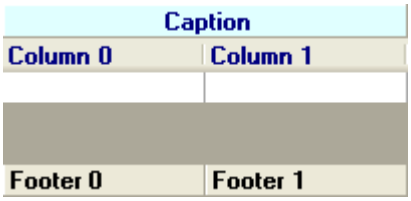
In the **Style Collection Editor**, expand the Normal node, select its Font property, and set **Bold** to **True**. Note that the column headers, column footers, and list caption are all bold, since all built-in styles inherit from the Normal style or one of its children.



Next, expand the Heading node and select its ForeColor property. Choose **Web** from the color tabs, then select **Navy**. Note that the text color of both the column headers and the list's caption bar is now white, since the Caption style inherits its color properties from the Heading style. The column footers remain the same because the Footer style inherits from Normal, not Heading.



Finally, expand the Caption node and select its BackColor property. Choose **Web** from the color tabs, then select **LightCyan**. Note that the background color of the column headers is not changed, and that the Caption style continues to inherit its text color from its parent style, Heading.



Modifying Named Styles

You can change the appearance of the overall list at design time by using .NET's collection editors to modify the GridStyleCollection object. For example, to force all column headers to center their caption text, you can change the HorizontalAlignment property of the built-in Heading style to Center.

The following statement accomplishes the same result in code:

To write code in Visual Basic

```
Visual Basic
Me.C1List1.Styles("Heading").HorizontalAlignment = C1.Win.C1List.AlignHorzEnum.Center
```

To write code in C#

```
C#
this.c1List1.Styles["Heading"].HorizontalAlignment =
C1.Win.C1List.AlignHorzEnum.Center;
```

However, you are not required to use the **Style Collection Editor** or manipulate named members of the GridStyleCollection in code, as the list and its component objects expose several properties that return Style objects. As the next section describes, you can fine tune the appearance of the list by manipulating these objects directly. For more information see Using the Style Collection Editor.

Working with Style Properties

Just as a document template in Microsoft Word specifies the overall appearance of individual paragraphs in a document, the named members of the GridStyleCollection object provide an overall display template for a C1List or C1Combo control. However, to customize the appearance of individual **Split** or C1DisplayColumn objects, you need to modify the appropriate Style object property:

Property	Description
CaptionStyle	Gets or sets the Style object that controls the appearance of the caption area.
EvenRowStyle	Gets or sets the Style object that controls the appearance of an even-numbered row when usingAlternatingRows.
FooterStyle	Gets or sets the Style object that controls the appearance of column footers.
HeadingStyle	Gets or sets the Style object that controls the appearance of the grids column headers.
HighlightRowStyle	Gets or sets the Style object that controls the current row/cell when the **MarqueeStyle** is set to **Highlight Row/Cell**.
OddRowStyle	Gets or sets the Style object that controls the appearance of an odd-numbered row when using AlternatingRows.
SelectedStyle	Gets or sets the Style object that controls the appearance of selected rows and columns.
Style	Gets or sets the root Style object for the Split.

Modifying a Style Property Directly

You can customize the appearance of a list component by modifying one or more members of an appropriate style property. For example, to make the list's caption text bold, you can change the Font property associated with the CaptionStyle property. At design time, this is done by expanding the CaptionStyle tree node on the Properties window, expanding the Font node, and setting **Bold** to **True**. The change is committed to the list when you click out of this particular property.

Note that if you switch to the **Style Collection Editor**, you will see that the built-in Caption style has not changed.

This means that the following statements are not equivalent:

To write code in Visual Basic

```
Visual Basic
Me.C1List1.CaptionStyle.Font = New Font(Me.C1List1.CaptionStyle.Font, FontStyle.Bold)
Me.C1List1.Styles("Caption").Font = New Font(Me.C1List1.Styles("Caption").Font,
FontStyle.Bold)
```

To write code in C#

```
C#
this.c1List1.CaptionStyle.Font = new Font(this.c1List1.CaptionStyle.Font, FontStyle.Bold);
this.c1List1.Styles["Caption"].Font = new Font(this.c1List1.Styles["Caption"].Font,
FontStyle.Bold);
```

The first statement specifies the font of the list's caption bar without changing the named Caption style. The second statement changes the named Caption style, which may or may not affect the display of the list's caption bar, depending on whether the Font member of the CaptionStyle property was specialized previously. For example, if you change the list's CaptionStyle font to Arial, then change the font of the built-in Caption style to Courier, the list caption will remain Arial. However, if you never change the CaptionStyle font, then any font change to the built-in Caption style will be instantly reflected in the list's caption bar.

Named Styles vs. Anonymous Styles

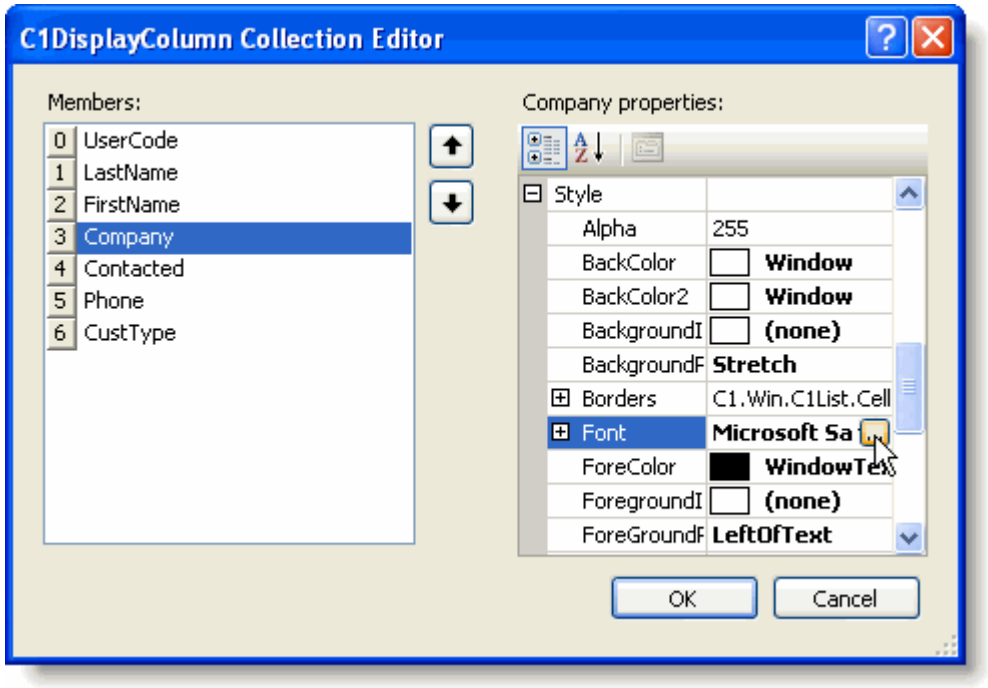
When setting style properties at design time, it is important to understand the distinction between named styles and the *anonymous styles* exposed by list properties.

Named styles provide templates that govern the appearance of the list, its splits, and its columns. In the designer, you can create, modify, and delete named styles using the **Style Collection Editor**. In code, the GridStyleCollection is used to represent the same set of named Style objects.

Anonymous styles are **not** members of the GridStyleCollection, however. They are provided so that you can easily and directly customize the appearance of an individual split or column without having to define a separate named style.

The following analogy should help to clarify the distinction between named and anonymous styles. Consider a Microsoft Word document that consists of several paragraphs based on the default normal style. Suppose that one of the paragraphs is a quotation that needs to be indented and displayed in italics. If the document is part of a larger work that contains several quotations, it makes sense to define a special style for that purpose and apply it to all paragraphs that contain quotations. If the document is an early draft or is not likely to be updated, defining a style for one paragraph is overkill, and it would be more convenient to apply indentation and italics to the quotation itself.

In this analogy, specifying paragraph attributes directly is akin to setting the members of a property that returns an anonymous style. For example, if you want to change the font of a particular list column, you can modify the Font member of the column's Style property in the **C1DisplayColumn Collection Editor**.



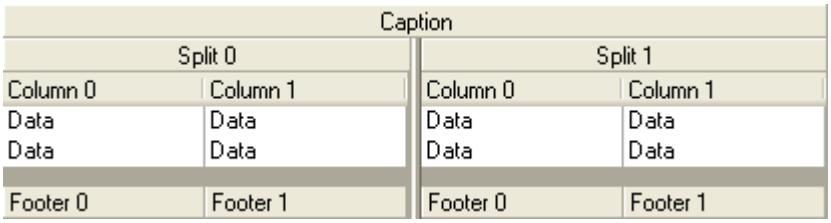
Note that modifying an anonymous style is just like modifying a named style. You first expand the desired Style object node in a property tree, then select and edit one or more of its member properties.

Anonymous Style Inheritance

Just as one named style can inherit font, color, and formatting characteristics from another, an anonymous style in a **Split** object can inherit from its counterpart in the containing C1List control. Similarly, an anonymous style in a C1DisplayColumn object can inherit from its counterpart in the containing **Split** object. Since the C1Combo control does not have a Splits collection, the anonymous styles of its C1DisplayColumn objects can inherit values from the control itself.

When a list is first created, its Style property inherits all of its attributes from the built-in Normal style, which controls the appearance of all data cells. Any changes to the Normal style are propagated to all splits, and in turn to the columns within each split. However, you can change the appearance of all data cells within a **Split** or C1DisplayColumn object by modifying the members of its anonymous Style property.

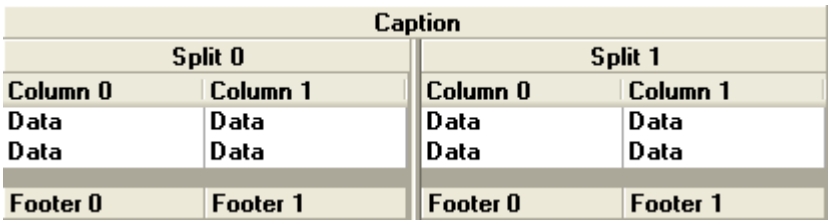
Consider the following list layout, which uses the default values of all built-in styles and contains two identical splits.



All of the subsequent examples use this layout as a starting point. For clarity, the examples use code to illustrate the relationships between style properties and the list's display; however, you can perform the same operations at design time using the list's collection editors.

Example 1 - Inheriting from Containing Splits

Since the default values of all built-in styles are in effect, columns inherit from their containing splits, which in turn inherit from the list as a whole. Therefore, this statement affects not only data cells, but all headers, footers, and caption bars. This statement has the same *visual* effect as changing the Normal style directly using the **Style Collection Editor**; however, the built-in Normal style itself is not changed.



The following code inherits values from the containing splits:

To write code in Visual Basic

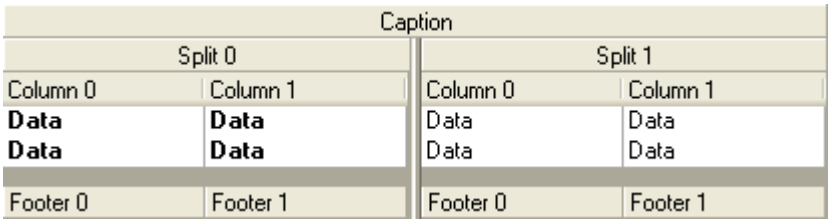
```
Visual Basic
Dim myfont As Font
myfont = New Font(Me.C1List1.Styles("Normal").Font, FontStyle.Bold)
Me.C1List1.Styles("Normal").Font = myfont
```

To write code in C#

```
C#
Font myfont;
myfont = new Font(this.c1List1.Styles["Normal"].Font, FontStyle.Bold);
this.c1List1.Styles["Normal"].Font = myfont;
```

Example 2 - Affecting Data Cells Only in the First Split

In this example, only the data cells of the first split are affected. This is because the split caption, column headers, and column footers inherit their fonts from the built-in styles Caption, Heading, and Footing, respectively.



The following code affects data cells only in the first split:

To write code in Visual Basic

Visual Basic

```
Dim myfont As Font
myfont = New Font(Me.C1List1.Splits(0).Style.Font, FontStyle.Bold)
Me.C1List1.Splits(0).Style.Font = myfont
```

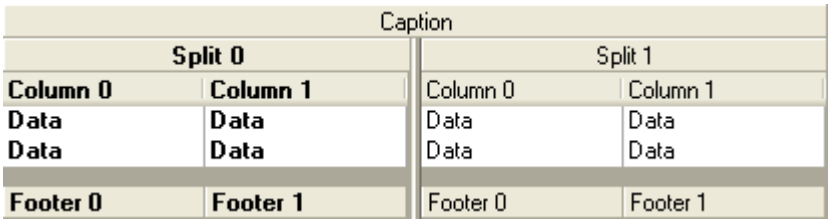
To write code in C#

C#

```
Font myfont;
myfont = new Font(this.c1List1.Splits[0].Style.Font, FontStyle.Bold);
this.c1List1.Splits[0].Style.Font = myfont;
```

Example 3 - Affecting All Elements Only in the First Split

This example extends the previous one to render all elements of the first split in bold. In addition to the Style property, it is necessary to set the CaptionStyle, HeadingStyle, and FooterStyle properties.



The following code affects all elements only in the first split:

To write code in Visual Basic

Visual Basic

```
Dim myfont As Font
Dim myfont1 As Font
Dim myfont2 As Font
Dim myfont3 As Font

myfont = New Font(Me.C1List1.Splits(0).Style.Font, FontStyle.Bold)
Me.C1List1.Splits(0).Style.Font = myfont

myfont1 = New Font(Me.C1List1.Splits(0).CaptionStyle.Font, FontStyle.Bold)
Me.C1List1.Splits(0).CaptionStyle.Font = myfont1

myfont2 = New Font(Me.C1List1.Splits(0).HeadingStyle.Font, FontStyle.Bold)
Me.C1List1.Splits(0).HeadingStyle.Font = myfont2
```

```
myfont3 = New Font(Me.C1List1.Splits(0).FooterStyle.Font, FontStyle.Bold)
Me.C1List1.Splits(0).FooterStyle.Font = myfont3
```

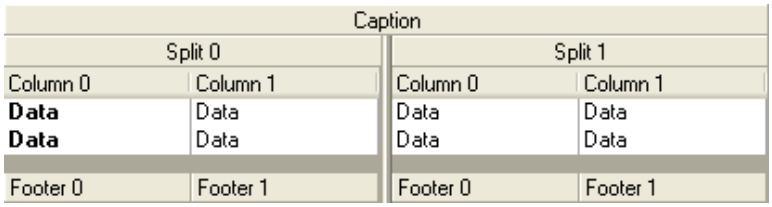
To write code in C#

C#

```
Font myfont;
Font myfont1;
Font myfont2;
Font myfont3;

myfont = new Font(this.c1List1.Splits[0].Style.Font, FontStyle.Bold);
this.c1List1.Splits[0].Style.Font = myfont;

myfont1 = new Font(this.c1List1.Splits[0].CaptionStyle.Font, FontStyle.Bold);
this.c1List1.Splits[0].CaptionStyle.Font = myfont1;

myfont2 = new Font(this.c1List1.Splits[0].HeadingStyle.Font, FontStyle.Bold);
this.c1List1.Splits[0].HeadingStyle.Font = myfont2;

myfont3 = new Font(this.c1List1.Splits[0].FooterStyle.Font, FontStyle.Bold);
this.c1List1.Splits[0].FooterStyle.Font = myfont3;
```

Example 4 - Affecting Data Cells Only in the First Column of the First Split

In this example, only the data cells of the first column of the first split are affected. This is because the column headers and column footers inherit their fonts from the built-in styles Heading and Footing, respectively.



The following code affects data cells only in the first column of the first split:

To write code in Visual Basic

Visual Basic

```
Dim myfont As Font
myfont = New Font(Me.C1List1.Splits(0).DisplayColumns(0).Style.Font, FontStyle.Bold)
Me.C1List1.Splits(0).DisplayColumns(0).Style.Font = myfont
```

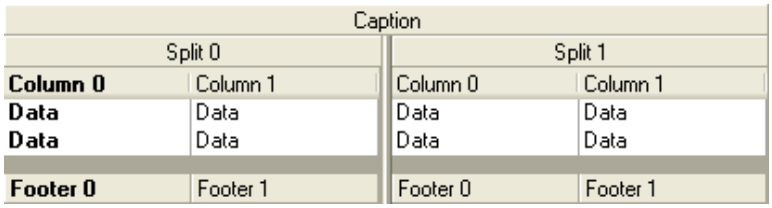
To write code in C#

C#

```
Font myfont;
myfont = new Font(this.c1List1.Splits[0].DisplayColumns[0].Style.Font, FontStyle.Bold);
this.c1List1.Splits[0].DisplayColumns[0].Style.Font = myfont;
```

Example 5 - Affecting All Elements Only in the First Column of the First Split

This example extends the previous one to render all elements of the first column of the first split in bold. In addition to the Style property, it is necessary to set the HeadingStyle and FooterStyle properties.



The following code affects all elements only in the first column of the first split:

To write code in Visual Basic

Visual Basic

```
Dim myfont As Font
Dim myfont1 As Font
Dim myfont2 As Font

myfont = New Font(Me.C1List1.Splits(0).DisplayColumns(0).Style.Font, FontStyle.Bold)
Me.C1List1.Splits(0).DisplayColumns(0).Style.Font = myfont

myfont1 = New Font(Me.C1List1.Splits(0).DisplayColumns(0).HeadingStyle.Font, FontStyle.Bold)
Me.C1List1.Splits(0).DisplayColumns(0).HeadingStyle.Font = myfont1

myfont2 = New Font(Me.C1List1.Splits(0).DisplayColumns(0).FooterStyle.Font, FontStyle.Bold)
Me.C1List1.Splits(0).DisplayColumns(0).FooterStyle.Font = myfont2
```

To write code in C#

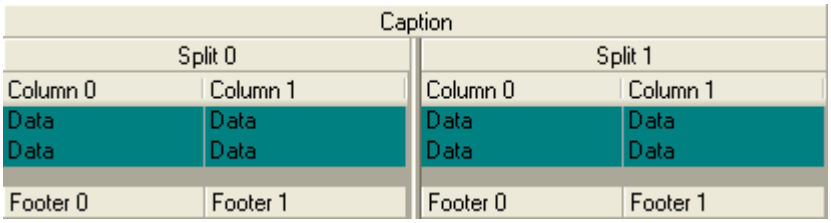
C#

```
Font myfont;
Font myfont1;
Font myfont2;

myfont = new Font(this.c1List1.Splits[0].DisplayColumns[0].Style.Font, FontStyle.Bold);
this.c1List1.Splits[0].DisplayColumns[0].Style.Font = myfont;

myfont1 = new Font(this.c1List1.Splits[0].DisplayColumns[0].HeadingStyle.Font,
FontStyle.Bold);
this.c1List1.Splits[0].DisplayColumns[0].HeadingStyle.Font = myfont1;

myfont2 = new Font(this.c1List1.Splits[0].DisplayColumns[0].FooterStyle.Font,
FontStyle.Bold);
this.c1List1.Splits[0].DisplayColumns[0].FooterStyle.Font = myfont2;
```

Example 6 - Changing the BackColor Property

In the first example, setting the Font member of the list's Style property affected the entire list, including each caption bar, column header, and column footer. However, the same is not true of the BackColor and ForeColor properties.

