ComponentOne

Input 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

Input for WinForms Overview

Enhancing the functionality and appearance of the standard controls, **Input for WinForms** consists of a suite of twelve controls that support visual styles (including Office 2010) and data binding. The powerful masked editing capability, rich formatting model, and localization enables you to use these input controls in your applications with increased performance. Display dynamic data in a visual format, manage dataset navigation, display or edit dates, and much more.

Getting Started

- C1Input Controls
- Input for WinForms Tutorials
- Input for WinForms Task-Based Help

Differences from the .NET 1.x Version

Please note that some changes in this version are not compatible with the 1.x versions and may require (minimal) changes to the source code.

Changes from the .NET 1.x version include the following:

- C1Input 2.x does not reference or need the C1Common assembly.
- Flag properties, such as **FormatInfo**, **MaskInfo**, **PreValidatation**, and **PostValidation**, do not show the Inherit flags at design time since those flags are set automatically.
- Most complex properties can now be reset to the default value in the designer.
- The **CopyWithLiterals** property was removed from C1TextBox as it is duplicated in MaskInfo.
- The **AllowDbNull** property was removed from C1TextBox and is only available as a sub-property of the PostValidation property.
- C1DropDownControl now allows the option to specify custom images for its buttons via the new ButtonImages Properties.

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

Display dynamic data in a visual format, manage dataset navigation, display or edit dates, and much more Benefit from using **Input for WinForms**, featuring:

- **Support for Office 2007 and 2010 visual styles**
  All C1Input controls support visual styles, including Office 2007 style. Provide attractive and consistent look and feel to your application, especially when used with other ComponentOne controls supporting visual styles.
- **The ability to display dynamic data**
  C1Input controls function in unbound and bound mode. In bound mode, a control's value is bound to a data source field.
- **Extensive data-binding support**
  Supports data binding to all .NET data sources, including ADO.NET data source objects and DataObjects components.
- **Powerful and customizable masked editing capabilities**
  C1TextBox and all derived controls support powerful masked editing including date and time formats, numeric range, and custom format support.
- **Support for regular expressions in mask format**
  Regular expressions in mask format make validation of complex input data easy. The regular expressions define the pattern of data by using keywords such as \A, {}, and so on. They also provide keywords to validate Japanese characters.
- **The power to format data in almost any way imaginable**
  A rich formatting model enables developers to customize the appearance of a control's text, border, color, and so on.
- **Support for data validation**
  Supports data validation both of the raw input string (PreValidation) and of the typed value entered by the user (PostValidation).
- **Support for a wide range of cultures**
  Define the cultural setting used by the control – this applies to string comparison, numeric and date time formats, and special characters.
- **Specialized drop-down editors**
  C1DropDownControl allows you to attach your own logic to the spin buttons and your own drop-down form/editor to the drop-down button.
- **Promp input error detection**
  Completely customizable error-handling behavior – detect an error while parsing or validating input value, and respond by showing an error message.
- **Drop-down and increment buttons**
  The specialized C1Input controls for date-time and numeric editing, C1DateEdit, C1NumericEdit, and C1ComboBox controls support drop-down and increment/decrement (up/down) buttons.
- **Slidable Thumbs**
  C1RangeSlider control provides movable thumbs which slide on a bar, enabling you to add numeric data selection to your applications.
- **Various formatting modes to choose from**
  Different formats available - display mode (used for a read-only or non-editing mode control) and edit mode.
- **Ability to quickly resolve NULL and empty values**
  Provides flexible rules for handling NULL and empty values, allowing the programmer to resolve this problem in practically any circumstance.
- **Customizable Appearance**
  Appearance of C1Input controls can be customized, owing to diverse properties and powerful theming capabilities. These provide flexible mechanism for adjusting look of the controls. The C1Input Control also provides two independent controls, C1ColorPicker and C1FontPicker. The **C1ColorPicker** control provides a rich, interactive color selection interface to select color for your control and **C1FontPicker** control provides functionality to choose fonts for a text.

Design-Time Support

C1Input provides visual editing to make it easier to configure the **C1Input** controls. This section describes how to use **C1Input's** design-time environment to configure the **C1Input** controls.

Context Menu

You can use the **C1Input** control's context menu for additional resources at design time.

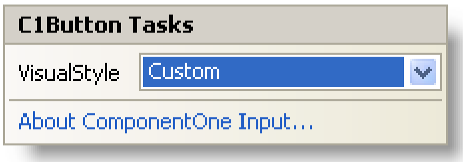
Tasks Menu

In Visual Studio 2005 2008, and 2010 the **C1Input** controls include a smart tag. A smart tag represents a short-cut tasks menu that provides the most commonly used properties in each control. You can invoke each control's Tasks menu by clicking on the smart tag (▶) in the upper-right corner of the control. For more information on how to use the smart tags for each control in **C1Input**, see the following topics.

C1Button Tasks Menu

In the **C1Button Tasks** menu you can quickly and easily set the VisualStyle property for the C1Button control.

To access the **C1Button Tasks** menu, click on the smart tag (▶) in the upper-right corner of the control. This will open the **C1Button Tasks** menu.



C1Button Tasks

VisualStyle: Custom

The **C1Button Tasks** menu operates as follows:

- **VisualStyle**

  Clicking the drop-down arrow in the **VisualStyle** drop-down opens a list of different VisualStyle enumeration options, such as System, Office2007Blue, Office2007Black, Office2007Silver, Custom, Office2010Blue, Office2010Black, and, Office2010Silver.The default value is **Custom**.

  For more information on how to set the VisualStyle property, see Customizing Appearance Using Visual Styles.

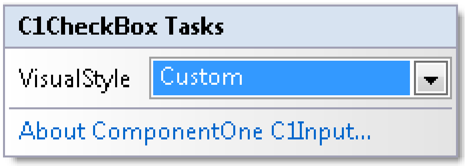
- **About Input**

  Clicking the **About ComponentOne Input** link displays the **About ComponentOne Input** dialog box, which is helpful in finding the version number of the control and online resources.

C1CheckBox Tasks Menu

In the **C1CheckBox Tasks** menu you can quickly and easily set the **VisualStyle** property for the C1DateEdit control.

To access the **C1CheckBox Tasks** menu, click on the smart tag (▶) in the upper-right corner of the control. This will open the **C1CheckBox Tasks** menu.



The **C1CheckBox Tasks** menu operates as follows:

- **VisualStyle**

  Clicking the drop-down arrow in the **VisualStyle** drop-down opens a list of different VisualStyle enumeration options, such as System, Office2007Blue, Office2007Black, Office2007Silver, Custom, Office2010Blue, Office2010Black, and Office2010Silver. The default value is **Custom**.

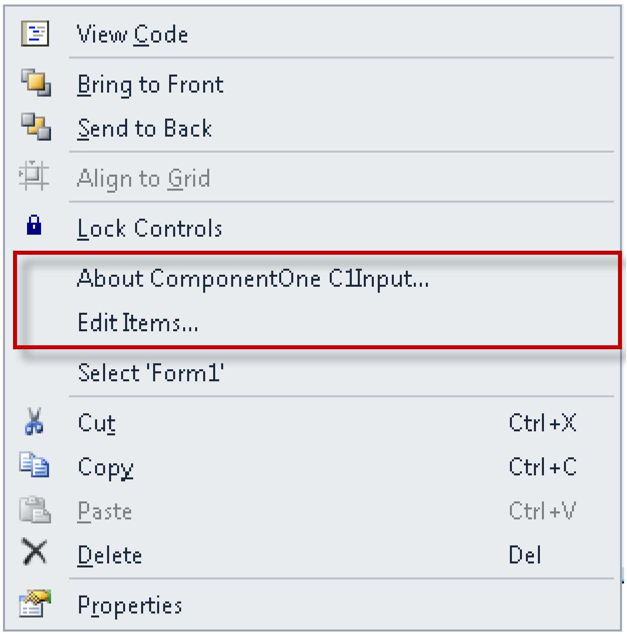
  For more information on how to set the **VisualStyle** property, see Customizing Appearance Using Visual Styles.

- **About Input**

  Clicking the **About ComponentOne Input** link displays the **About ComponentOne C1Input** dialog box, which is helpful in finding the version number of the control and online resources.

C1ComboBox Context Menu

The C1ComboBox control provides a context menu for additional resources at design time. Right-click on the C1ComboBox control to open its context menu.



The C1Input context menu operates as follows:

About ComponentOne Input
Clicking the About ComponentOne Input link displays the About ComponentOne Input dialog box, which is helpful in finding the version number of the control and online resources.

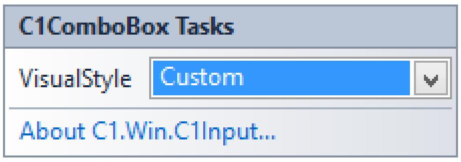
Edit Items...

Clicking the **Edit Items...** opens the **String Collection Editor** where you enter strings in the collection one per line.

C1ComboBox Tasks Menu

In the **C1ComboBox Tasks** menu you can quickly and easily set the **VisualStyle** property for the **C1ComboBox** control.

To access the **C1ComboBox Tasks** menu, click on the smart tag (▶) in the upper-right corner of the control. This will open the C1ComboBox Tasks menu.



The **C1ComboBox Tasks** menu operates as follows:

- **VisualStyle**

  Clicking the drop-down arrow in the VisualStyle drop-down opens a list of different VisualStyle enumeration options, such as System, Office2007Blue, Office2007Black, Office2007Silver, Custom, Office2010Blue, Office2010Black, and Office2010Silver. The default value is Custom.

  For more information on how to set the VisualStyle property, see Customizing Appearance Using Visual Styles.

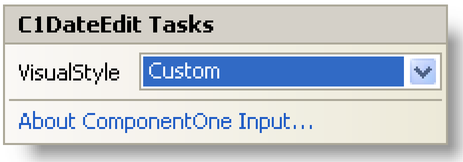
- **About ComponentOne Input**

  Clicking the **About ComponentOne Input** link displays the **About ComponentOne C1Input** dialog box, which is helpful in finding the version number of the control and online resources.

C1DateEdit Tasks Menu

In the **C1DateEdit Tasks** menu you can quickly and easily set the **VisualStyle** property for the C1DateEdit control.

To access the **C1DateEdit Tasks** menu, click on the smart tag (▶) in the upper-right corner of the control. This will open the **C1DateEdit Tasks** menu.



The **C1DataEdit Tasks** menu operates as follows:

- **VisualStyle**

  Clicking the drop-down arrow in the **VisualStyle** drop-down opens a list of different VisualStyle enumeration options, such as System, Office2007Blue, Office2007Black, Office2007Silver, Custom, Office2010Blue, Office2010Black, and Office2010Silver. The default value is **Custom**.

  For more information on how to set the **VisualStyle** property, see Customizing Appearance Using Visual Styles.

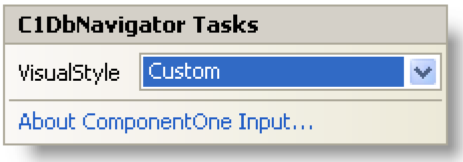
- **About ComponentOne Input**

  Clicking the **About ComponentOne Input** link displays the **About ComponentOne Input** dialog box, which is helpful in finding the version number of the control and online resources.

C1DbNavigator Tasks Menu

In the **C1DbNavigator Tasks** menu you can quickly and easily set the VisualStyle property for the C1DbNavigator control.

To access the **C1DbNavigator Tasks** menu, click on the smart tag (▶) in the upper-right corner of the control. This will open the **C1DbNavigator Tasks** menu.



C1DbNavigator Tasks

VisualStyle: Custom

About ComponentOne Input...

The **C1DbNavigator Tasks** menu operates as follows:

- **VisualStyle**

  Clicking the drop-down arrow in the **VisualStyle** drop-down opens a list of different VisualStyle enumeration options, such as System, Office2007Blue, Office2007Black, Office2007Silver, and Custom, Office2010Blue, Office2010Black, and Office2010Silver. The default value is **Custom**.

  For more information on how to set the VisualStyle property, see Customizing Appearance Using Visual Styles.

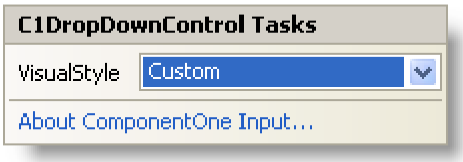
- **About ComponentOne Input**

  Clicking the **About ComponentOne Input** link displays the **About ComponentOne Input** dialog box, which is helpful in finding the version number of the control and online resources.