Since the built-in Caption, Heading, and Footing styles override both of these properties, only the data cells of the list are displayed with a teal background.



The following code chages the font member of the **Style** property:

To write code in Visual Basic

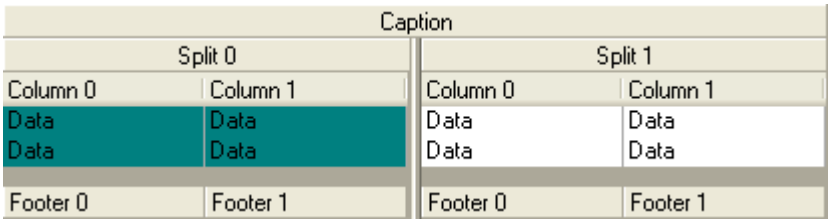
```
Visual Basic
Me.C1List1.Style.BackColor = System.Drawing.Color.Teal
```

To write code in C#

```
C#
this.c1List1.Style.BackColor = System.Drawing.Color.Teal;
```

Example 7 - Changing the Data Cells in Only the First Split

In this example, only the data cells of the first split are affected. This is because the split caption, column headers, and column footers inherit their background colors from the built-in styles Caption, Heading, and Footing, respectively.



The following code changes the data cells in only the first split:

To write code in Visual Basic

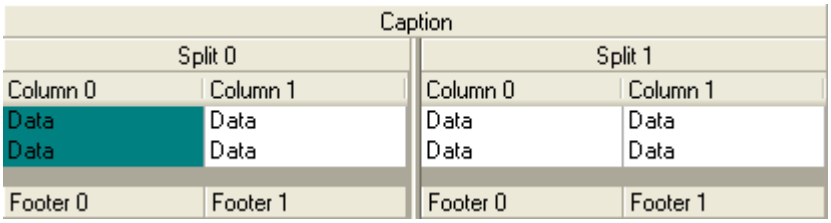
```
Visual Basic
Me.C1List1.Splits(0).Style.BackColor = System.Drawing.Color.Teal
```

To write code in C#

```
C#
this.c1List1.Splits[0].Style.BackColor = System.Drawing.Color.Teal;
```

Example 8 - Changing the Data Cells in Only the First Column of the First Split

In this example, only the data cells of the first column of the first split are affected. This is because the column headers and column footers inherit their background colors from the built-in styles Heading and Footing, respectively.



The following code changes the data cells in only the first column of the first split:

To write code in Visual Basic

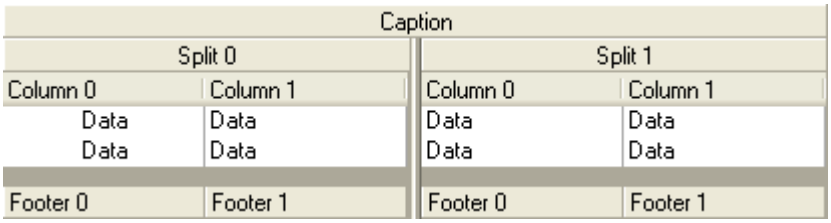
```
Visual Basic
Me.C1List1.Splits(0).DisplayColumns(0).Style.BackColor = System.Drawing.Color.Teal
```

To write code in C#

```
C#
this.c1List1.Splits[0].DisplayColumns[0].Style.BackColor = System.Drawing.Color.Teal;
```

Example 9 - Setting the Alignment Property of C1DisplayColumn Objects

Setting the HorizontalAlignment property of a C1DisplayColumn object affects not only its data cells, but also its header and footer. The reason for this is that the default setting of the HorizontalAlignment property for the built-in Heading and Footing styles, which is inherited from Normal, is set to General. For data cells, the general setting means that the underlying data type determines whether the cell text is left, center, or right aligned; for column headers and footers, the general setting means that the column's data cell alignment should be followed.



The following code sets the alignment of C1DisplayColumn objects:

To write code in Visual Basic

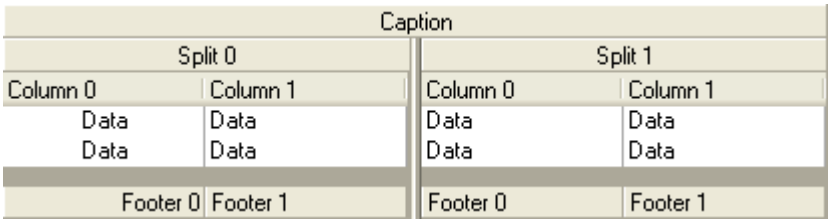
```
Visual Basic
Me.C1List1.Splits(0).DisplayColumns(0).Style.HorizontalAlignment =
C1.Win.C1List.AlignHorzEnum.Center
```

To write code in C#

```
C#
this.c1List1.Splits[0].DisplayColumns[0].Style.HorizontalAlignment =
C1.Win.C1List.AlignHorzEnum.Center;
```

Example 10 - Setting the Alignment for Column Headers and Footers

This example illustrates the distinction between general and specific alignment for column headers and footers. If the HorizontalAlignment member of the HeadingStyle (or FooterStyle) property is not set to General, then the header (or footer) is aligned independently of the data cells.



The following code sets the alignment for column headers:

To write code in Visual Basic

Visual Basic

```
Me.C1List1.Splits(0).DisplayColumns(0).HeadingStyle.HorizontalAlignment =
C1.Win.C1List.AlignHorzEnum.Near
Me.C1List1.Splits(0).DisplayColumns(0).FooterStyle.HorizontalAlignment =
C1.Win.C1List.AlignHorzEnum.Far
Me.C1List1.Splits(0).DisplayColumns(0).Style.HorizontalAlignment =
C1.Win.C1List.AlignHorzEnum.Center
```

To write code in C#

C#

```
this.c1List1.Splits[0].DisplayColumns[0].HeadingStyle.HorizontalAlignment =
C1.Win.C1List.AlignHorzEnum.Near;
this.c1List1.Splits[0].DisplayColumns[0].FooterStyle.HorizontalAlignment =
C1.Win.C1List.AlignHorzEnum.Far;
this.c1List1.Splits[0].DisplayColumns[0].Style.HorizontalAlignment =
C1.Win.C1List.AlignHorzEnum.Center;
```

Applying Styles to Cells

List for WinForms gives you three ways to control the display characteristics of individual cells:

- **By Status**

  Each list cell has a cell status that identifies its disposition (any combination of current, modified, part of a selected row, or part of a highlighted row). Using the AddCellStyle method, you can set style attributes that apply to any possible combination of cell status values.

- **By Contents**

  You can specify a pattern (called a regular expression) that is used to perform pattern matching on cell contents. When the contents match the pattern supplied in the AddRegexCellStyle method, **List for WinForms** will automatically apply pre-selected style attributes to the cell.

- **By Custom Criteria**

Using the FetchCellStyle (or FetchRowStyle) event, you can make decisions about cell colors and fonts each time a cell (or row) is displayed.

You can use Style objects defined at design time as arguments to the AddCellStyle and AddRegexCellStyle methods. Or, you can create a temporary style in code and use it to specialize one or more attributes.

The FetchCellStyle and FetchRowStyle events pass a temporary Style object as the final parameter. By setting its properties, you can control the appearance of the cell specified by the other event parameters.

In this version of **List for WinForms**, per-cell font and color control can only be achieved by writing code. However, by creating styles at design time, you can keep this code to a minimum. To learn how to create named styles at design time, see Using the Style Collection Editor.

Specifying Cell Status Values

List for WinForms recognizes 16 distinct cell status values which are used in code to indicate the disposition of a cell. A cell status value is a combination of four separate conditions. These conditions are enumerations that have the *flag* attribute, which means that they can be combined with the *Or* operator:

Condition	Description
Current Cell	The cell is the current cell as specified by the Bookmark, Col, and Split properties. At any given time, only one cell can have this status.
Marquee Row	The cell is part of a highlighted row marquee.
Selected Row	The cell is part of a row selected by the user or in code. The SelectedRowCollection contains a bookmark for each selected row.

List for WinForms defines the following constants corresponding to these cell conditions:

Constant	Description
CurrentCell	Applies to the current cell
MarqueeRow	Applies to cells in a highlighted row marquee
SelectedRow	Applies to cells in a selected row

List for WinForms also defines the following constants, which are **not** meant to be combined with those listed earlier:

Constant	Description
AllCells	Applies to all cells
NormalCell	Applies to cells without status conditions

Use AllCells to refer to all cells regardless of status. Use NormalCell to refer to only those cells without any of the three basic cell conditions described earlier.

Applying Cell Styles by Status

Each cell in the **List for WinForms** display has a status value which identifies its disposition (any combination of current, modified, part of a selected row, or part of a highlighted row). Using the AddCellStyle method, you can set style attributes which apply to any possible combination of cell status values. The AddCellStyle method is supported by the C1List, C1Combo, **Split**, and C1DisplayColumn objects, enabling you to control the range of cells for which certain conditions apply.

For each unique status combination, you can set the color, font, and picture attributes to be used for cells of that

status. When a cell's status changes, **List for WinForms** checks to see if any style property overrides are defined for that cell, and applies those attributes to the cell when it is displayed. Style objects are used to specify the color and font for a cell, as in the following example:

To write code in Visual Basic

Visual Basic

```
Dim S As New C1.Win.C1List.Style()
Dim myfont As Font

myfont = New Font(S.Font, FontStyle.Bold)
S.Font = myfont
S.ForeColor = System.Drawing.Color.Red

Me.C1List1.AddCellStyle(C1.Win.C1List.CellStyleFlag.CurrentCell, S)
```

To write code in C#

C#

```
C1.Win.C1List.Style S = new C1.Win.C1List.Style();
Font myfont;

myfont = new Font(S.Font, FontStyle.Bold);
S.Font = myfont;

S.ForeColor = System.Drawing.Color.Red;
this.c1List1.AddCellStyle(C1.Win.C1List.CellStyleFlag.CurrentCell, S);
```

Here, a new temporary style object is created to specify the color and font (red text, bold) to be applied to the current cell throughout the entire list. Since the style object's BackColor property is not set explicitly, the background color of the current cell is not changed.

You can also use styles defined at design time as arguments to the AddCellStyle method:

To write code in Visual Basic

Visual Basic

```
Dim S As C1.Win.C1List.Style
S = Me.C1List1.Styles("RedBold")
Me.C1List1.AddCellStyle(C1.Win.C1List.CellStyleFlag.CurrentCell, S)
```

To write code in C#

C#

```
C1.Win.C1List.Style S;
S = this.c1List1.Styles["RedBold"];
this.c1List1.AddCellStyle(C1.Win.C1List.CellStyleFlag.CurrentCell, S);
```

The preceding example can be simplified since the AddCellStyle method accepts a style name as well as an actual style object:

To write code in Visual Basic

Visual Basic

```
Me.C1List1.AddCellStyle(C1.Win.C1List.CellStyleFlag.CurrentCell, S)
```

To write code in C#

C#

```
this.c1List1.AddCellStyle(C1.Win.C1List.CellStyleFlag.CurrentCell, S);
```

All of the preceding examples cause the text of the current cell to appear in red and bold. However, it is important to note that the status **CurrentCell** applies only to cells which have **only** this status.

Although this method of specifying cell conditions offers more control and flexibility, it also requires that additional code be written for some common cases.

Calls to AddCellStyle take effect immediately, and can be used for interactive effects as well as overall list characteristics.

Applying Cell Styles by Contents

You can tell **List for WinForms** to automatically apply colors and fonts to particular cells, based upon their displayed contents. To do so, you provide a pattern, called a *regular expression*, which the list tests against the displayed value of each cell. Using the AddRegexCellStyle method, you can associate a regular expression with a set of style attributes, and then apply them to any possible combination of cell status values. The AddRegexCellStyle method is supported by the C1List, C1Combo, **Split**, and C1DisplayColumn objects, allowing you to control the range of cells for which certain conditions apply.

The AddRegexCellStyle method is similar to the AddCellStyle method, but it requires an additional argument for the regular expression string. As with AddCellStyle, you can use either temporary or named styles. The following example uses a temporary style to display all cells in the first column that contain the string "Computer" in bold:

To write code in Visual Basic

Visual Basic

```
Dim S As New C1.Win.C1List.Style()
Dim myfont As Font

myfont = New Font(S.Font, FontStyle.Bold)
S.Font = myfont

Me.C1List1.Splits(0).DisplayColumns(0).AddRegexCellStyle(C1.Win.C1List.CellStyleFlag.AllCells,
S, "Computer")
```

To write code in C#

C#

```
C1.Win.C1List.Style S = new C1.Win.C1List.Style();
Font myfont;
```

```
myfont = new Font(S.Font, FontStyle.Bold);
S.Font = myfont;

this.c1List1.Splits[0].DisplayColumns[0].AddRegexCellStyle(C1.Win.C1List.CellStyleFlag.AllCells,
S, "Computer");
```

This feature allows you to implement "visual queries" that attach distinctive font or color attributes to cells that match a certain pattern.

Applying Cell Styles by Custom Criteria

For cases where regular expressions are insufficient to express your formatting requirements, you can use the FetchCellStyle event to customize fonts and colors on a per-cell basis. This event will only be fired for columns that have the FetchStyle property set to **True**.

For example, you may want to provide color coding for values that fall within a certain range. The following code assumes that the FetchStyle property is **True** for a single column of numeric data, and handles the FetchCellStyle event to display values greater than 1000 in blue:

To write code in Visual Basic

Visual Basic

```
Private Sub C1List1_FetchCellStyle(ByVal sender As Object, ByVal e As
C1.Win.C1List.FetchCellStyleEventArgs) Handles C1List1.FetchCellStyle
    Dim N As Long
    N = Val(Me.C1List1.Columns(e.Col).CellText(e.Row))

    If N > 1000 Then
 e.CellStyle.ForeColor = System.Drawing.Color.Blue
    End If
End Sub
```

To write code in C#

C#

```
private void C1List1_FetchCellStyle( object sender,
C1.Win.C1List.FetchCellStyleEventArgs e)
{
    long N;
    N = long.Parse(this.C1List1.Columns(e.Col).CellText(e.Row));

    if ( N > 1000 )
    {
```

```
 e.CellStyle.ForeColor = System.Drawing.Color.Blue;
    }
}
```

The Split, Row, and Col arguments identify which cell the list is displaying. The CellStyle argument conveys formatting information from your application to the list. Since the CellStyle argument is a Style object, you can also change a cell's font characteristics in the FetchCellStyle event:

To write code in Visual Basic

Visual Basic

```
If N > 1000 Then
    e.CellStyle.Font = New Font(e.CellStyle.Font, FontStyle.Italic)
End If
```

To write code in C#

C#

```
if ( N > 1000 )
{
    e.CellStyle.Font = new Font(e.CellStyle.Font, FontStyle.Italic);
}
```

The FetchCellStyle event can also be used to apply formatting to one cell based upon the values of other cells, or even other controls. For example, suppose that you want to:

- Make the cell text red in column 4 if column 1 minus column 2 is negative.
- Make the cell text bold in column 7 if it matches the contents of a text box.

In this case, you need to set the FetchStyle property to **True** for columns 4 and 7, and handle the FetchCellStyle event as follows:

To write code in Visual Basic

Visual Basic

```
Private Sub C1List1_FetchCellStyle(ByVal sender As Object, ByVal e As
C1.Win.C1List.FetchCellStyleEventArgs) Handles C1List1.FetchCellStyle
    Select Case e.Col
 Case 4
     Dim Col1 As Long, Col2 As Long
     Col1 = CLng(Me.C1List1.Columns(1).CellText(e.Row))
     Col2 = CLng(Me.C1List1.Columns(2).CellText(e.Row))
     If Col1 - Col2 < 0 Then
         e.CellStyle.ForeColor = System.Drawing.Color.Red
     End If
 Case 7
     Dim S As String
     S = Me.C1List1.Columns(7).CellText(e.Row)
     If S = TextBox1.Text Then
         e.CellStyle.Font = New Font(e.CellStyle.Font, FontStyle.Bold)
     End If
 Case Else
     Debug.WriteLine ("FetchCellStyle not handled: " & e.Col)
```

```
    End Select
End Sub
```

To write code in C#

C#

```
private void C1List1_FetchCellStyle( object sender,
C1.Win.C1List.FetchCellStyleEventArgs e)
{
    switch (e.Col)
    {
 case 4:
     long Col1, Col2;
     Col1 = long.Parse(this.C1List1.Columns[1].CellText(e.Row));
     Col2 = long.Parse(this.C1List1.Columns[2].CellText(e.Row));
     if ( Col1 - Col2 < 0 )
     {
         e.CellStyle.ForeColor = System.Drawing.Color.Red;
     }
     break;
 case 7:
     string  S;
     S = this.C1List1.Columns[7].CellText(e.Row);
     if ( S == TextBox1.Text )
     {
         e.CellStyle.Font = new Font(e.CellStyle.Font, FontStyle.Bold);
     }
     break;
 default:
     Console.WriteLine ("FetchCellStyle not handled: " + e.Col);
     break;
    }
}
```

For efficiency reasons, you should only set FetchStyle to **True** for columns that you plan to handle in the FetchCellStyle event.

Note: The preceding examples use the CellText method for simplicity. However, the CellText and CellValue methods always create and destroy an internal clone of the dataset each time they are called, which may make them too inefficient to use in the FetchCellStyle event.

If you need to customize fonts and colors on a per-row instead of a per-cell basis, use the FetchRowStyle event, which will only be fired once per row for lists that have the FetchRowStyles property set to **True**. The syntax for this event is as follows:

To write code in Visual Basic

Visual Basic

```
Private Sub C1List1_FetchRowStyle(ByVal sender As Object, ByVal e As
C1.Win.C1List.FetchRowStyleEventArgs) Handles C1List1.FetchRowStyle
```

To write code in C#

C#

```
private void c1List1_FetchRowStyle(object sender,
C1.Win.C1List.FetchRowStyleEventArgs e)
```

Although you can use the FetchRowStyle event to implement an alternating row color scheme, an easier and more efficient way to accomplish the same task would be to use the AlternatingRows property, together with the built-in EvenRow and OddRow styles.