C1DropDownControl Tasks Menu

In the **C1DropDownControl Tasks** menu you can quickly and easily set the **VisualStyle** property for the C1DropDownControl control.

To access the **C1DropDownControl Tasks** menu, click on the smart tag (▶) in the upper-right corner of the control. This will open the **C1DropDownControl Tasks** menu.



C1DropDownControl Tasks

VisualStyle: Custom

About ComponentOne Input...

The **C1DropDownControl Tasks** menu operates as follows:

- **VisualStyle**

  Clicking the drop-down arrow in the **VisualStyle** drop-down opens a list of different VisualStyle enumeration options, such as System, Office2007Blue, Office2007Black, Office2007Silver, Custom, Office2010Blue, Office2010Black, and Office2010Silver. The default value is **Custom**.

  For more information on how to set the **VisualStyle** property, see Customizing Appearance Using Visual Styles.

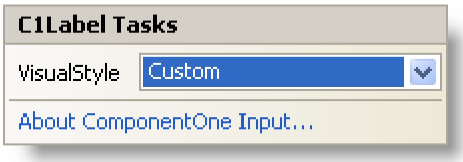
- **About ComponentOne Input**

  Clicking the **About ComponentOne Input** link displays the **About ComponentOne Input** dialog box, which is helpful in finding the version number of the control and online resources.

C1Label Tasks Menu

In the **C1Label Tasks** menu you can quickly and easily set the VisualStyle property for the C1Label control.

To access the **C1Label Tasks** menu, click on the smart tag (▶) in the upper-right corner of the control. This will open the **C1Label Tasks** menu.



The **C1Label Tasks** menu operates as follows:

- **VisualStyle**

  Clicking the drop-down arrow in the **VisualStyle** drop-down opens a list of different VisualStyle enumeration options, such as System, Office2007Blue, Office2007Black, Office2007Silver, Custom, Office2010Blue, Office2010Black, and Office2010Silver. The default value is **Custom**.

  For more information on how to set the VisualStyle property, see Customizing Appearance Using Visual Styles.

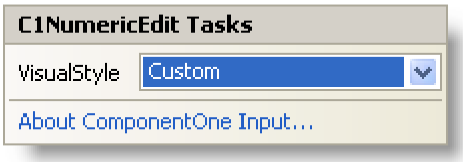
- **About ComponentOne Input**

  Clicking the **About ComponentOne Input** link displays the **About ComponentOne Input** dialog box, which is helpful in finding the version number of the control and online resources.

C1NumericEdit Tasks Menu

In the **C1NumericEdit Tasks** menu you can quickly and easily set the **VisualStyle** property for the C1NumericEdit control.

To access the **C1NumericEdit Tasks** menu, click on the smart tag (▶) in the upper-right corner of the control. This will open the **C1NumericEdit Tasks** menu.



The **C1NumericEdit Tasks** menu operates as follows:

- **VisualStyle**

  Clicking the drop-down arrow in the **VisualStyle** drop-down opens a list of different VisualStyle enumeration options, such as System, Office2007Blue, Office2007Black, Office2007Silver, Custom, Office2010Blue, Office2010Black, and Office2010Silver The default value is **Custom**.

  For more information on how to set the **VisualStyle** property, see Customizing Appearance Using Visual Styles.

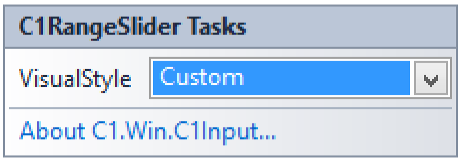
- **About ComponentOne Input**

  Clicking the **About ComponentOne Input** link displays the **About ComponentOne Input** dialog box, which is helpful in finding the version number of the control and online resources.

C1RangeSliderTasks Menu

In the **C1RangeSlider Tasks** menu you can quickly and easily set the VisualStyle property for the C1RangeSlider Control.

To access the **C1RangeSlider Tasks** menu, click on the smart tag (▸) in the upper-right corner of the control. This will open the **C1RangeSlider Tasks** menu.



VisualStyle Custom

The **C1RangeSlider Tasks** menu operates as follows:

- **VisualStyle**

  Clicking the drop-down arrow in the **VisualStyle** drop-down opens a list of different VisualStyle enumeration options, such as System, Office2007Blue, Office2007Black, Office2007Silver, Custom, Office2010Blue, Office2010Black, and Office2010Silver. The default value is **Custom**.

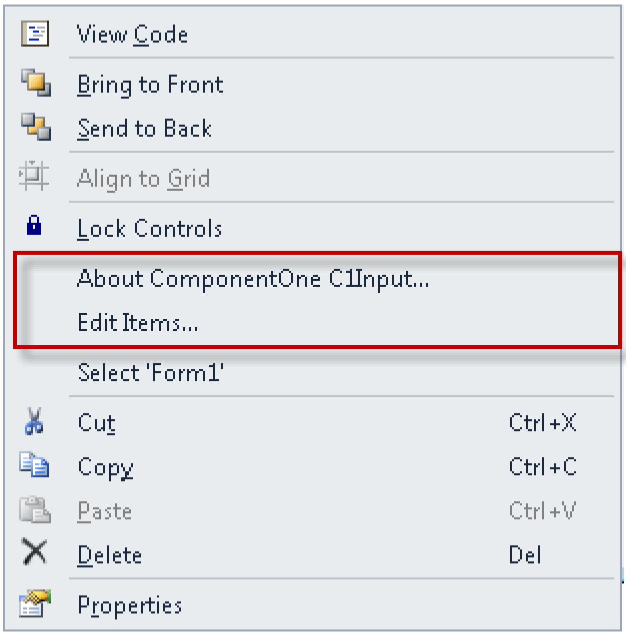
  For more information on how to set the **VisualStyle** property, see Customizing Appearance Using Visual Styles.

- **About C1.Win.C1 Input**

  Clicking the **About C1.Win.C1Input** link displays the **About ComponentOne Input** dialog box, which is helpful in finding the version number of the control and online resources.

C1SplitButton Context Menu

The C1SplitButton control provides a context menu for additional resources at design time. Right-click on the C1SplitButton control to open its context menu.



The C1Input context menu operates as follows:

About ComponentOne Input
Clicking the **About ComponentOne Input** link displays the **About ComponentOne Input** dialog box, which is helpful in finding the version number of the control and online resources.

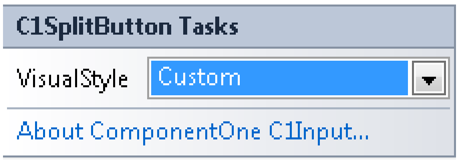
Edit Items...

Clicking the **Edit Items...** opens the **DropDownItem Collection Editor** where you can add or remove dropdown items.

C1SplitButton Tasks Menu

In the **C1SplitButton Tasks** menu you can quickly and easily set the VisualStyle property for the C1SplitButton control.

To access the C1SplittButton Tasks menu, click on the smart tag ( ) in the upper-right corner of the control. This will open the **C1SplitButton Tasks** menu.



C1SplitButton Tasks

VisualStyle: Custom

The **C1SplitButton Tasks** menu operates as follows:

- **VisualStyle**

  Clicking the drop-down arrow in the **VisualStyle** drop-down opens a list of different VisualStyle enumeration options, such as System, Office2007Blue, Office2007Black, Office2007Silver, Custom, Office2010Blue, Office2010Black, and, Office2010Silver.The default value is **Custom**.

  For more information on how to set the VisualStyle property, see Customizing Appearance Using Visual Styles.

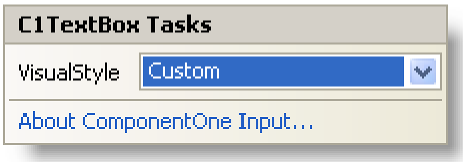
- **About ComponentOne Input**

  Clicking the **About ComponentOne Input** link displays the **About ComponentOne Input** dialog box, which is helpful in finding the version number of the control and online resources.

C1TextBox Tasks Menu

In the **C1TextBox Tasks** menu you can quickly and easily set the VisualStyle property for the C1TextBox control.

To access the **C1TextBox Tasks** menu, click on the smart tag (▶) in the upper-right corner of the control. This will open the **C1TextBox Tasks** menu.



C1TextBox Tasks

VisualStyle: Custom

The **C1TextBox Tasks** menu operates as follows:

- **VisualStyle**

  Clicking the drop-down arrow in the **VisualStyle** drop-down opens a list of different VisualStyle enumeration options, such as System, Office2007Blue, Office2007Black, Office2007Silver, Custom, Office2010Blue, Office2010Black, and Office2010Silver .The default value is **Custom**.

  For more information on how to set the VisualStyle property, see Customizing Appearance Using Visual Styles.

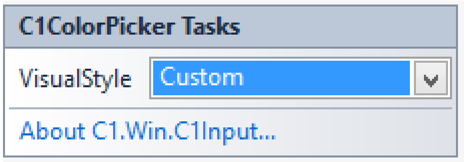
- **About ComponentOne Input**

  Clicking the **About ComponentOne Input** link displays the **About ComponentOne Input** dialog box, which is helpful in finding the version number of the control and online resources.

C1ColorPicker Tasks Menu

In the **C1ColorPicker Tasks** menu, you can quickly and easily set the VisualStyle property for the **C1ColorPicker** control.

To access the **C1ColorPicker Tasks** menu, click on the smart tag (▶) in the upper-right corner of the control. This will open the **C1ColorPicker Tasks** menu.



The **C1ColorPicker Tasks** menu operates as follows:

- **VisualStyle**

  Clicking the drop-down arrow in the **VisualStyle** drop-down opens a list of different VisualStyle enumeration options, such as System, Office2007Blue, Office2007Black, Office2007Silver, Custom, Office2010Blue, Office2010Black, and Office2010Silver. The default value is **Custom**.

  For more information on how to set the **VisualStyle** property, see Applying Visual Styles to C1ColorPicker and Customizing Appearance Using Visual Styles.

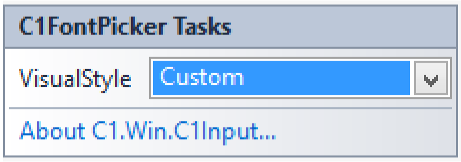
- **About C1.Win.C1Input**

  Clicking the **About C1.Win.C1Input** link displays the **ComponentOne** dialog box, which is helpful in finding the information about the version number, licensing, and online resources of the control.

C1FontPicker Tasks Menu

In the **C1FontPicker Tasks** menu, you can quickly and easily set the VisualStyle property for the **C1FontPicker** control.

To access the **C1FontPicker Tasks** menu, click on the smart tag (▶) in the upper-right corner of the control. This will open the **C1FontPicker Tasks** menu.



The **C1FontPicker Tasks** menu operates as follows:

- **VisualStyle**

  Clicking the drop-down arrow in the **VisualStyle** drop-down opens a list of different VisualStyle enumeration options, such as System, Office2007Blue, Office2007Black, Office2007Silver, Custom, Office2010Blue, Office2010Black, and Office2010Silver. The default value is **Custom**.

  For more information on how to set the **VisualStyle** property, see Applying Visual Styles to C1FontPicker and Customizing Appearance Using Visual Styles.

- **About C1.Win.C1Input**

  Clicking the **About C1.Win.C1Input** link displays the **ComponentOne** dialog box, which is helpful in finding the information about the version number, licensing, and online resources of the control.

Using the C1Input Controls

The following sections describe various aspects of using the **C1Input** controls.

C1Input Controls

The current version of **C1Input** contains twelve controls:

- **C1Button**

  A standard button type control derived from the standard System.Windows.Forms.Button. The C1Button control supports additional visual styles.

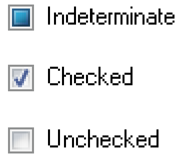
- **C1CheckBox**

  A window control used for setting or changing the value of an item as true, false, or indeterminate. The default **C1CheckBox** control is a two-state checkbox, but can be a three-state checkbox when the **ThreeState** property is enabled. The property **CheckState** determines the appearance of the checkbox state.

  The following table describes the three check box states and how it visually affects each checkbox.

Checkbox State	Description
Indeterminate	A dark shaded gray box appears in the box when only a few of the checkbox items are selected.
Checked	A checkmark appears in the checkbox item when the item is selected.
Unchecked	An empty checkbox appears in the checkbox when the item is not selected.

  The following image displays each check box state for the **C1CheckBox** control: indeterminate, checked, and unchecked.