Cell Style Evaluation Order

The following list defines the order in which cell styles are applied relative to the anonymous styles of a list, split, or column:

1. Style property, C1List control. The default named parent of this anonymous style is Normal.
2. **Style** property, **Split** object. By default, this anonymous style inherits from its C1List control counterpart.
3. **EvenRowStyle** and **OddRowStyle** properties, **Split** object. By default, these anonymous styles inherit from their C1List control counterparts, which in turn have default named parents of EvenRow and OddRow. These properties apply only if the AlternatingRows property is **True**.
4. Style property, C1DisplayColumn object. By default, this anonymous style inherits from its **Split** object counterpart.
5. FetchRowStyle event. This event fires only if the FetchRowStyles property is **True** for a list or split.
6. **SelectedStyle** property, **Split** object. By default, this anonymous style inherits from its C1List control counterpart, which in turn has a default named parent of Selected. This property applies only to selected rows; that is, rows whose bookmarks have been added to the SelectedRowCollection through code or user interaction.
7. **HighlightRowStyle** property, **Split** object. By default, this anonymous style inherits from its C1List control counterpart, which in turn has a default named parent of HighlightRow. This property applies only to highlighted rows.
8. AddCellStyle and AddRegexCellStyle methods, if called. Cell styles specified at the C1DisplayColumn object level have the highest priority, followed by those specified at the **Split** object and C1List control levels. Within an object level, cell styles are tested in the order in which they were added in code. Cell styles do not inherit from one another; as soon as a match is found, testing stops.
9. FetchCellStyle event. This event fires only if the FetchStyle property is **True** for a C1DisplayColumn object. Thus, you always have final control over the rendering of a cell via the FetchCellStyle event.

Applying Pictures to List Elements

We extend the concept of styles to include background and foreground pictures, enabling you to add adornments to headers, footers, and caption bars, to specify a background pattern for data cells, and to render picture data in cells without having to populate a ValueItems object. The following properties of the Style object determine how pictures are displayed:

Property	Description
BackgroundImage	Sets/returns a style's background picture.
BackgroundPictureDrawMode	Controls how a style's background picture is displayed.
ForegroundImage	Sets/returns a style's foreground picture.
ForegroundPicturePosition	Controls how a style's foreground picture is positioned.

Since picture properties follow the same inheritance rules as other style attributes, any technique described earlier in this chapter also works with pictures. This means that you can attach pictures to a list element using any of the following methods:

- Setting the BackgroundImage or ForegroundImage property of a built-in named style in the designer or in code.
- Setting the BackgroundImage or ForegroundImage property of an anonymous style in the designer or in code.
- Calling the AddCellStyle or AddRegexCellStyle methods.
- Writing a handler for the FetchCellStyle or FetchRowStyle events.

Displaying Background Pictures

You can use background pictures to customize static list elements such as caption bars, column headers, and column footers. For example, the following code applies a colored gradient bitmap to the BackgroundImage member of the Style object returned by the list's CaptionStyle property:

To write code in Visual Basic

Visual Basic

```
Dim myfont As Font
Me.C1List1.CaptionStyle.BackgroundImage = System.Drawing.Image.FromFile("c:\bubbles.bmp")
Me.C1List1.CaptionStyle.BackgroundPictureDrawMode =
C1.Win.C1List.BackgroundPictureDrawModeEnum.Tile
Me.C1List1.CaptionStyle.ForeColor = System.Drawing.Color.White
myfont = New Font(Me.C1List1.CaptionStyle.Font, FontStyle.Bold)
Me.C1List1.CaptionStyle.Font = myfont
```

To write code in C#

C#

```
Font myfont;
this.c1List1.CaptionStyle.BackgroundImage =
System.Drawing.Image.FromFile(@"c:\bubbles.bmp");
this.c1List1.CaptionStyle.BackgroundPictureDrawMode =
C1.Win.C1List.BackgroundPictureDrawModeEnum.Tile;
this.c1List1.CaptionStyle.ForeColor = System.Drawing.Color.White;
myfont = new Font(this.c1List1.CaptionStyle.Font, FontStyle.Bold);
this.c1List1.CaptionStyle.Font = myfont;
```

This code also adjusts the color of the caption text and makes it bold, producing the following display:

You can achieve the same effect at design time by editing either the built-in Caption style on the **Style Collection Editor**, or the members of the C1ListBase.CaptionStyle property in the Properties window.

By default, background pictures are centered within the associated list element. Depending upon the height of the background bitmap, you may need to adjust the value of the BackgroundPictureDrawMode property to ensure that the entire area is filled. This property determines whether the picture is centered, tiled, or stretched to fit the entire area, as

shown in the following figure:

Center	Tile	Stretch
		

You can also use background pictures within data cells to produce interesting visual effects. For example, the following patterns were designed to be replicated in adjacent rows.

By eliminating the dividing lines between data rows and the column header dividers, these patterns can be used to produce the following display.

The trick is to insert an empty unbound column on the left to display the binder rings. Add a column using the **C1List Designer** (see Using the C1List Designer). The following code demonstrates how to display the list:

To write code in Visual Basic

Visual Basic

```
' Give the list a flat appearance and remove row dividers, and scroll bars. Assume that
the ScaleMode of the containing form is in pixels.
Me.C1List1.Splits(0).HeadingStyle.ForeColor = System.Drawing.Color.White
Me.C1List1.RowDivider.Style = C1.Win.C1List.LineStyleEnum.None
Me.C1List1.HeadingStyle.Borders.BorderType = C1.Win.C1List.BorderTypeEnum.None
Me.C1List1.ItemHeight = 16
```

```
Me.C1List1.HScrollBar.Style = C1.Win.C1List.ScrollBarStyleEnum.Always
Me.C1List1.VScrollBar.Style = C1.Win.C1List.ScrollBarStyleEnum.Always

' Set the background pattern to be used by data cells in the default split (so as not to
disturb the Normal style).
Me.C1List1.Splits(0).Style.BackgroundImage =
System.Drawing.Image.FromFile("binderpaper.bmp")
Me.C1List1.Splits(0).Style.BackgroundPictureDrawMode =
C1.Win.C1List.BackgroundPictureDrawModeEnum.Tile

' Create an empty unbound column on the left to hold the binder rings. Remove its
dividing lines and set the BackroundBitmap property of its Style object.
Dim C As C1.Win.C1List.C1DisplayColumn
C = C1List1.Splits(0).DisplayColumns(0)

C.Width = 48
C.Visible = True
C.Style.BackgroundImage = System.Drawing.Image.FromFile("binderrings.bmp")
C.HeaderDivider = False
C.ColumnDivider.Style = C1.Win.C1List.LineStyleEnum.None

' Scroll the unbound column into view.
Me.C1List1.Col = 0

' Use a small corner of the binder ring bitmap as the background of the column headers,
and adjust the font and text color accordingly.
Dim myfont As Font
Dim S As New C1.Win.C1List.Style()

Me.C1List1.HeadingStyle.BackgroundImage =
System.Drawing.Image.FromFile("bindercorner.bmp")
Me.C1List1.HeadingStyle.BackgroundPictureDrawMode =
C1.Win.C1List.BackgroundPictureDrawModeEnum.Tile
myfont = New Font(S.Font, FontStyle.Bold)
Me.C1List1.HeadingStyle.Font = myfont
Me.C1List1.HeadingStyle.ForeColor = System.Drawing.Color.White
```

To write code in C#

C#

```
// Give the list a flat appearance and remove row dividers, and scroll bars. Assume that
the ScaleMode of the containing form is in pixels.
this.c1List1.Splits[0].HeadingStyle.ForeColor = System.Drawing.Color.White;
this.c1List1.RowDivider.Style = C1.Win.C1List.LineStyleEnum.None;
this.c1List1.HeadingStyle.Borders.BorderType = C1.Win.C1List.BorderTypeEnum.None;
this.c1List1.ItemHeight = 16;
this.c1List1.HScrollBar.Style = C1.Win.C1List.ScrollBarStyleEnum.Always;
this.c1List1.VScrollBar.Style = C1.Win.C1List.ScrollBarStyleEnum.Always;

// Set the background pattern to be used by data cells in the default split (so as not to
disturb the Normal style).
this.c1List1.Splits[0].Style.BackgroundImage =
System.Drawing.Image.FromFile("binderpaper.bmp");
```

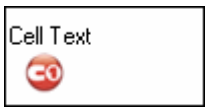
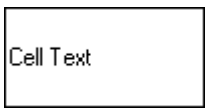
```
this.c1List1.Splits[0].Style.BackgroundPictureDrawMode =
C1.Win.C1List.BackgroundPictureDrawModeEnum.Tile;

// Create an empty unbound column on the left to hold the binder rings. Remove its
dividing lines and set the BackroundBitmap property of its Style object.
C1.Win.C1List.C1DisplayColumn C;
C = this.c1List1.Splits[0].DisplayColumns[0];

C.Width = 48;
C.Visible = true;
C.Style.BackgroundImage = System.Drawing.Image.FromFile("binderrings.bmp");
C.HeaderDivider = false;
C.ColumnDivider.Style = C1.Win.C1List.LineStyleEnum.None;

// Scroll the unbound column into view.

this.c1List1.Col = 0;

// Use a small corner of the binder ring bitmap as the background of the column headers,
and adjust the font and text color accordingly.
Font myfont;
C1.Win.C1List.Style S = new C1.Win.C1List.Style();

this.c1List1.HeadingStyle.BackgroundImage =
System.Drawing.Image.FromFile("bindercorner.bmp");
this.c1List1.HeadingStyle.BackgroundPictureDrawMode =
C1.Win.C1List.BackgroundPictureDrawModeEnum.Tile;
myfont = new Font(S.Font, FontStyle.Bold);
this.c1List1.HeadingStyle.Font = myfont;
this.c1List1.HeadingStyle.ForeColor = System.Drawing.Color.White;
```

Displaying Foreground Pictures

You can use foreground pictures to add visual cues to static list elements such as caption bars, column headers, and column footers. Foreground pictures are specified by the ForegroundImage property of the Style. Foreground pictures can be displayed beside some text or in place of it, but can't be displayed over text. For an example of Foreground picture positions, see below.

Foreground pictures have the ForegroundPicturePosition property which specifies where a foreground picture is situated in comparison to the cell text. The values and their representations are displayed below:

Position	Display
Near	
Far	
LeftOfText	

Position	Display
RightOfText	
TopOfText	
BottomOfText	
PictureOnly	
TextOnly	

List for WinForms Samples

Please be advised that this ComponentOne software tool is accompanied by various sample projects and/or demos which may make use of other development tools included with ComponentOne Studio.

Samples

Please refer to the pre-installed product samples through the following path:

Documents\ComponentOne Samples\WinForms

The following samples are available for **List for WinForms**:

Sample	Description
Tutorial 1	In this tutorial, you will learn how to bind **C1List** to a **DataSet** and create a fully functional database browser. You will also learn about the basic properties of the **C1List** control.
Tutorial 2	In this tutorial, you will learn how to bind **C1Combo** to a **DataSet**. You will also learn about the basic properties of the **C1Combo** control.
Tutorial 3	This tutorial demonstrates how you can manually add information to a list using the **AddItem Mode** without having to bind **C1List** to a **DataSet**.
Tutorial 4	This tutorial demonstrates how to link multiple **C1List** and **C1Combo** controls using the **Change** event to trigger related actions. This technique is particularly useful for displaying master-detail relationships.
Tutorial 5	In this tutorial, you will learn how to use **C1List** to display the results of ad-hoc SQL queries. In addition, it will also outline how to set up a connection to a **DataSet** at run-time.
Tutorial 6	In this tutorial, you will learn how to select and highlight records that satisfy specified criteria. A group of similar items is generally implemented as a collection in **C1List**. When manipulating a group of items in **C1List**, use techniques similar to those described here.
Tutorial 7	In this tutorial, you will learn how to use the **UnboundColumnFetch** event to display two fields (**First** and **Last**) together in one column.
Tutorial 8	In this tutorial, you will learn how to use the **Translate** and **DisplayValue** properties to assign text to numerical values for a more descriptive list.
Tutorial 9	In this tutorial, you will learn how to use the **Translate** and **DisplayValue** properties to display images in the column.
Tutorial 10	In this tutorial, you will learn how to change the list's display to highlight rows by creating row styles depending upon a value in the list. List uses the **FetchRowStyle** event to create style characteristics and apply them to rows dynamically.
Tutorial 11	In this tutorial, you will learn how to use the **AlternatingRows** property and built-in styles to apply alternating colors to list rows to improve their readability.
Tutorial 12	In this tutorial, you will learn how to write code to create a list with two horizontal splits, and then "fix" a pair of columns in the leftmost split.
Tutorial 13	In this tutorial you will learn to use the **SelectionMode** property to allow users to select multiple items in a list using a checkbox.
Tutorial 14	In this tutorial, you will learn how to use **Style** objects to display arbitrary bitmaps within list cells. In addition to font, color, and alignment settings, Style objects support both background and foreground

Sample	Description
	pictures. Background pictures can be centered, tiled, or stretched to fill the entire cell. Foreground pictures can be positioned relative to the cell text, and can also be displayed using a transparent color.
Tutorial 15	In this tutorial you will learn to format your list with the **OwnerDrawCell** event.
Tutorial 16	In this tutorial, you will learn how to use the **ScrollTips** property and the **FetchScrollTips** event to provide a tip box as the user scrolls through a list. You will also learn to create borders and colored borders for each item.
Tutorial 17	In this tutorial you will utilize the **Find** method in List. The **Find** method allows you to perform custom searches within the control.
Tutorial 18	In this tutorial, you will learn how to search a record from a list using the **MatchEntry**, **MatchCol**, **MatchCompare** and **MatchEntryTimeout** properties.
Tutorial 19	In this tutorial, you will learn how to use the **LookUp** feature to change the value of a **C1Combo** box.
Tutorial 20	In this tutorial, you will learn how to use the **LimitToList** feature to prevent users from entering an item which does not appear in the combo box list.
Tutorial 21	Design-time support for the **AddItem** mode of **C1List** is now available. The following steps demonstrate how to add items and set a layout at design time.

List for WinForms Task-Based Help

The task-based help assumes that you are familiar with programming in Visual Studio, know what a DataSet is, and know how to use bound controls in general. By following the steps outlined in the help, you will be able to utilize the features of List for WinForms.

The help uses an Access database, C1NWind.mdb, for bound data. It is assumed C1NWind.mdb is in the \common directory that was added, by default, during the WinForms Edition installation (**Documents\ComponentOne Samples\Common**), and it is referred to by filename instead of the full pathname for the sake of brevity.

> **Note**: Depending on where you store the projects and database files, you may need to change the location of the C1NWind.mdb reference in the projects.

Each task-based help topic also assumes that you have created a new .NET project and have placed a bound C1List component on the form. For additional information on creating a .NET project and binding List for WinForms to a data source, see Tutorial 1 - Binding C1List to a DataSet or Tutorial 2 - Binding C1Combo to a DataSet.

In the following task-based help topics, the C1ListDemo.mbd database connection is used with the SQL statement:

```
SELECT Company, Contacted, CustType, FirstName, LastName, Phone, UserCode FROM
Customers
```

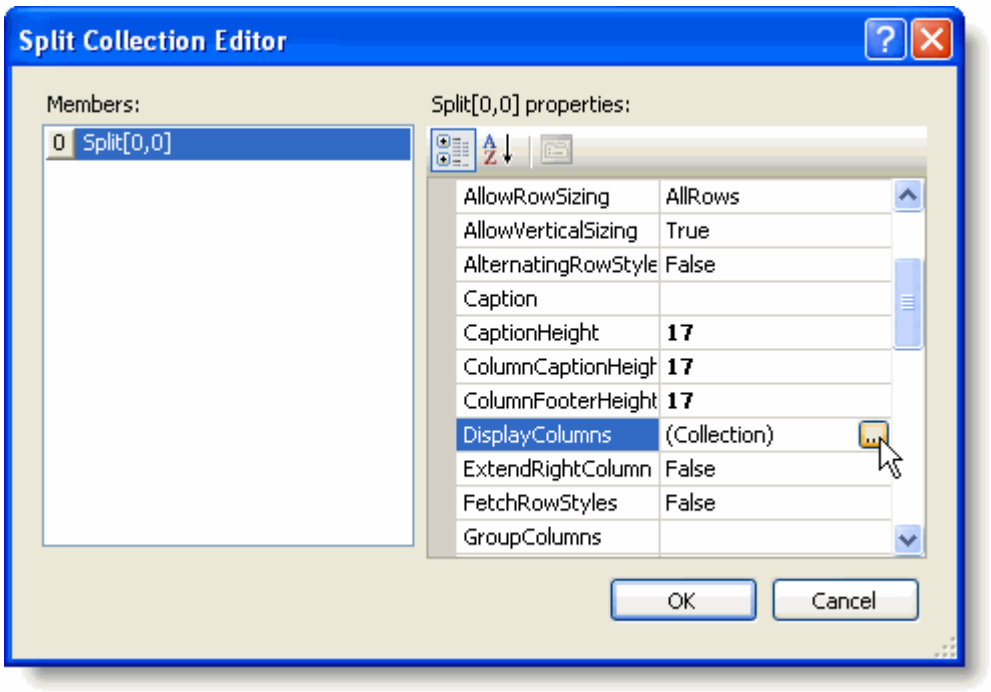
The generated DataSet is DsCustomer.

Applying a Style to Specific Cell Values

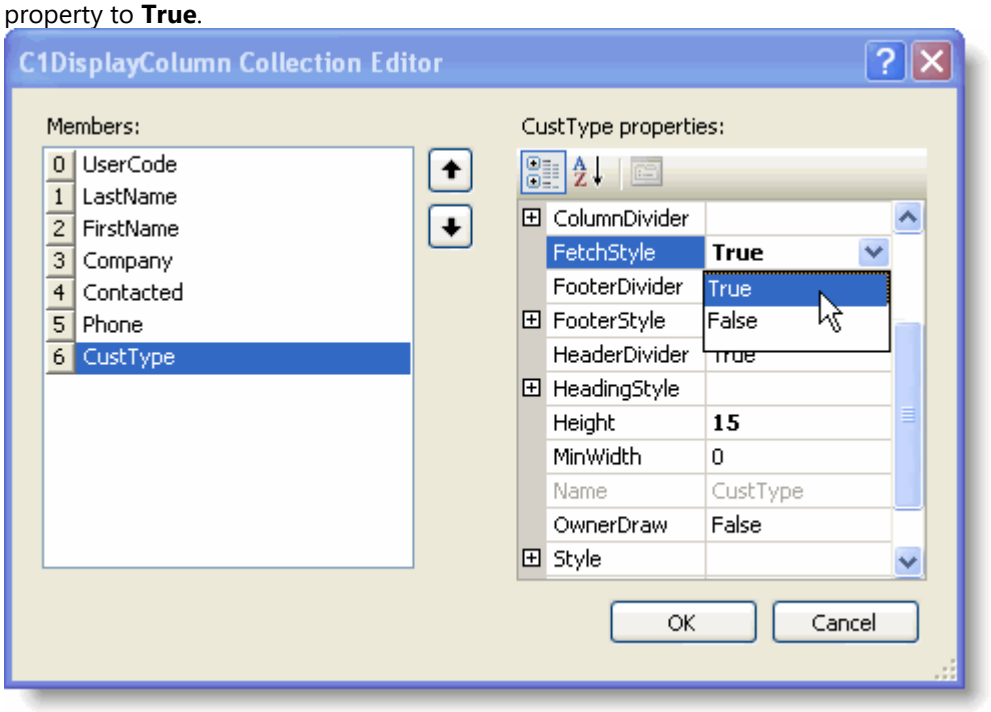
To apply a style to specific values in a cell, use the FetchStyle property and the FetchCellStyle event. You can set the FetchStyle property to True using the designer or in code:

Using the Designer

1. Click the **ellipsis** button next to the Splits property in the Properties window to open the **Split Collection Editor**.
2. In the **Split Collection Editor**, click the **ellipsis** button next to DisplayColumns property to open the **C1DisplayColumn Collection Editor**.