- **C1ComboBox**

  A composite combobox control that allows users to view a drop-down list of options and select one or more options from the list.

- **C1TextBox**

  The main data bound control used for entering and editing information of any data type in a text form. Supports data formatting, edit mask, data validation and other features. C1TextBox also supports formatted and masked editing of all data types, including special features for date-time formats. Apart from being the main data editor control, C1TextBox also serves as the base class for specialized controls such as C1NumericEdit and C1DateEdit. C1TextBox derives from the standard System.Windows.Forms.TextBox control.

- **C1DropDownControl**

A control derived from C1TextBox, C1DropDownControl supports all C1TextBox formatting, validation, and other features. Like the other two C1TextBox-derived controls, it also supports **UpDown** (spin) and drop-down buttons. However, unlike those specialized controls, C1DropDownControl allows you to attach your own logic to the **UpDown** button and your own drop-down form/editor to the drop-down button.

- **C1DateEdit**

  A data bound control derived from C1TextBox specialized for editing date and time values. In addition to C1TextBox functionality, C1DateEdit supports a drop-down calendar and an up/down (spin) button for changing the value.

- **C1DbNavigator**

  A data bound control which provides buttons for convenient navigation over data source rows. It enables movement to the first, last, previous and next row and common data actions such as updating the data source and refreshing data.

- **C1Label**

  A read-only data bound control that displays formatted data. C1Label derives from the standard System.Windows.Forms.Label control.

- **C1NumericEdit**

  A data bound control derived from C1TextBox specialized for editing numeric values. In addition to C1TextBox functionality, C1NumericEdit supports a drop-down calculator and an up/down (spin) button that can be used to increment/decrement the value.

- **C1PictureBox**

  A data bound control which shows picture images stored in a data source field, derives from System.Windows.Forms.PictureBox.

- **C1RangeSlider**

  A control built on top of System.Windows.Forms.Control, C1RangeSlider enables numeric data selection over a range. The range is defined by two thumbs- upper value thumb and lower value thumb, which move over a Range Bar.

- **C1SplitButton**

  A composite button control supporting additional visual styles and drop down item list.

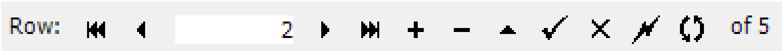
C1DbNavigator Control Overview

The C1DbNavigator class represents the **C1DbNavigator** control. It is a data bound control that provides buttons for convenient navigation over data source rows. It enables movement to the first, last, previous and next row and common data actions such as updating the data source and refreshing data.

The NavigatorButtonEnum gets the list of available buttons for the C1DbNavigator control which are the following: Add, Apply, Cancel, Delete, Edit, First, Last, Next, Position, Previous, Refresh, and Update.

The **Position** value of the NavigatorButtonEnum is used in the BeforeAction event when the text in Position text box is changed. For an example see, Changing the Navigation in the Navigator.

The **C1DbNavigator** control includes the following buttons that can be used to navigate and edit the records in a dataset:



Button	Description
⏮	First button. Moves to the first row in the record. Visible by default.
◀	Previous button. Moves to the previous row in the record. Visible by default.
▶	Next button. Moves to the next row in the record. Visible by default.
⏭	Last button. Moves to the last row in the record. Visible by default.
+	Add button. Adds a row to the record. Not visible by default.
−	Delete button. Deletes the row in the record. Not visible by default.
▲	Edit button. Edits the row in the record. Not visible by default.
✓	Apply button. Applies the changes made in the record. Not visible by default.
✕	Cancel button. Cancels the changes. Not visible by default.
⚡	Update button. Updates the record. Not visible by default.
⟳	Refresh button. Refreshes the record. Not visible by default.

C1DbNavigator Appearance

C1DbNavigator's buttons, border, and user interface strings can easily be customized by using C1DbNavigator's properties.

C1DbNavigator's Button Properties

The following table lists and describes the properties used to customize the buttons on the **C1DbNavigator** control:

Property	Description
C1DbNavigator.ButtonSize	The size of the navigator buttons.
C1DbNavigator.ButtonStyle	Gets the navigator button style which can be flat or standard.
C1DbNavigator.ButtonTextAlign	Controls how the text is positioned relative to the image in the navigator buttons.
C1DbNavigator.ButtonTexts	Gets or sets the texts displayed on the buttons.
C1DbNavigator.ButtonToolTips	The string collection defining the navigator button tooltips.
C1DbNavigator.ColorButtons	Specifies if navigator buttons have color bitmaps.

C1DbNavigator.ColorWhenHover	If true, navigator buttons show color bitmaps when the mouse hovers over them.
C1DbNavigator.VisibleButtons	Specifies which buttons are visible.

C1DbNavigator's User Interface Strings

The following table lists and describes the properties used to customize the user interface strings on the C1DbNavigator:

UIString	Description
Row:	Represents the current selected row number in the record.
of {0}	Represents the total amount of rows in the record.
(inactive)	Represents the position textbox where the text is displayed.
(empty)	
Confirmation	Appears in the title of the dialog box that appears when you click on the delete button to delete a row in the record.
Do you want to delete the row?	Appears in the content area of the confirmation dialog box.

C1DbNavigator Position Textbox Property

You can use the Text property to get or set the text in the position textbox. If the Position textbox is not visible, it returns empty string.

If you set the Text property when the Position textbox is not visible, the action has no effect.

Changing the Text property causes the data source position change.

C1DbNavigator Behavior

C1DbNavigator's includes several events for controlling its behavior. For example, the C1DbNavigator's behavior can change when any of the buttons are pressed, if the current row or if the fields have been changed in the record, if there is an exception thrown when clicking one of its buttons, or if the visual property style has changed.

The following table lists the C1DbNavigator events:

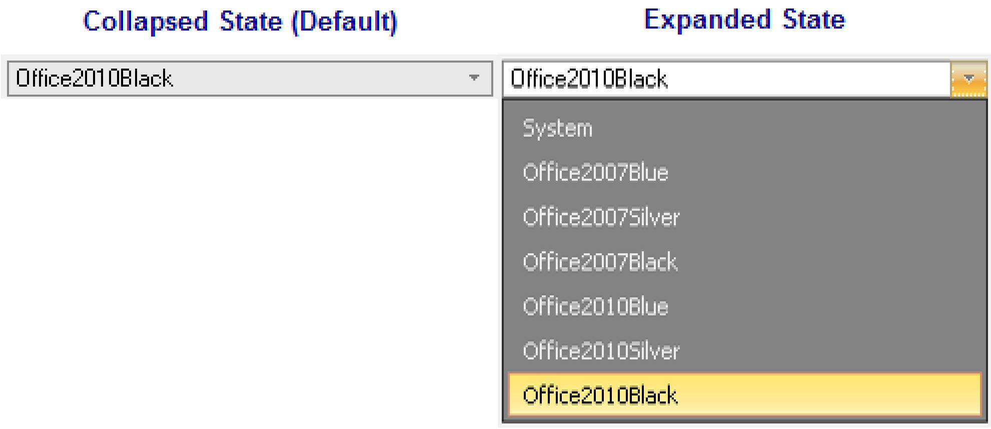
Event	Description
Adding	Occurs when the Add button is pressed.
BeforeAction	Occurs when a button is clicked, before the action is executed.
ButtonClick	Occurs when a navigator button has been pressed, after the button action is performed.
ButtonCursorChanged	Event fired when the value of the ButtonCursor property is changed.
Deleting	Occurs when the Delete button is pressed.
Editing	Occurs when the Edit button is pressed.
Error	Occurs when an exception is thrown performing an action on button click.
ItemChanged	Occurs when the current row has been modified, some of its fields changed.
PositionChanged	Occurs when the position has changed.

RefreshData	Occurs when Refresh button is pressed.
TextChanged	Occurs when the C1.Win.C1Input.C1DbNavigator.Text property value changes.
UpdateData	Occurs when Update button is pressed.
VisualStyleChanged	Occurs when the VisualStyle property has changed.

C1ComboBox Control Overview

C1ComboBox is a composite control that is used for displaying a list of selectable items. It functions similar to the **ListBox** control, but it takes up less space since the items can be hidden. Items can be added to the **C1ComboBox** through the **Items** property or they could be bound to data via an arrary of strings or binding source. C1ComboBox includes the following elements: **Textbox**, **Button**, and **DropDownList**. In the textbox you can type anything or you can click the button to select an item from the DropDownList. For more information see C1ComboBox Elements. In its default state the C1ComboBox control appears collapsed and only displays one item inside the textbox area. In its expanded state the C1ComboBox control appears expanded and displays a dropdown listbox of selectable items.

The following image illustrates the C1ComboBox in its collapsed and expanded states:

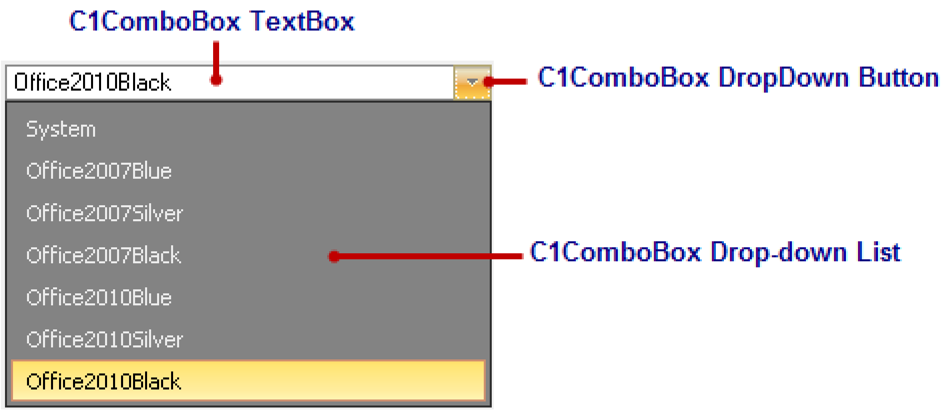


In a typical combobox control, a dropdown button appears to the right and functions as a dropdownlist where you can quickly choose from a list of options. However, in C1ComboBox you can add more functionality and create a numeric up/down button to edit numeric values or you can add a modal button if you need to show a modal dialog in your combobox. For more information see C1ComboBox Button Appearance.

C1ComboBox Elements

This section provides a visual and descriptive overview of the elements that comprise the C1ComboBox control.

The C1ComboBox control is made up of an editable text box and drop-down list.



C1ComboBox Text Box

Users can enter text in the C1ComboBox textbox at runtime or you can assign strings to the **Text** property. If you assign a value to the **Text** property, the current value in the textbox of the C1ComboBox will changes.

As text is entered into the C1ComboBox textbox, C1ComboBoxItems matching the characters will appear in the drop-down list. For example, if you enter "Sm" in the text box for a C1ComboBox listing customer names, any customer names starting with "Sm" appear in the drop-down list.

C1ComboBox DropDown Button

The default C1ComboBox displays a dropdown style button, but you can choose whether or not to show the button through the **VisibleButtons** property. In addition to the dropdown button you could also display the **Updown** button or your own custom button. A modal button can be displayed next to the dropdown, updown, or custom button or you can display it separately. For more information see C1ComboBox Button Appearance.

The size of the button can be modified using the **ButtonWidth** property.

Drop-down List of C1ComboBoxItems

The drop-down list is made up of C1ComboBoxItems and is only visible at run time. You can access the drop-down list by clicking the trigger button, or drop-down arrow, next to the C1ComboBox text box.

Items can be added to the C1ComboBox at design time through the **String Collection Editor**. In the String Collection Editor you can type the items line-by-line. Items could also be added dynamically during runtime through the **Items** property. For an example see, Adding Items to C1ComboBox.

ComboBox Item Modes

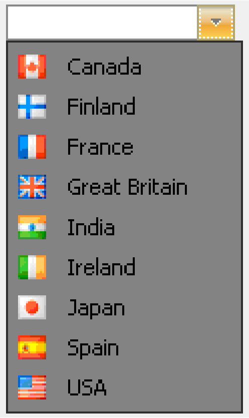
You can use the ItemMode property to build item presentation.

There are three options which include the following:

Default

In the default option, each C1ComboBoxItem is a string and can also be an image.