3. In the **C1DisplayColumn Collection Editor**, select a column from the **Members** list and set the FetchStyle property to **True**.



4. When finished click **OK** to close the **C1DisplayColumn Collection Editor** and the **Splits Collection Editor**.

Using Code

Alternatively, you can add code to set the FetchStyle property. In the following example the code was added to the Button1_Click event:

To write code in Visual Basic

Visual Basic
`Me.C1List1.Splits(0).DisplayColumns.Item("CustType").FetchStyle = True`

To write code in C#

C#
```
this.c1List1.Splits[0].DisplayColumns["CustType"].FetchStyle = true;
```

Add the following FetchCellStyle event to set all *CustType* greater than 3 to bold and blue.

To write code in Visual Basic

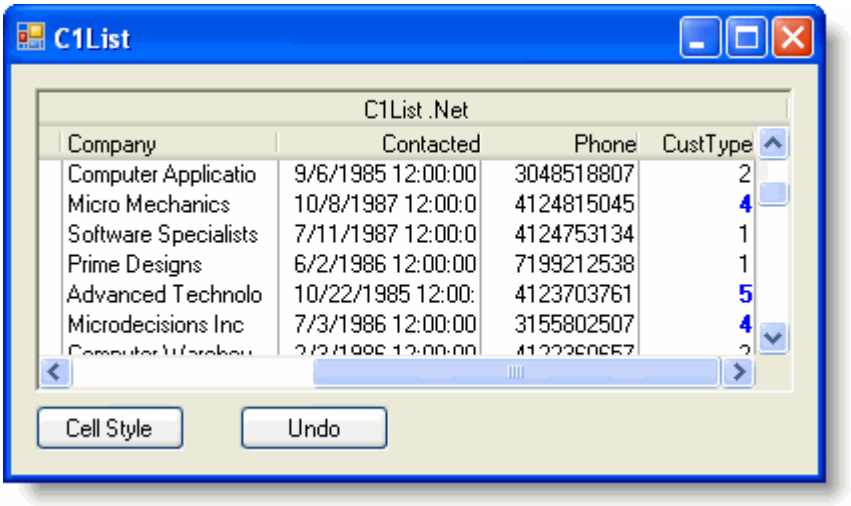
Visual Basic
```
Private Sub C1List1_FetchCellStyle(ByVal sender As Object, ByVal e As
C1.Win.C1List.FetchCellStyleEventArgs) Handles C1List1.FetchCellStyle
    Dim custtype As Long
    custtype = Val(Me.C1List1.Columns(e.Col).CellText(e.Row))
    If custtype > 3 Then
        e.CellStyle.ForeColor = System.Drawing.Color.Blue

        e.CellStyle.Font = New Font(e.CellStyle.Font, FontStyle.Bold)
    End If
End Sub
```

To write code in C#

C#
```
private void c1List1_FetchCellStyle(object sender,
C1.Win.C1List.FetchCellStyleEventArgs e)
{
    long custtype;
    custtype = long.Parse(this.c1List1.Columns[e.Col].CellText(e.Row));
        if (custtype > 3)
    {
        e.CellStyle.ForeColor = System.Drawing.Color.Blue;
        e.CellStyle.Font = new Font(e.CellStyle.Font, FontStyle.Bold);
    }
}
```

This topic illustrates the following:

When the **Cell Style** button is clicked, values in the *CustType* column greater than 3 are bold and blue.



Undo Applying a Style to Specific Cell Values

To undo applying a style to specific values, set the FetchStyle property to False by adding the following code. In this example the code was added to the **Undo** button's Click event.

To write code in Visual Basic

Visual Basic

```
Me.C1List1.Splits(0).DisplayColumns.Item("CustType").FetchStyle = False
```

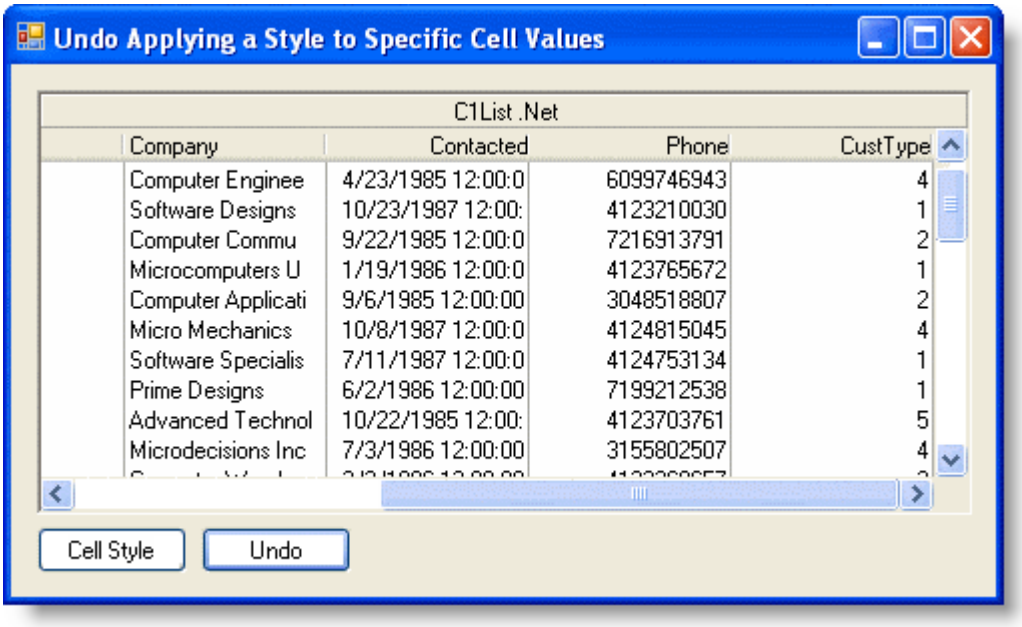
To write code in C#

C#

```
his.c1List1.Splits[0].DisplayColumns["CustType"].FetchStyle = false;
```

This topic illustrates the following:

When the **Undo** button is clicked, the rows return to the default color.

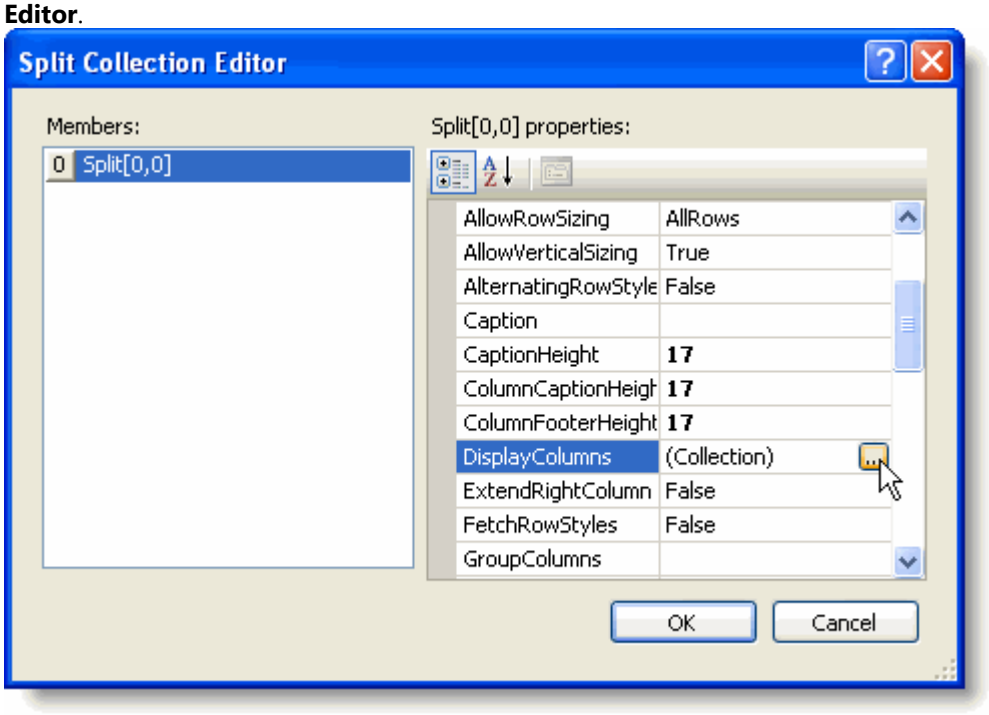


Changing the Font Style of a Column

To change the font style of a column, set the Style.Font property either in the designer or in code.

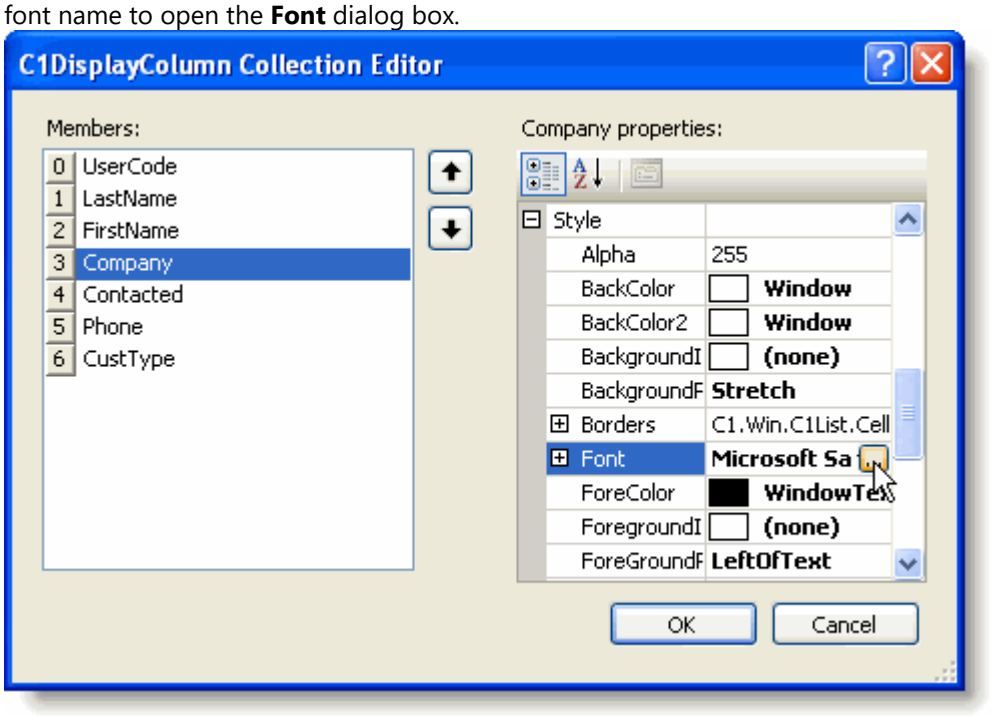
In the Designer

1. Click the **ellipsis** button next to the Splits property in the Properties window to open the **Split Collection Editor**.
2. Click the **ellipsis** button next to the DisplayColumns property to open the **C1DisplayColumn Collection Editor**.

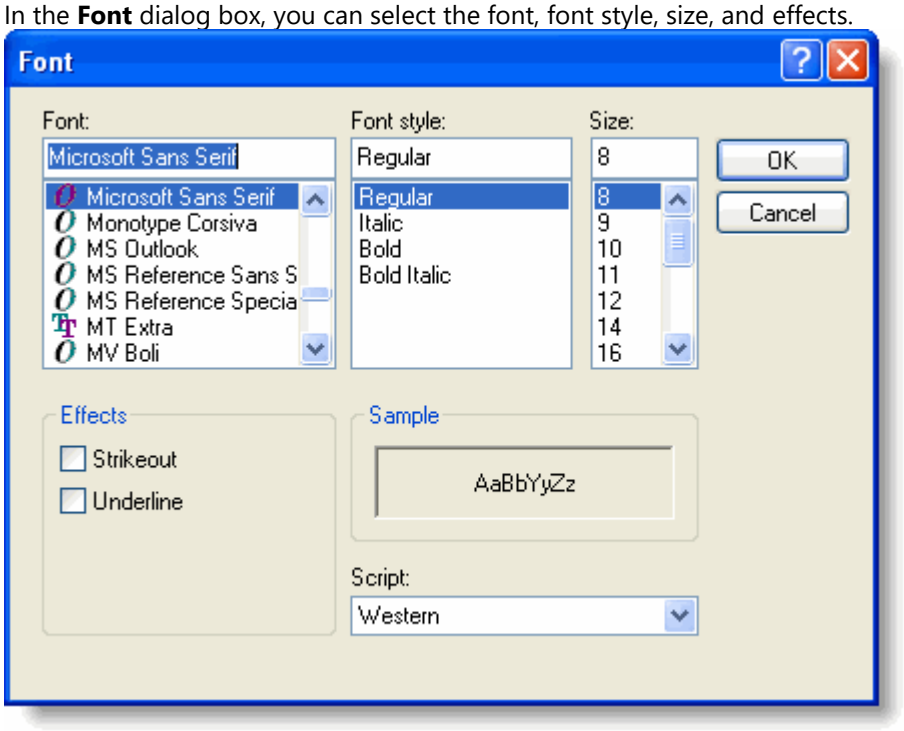


3. In the **C1DisplayColumn Collection Editor**, select the column that you would like to change in the **Members** list.
4. In the Properties list, expand the Style class and locate the Font property. Click the **ellipsis** button next to the

font name to open the **Font** dialog box.



5. In the **Font** dialog box, you can select the font, font style, size, and effects.



6. When finished click **OK** to close the **Font** dialog box, the **C1DisplayColumn Collection Editor**, and the **Splits Collection Editor**.

In Code

Add the following code, in this example the code was added to the Button1_Click event.

To write code in Visual Basic

Visual Basic

```
Dim fntFont As Font
fntFont = New Font("Impact",
Me.C1List1.Splits(0).DisplayColumns.Item("Company").Style.Font.Size,
FontStyle.Italic)
Me.C1List1.Splits(0).DisplayColumns.Item("Company").Style.Font = fntFont
```

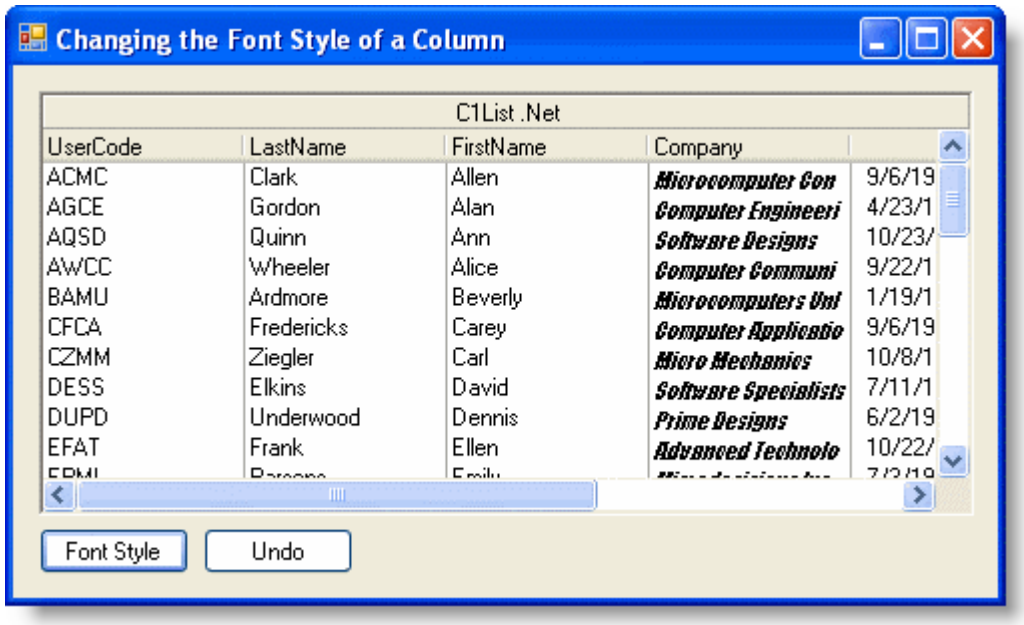
To write code in C#

C#

```
Font fntFont;
fntFont = new Font("Impact",
this.c1List1.Splits[0].DisplayColumns["Company"].Style.Font.Size, FontStyle.Italic);
this.c1List1.Splits[0].DisplayColumns["Company"].Style.Font = fntFont;
```

This topic illustrates the following:

When the **Font Style** button is clicked, the font in the *Company* column is changed to Impact and italic.



Undo Changing the Font Style of a Column

To undo changing the font style of a column, either set the font in the designer to Microsoft Sans Serif or in code by adding the following code. In this example the code was added to the **Undo** button's Click event.

To write code in Visual Basic

Visual Basic

```
Me.C1List1.Splits(0).DisplayColumns.Item("Company").Style.Font = Font
```

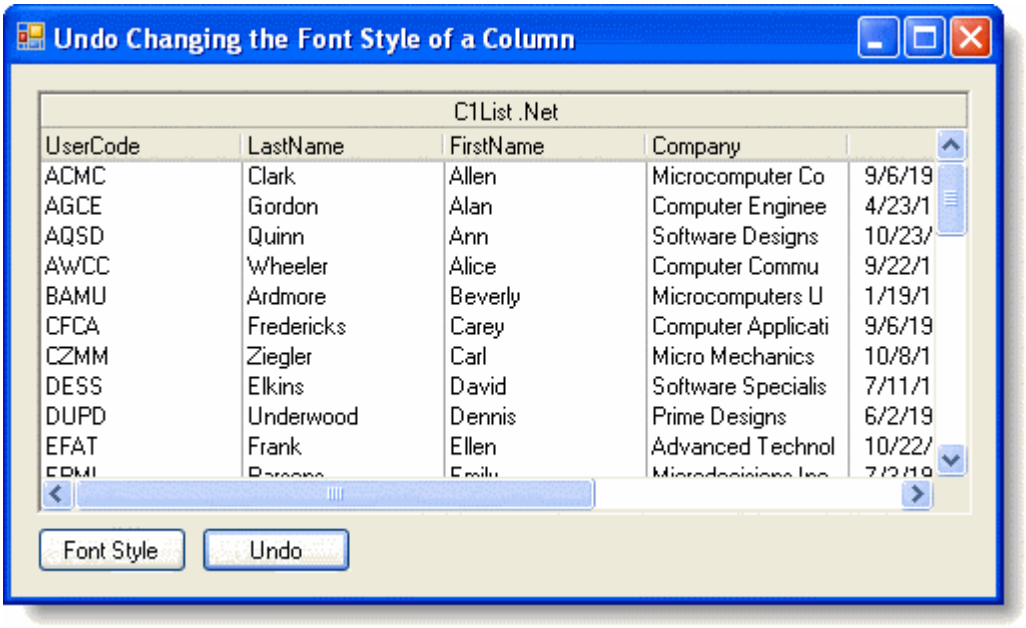
To write code in C#

C#

```
this.c1List1.Splits[0].DisplayColumns["Company"].Style.Font = Font;
```

This topic illustrates the following:

When the **Undo** button is clicked, the font in the *Company* column is returned to the default font.



Clearing a Sort

To undo a sort, use the DataView.Sort property to return the data to the default view. Add the following code, in this example the code was added to the Button1_Click event:

To write code in Visual Basic

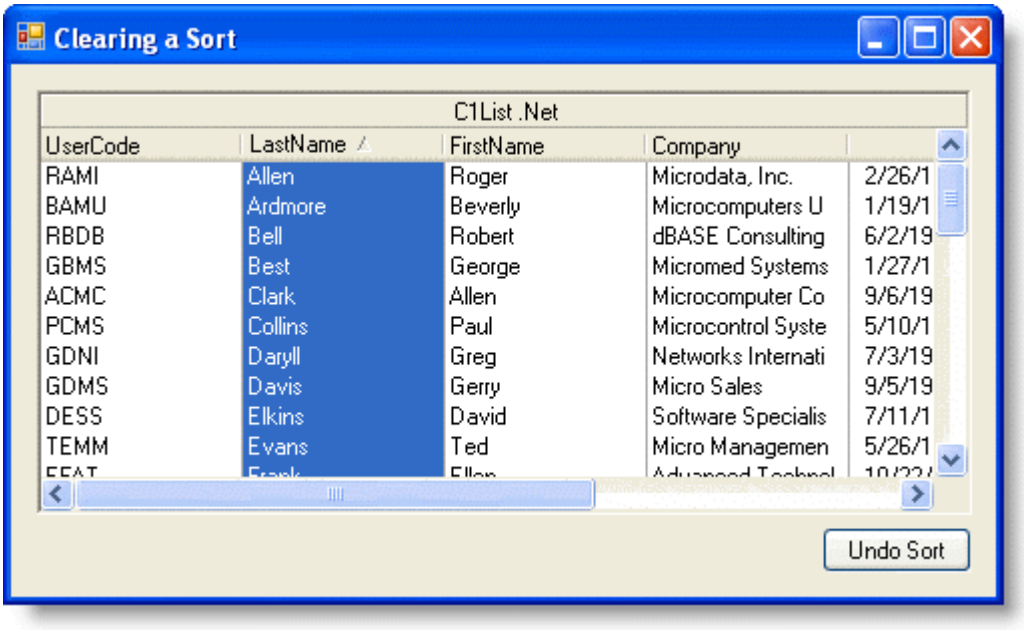
Visual Basic
`Me.CustomersBindingSource.Sort = ""`

To write code in C#

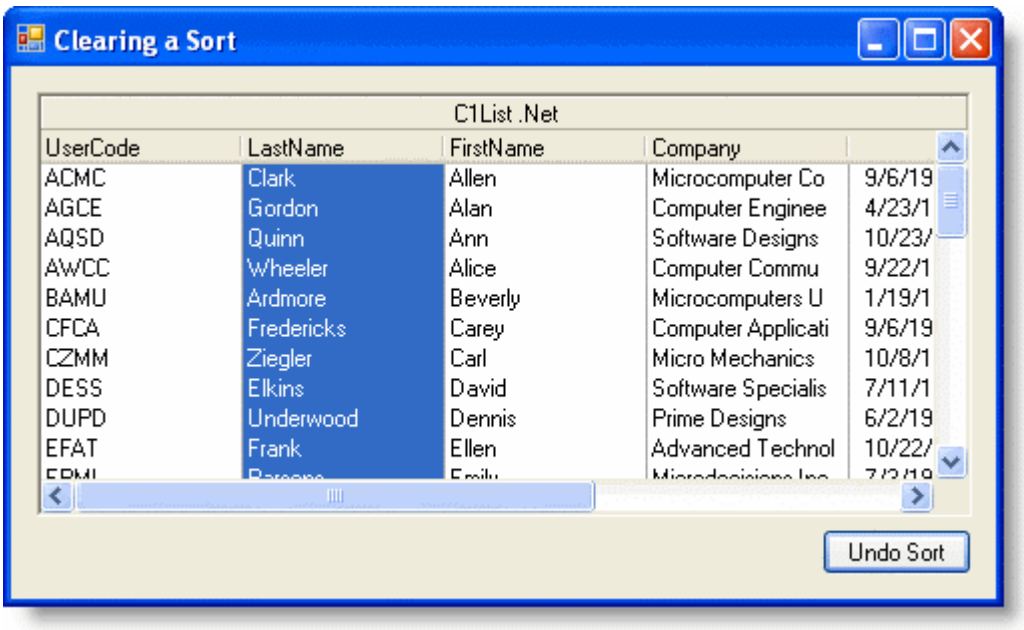
C#
`this.CustomersBindingSource.Sort = "";`

This topic illustrates the following:

Here is the table sorted by the *LastName* column:



After clicking the **Undo Sort** button, the sort is undone.



Displaying Multiple Columns in C1Combo

You can display multiple columns in the C1Combo control through the Close event of the C1Combo to include multiple column values in the display area of the C1Combo control.

1. Bind the DataSet to the C1Combo control. For more information on how to bind to a C1Combo control, see Tutorial 2 - Binding C1Combo to a DataSet.
2. Set the columns you would like visible using the **C1DisplayColumn Collection Editor**. For more information on how to make columns visible, see Hiding or Displaying a Column. For this example, hide all columns except for *Company*.
3. Add the following Close event for C1Combo:

   To write code in Visual Basic

Visual Basic

```
Dim _textChanging As Boolean = False

Private Sub C1Combo1_Close(ByVal sender As Object, ByVal e As System.EventArgs)
Handles C1Combo1.Close
    Dim index As Integer = Me.C1Combo1.SelectedIndex()
    If (index >= 0) Then
        _textChanging = True
        Me.C1Combo1.Text = (Me.C1Combo1.Columns("FirstName").CellText(index) +
(" " + (Me.C1Combo1.Columns("LastName").CellText(index) + (" at " +
Me.C1Combo1.Columns("Company").CellText(index)))))
        _textChanging = False
    End If
End Sub

Private Sub C1Combo1_SelChange(ByVal sender As Object, ByVal e As
System.ComponentModel.CancelEventArgs)
    e.Cancel = _textChanging

End Sub
```

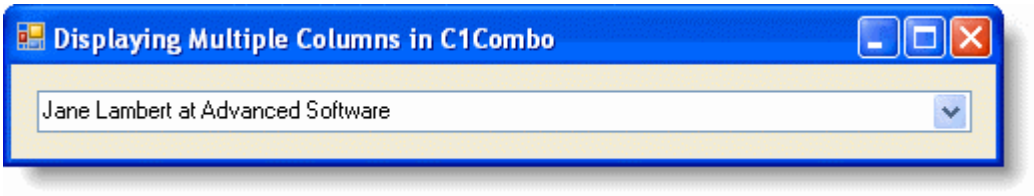
To write code in C#

C#

```
bool _textChanging = false;

private void c1Combo1_Close(object sender, System.EventArgs e)
{
    int index = this.c1Combo1.SelectedIndex;
    if (index >= 0)
    {
        _textChanging = true;
        this.c1Combo1.Text = this.c1Combo1.Columns["FirstName"].CellText(index)
+ " " + this.c1Combo1.Columns["LastName"].CellText(index) + " at " +
this.c1Combo1.Columns["Company"].CellText(index);
        _textChanging = false;
    }
}

private void c1Combo1_SelChange(object sender,
System.ComponentModel.CancelEventArgs e)
{
    e.Cancel = _textChanging;
}
```

This topic illustrates the following:

When a company is selected from the list, the first and last name of the contact and the company appears in the C1Combo control.

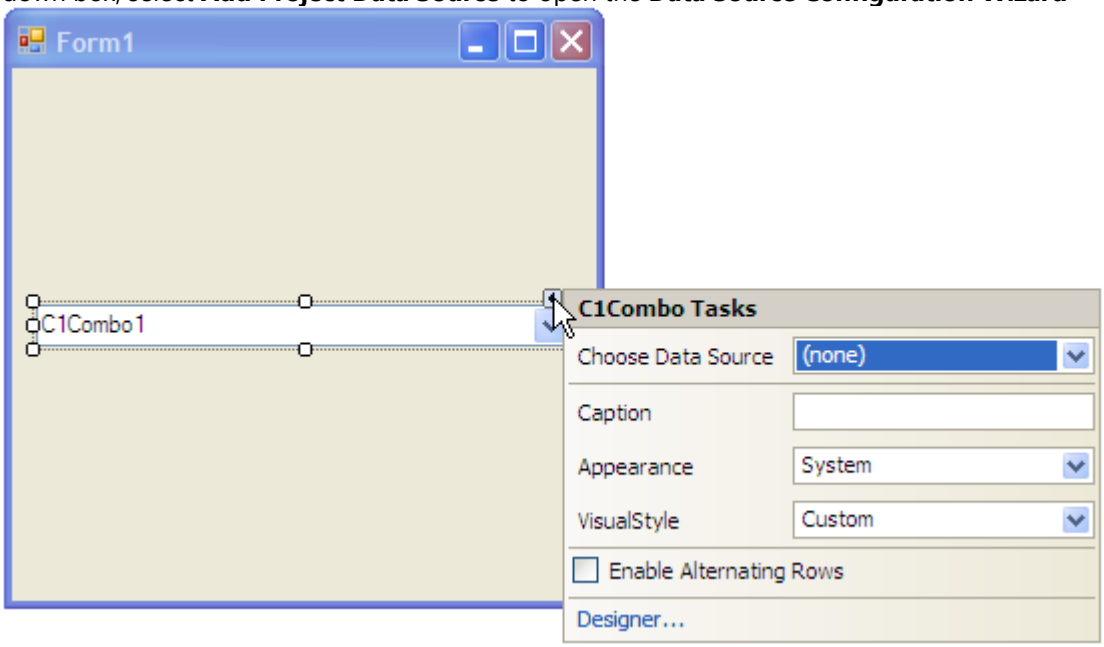


Creating a ComboBox with AutoCompletion and KeyDown

By manipulating the properties of the C1Combo control, you can customize how users will view and utilize a ComboBox. The AutoCompletion property allows users the option to have text completed for them without entering the entire text. The KeyDown event allows users to press the down arrow key and see the entire contents of the drop down box.

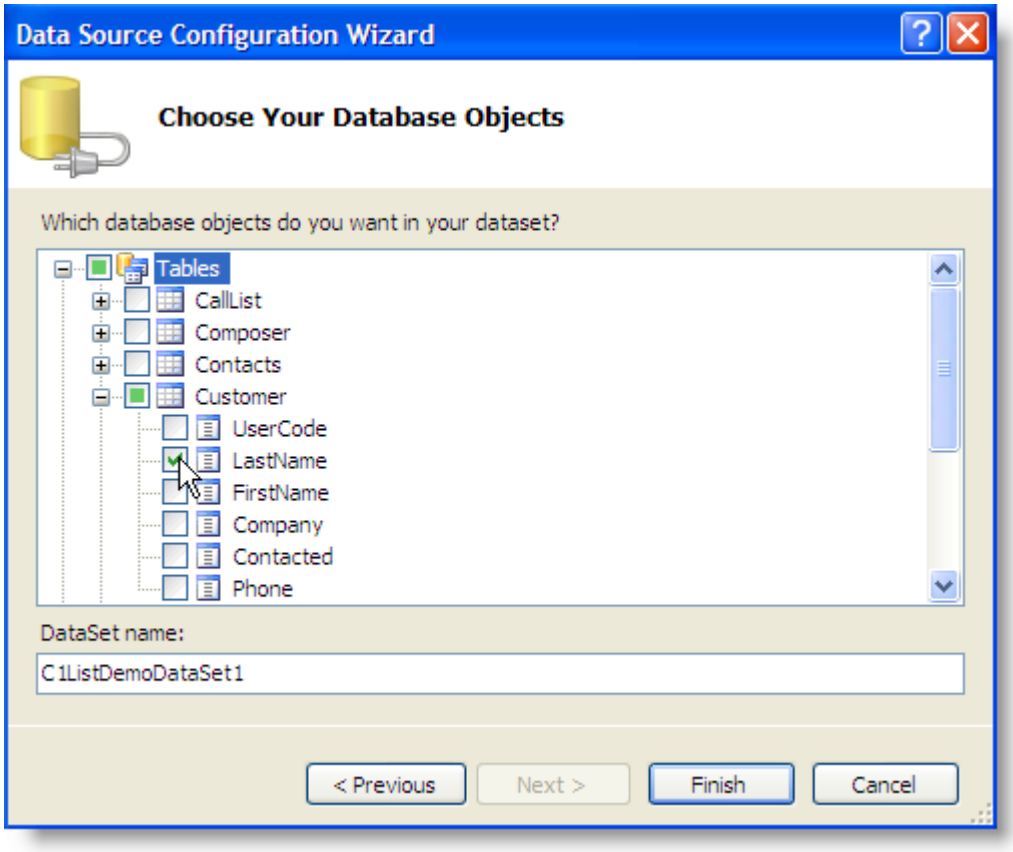
To enable AutoCompletion and KeyDown

1. Create a .NET project then add the C1List component to your form.
2. Add the C1Combo control to your form.
3. Click the C1Combo smart tag (▶) to open the **C1Combo Tasks** menu. From the **Choose Data Source** drop-down box, select **Add Project Data Source** to open the **Data Source Configuration Wizard**



4. In the **Data Source Configuration Wizard**, select **Database** and click **Next**.
5. Select **Dataset** and then click **Next**.
6. Click the **New Connection** button to create a new connection or choose one from the drop-down list.
7. In the **Add Connection** dialog window, click **Browse** to specify the location of the data and enter the correct login information. Either select a connection to **C1NWind.mdb** (on installation located in **Documents\ComponentOne Samples\Common**) or create a new connection to this database.
8. Click the **Test Connection** button to make sure that you have successfully connected to the database or server and click **OK**. The new string appears on the page.
9. Click the **Next** button to continue. A dialog box will appear asking if you would like to add the data file to your project and modify the connection string. Click **No**.
10. Save the connection string in the application configuration file by checking the **Yes**, save the connection as box and entering a name. Click **Next** to continue.
11. On the **Choose Your Database Objects** page, select the tables and fields that you would like in your dataset. For this example expand the **Tables** node, then expand the **Customers** node, then place a check next to the

LastName box. Click **Finish** to exit the wizard.

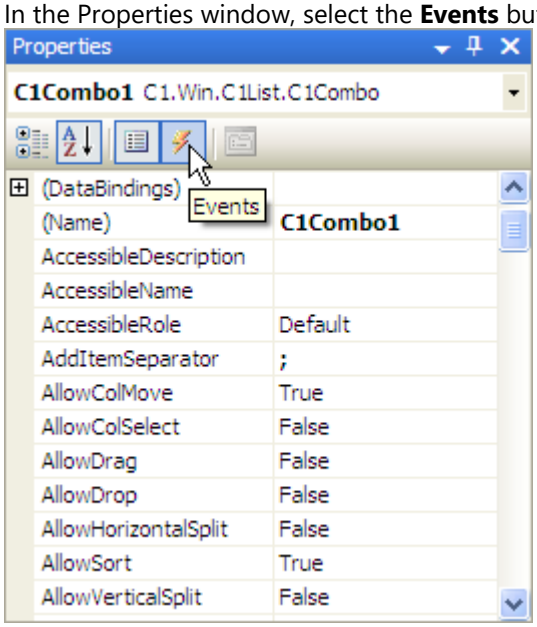


A DataSet and connection string are added to your project.

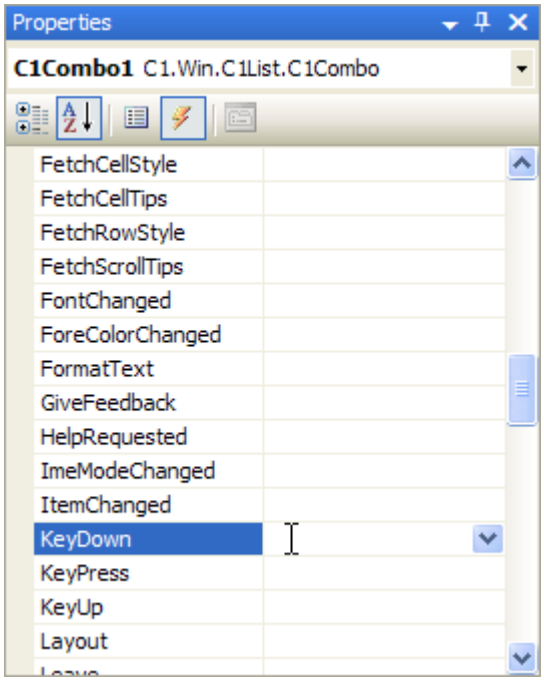
12. Set the AutoCompletion property and the AutoDropDown property to **True**.
13. Set the MatchCompare property to **PartiallyEqual**. This enhances the auto complete feature by specifying a comparison method for input strings.

> **Note:** For values other than **MatchCompareEnum.Equal** and **MatchCompareEnum.PartiallyEqual**, it is advisable to turn off **AutoCompletion**.

14. In the Properties window, select the **Events** button.



15. Double-click the blank space to the right of the KeyDown event. This will add an event to your code.



16. Add the following code to the C1Combo1_KeyDown event:

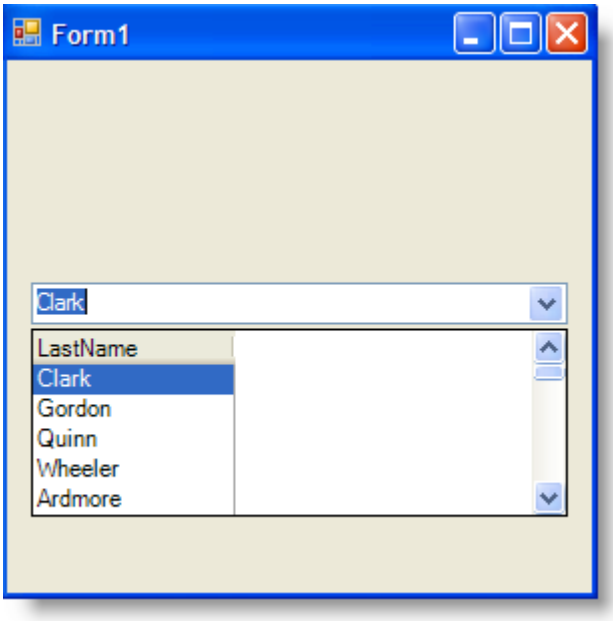
To write code in Visual Basic

```
Visual Basic
Me.C1Combo1.DroppedDown = True
```

To write code in C#

```
C#
this.C1Combo1.DroppedDown = true;
```

When you run the application, the AutoCompletion feature is available and, just by pressing the down arrow on the keyboard, the contents of the drop-down menu are visible.