The following image illustrates a C1ComboBox with the Default option:

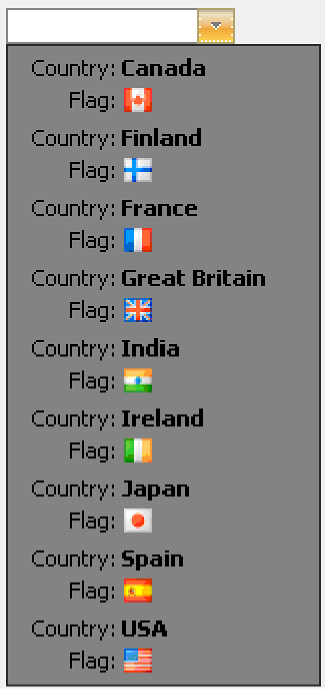


For an example see the **ComboBoxImages** sample.

HtmlPattern

In the HtmlPattern, each item is built from a HTML pattern and bound item data.

The following image illustrates a C1ComboBox with the HtmlPattern:



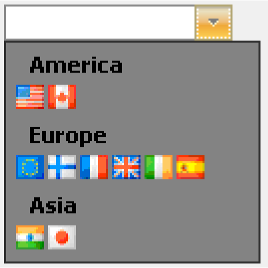
The HtmlPattern used above is set to the following: <table><tr><td>Country:</td><td><b>{Text}</b></td></tr><tr><td align="right">Flag:</td><td><img src='{Text}'></td></tr></table>

For an example see the **ComboBoxItemModes** sample.

Html

In the Html option, each item is a fragment of a HTML subset.

The following image illustrates a C1ComboBox with the Html option:



The Html used above is set to the following:

For an example see the **ComboBoxItemModes** sample.

C1ComboBox Styling

If you are not using one of the predefined themes you can set the C1ComboBox's **VisualStyle** property to **Custom** to create your own style for the C1CombBox control.

C1ComboBox includes the following apppearance properties:

Property	Description
DefaultItemForeColor	Gets or sets the default text color for the items in the C1ComboBox.
DropDownBackColor	Gets or sets the background color for the dropdown form in the C1ComboBox.
DropDownBorderColor	Gets or sets the border color for the dropdown form in the C1ComboBox.
HotItemBackColor	Gets or sets the background color of the combobox items when you mouse over them.
HotItemBorderColor	Gets or sets the border color for the combobox items when mouse over the border.
HotItemForeColor	Gets or sets the forecolor for the combobox items when you mouse over them.
Padding	Gets or sets the padding within the dropdown form.
TextSpacing	Gets or sets the textual parts of the combobox items.

C1ComboBox Button Appearance

The default C1ComboBox appears with one dropdown button to the right of the textbox. You can determine the button's visibility and what type of button to use through the **C1ComboBox.VisibleButtons** property.

The **C1ComboBox.VisibleButtons** property provides the following possible values:

Value	Appearance or Description
None	C1ComboBox appears with no dropdown button.
UpDown	Gets or sets the background color for the dropdown form in the C1ComboBox.
DropDown	Displays the default image for the DropDown button.

Modal	Displays ther default image for the Modal button. It appears to the right of the dropdown button if the dropdown button is enabled too.
Custom	Gets or sets the custom button for the C1ComboBox control.

Once the button style is determined you can use the default button image for the selected button or you can create a custom image for it. The following properties can be used to apply the custom image for each button style (Custom, UpDown, DropDown, and Modal):

- **ButtonImages.CustomImage** - Applies the image to the Custom button.
- **ButtonImages.DownImage** - Applies the image to the Down button.
- **ButtonImages.DropImage** - Applies the image to the DropDown button.
- **ButtonImages.ModalImage** - Applies the image to the Modal buton.
- **ButtonImages.Up** - Applies the image to the Up button.

ComboBox DataBinding

C1ComboBox can be bound to an Enum and an Array at runtime or it can be bound at design time through an array or strings or bindingsource. Binding C1ComboBox to data gives you the ability to browse data in a database, enter new data, or add existing data.

The **C1ComboBoxFeatures** sample shows how to use the following different methods for binding data to C1CombBox:

- How to bind C1ComboBox to an Enum at runtime
- How to bind C1ComboBox to an Array at runtime
- How to bind C1ComboBox to an Array at design time
- How to bind C1ComboBox to a bindingsource

Adding Images to Items in the ComboBox

You can easily use the images from the ImageList to add images to each item in the dropdown list of a C1ComboBox control.

To add images to C1ComboBoxItems at Design time, complete the following:

1. Add the C1ComboBox control to your form.
2. Add items to C1ComboBox.Items collection using the String Collection Editor.
3. Add the ImageList control to your form.
4. Add images to the imageList1.
5. Set keys (Name) of the images equal to the items in C1ComboBox.Items.

To add images to C1ComboBoxItems at Run time, add the following code:

To write code in Visual Basic

Visual Basic

```
c1ComboBox1.ItemsImageList = imageList
imageList.Images.Add("First item", Image.FromFile("First.png"))
c1ComboBox1.Items.Add("First item")
```

To write code in C#

C#

```
c1ComboBox1.ItemsImageList = imageList;
imageList.Images.Add("First item", Image.FromFile("First.png"));
c1ComboBox1.Items.Add("First item");
```

Adding Items to C1ComboBox

You can easily add items to C1ComboBox programatically using the **Add** method or you can add them at design time through the **String Collection Editor**. If you have more than one item, the **Add** method will add the new item(s) in the next position. If you need to add an item or object at a specific position in the list you can use the **Insert** method. An entire array can be added to the ComboBox by using the **AddRange** method to add the object or string of items to the C1ComboBox.

To Add Items Programatically

To add items to the C1ComboBox using the Add method of the C1ComboBox class. The collection is referenced using the Items property.

To write code in Visual Basic

Visual Basic

```
c1ComboBox1.Items.Add("Pittsburgh")
```

To write code in C#

C#

```
c1ComboBox1.Items.Add("Pittsburgh");
```

To Add Items Using the String Collection Editor

1. On the form, right-click on the **C1ComboBox** control and select **Edit Items**. The **String Collection Editor** appears.
2. In the **String Colllection Editor**, enter the string and then press **Enter** to add the next string in position.

To Insert the String or Object at the Desired Position

The following example inserts the string, Chicago, in the fifth position:

To write code in Visual Basic

Visual Basic

```
c1ComboBox1.Items.Insert(4, "Chicago")
```

To write code in C#

C#

```
c1ComboBox1.Items.Insert(4, "Chicago");
```

To Pass an Array through Strings

To pass an array through strings, complete the following:

1. Add the **C1ComboBox** control to the Form.
2. Add a **Button** control to the Form.
3. Create the following **Button_Click** event handler and add the following code to pass the array through strings to the C1ComboBox:

   To write code in Visual Basic

   Visual Basic

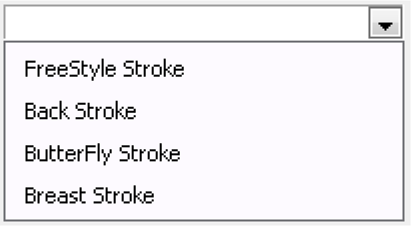
   ```
   Type your Drop Down Section text here.
   ```

   C#

   C#

   ```
   private void button1_Click(object sender, EventArgs e)
          {
              string[] items = { "FreeStyle Stroke", "Back Stroke", "ButterFly Stroke", "Breast Stroke"};
              c1ComboBox1.Items.AddRange(items);
          }
   ```

4. Run your project and click on the **Button**.
5. Click on the dropdown button on the C1ComboBox control and notice the string of items appear in the dropdownlist:



FreeStyle Stroke
Back Stroke
ButterFly Stroke
Breast Stroke

Removing Items from C1ComboBox

All Items or specific items can easily be removed from the C1ComboBox programmatically or at design time through the **Strings Collection Editor**.

Removing All Items Programatically

To programatically remove all items from C1ComboBox, complete the following:

To write code in Visual Basic

Visual Basic

```
c1ComboBox1.Items.Clear()
```

To write code in C#

C#

```
c1ComboBox1.Items.Clear();
```

Removing an Item Programatically

To programatically remove an item use the **Remove** or **RemoveAt** methods. The Remove method removes the specified item or selected item. The RemoveAt method removes the item with the specified index number.

The following code shows how to use the Remove and RemoveAt methods to remove specific items or remove selected items:

To write code in Visual Basic

Visual Basic

```
' To remove item with index 0:
c1ComboBox1.Items.RemoveAt(0)
' To remove currently selected item:
c1ComboBox1.Items.Remove(ComboBox1.SelectedItem)
' To remove "Chicago" item:
c1ComboBox1.Items.Remove("Chicago")
```

To write code in C#

C#

```
// To remove item with index 0:
comboBox1.Items.RemoveAt(0);
// To remove currently selected item:
comboBox1.Items.Remove(comboBox1.SelectedItem);
// To remove "Chicago" item:
comboBox1.Items.Remove("Chicago");
```

Populating C1ComboBox with Data Using SelectedItemChanged Event

To populate the C1ComboBox with data once a specific item in the combobox has been selected, use the SelectedItemChanged event like the following:

1. Add two C1ComboBoxes on the Form.
2. Add the string, "Pittsburgh", to the first C1ComboBox using the String Collection Editor.
3. Double click on the **SelectedItemChanged** event in C1ComboBox Properties window to create an event handler for the SelectedItemChanged event.
4. Add the following code to the SelectedIndexChanged event:

   To write code in Visual Basic

   Visual Basic

   ```
   Private Sub comboBox1_SelectedItemChanged(sender As Object, e As EventArgs)
     c1ComboBox2.Items.Clear()
   ```

```
    If c1ComboBox1.SelectedItem.ToString() = "Pittsburgh" Then
        c1ComboBox2.Items.Add("Pittsburgh is known as the Steel City and the City of Bridges.")
        c1ComboBox2.Items.Add("Pittsburgh has 446 Bridges")
    Else

        c1ComboBox2.Items.Add("You did not select Pittsburgh.")
    End If
End Sub
```

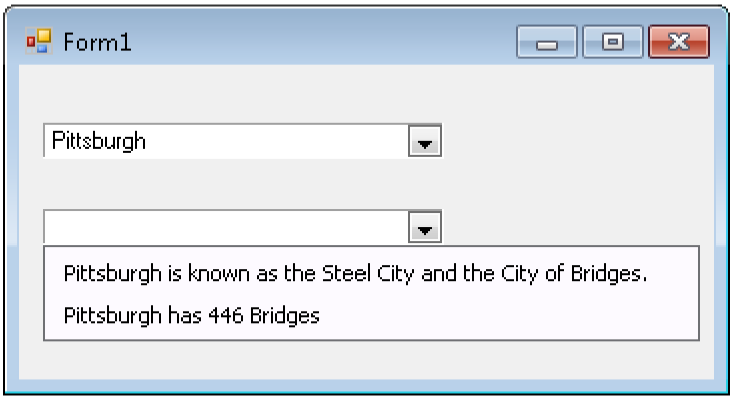
To write code in C#

C#

```
private void comboBox1_SelectedItemChanged(object sender, EventArgs e)
        {
            c1ComboBox2.Items.Clear();
            if (c1ComboBox1.SelectedItem.ToString() == "Pittsburgh")
            {
                c1ComboBox2.Items.Add("Pittsburgh is known as the Steel City and the City of Bridges.");
                c1ComboBox2.Items.Add("Pittsburgh has 446 Bridges");
            }
            else
            {
                c1ComboBox2.Items.Add("You did not select Pittsburgh.");

            }
        }
```

5. Run your project and select Pittsburgh in the first C1ComboBox.
6. Click on the dropdown button in the second C1ComboBox and notice the items were added to the dropdownlist based on what you added in the SelectedItemChanged event.



Pittsburgh

Pittsburgh is known as the Steel City and the City of Bridges.