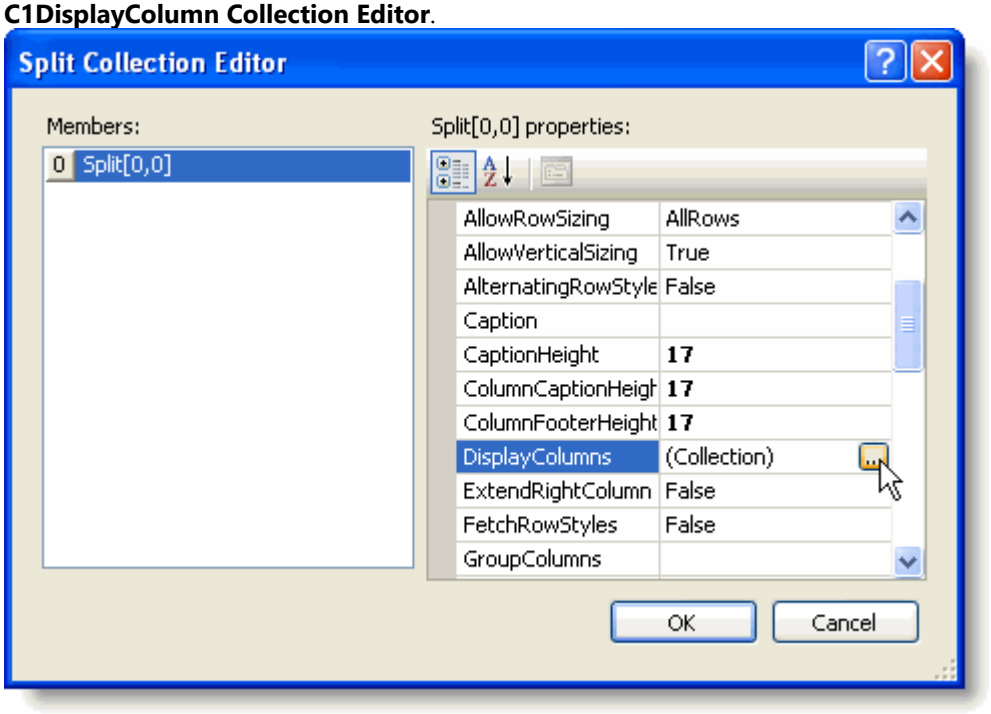


Hiding or Displaying a Column

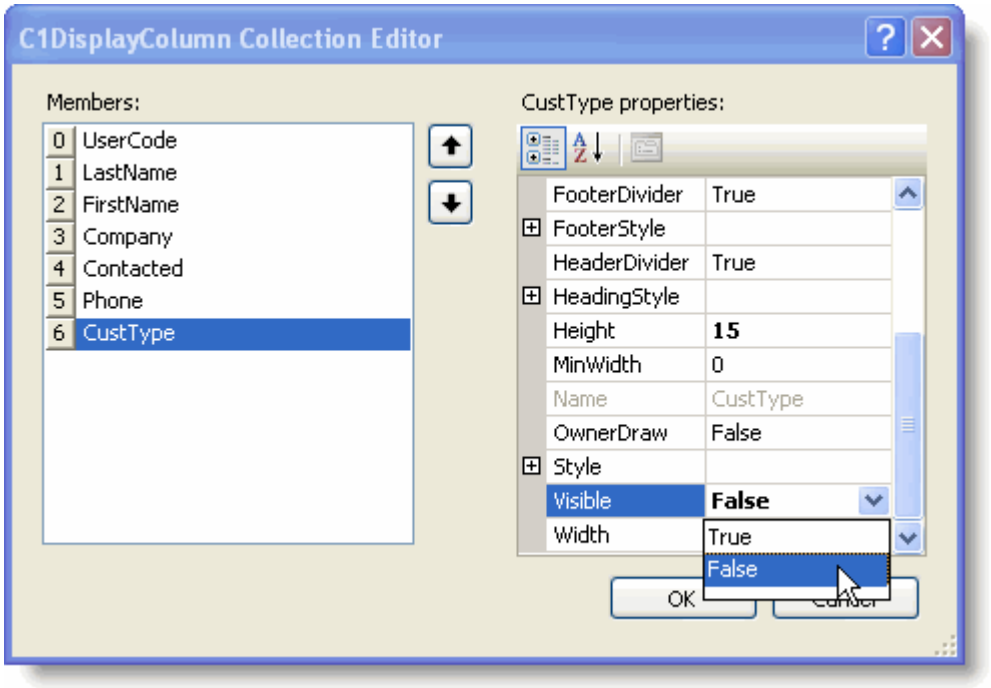
To hide a column on a C1List control, set the Visible property for the column to False either in the **C1DisplayColumn Collection Editor** or in code. To make a column visible, set the Visible property to **True**.

In the Designer

1. Open the **Split Collection Editor** in the Properties window by clicking the **ellipsis** button after the Splits property.
2. In the **Split Collection Editor**, click the **ellipsis** button next to the DisplayColumns property to open the **C1DisplayColumn Collection Editor**.



3. In the **C1DisplayColumn Collection Editor**, set the Visible property to **False** for the column that you would like hidden.



In Code

Add the following code to the Form_Load event:

To write code in Visual Basic

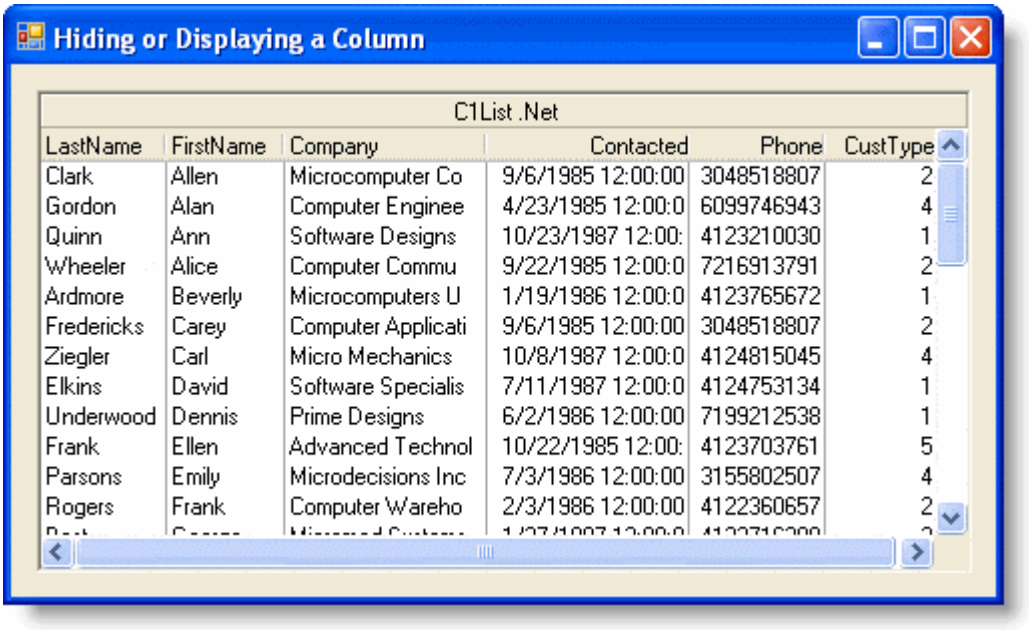
```
Visual Basic
Me.C1List1.Splits(0).DisplayColumns.Item("UserCode").Visible = False
```

To write code in C#

```
C#
this.c1List1.Splits[0].DisplayColumns["UserCode"].Visible = false;
```

This topic illustrates the following:

The final result will look like this, where the *UserCode* column is hidden.



Hiding Rows Using RowFilter

To hide a row on a C1List control bound to a DataSource, use a filter on the default view of the DataTable.

Single Filter

For a single filter, add the following code to the Form_Load event to create a filter with the criteria for the rows that you would like to remain visible:

To write code in Visual Basic

Visual Basic

```
Me.CustomersBindingSource.Filter = "(CustType='1' OR CustType='3') AND Company LIKE
'%Computer%'"
```

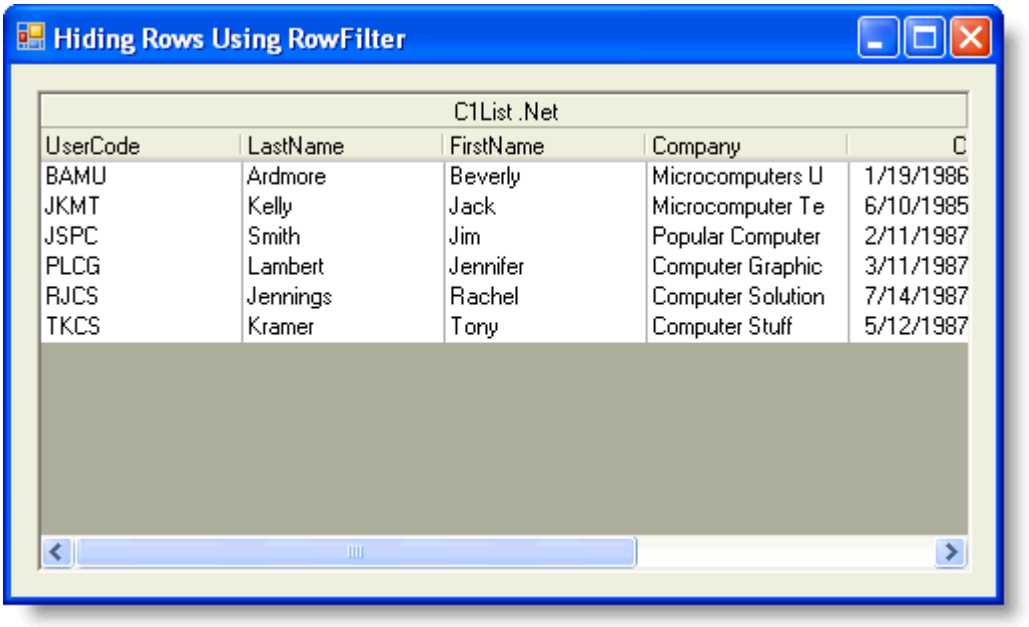
To write code in C#

C#

```
this.CustomersBindingSource.Filter = "(CustType='1' OR CustType='3') AND Company LIKE
'%Computer%'";
```

This topic illustrates the following:

The final result will look like this, where only companies with the word computer and *CustType* equal to 1 or 3 appear:



Multiple Filters

You can also create multiple filters for a C1List control using a ComboBox to choose which filter you would like to apply. Add the following SelectedIndexChanged event of the ComboBox:

To write code in Visual Basic

Visual Basic

```
Private Sub ComboBox1_SelectedIndexChanged(ByVal sender As Object, ByVal e As
System.EventArgs) Handles ComboBox1.SelectedIndexChanged

    Select Case ComboBox1.SelectedItem

        Case "Prospective"

            Me.CustomersBindingSource.Filter = "CustType='1' AND Company LIKE
'%Computer%'"

        Case "Normal"

            Me.CustomersBindingSource.Filter = "CustType='2' AND Company LIKE
'%Computer%'"

        Case "Buyer"

            Me.CustomersBindingSource.Filter = "CustType='3' AND Company LIKE
'%Computer%'"

        Case "Distributor"

            Me.CustomersBindingSource.Filter = "CustType='4' AND Company LIKE
'%Computer%'"
```

```
        Case "Other"

            Me.CustomersBindingSource.Filter = "CustType='5' AND Company LIKE
'%Computer%'"

    End Select

End Sub
```

To write code in C#

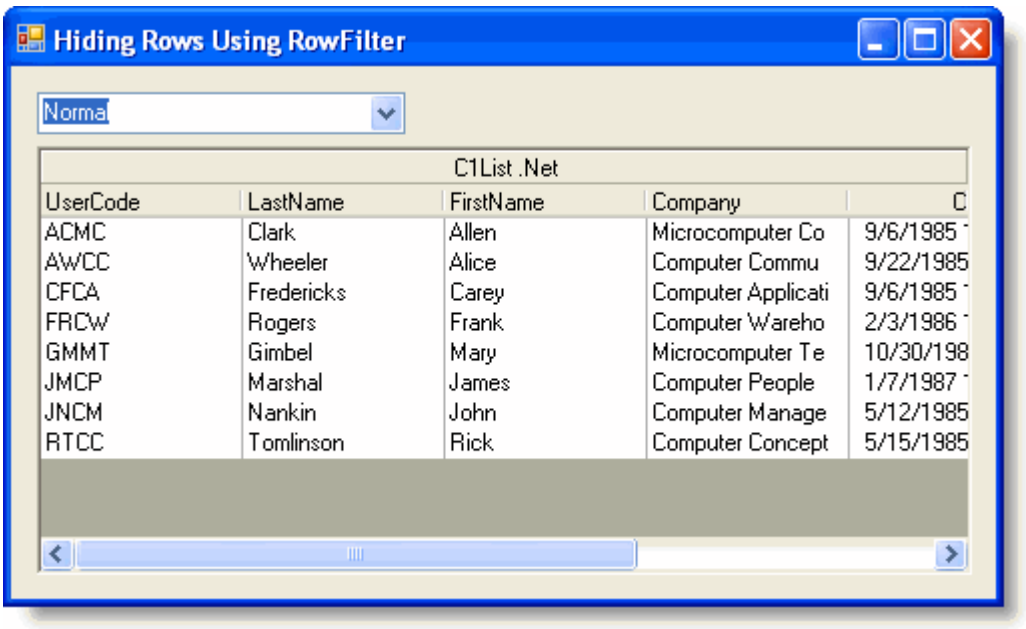
C#

```
private void comboBox1_SelectedIndexChanged(object sender, System.EventArgs e)

{

    switch (comboBox1.SelectedItem)

    {

        case "Prospective" :

        {

            this.CustomersBindingSource.Filter = "CustType'1' AND Company LIKE
'%Computer%'";

            break;

        }

        case "Normal" :

        {

            this.CustomersBindingSource.Filter = "CustType'2' AND Company LIKE
'%Computer%'";

            break;

        }

        case "Buyer" :

        {

            this.CustomersBindingSource.Filter = "CustType'3' AND Company LIKE
'%Computer%'";

            break;
```

```
        }

        case "Distributor" :

        {

            this.CustomersBindingSource.Filter = "CustType'4' AND Company LIKE
'%Computer%'";

            break;

        }

        case "Other" :

        {

            this.CustomersBindingSource.Filter = "CustType'5' AND Company LIKE
'%Computer%'";

            break;

        }

    }

}
```

This topic illustrates the following:

When a customer type is selected from the drop-down list, only companies matching that customer type and containing the word computer will appear.



Note: If there are no results that meet the filter requirements, then nothing will appear in the C1List control.

Undo Hiding Rows Using RowFilter

To undo hiding a row, use the RowFilter property to return to the default view. Add the following RowFilter code, in this example the code was added to the **Clear RowFilter** button's Click event:

To write code in Visual Basic

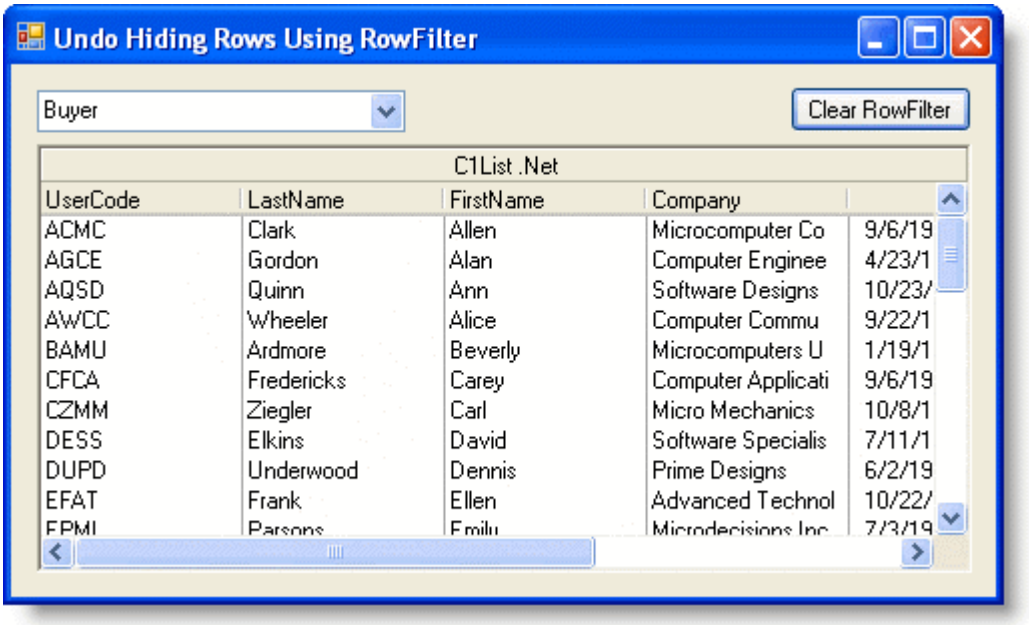
```
Visual Basic
Me.CustomersBindingSource.Filter = ""
```

To write code in C#

```
C#
this.CustomersBindingSource.Filter = "";
```

This topic illustrates the following:

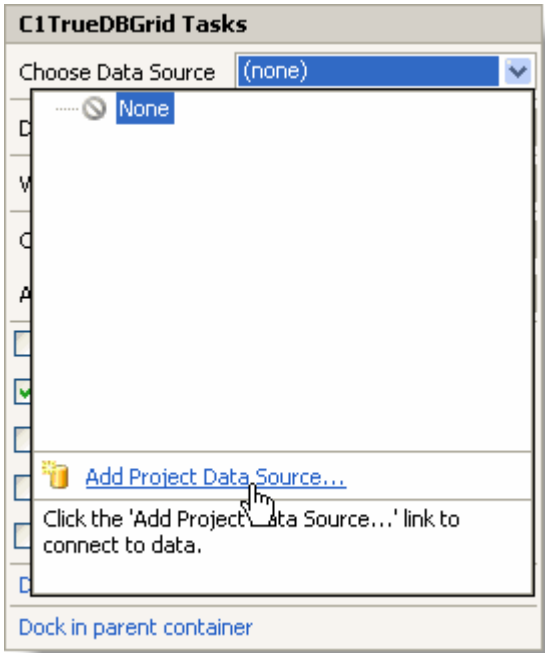
When the **Clear RowFilter** button is clicked, all of the rows in the C1List control are visible.



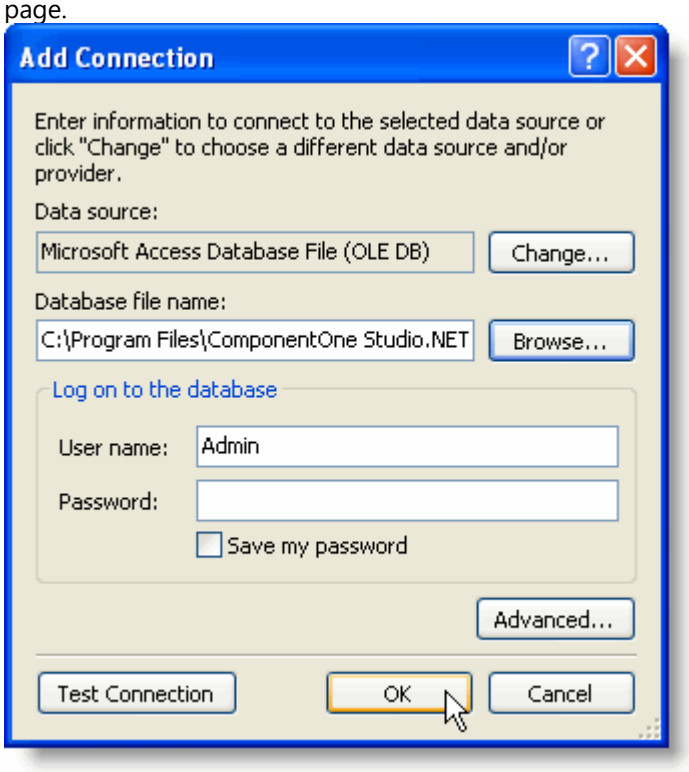
Modifying the DataSource

To change the location of the **C1NWind.mdb** reference, you can edit the DataSource property of C1List.

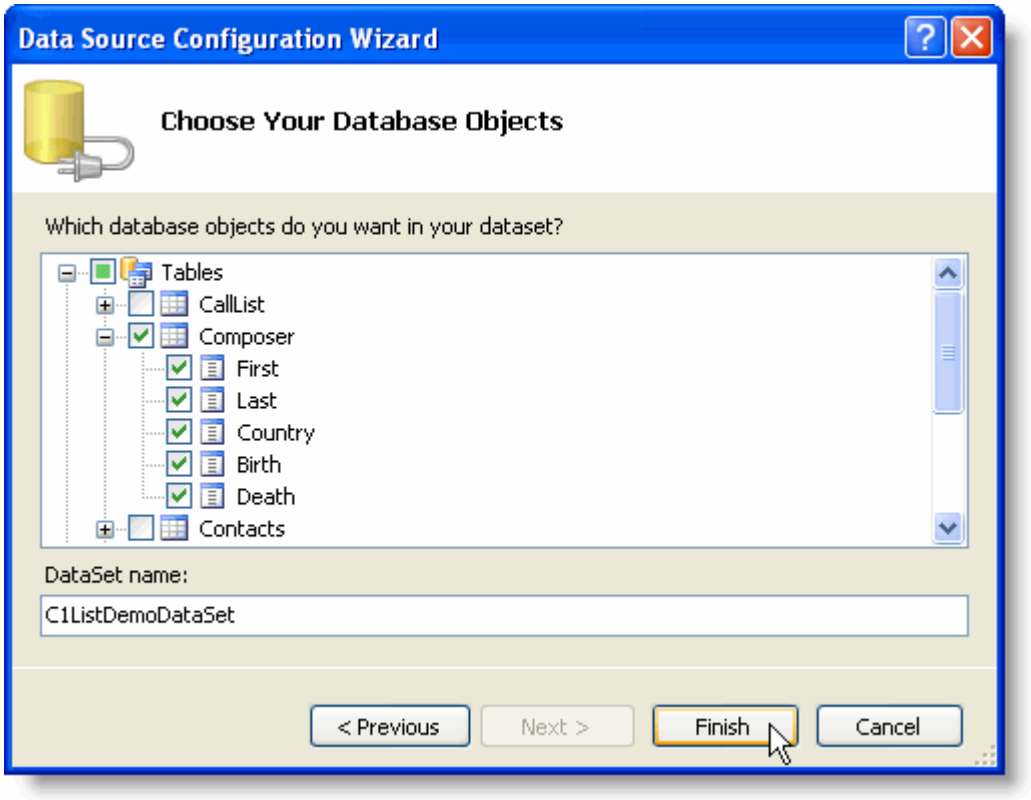
1. Click the C1List smart tag to open the C1List **Tasks** menu, select **Add Project Data Source** from the **Choose Data Source** drop-down box to open the **Data Source Configuration Wizard**.



2. The **Data Source Configuration Wizard** appears. Select **Database** on the **Choose a Data Source Type** page, and click **Next**.
3. Click the **New Connection** button to create a new connection or choose one from the drop-down list.
4. The **Add Connection** dialog window appears. Click **Browse** to specify the location of the data and enter the correct login information. Click the **Test Connection** button to make sure that you have successfully connected to the database or server and click **OK**. The new string appears in the on the **Choose Your Data Connection** page.



5. Click the **Next** button to continue. A dialog box will appear asking if you would like to add the data file to your project and modify the connection string. Click **No**.
6. Save the connection string in the application configuration file by checking the **Yes**, save the connection as box and entering a name. Click **Next** to continue.
7. On the **Choose Your Database Objects** page, select the tables and fields that you would like in your dataset. Enter a name for your DataSet in the **DataSet name** box and click **Finish** to exit the wizard.



A DataSet and connection string are added to your project. Additionally, Visual Studio automatically creates the code to fill the DataSet.

Selecting a Row by Data

To select a row by a value in a specific row, use the SelectedValue property.

1. Bind the DataSet to the C1List component.
2. Set the following properties in the Properties window:

Property	Value
C1ListBase.DisplayMember	Company
C1ListBase.ValueMember	Phone

Or in code, add the following to the Form_Load event:

To write code in Visual Basic

```
Visual Basic
Me.C1List1.DisplayMember = "Company"
Me.C1List1.ValueMember = "Phone"
```

To write code in C#

```
C#
this.c1List1.DisplayMember = "Company";
this.c1List1.ValueMember = "Phone";
```

3. Use the SelectedValue property to set the row value. In this example, the value is 4123210030, the phone number of Ann Quinn at Software Designs. Add the following code to the Form_Load event:

To write code in Visual Basic

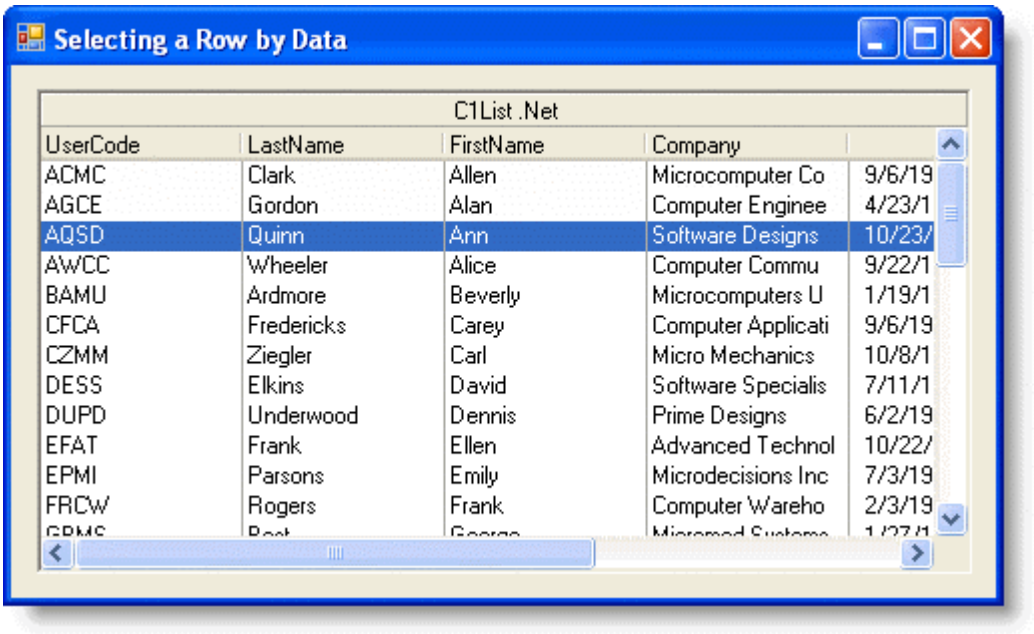
```
Visual Basic
Me.C1List1.SelectedValue = 4123210030
```

To write code in C#

```
C#
this.c1List1.SelectedValue = 4123210030;
```

This topic illustrates the following:

When finished, run the application. The result will look like this, with the SelectedValue highlighted.



Setting the Foreground Color of a Row

To set the foreground color of a row, use the FetchRowStyles property and the FetchRowStyle event.

1. Set the FetchRowStyles property to **True**:

In the Designer

Locate the FetchRowStyles property in the Properties window and set it to **True**:

In Code

Alternatively you can add code to set the FetchRowStyles property. Add the following code to the project, for example to the Button1_Click event:

To write code in Visual Basic

```
Visual Basic
Me.C1List1.FetchRowStyles = True
```

To write code in C#

```
C#
this.c1List1.FetchRowStyles = true;
```

2. Add the following FetchRowStyle event to identify the customer type by color.

To write code in Visual Basic

```
Visual Basic
Private Sub C1List1_FetchRowStyle(ByVal sender As Object, ByVal e As
C1.Win.C1List.FetchRowStyleEventArgs) Handles C1List1.FetchRowStyle
       If C1List1.Columns("CustType").CellText(e.Row) = "1" Then
              e.CellStyle.ForeColor = System.Drawing.Color.DarkGoldenrod
       ElseIf C1List1.Columns("CustType").CellText(e.Row) = "2" Then
              e.CellStyle.ForeColor = System.Drawing.Color.Crimson
       ElseIf C1List1.Columns("CustType").CellText(e.Row) = "3" Then
              e.CellStyle.ForeColor = System.Drawing.Color.DarkGreen
       ElseIf C1List1.Columns("CustType").CellText(e.Row) = "4" Then
              e.CellStyle.ForeColor = System.Drawing.Color.DarkCyan
       ElseIf C1List1.Columns("CustType").CellText(e.Row) = "5" Then
              e.CellStyle.ForeColor = System.Drawing.Color.DarkOrchid
       End If
End Sub
```

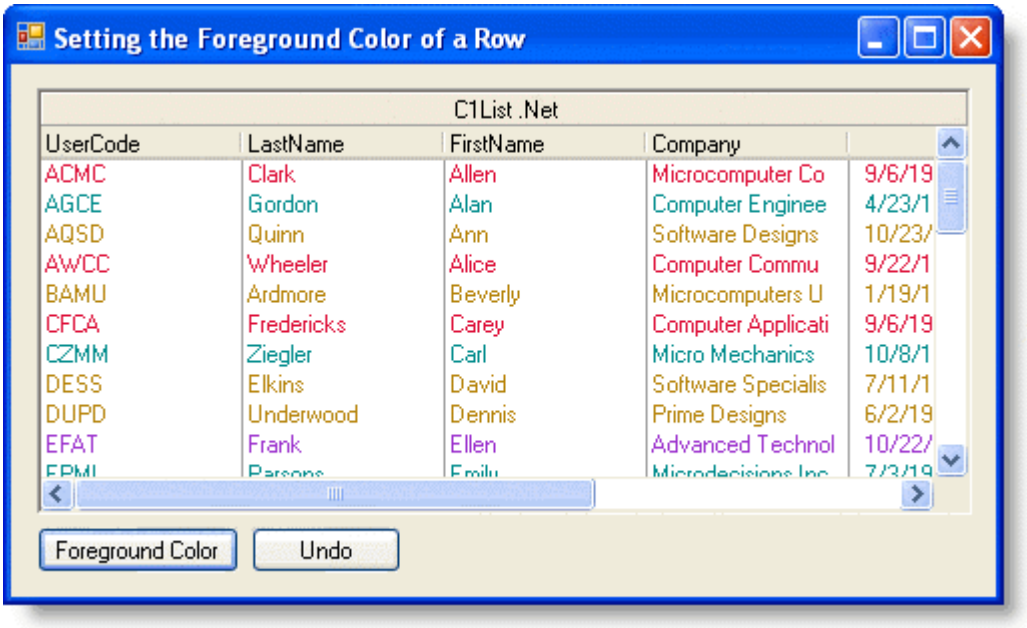
To write code in C#

```
C#
private void C1List1_FetchRowStyle( object sender,
C1.Win.C1List.FetchRowStyleEventArgs e)
{
       if (c1List1.Columns["CustType"].CellText(e.Row) == "1")
       {
              e.CellStyle.ForeColor = System.Drawing.Color.DarkGoldenrod;
       }
       else if (c1List1.Columns["CustType"].CellText(e.Row) == "2")
       {
              e.CellStyle.ForeColor = System.Drawing.Color.Crimson;
       }
       else if (c1List1.Columns["CustType"].CellText(e.Row) == "3")
       {
              e.CellStyle.ForeColor = System.Drawing.Color.DarkGreen;
       }
       else if (c1List1.Columns["CustType"].CellText(e.Row) == "4")
       {
              e.CellStyle.ForeColor = System.Drawing.Color.DarkCyan;
```

```
        }
        else if (c1List1.Columns["CustType"].CellText(e.Row) == "5")
        {
                e.CellStyle.ForeColor = System.Drawing.Color.DarkOrchid;
        }
}
```

This topic illustrates the following:

When the **Foreground Color** button is clicked, the customer type is identified by color.



Undo Setting the Foreground Color of a Row

To undo setting the foreground color of a row, set the FetchRowStyles property to False by adding the following code. In this example the code was added to the **Undo** button's Click event:

To write code in Visual Basic

Visual Basic

```
Me.C1List1.FetchRowStyles = False
```

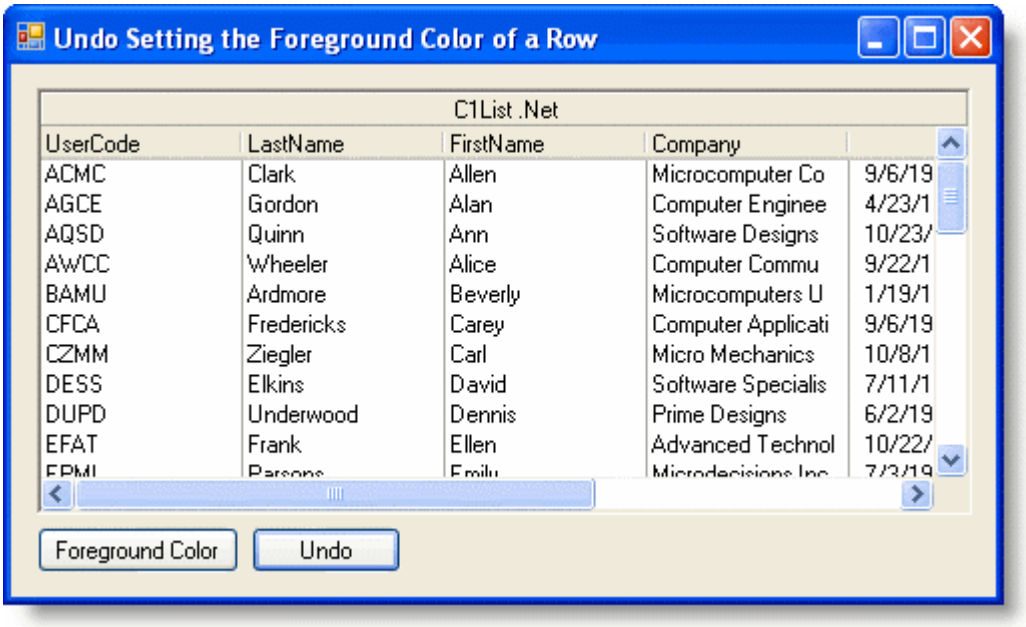
To write code in C#

C#

```
this.c1List1.FetchRowStyles = false;
```

This topic illustrates the following:

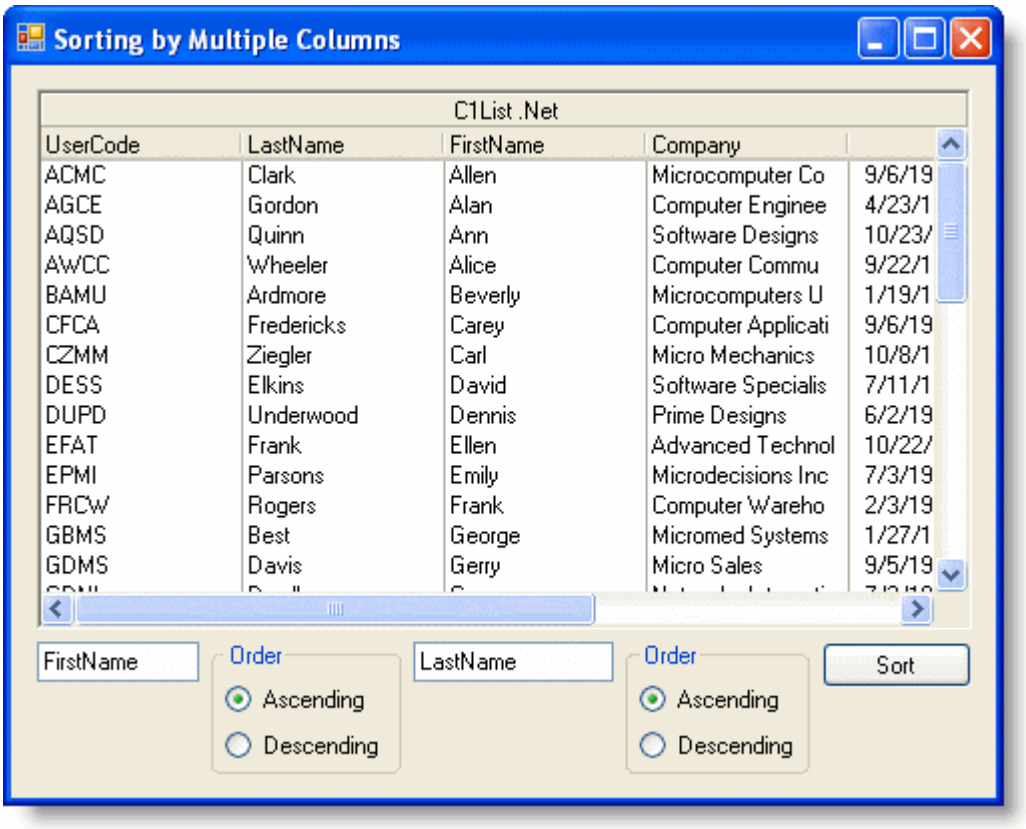
When the **Undo** button is clicked, the rows return to the default color.



Sorting by Multiple Columns

To sort by multiple columns, use the DataView.Sort property to specify which columns you would like to sort by.

This is the C1List before the data sort:



Notice that on this form, you can choose which columns you would like to sort by and how you would like to sort them, either in ascending or descending order. After the user enters the column names and chooses the order, clicking the Sort button sorts the data.