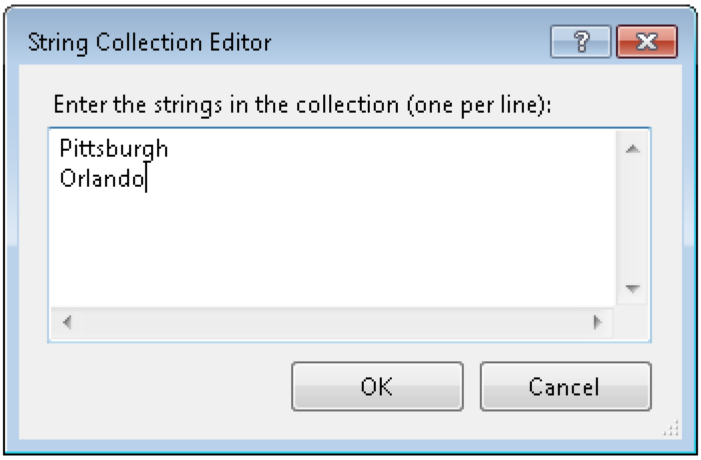
Pittsburgh has 446 Bridges

Populating C1Combbox with Data Using the SelectedIndexChanged Event

To populate the C1ComboBox with data once a specific index in the combobox has been selected, use the SelectedIndexChanged event like the following:

1. Add two **C1ComboBoxes** on the Form.
2. In the first C1ComboBox add the items line by line in the **String Collection Editor** so it appears like the

following:



```
Pittsburgh
Orlando
```

3. Double click on the **SelectedIndexChanged** event in the C1ComboBox Properties window to create an event handler for the **SelectedIndexChanged** event.
4. Add the following code to the **SelectedIndexChanged** event:

To write code in Visual Basic

Title Text

```
 Private Sub comboBox1_SelectedIndexChanged(sender As Object, e As EventArgs)
   c1ComboBox2.Items.Clear()
   If c1ComboBox1.SelectedIndex.ToString() = "1" Then

       c1ComboBox2.Items.Add("You selected the second item.")
   Else

       c1ComboBox2.Items.Add("You did not select the second item.")
   End If
End Sub
```

To write code in C#

C#

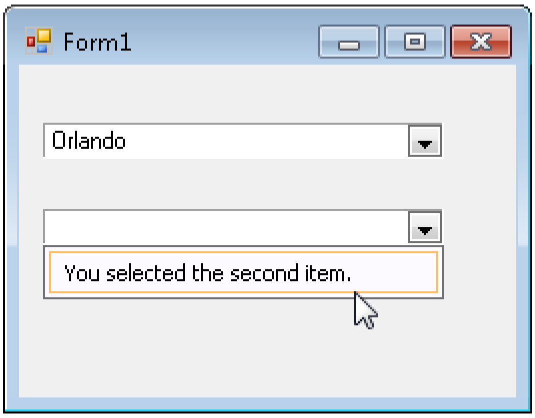
```
 private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
        {
            c1ComboBox2.Items.Clear();
            if (c1ComboBox1.SelectedIndex.ToString() == "1")
            {
                c1ComboBox2.Items.Add("You selected the second item.");

            }
            else
            {
                c1ComboBox2.Items.Add("You did not select the second item.");

            }
        }
```

4. Run your project and select the first index or second item, "Orlando" in the first C1ComboBox.
5. Click on the dropdown button in the second C1ComboBox and notice the items were added to the

dropdownlist based on what you added in the **SelectedIndexChanged** event.



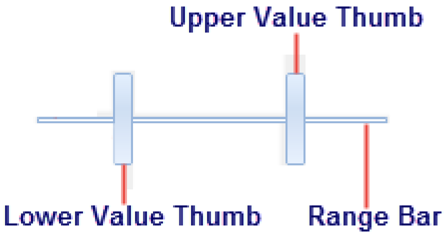
C1RangeSlider Control Overview

C1RangeSlider is a simple control that enables you to select a range of numeric data with lower value thumb and upper value thumb. These thumbs define start and end values of the range. When you drag the thumb towards the left (or down) on Range Bar you reduce its value, and you increase the value when drag it towards the right (or up). The control has minimum and maximum bounds for thumb values. Thumb value cannot be less than the minimum bound and more than the maximum bound. Moreover, lower value cannot be greater than the upper value, and otherwise.

The control can be oriented horizontally or vertically.

C1RangeSlider Elements

When you add C1Range Slider control to your form it exists as a completely functional slider control, with two thumbs to set range values and a range bar on which the thumbs move.



C1RangeSlider Features

This section describes key features of C1RangeSlider control.

Maximum and Minimum Values

The Maximum and Minimum properties enable you to set upper and lower allowable bounds for range in C1RangeSlider control. Lower value thumb cannot be set to a value less than the Minimum, and Upper Value thumb cannot be set to a value greater than the Maximum.

By default Minimum is set to 0 and Maximum is set to 100.

Set the Minimum and Maximum values using Minimum and Maximum properties from **Properties** pane of the control, or through code:

To write code in Visual Basic

Visual Basic

```
Me.c1RangeSlider1.Minimum = 10
Me.c1RangeSlider1.Maximum = 100
```

To write code in C#

C#

```
this.c1RangeSlider1.Minimum = 10;
this.c1RangeSlider1.Maximum = 100;
```

Orientation

C1RangeSlider control can be displayed Horizontally or Vertically, using Orientation property. By default, the control is oriented horizontally.

You can easily change the orientation through Orientation property in Properties pane, or through code:

To write code in Visual Basic

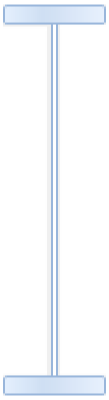
Visual Basic

```
Me.c1RangeSlider1.Orientation = Orientation.Vertical
```

To write code in C#

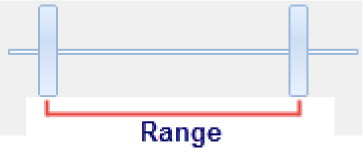
C#

```
this.c1RangeSlider1.Orientation = Orientation.Vertical;
```



Thumb Values and Range

Range in C1RangeSlider control is the difference between UpperValue and LowerValue properties. These two values are linked with movable thumbs which are used to set the numeric range on the control.



Setting the Thumb Values

Thumb values can be set using UpperValue and LowerValue properties in properties pane or through code.

By default, the UpperValue property is set to 100, and LowerValue property is set to 0.

To write code in Visual Basic

Visual Basic

```
Me.c1RangeSlider1.LowerValue = 24
Me.c1RangeSlider1.UpperValue = 88
```

To write code in C#

C#

```
this.c1RangeSlider1.LowerValue = 24;
this.c1RangeSlider1.UpperValue = 88;
```

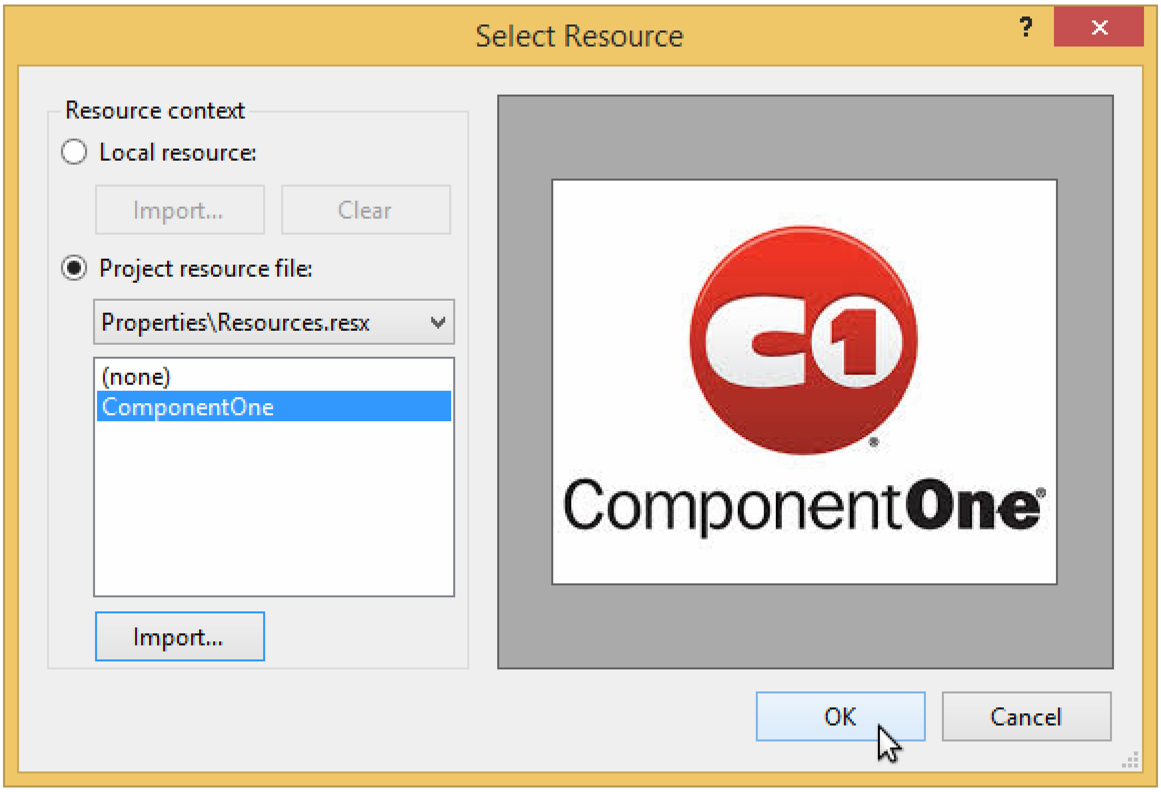
C1RangeSlider Appearance

You can customize the appearance of C1RangeSlider control by adjusting background image, thumb style and range bar style.

Background Image

Complete the following steps to set background image to **C1RangeSlider** control:

1. In Solution Explorer, right-click the project name and select **Add** > **New Folder**.
2. Rename the folder **Resources**.
3. Add the desired image to the **Resources** folder of your local project.
4. In Solution Explorer, click the **Show All Files**( ) button.
5. Right-click the image kept in **Resources** folder and select **Include In Project**.
6. Right-click C1RangeSlider control and select **Properties**.
7. In the **Properties** pane expand the **Appearance** node.
8. Click the ellipsis(...) button next to **BackgroundImage** property. **Select Resources** dialog box appears.
9. Click the **Import** button and browse to **Resources** folder in your project.
10. Select the image and click **OK** to save and close the **Select Resource** dialog box.



Setting the Background Image Layout

1. In **Properties** pane, click the dropdown corresponding to **BackgroundImageLayout** property.
2. Select the appropriate layout for the background image of your C1RangeSlider control.

Bar Style

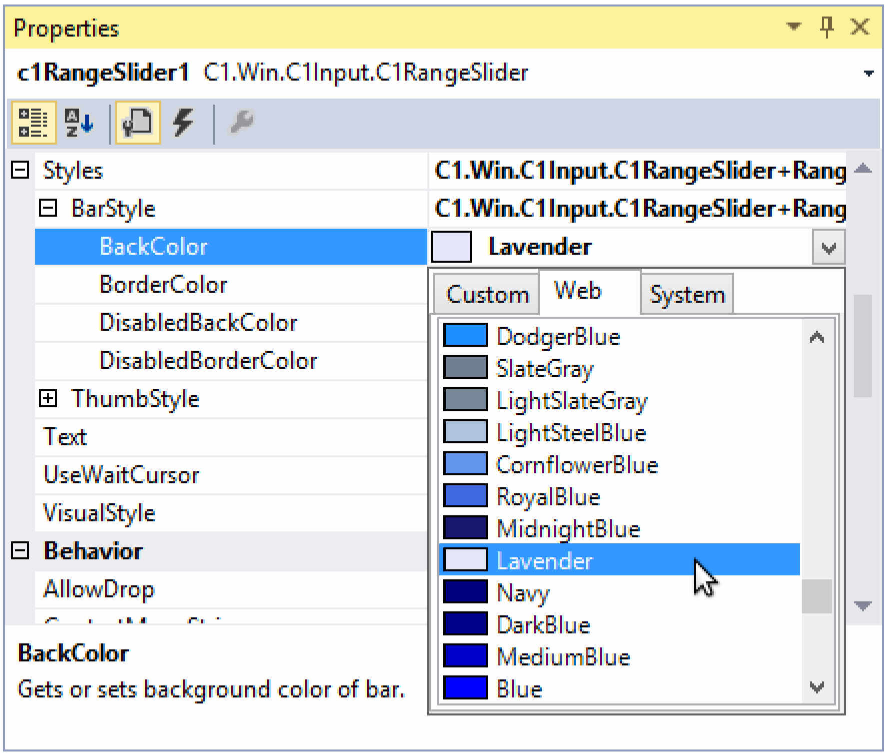
You can manage the appearance of C1RangeSlider control through various **Bar Style** options.

BackColor

Back color of **C1RangeSlider** bar can be changed at design time or through code.

To change the back color in design time complete the following:

1. Right-click the C1RangeSlider control and select **Properties** option.
2. In **Properties** pane expand the **Styles** node.
3. Select **BackColor** property from **BarStyle** collection.
4. In the drop-down menu corresponding to **BackColor** property, select **Lavender**.



c1RangeSlider1 C1.Win.C1Input.C1RangeSlider

⊟ Styles	C1.Win.C1Input.C1RangeSlider+Rang
⊟ BarStyle	C1.Win.C1Input.C1RangeSlider+Rang
BackColor	Lavender
BorderColor	
DisabledBackColor	
DisabledBorderColor	
⊞ ThumbStyle	
Text	
UseWaitCursor	
VisualStyle	
⊟ Behavior	
AllowDrop	

BackColor

Gets or sets background color of bar.