Add the following code to your form, in this example the code was added to the Button1_Click event:

To write code in Visual Basic

Visual Basic

```
If RadioButton1.Checked = True And RadioButton3.Checked = True Then
       Dim SortExpression = TextBox1.Text + " ASC" + "," + TextBox2.Text + " ASC"
       Me.CustomersBindingSource.Sort = SortExpression
ElseIf RadioButton1.Checked = True And RadioButton4.Checked = True Then
       Dim SortExpression = TextBox1.Text + " ASC" + "," + TextBox2.Text + " DESC"
       Me.CustomersBindingSource.Sort = SortExpression
ElseIf RadioButton2.Checked = True And RadioButton3.Checked = True Then
       Dim SortExpression = TextBox1.Text + " DESC" + "," + TextBox2.Text + " ASC"
       Me.CustomersBindingSource.Sort = SortExpression
ElseIf RadioButton2.Checked = True And RadioButton4.Checked = True Then
       Dim SortExpression = TextBox1.Text + " DESC" + "," + TextBox2.Text + " DESC"
       Me.CustomersBindingSource.Sort = SortExpression
End If
```

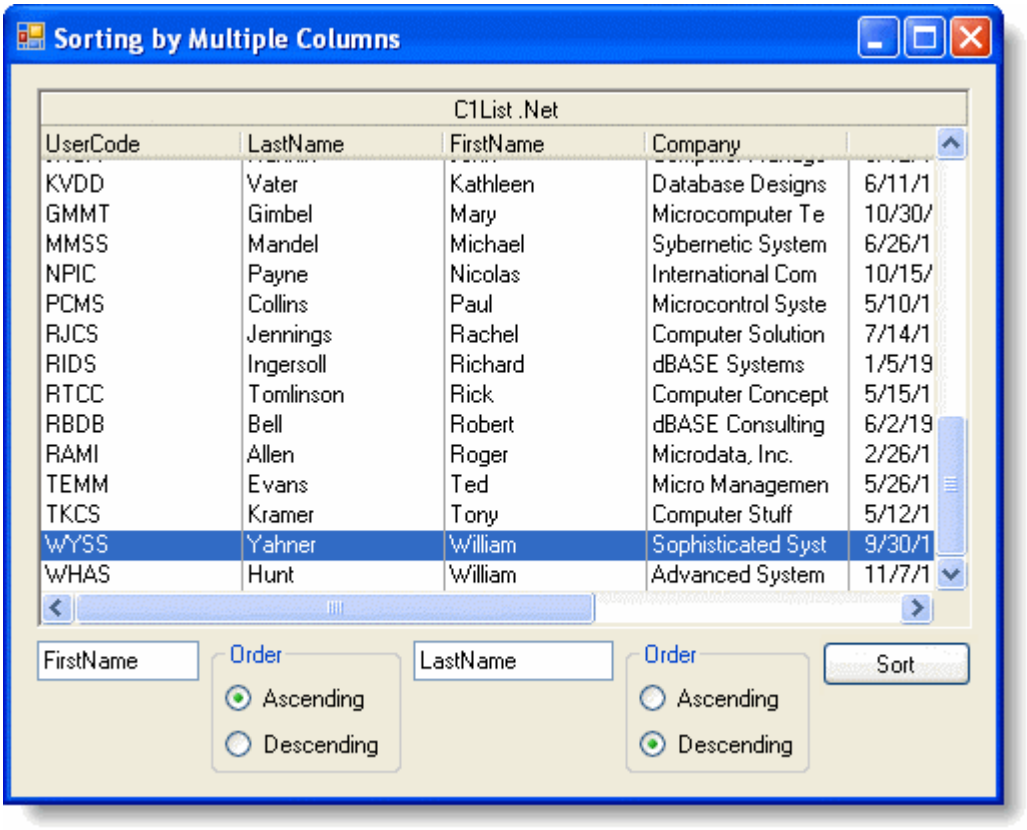
To write code in C#

C#

```
if (radioButton1.Checked == true && radioButton3.Checked == true)
{
       String SortExpression = textBox1.Text + " ASC" + "," + textBox2.Text + " ASC";
       this.CustomersBindingSource.Sort = SortExpression;
}
else if (radioButton1.Checked == true && radioButton4.Checked == true)
{
       String SortExpression = textBox1.Text + " ASC" + "," + textBox2.Text + "
DESC";
       this.CustomersBindingSource.Sort = SortExpression;
}
else if (radioButton2.Checked == true && radioButton3.Checked == true)
{
       String SortExpression = textBox1.Text + " DESC" + "," + textBox2.Text + "
ASC";
       this.CustomersBindingSource.Sort = SortExpression;
}
else if (radioButton2.Checked == true && radioButton4.Checked == true)
{
       String SortExpression = textBox1.Text + " DESC" + "," + textBox2.Text + "
DESC";
       this.CustomersBindingSource.Sort = SortExpression;
}
```

This topic illustrates the following:

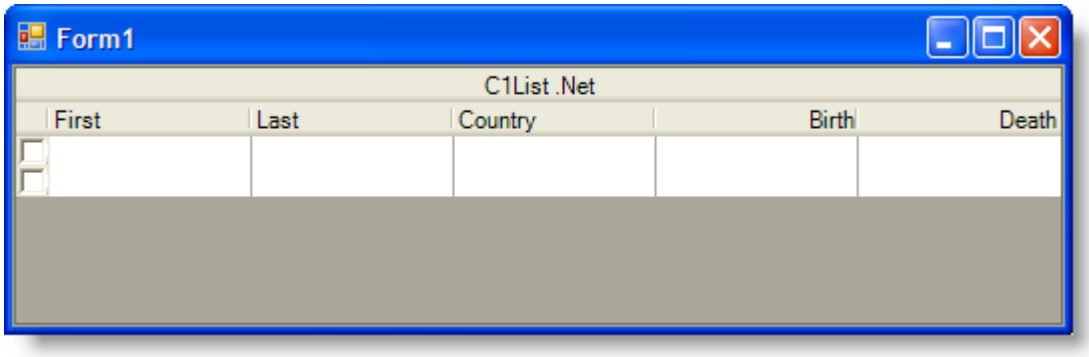
You can now enter column names into the textboxes and choose the sort order. Notice how choosing different orders on the columns changes the order of the data, in this case William Yahner appears before William Hunt because the *LastName* column is set to descending order.



Selecting a Row with a CheckBox

In **List for WinForms** you can create a list that allows users to select and highlight an entire row of information. Clicking anywhere on that row will select and highlight the row. If you want users to have to click in a specific area a checkbox, for example to select and highlight a row or column. Follow these steps:

1. Start with the project you completed at the end of the Quick Start Step 2.
2. From the Properties window, select the C1List control. Change the SelectionMode property from **One** to **CheckBox**. Your Form should look similar to this:



3. From the Visual Studio menu, select **View | Code**.
4. Add the following code to the Form_Load event:

To write code in Visual Basic

Visual Basic

```
Private Sub C1List1_MouseClick(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles C1List1.MouseClick
```

```
    Dim row, col As Integer
    Me.C1List1.CellContaining(e.X, e.Y, row, col)
    If col <> -1 Then
                    If Me.C1List1.SelectedIndices.Contains(row) Then
                                    Me.C1List1.SetSelected(row, False)
                    Else
                                    Me.C1List1.SetSelected(row, True)
                    End If
    End If
End Sub
```

To write code in C#

C#

```
private void c1List1_MouseClick(object sender,
System.Windows.Forms.MouseEventArgs e)
{
        int row;

        int col;

        this.c1List1.CellContaining(e.X, e.Y, out row, out col);

        if ((col != -1)) {

                if (this.c1List1.SelectedIndices.Contains(row)) {

                        this.c1List1.SetSelected(row, false);

                }

                else {

                        this.c1List1.SetSelected(row, true);

                }

        }
```

5. Press the F5 key to run your application.

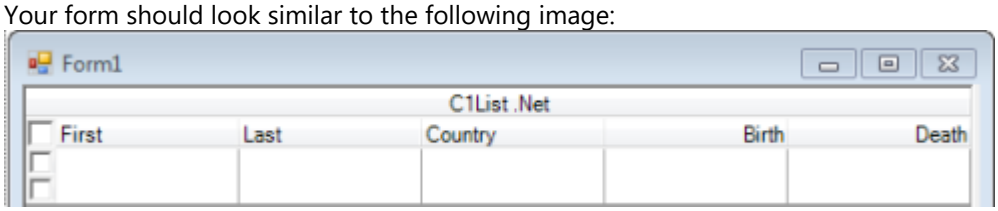
   Notice that you can no longer select an entire row just by clicking anywhere on that row. To select and highlight the entire row, you must click the checkbox to the left of the row.

Selecting All Rows with a CheckBox

The Show Header Checkbox property allows you to select an entire list of information instead of having to check each individual row or column. Follow these steps to apply a Header Checkbox to your list:

1. Start with the project you completed at the end of the Quick Start Step 2.

2. Select the C1List control on your form to open the Properties window. Change the SelectionMode property from **One** to **Checkbox**.
3. Change the ShowHeaderCheckBox property to **True**.
   Your form should look similar to the following image:

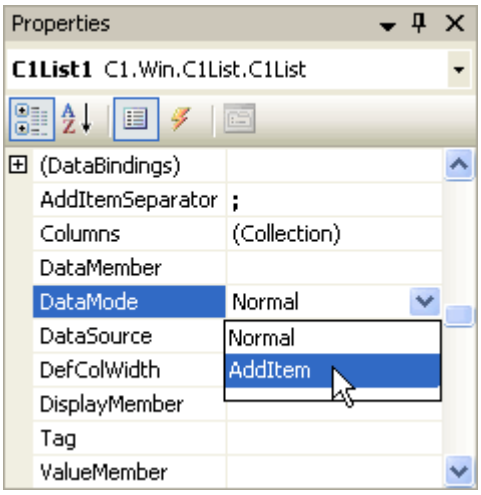


4. Press the F5 key to run your application. Note that the **C1List** now has a checkbox appearing in the header that allows you to select the entire list.

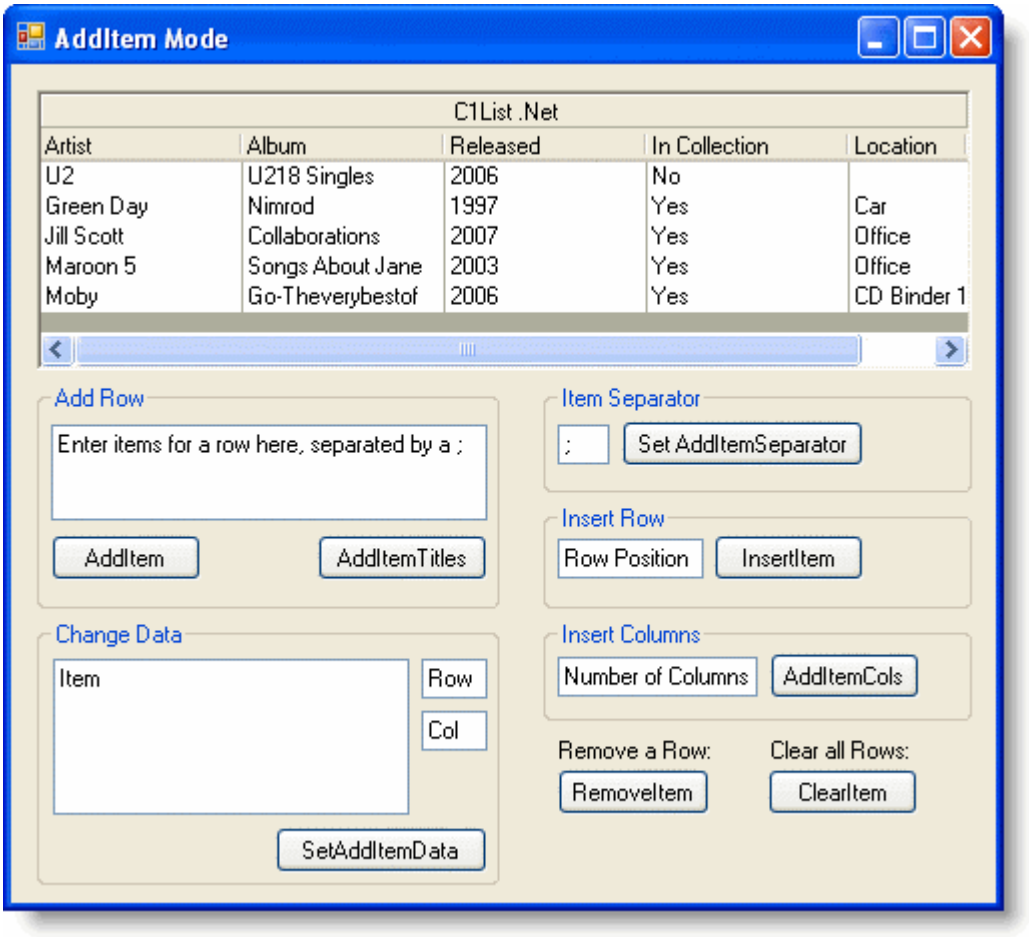
Using AddItem Mode

In **AddItem** mode, you can modify a list manually. The application supplies and maintains the data. Tutorial 3 - AddItem Mode shows how to use the AddItem, RemoveItem and ClearItems methods, and Tutorial 21 - Design-Time Support for C1Combo's AddItem Mode shows you how to use the **C1List Designer** to use **AddItem** mode. This section will expand on the tutorials and show you how to use the InsertItem, AddItemTitles, and SetAddItemData methods, as well as the AddItemSeparator and AddItemCols properties.

Begin by setting the DataMode to **AddItem** in the Properties window.

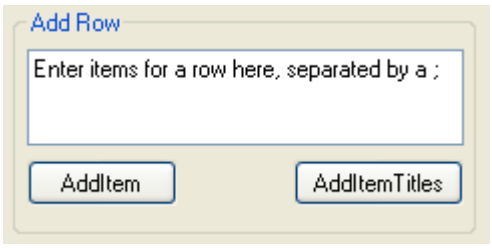


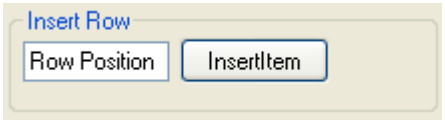
The following picture uses the **AddItem** mode methods and properties that will be discussed in the following sections. The AddItem, RemoveItem, and ClearItems methods are discussed in Tutorial 3 - AddItem Mode, as well as in List for WinForms' AddItem Mode.



Inserting a Row in a Specific Location

The InsertItem method allows you to insert a row at a specific index in the list.





To insert an item using the InsertItem method, add the following code. In this example, the text from **AddRowBox** will be inserted in the position entered into **PostitionBox** when the **InsertItem** button is clicked.

To write code in Visual Basic

```
Visual Basic
Me.C1List1.InsertItem(AddRowBox.Text, PositionBox.Text)
```

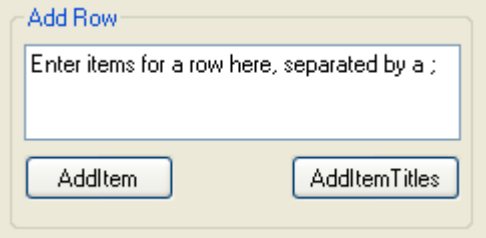
To write code in C#

C#

```
this.c1List1.InsertItem(AddRowBox.Text, PositionBox.Text);
```

Adding or Changing Titles

The AddItemTitles method allows you to add or change the titles of the list.



To add or change the titles, add the following code. In this example, the text entered into **AddRowBox** will become the titles of the columns when the **AddItemTitles** button is clicked.

To write code in Visual Basic

Visual Basic

```
Me.C1List1.AddItemTitles(AddRowBox.Text)
```

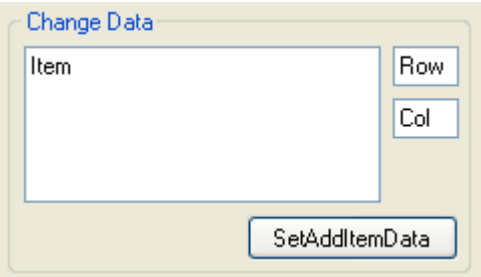
To write code in C#

C#

```
this.c1List1.AddItemTitles(AddRowBox.Text);
```

Adding or Changing Data in a Specific Cell

The SetAddItemData method allows you to add or change the value of a cell at a specific row and column.



To change the value of a cell, add the following code. In this example, the text entered into **ItemBox** will be entered into the row entered into **RowBox** and the column entered into **ColBox** when the **SetAddItemData** button is clicked.

To write code in Visual Basic

Visual Basic

```
Me.C1List1.SetAddItemData(ItemBox.Text, RowBox.Text, ColBox.Text)
```

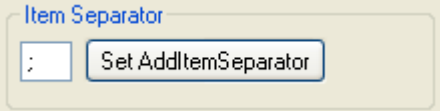
To write code in C#

C#

```
this.c1List1.SetAddItemData(ItemBox.Text, RowBox.Text, ColBox.Text);
```

Setting the Item Separator

The AddItemSeparator property allows you to set a custom separator. The default separator is the semicolon. The AddItemSeparator can be set in code or in the designer.



To set the AddItemSeparator property, add the following code to your project. In this example, the separator is defined by the text entered into **SeparatorBox** when the **Set AddItemSeparator** button is clicked:

To write code in Visual Basic

Visual Basic

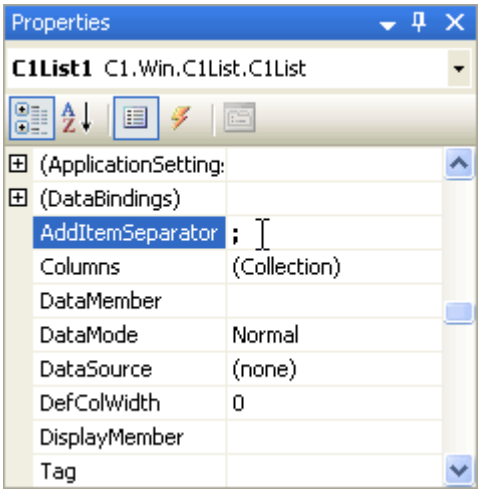
```
Me.C1List1.AddItemSeparator = SeparatorBox.Text
```

To write code in C#

C#

```
this.c1List1.AddItemSeparator = SeparatorBox.Text;
```

To set it in the designer, locate the AddItemSeparator property in the Properties window and enter the separator that you would like to use.



Adding Columns

The AddItemCols property allows you to control how many columns appear in the list.

To set the AddItemCols property, add the following code. In this example, the number of columns is determined from the number in **AddColBox** when the **AddItemCols** button is clicked.

To write code in Visual Basic

Visual Basic

```
Me.C1List1.AddItemCols = AddColBox.Text
```

To write code in C#

C#

```
this.c1List1.AddItemCols = AddColBox.Text;
```