To change the back color at run-time, add the following code in FormLoad event:

To write code in Visual Basic

```
Visual Basic
Me.C1RangeSlider1.Styles.BarStyle.BackColor = System.Drawing.Color.Lavender
```

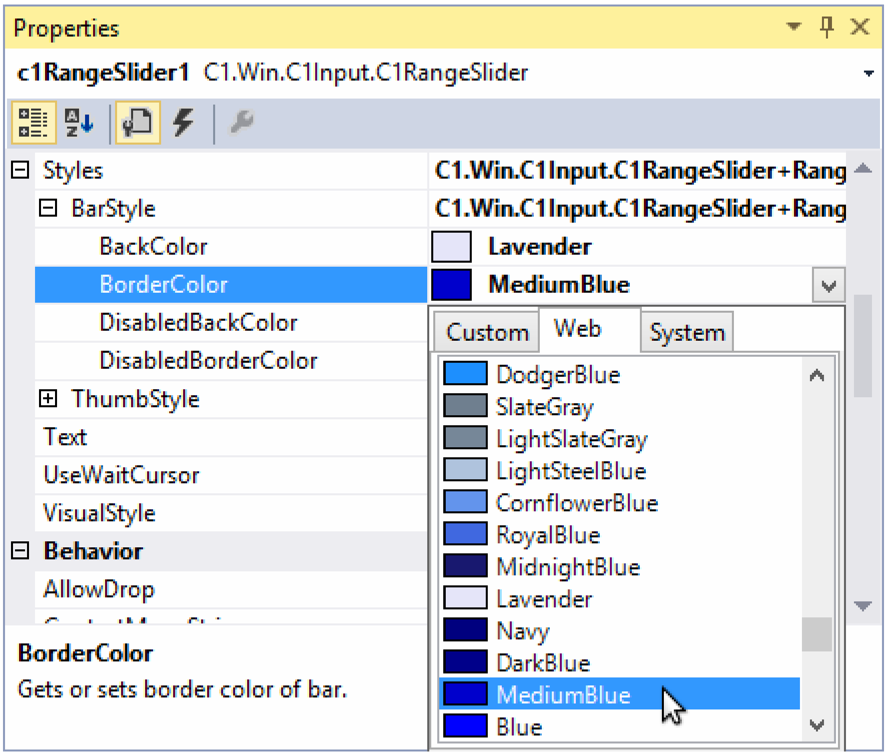
To write code in C#

```
C#
this.c1RangeSlider1.Styles.BarStyle.BackColor = System.Drawing.Color.Lavender;
```

BorderColor

To change the border color in design time complete the following:

1. Right-click the C1RangeSlider control and select **Properties** option.
2. In **Properties** pane expand the **Styles** node.
3. Select **BorderColor** property from **BarStyle** collection.
4. In the drop-down menu corresponding to **BorderColor** property, select **MediumBlue**.



Properties

c1RangeSlider1 C1.Win.C1Input.C1RangeSlider

⊟ Styles	C1.Win.C1Input.C1RangeSlider+Rang
⊟ BarStyle	C1.Win.C1Input.C1RangeSlider+Rang
BackColor	Lavender
BorderColor	MediumBlue
DisabledBackColor	
DisabledBorderColor	
⊞ ThumbStyle	
Text	
UseWaitCursor	
VisualStyle	
⊟ Behavior	
AllowDrop	

BorderColor

Gets or sets border color of bar.

To change the border color at run-time, add the following code in FormLoad event:

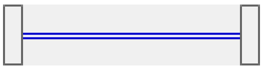
To write code in Visual Basic

Visual Basic
Me.C1RangeSlider1.Styles.BarStyle.BorderColor = System.Drawing.Color.MediumBlue

To write code in C#

C#
this.c1RangeSlider1.Styles.BarStyle.BorderColor = System.Drawing.Color.MediumBlue;

Run your project and observe the customizations. Following image shows the changed back color and border color of range bar in **C1RangeSlider** control:

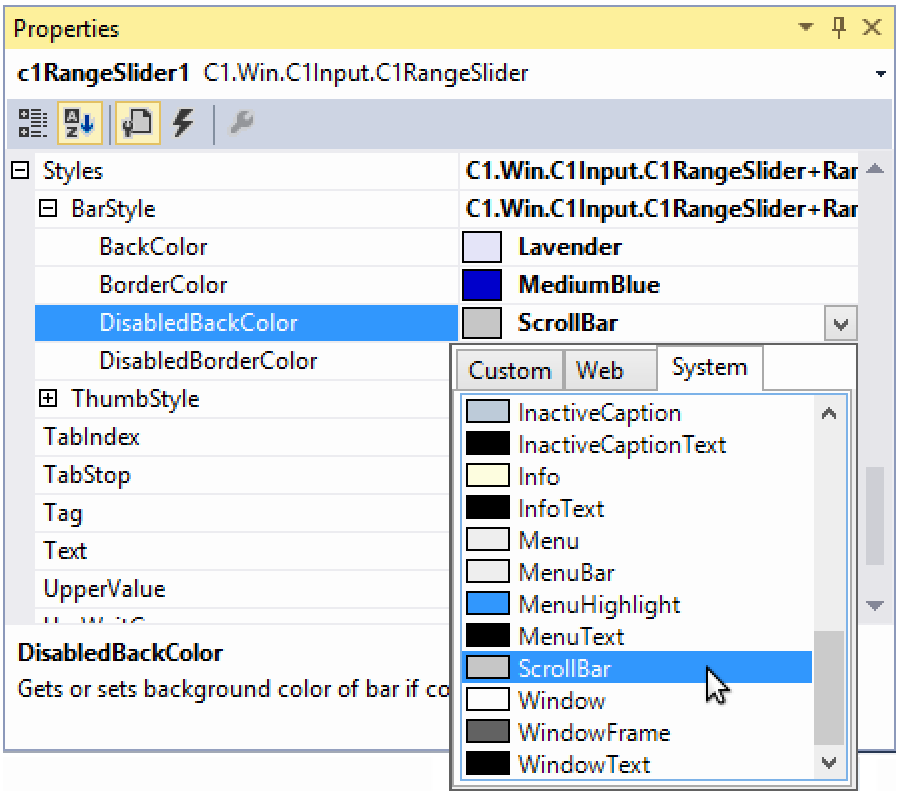


DisabledBackColor

DisabledBackColor property enables you to set the background color of range bar, which will be visible when the C1RangeSlider control is disabled.

To change the **DisabledBackColor** in design time complete the following:

1. Right-click the C1RangeSlider control and select **Properties** option.
2. In **Properties** pane expand the **Styles** node.
3. Select **DisabledBackColor** property from **BarStyle** collection.
4. In the drop-down menu corresponding to **DisabledBackColor** property, select **ScrollBar**.



To change the DisabledBackColor at run-time, add the following code in FormLoad event:

To write code in Visual Basic

Visual Basic

```
Me.C1RangeSlider1.Styles.BarStyle.DisabledBackColor =
System.Drawing.SystemColors.ScrollBar
```

To write code in C#

C#

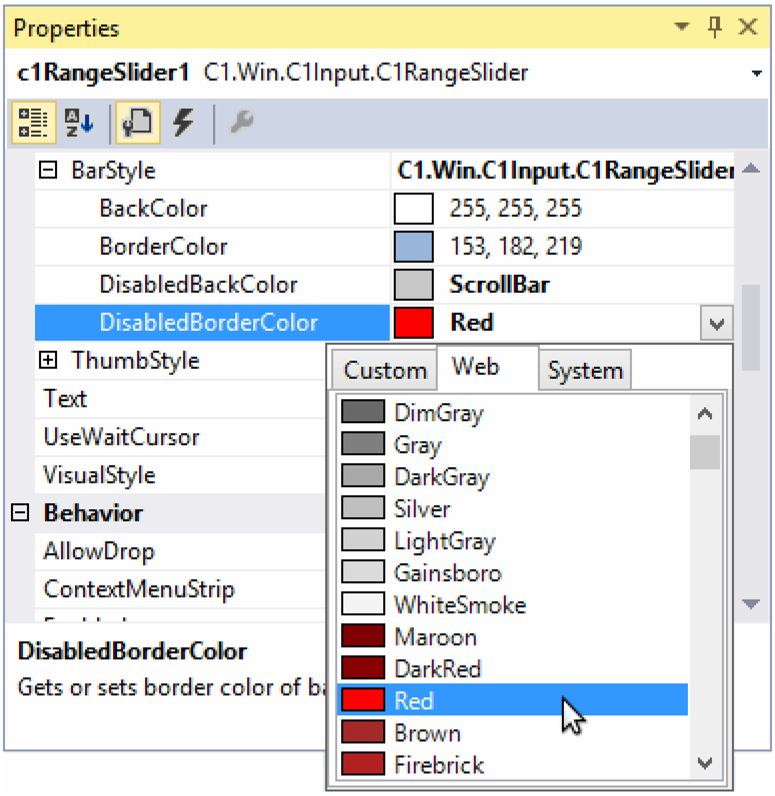
```
this.c1RangeSlider1.Styles.BarStyle.DisabledBackColor =
System.Drawing.SystemColors.ScrollBar;
```

DisabledBorderColor

DisabledBorderColor property enables you to set the border color of range bar, which will be visible when the C1RangeSlider control is disabled.

To change the **DisabledBorderColor** in design time complete the following:

1. Right-click the C1RangeSlider control and select **Properties** option.
2. In **Properties** pane expand the **Styles** node.
3. Select **DisabledBorderColor** property from **BarStyle** collection.
4. In the drop-down menu corresponding to **DisabledBorderColor** property, select **Red**.



Properties

c1RangeSlider1 C1.Win.C1Input.C1RangeSlider

⊟ BarStyle	C1.Win.C1Input.C1RangeSlider
BackColor	□ 255, 255, 255
BorderColor	153, 182, 219
DisabledBackColor	**ScrollBar**
DisabledBorderColor	🟥 **Red**
⊞ ThumbStyle	
Text	
UseWaitCursor	
VisualStyle	
⊟ **Behavior**	
AllowDrop	
ContextMenuStrip	

DisabledBorderColor
Gets or sets border color of ba

To change the DisabledBorderColor at run-time, add the following code in FormLoad event:

To write code in Visual Basic

Visual Basic

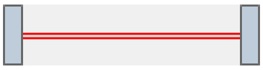
```
Me.C1RangeSlider1.Styles.BarStyle.DisabledBorderColor = System.Drawing.Color.Red
```

To write code in C#

C#

```
this.c1RangeSlider1.Styles.BarStyle.DisabledBorderColor = System.Drawing.Color.Red;
```

Set the **Enabled** property to **False,** and run the project. Following image shows disabled back color and disabled border color of range bar in the **C1RangeSlider** control:



Thumb Style

You can manage the appearance of C1RangeSlider control through various **Thumb Style** options.

Back Color

Back color of C1RangeSlider thumbs can be changed at design time or through code

To change the back color in design time complete the following:

1. Right-click the **C1RangeSlider** control and select **Properties** option.
2. In **Properties** pane expand the **Styles** node.
3. Select **BackColor** property from **ThumbStyle** collection.

4. In the drop-down menu corresponding to BackColor property, select **Tan**.

To change the Back Color at run-time, add the following code in FormLoad event:

To write code in Visual Basic

Visual Basic

```
Me.C1RangeSlider1.Styles.ThumbStyle.BackColor = System.Drawing.Color.Tan
```

To write code in C#

C#

```
this.c1RangeSlider1.Styles.ThumbStyle.BackColor = System.Drawing.Color.Tan;
```

Border Color

Border color of C1RangeSlider thumbs can be changed at design time or through code

To change the back color in design time complete the following:

1. Right-click the **C1RangeSlider** control and select **Properties** option.
2. In **Properties** pane expand the **Styles** node.
3. Select **BorderColor** property from **ThumbStyle** collection.
4. In the drop-down menu corresponding to **BorderColor** property, select **ActiveCaptionText**.

To change the Border Color at run-time, add the following code in FormLoad event:

To write code in Visual Basic

Visual Basic

```
Me.C1RangeSlider1.Styles.ThumbStyle.BorderColor =
System.Drawing.SystemColors.ActiveCaptionText
```

To write code in C#

C#

```
this.c1RangeSlider1.Styles.ThumbStyle.BorderColor =
System.Drawing.SystemColors.ActiveCaptionText;
```

Corner Radius

CorneRadius property enables you to customise the appearance of thumbs of **C1RangeSlider** control. Thumbs with CornerRadius set to 0 will be rectangular. To add curvature to the thumbs, you can increase their corner radius.

Corner Radius of C1RangeSlider thumbs can be changed at design time or through code

To change CornerRadius of thumbs in design time complete the following:

1. Right-click the **C1RangeSlider** control and select **Properties** option.
2. In **Properties** pane expand the **Styles** node.
3. Select **CornerRadius** property from **ThumbStyle** collection.
4. Set **CornerRadius** to **4**.

To change the Corner Radius at run-time, add the following code in FormLoad event:

To write code in Visual Basic

Visual Basic

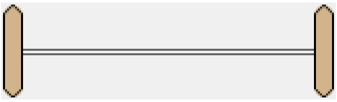
```
Me.C1RangeSlider1.Styles.ThumbStyle.CornerRadius = 4
```

To write code in C#

C#

```
this.c1RangeSlider1.Styles.ThumbStyle.CornerRadius = 4;
```

The following image shows customised thumb styles of **C1RangeSlider** control with corner radius 4.



Disabled Back Color

DisabledBackColor property enables you to set the background color of thumb, which will be visible when the C1RangeSlider control is disabled.

DisabledBackColor of C1RangeSlider thumbs can be changed at design time or through code

To change the **DisabledBackColor** in design time complete the following:

1. Right-click the **C1RangeSlider** control and select **Properties** option.
2. In **Properties** pane expand the **Styles** node.
3. Select **DisabledBackColor** property from **ThumbStyle** collection.
4. In the drop-down menu corresponding to **DisabledBackColor** property, select **InactiveCaption**.

To change the DisabledBackColor at run-time, add the following code in FormLoad event:

To write code in Visual Basic

Visual Basic

```
Me.C1RangeSlider1.Styles.ThumbStyle.DisabledBackColor =
System.Drawing.SystemColors.InactiveCaption
```

To write code in C#

C#

```
this.c1RangeSlider1.Styles.ThumbStyle.DisabledBackColor =
System.Drawing.SystemColors.InactiveCaption;
```

Disabled Border Color

DisabledBorderColor property enables you to set the border color of thumbs, which will be visible when the C1RangeSlider control is disabled.

DisabledBorderColor of C1RangeSlider thumbs can be changed at design time or through code.

To change the **DisabledBorderColor** in design time complete the following:

1. Right-click the **C1RangeSlider** control and select **Properties** option.
2. In **Properties** pane expand the **Styles** node.
3. Select **DisabledBorderColo**r property from **ThumbStyle** collection.
4. In the drop-down menu corresponding to **DisabledBorderColor** property, select **WindowFrame**.

To change the DisabledBorderColor at run-time, add the following code in FormLoad event:

To write code in Visual Basic

Visual Basic

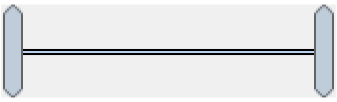
```
Me.C1RangeSlider1.Styles.ThumbStyle.DisabledBorderColor =
System.Drawing.SystemColors.WindowFrame
```

To write code in C#

C#

```
this.c1RangeSlider1.Styles.ThumbStyle.DisabledBorderColor =
System.Drawing.SystemColors.WindowFrame;
```

The following image shows customised thumb styles in disabled C1RangeSlider control.



Hovered Back Color

HoveredBackColor property enables you to set the back color of thumbs, which will be visible when mouse is over it.

HoveredBackColor of C1RangeSlider thumbs can be changed at design time or through code

To change the **HoveredBackColor** in design time complete the following:

1. Right-click the **C1RangeSlider** control and select **Properties** option.
2. In **Properties** pane expand the **Styles** node.
3. Select **HoveredBackColor** property from **ThumbStyle** collection.
4. In the drop-down menu corresponding to **HoveredBackColor** property, select **ActiveCaption**.

To change the HoveredBackColor at run-time, add the following code in FormLoad event:

To write code in Visual Basic

Visual Basic

```
Me.C1RangeSlider1.Styles.ThumbStyle.HoveredBackColor =
System.Drawing.SystemColors.ActiveCaption
```

To write code in C#

C#

```
this.c1RangeSlider1.Styles.ThumbStyle.HoveredBackColor =
System.Drawing.SystemColors.ActiveCaption;
```

Hovered Border Color

HoveredBorderColor property enables you to set the border color of thumbs, which will be visible when mouse is over it.

HoveredBackColor of C1RangeSlider thumbs can be changed at design time or through code

To change the **HoveredBorderColor** in design time complete the following:

1. Right-click the **C1RangeSlider** control and select **Properties** option.

2. In **Properties** pane expand the **Styles** node.
3. Select **HoveredBorderColor** property from **ThumbStyle** collection.
4. In the drop-down menu corresponding to **HoveredBorderColor** property, select **Blue**.

To change the HoveredBorderColor at run-time, add the following code in FormLoad event:

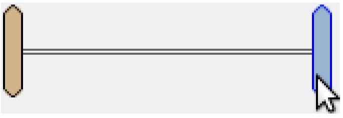
To write code in Visual Basic

Visual Basic
`Me.C1RangeSlider1.Styles.ThumbStyle.HoveredBorderColor = System.Drawing.Color.Blue`

To write code in C#

C#
`this.c1RangeSlider1.Styles.ThumbStyle.HoveredBorderColor = System.Drawing.Color.Blue;`

The following image shows customised thumb styles of C1RangeSlider control on mouse over.



Pressed Back Color

PressedBackColor property enables you to set the back color of thumbs, which will be visible when mouse-click is performed over it.

PressedBackColor of C1RangeSlider thumbs can be changed at design time or through code

To change the **PressedBackColor** in design time complete the following:

1. Right-click the **C1RangeSlider** control and select **Properties** option.
2. In **Properties** pane expand the **Styles** node.
3. Select **PressedBackColor** property from **ThumbStyle** collection.
4. In the drop-down menu corresponding to **PressedBackColor** property, select **Gold**.

To change the PressedBackColor at run-time, add the following code in FormLoad event:

To write code in Visual Basic

Visual Basic
`Me.C1RangeSlider1.Styles.ThumbStyle.PressedBackColor = System.Drawing.Color.Gold`

To write code in C#

C#
`this.c1RangeSlider1.Styles.ThumbStyle.PressedBackColor = System.Drawing.Color.Gold;`

Pressed Border Color

PressedBorderColor property enables you to set the border color of thumbs, which will be visible when mouse-click is performed over it.

PressedBorderColor of C1RangeSlider thumbs can be changed at design time or through code

To change the **PressedBorderColor** in design time complete the following:

1. Right-click the **C1RangeSlider** control and select **Properties** option.
2. In **Properties** pane expand the **Styles** node.
3. Select **PressedBorderColor** property from **ThumbStyle** collection.
4. In the drop-down menu corresponding to **PressedBorderColor** property, select **Orange**.

To change the PressedBorderColor at run-time, add the following code in FormLoad event:

To write code in Visual Basic

Visual Basic

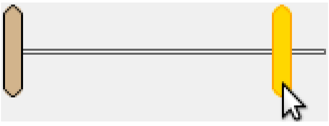
```
Me.C1RangeSlider1.Styles.ThumbStyle.PressedBorderColor = System.Drawing.Color.Orange
```

To write code in C#

C#

```
this.c1RangeSlider1.Styles.ThumbStyle.PressedBorderColor =
System.Drawing.Color.Orange;
```

The following image shows customised thumb styles of C1RangeSlider control when mouse-click is performed.

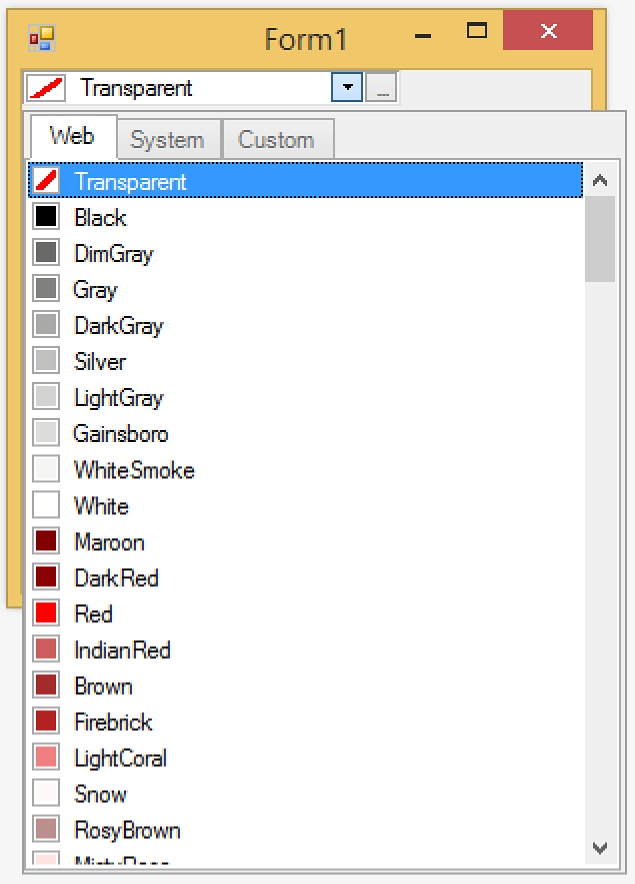


C1ColorPicker Control Overview

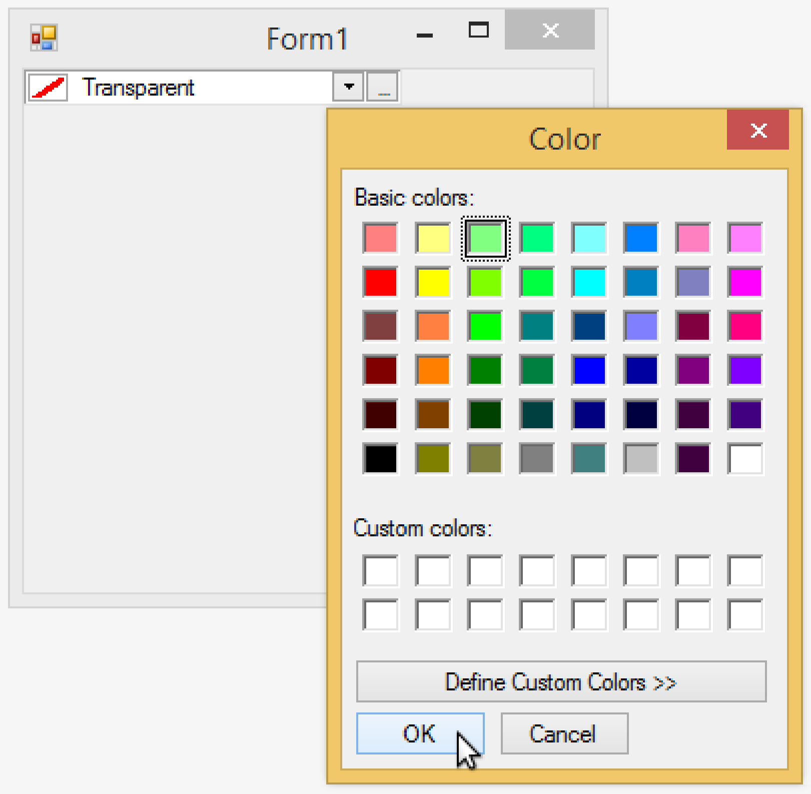
C1ColorPicker control is a color input editor that provides an interactive color selection interface. Users can select basic colors and can define custom colors from various options available in the **C1ColorPicker** to create a polished and professional looking application. **C1ColorPicker** also supports additional visual styles and themes that help users in further customizing their applications.

C1ColorPicker Elements

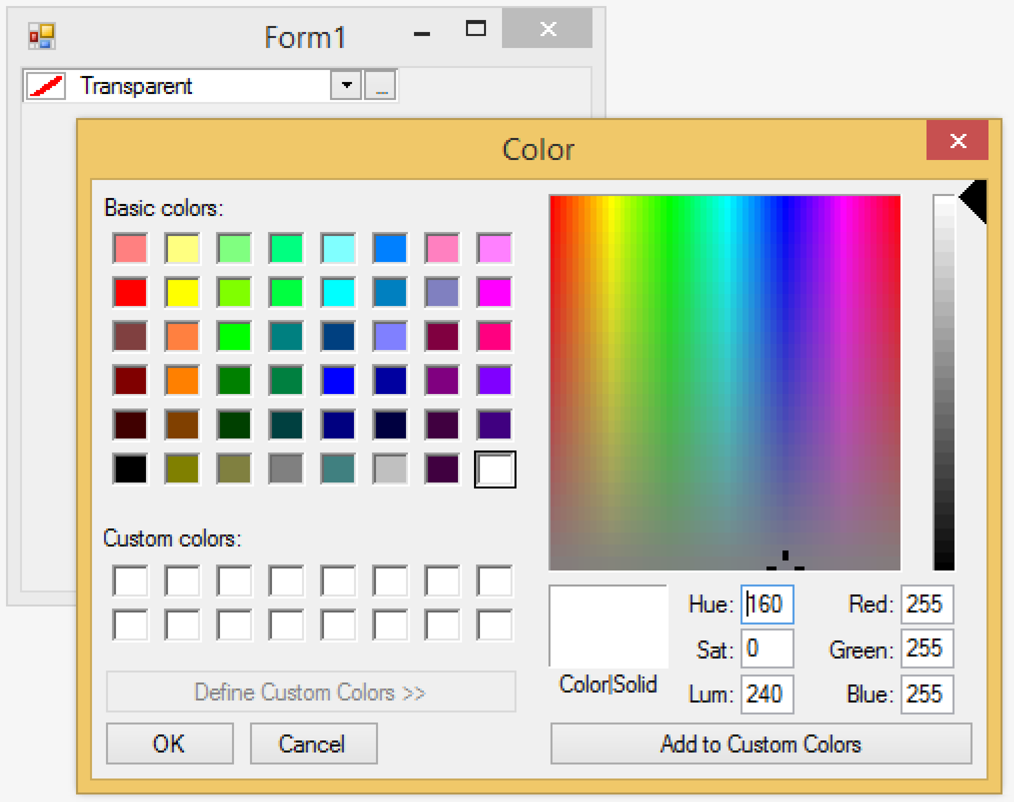
The **C1ColorPicker** control exists as a complete color selection control that you can customize further. When you click the drop-down button available on the C1ColorPicker, the control's interface looks similar to the following image:



The ellipses on the right of drop-down button let you select the basic colors that available in the color picker.



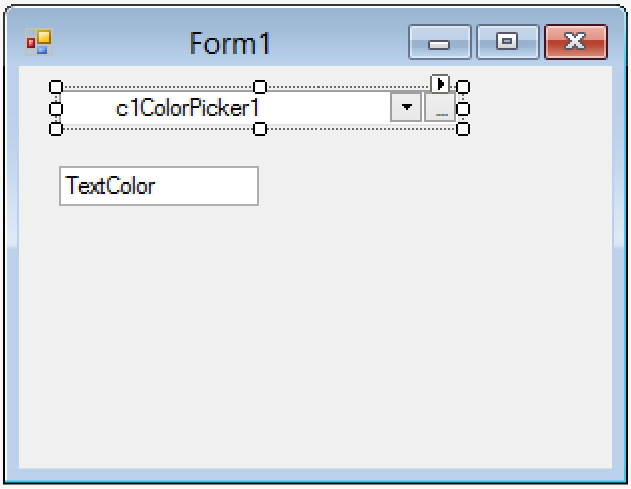
You can also define custom colors by clicking the **Default Custom Colors** button, creating a custom color from the color palette, and then adding the defined color to custom colors.



Working with C1ColorPicker

This topic illustrates how to use C1ColorPicker in **Windows Forms** applications. The steps to set up Visual Studio project and customize the application during runtime are as follows:

1. Create a Windows Forms project and add **C1ColorPicker** control to the Form.
2. Add a **RichTextBox** control to the Form. Set the **Text** property of the **RichTextBox** to **TextColor**. The Form appears like the image shown below:



3. Set the Color property of the C1ColorPicker control to a desired color from the palette. This enables the user to set a particular color as the current color instead of the default Transparent.
4. Add the following code to the **C1ColorPicker1_ValueChanged** event handler to change the color of the text on color selection:

VB copyCode

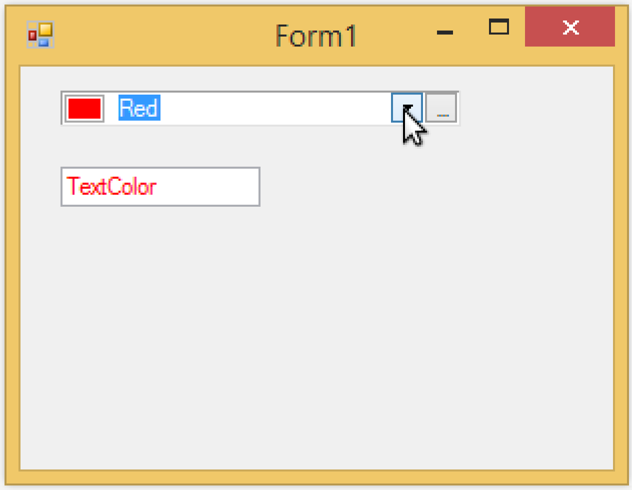
```
Private Sub C1ColorPicker1_ValueChanged(sender As Object, e As EventArgs)
```

```
Handles C1ColorPicker1.ValueChanged
    RichTextBox1.SelectionColor = DirectCast(DirectCast(sender,
C1ColorPicker).Value, Color)
End Sub
```

C# copyCode

```
private void c1ColorPicker1_ValueChanged(object sender, EventArgs e)
{
    richTextBox1.SelectionColor = (Color)c1ColorPicker1.Value;
}
```

5. Run the application. Now you can select the color from the drop-down of the **C1ColorPicker** and set the color of the text in the **RichTextBox** control. The following image shows the output where the text is set as red.



Applying Visual Styles to C1ColorPicker

Visual styles can be applied to the C1ColorPicker control using **VisualStyle** property. The smart tag on the top right corner of **C1ColorPicker** control lets users select the visual style they want to apply on the color picker.

To Change the Visual Style using the Smart Tag

Complete the following steps:

1. Select the **C1ColorPicker** control.
2. Click the smart tag (▶) to open the C1ColorPicker Tasks menu.
3. Click the visual style drop-down arrow and select a visual style to apply on the C1ColorPicker control.

To Change the Visual Style through Code

The code snippets refer the color picker application created in Working with C1ColorPicker.

1. Add the **ComboBox** control to the form.
2. Add the following line of code to the application to get the visual styles in the ComboBox:

VB

```
InitializeComponent()
ComboBox1.DataSource = [Enum].GetValues(GetType(VisualStyle))
```

C#

```
InitializeComponent();
comboBox1.DataSource = Enum.GetValues(typeof(VisualStyle));
```

3. Add code to the **comboBox1_SelectedIndexChanged** event handler so that the visual style of C1ColorPicker is updated on selecting an option from the combo box listing the available visual styles:

VB copyCode

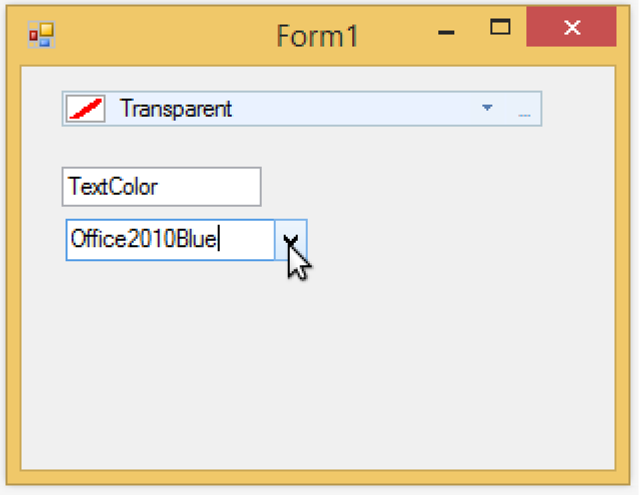
```
Private Sub ComboBox1_SelectedIndexChanged(sender As Object, e As EventArgs)
Handles ComboBox1.SelectedIndexChanged
    C1ColorPicker1.VisualStyle = DirectCast(ComboBox1.SelectedItem, VisualStyle)
End Sub
```

C# copyCode

```
private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    c1ColorPicker1.VisualStyle = (VisualStyle)comboBox1.SelectedItem;

}
```

4. Run the application.
Now you can select the visual style from the drop down of the combo box and set the visual style of the **C1ColorPicker**. The following image shows the output:

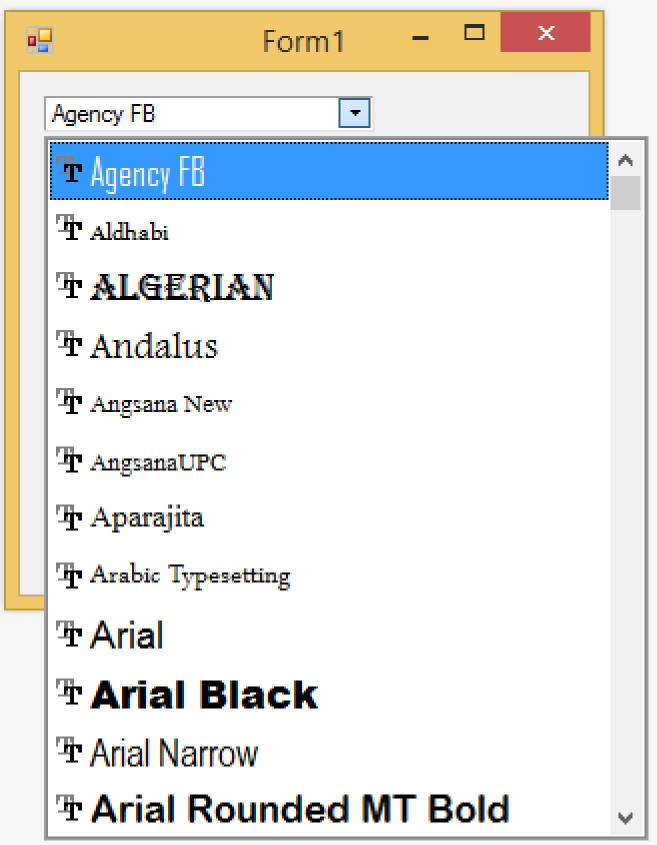


C1FontPicker Control Overview

C1FontPicker control is a font selection control that lets users select the desired font from a drop-down list. Users can select out of a list of more than 250 fonts available in the **C1FontPicker**. The **C1FontPicker** also supports additional visual styles and themes to choose from so that the users can further customize their applications.

C1FontPicker Elements

C1FontPicker control is a font selection control and when you click the drop-down button on **C1FontPicker**, the control's interface looks similar to the following image:

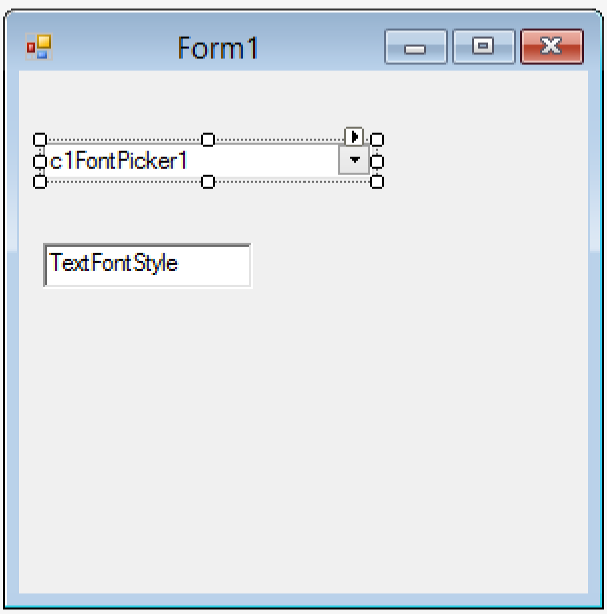


You can browse through the list to see the available fonts .

Working with C1FontPicker

This topic illustrates how to use C1FontPicker for **Windows Forms** applications. The steps to set up Visual Studio project and customize the application during runtime are as follows:

1. Create a Windows Forms project and add **C1FontPicker** control to the Form.
2. Add **RichTextBox** control to the Form. Set the **Text** property of the **RichTextBox** to **TextFontStyle**. The Form appears like the image shown below:



3. Add the following code to the **C1FontPicker1_ValueChanged** event handler to change the font of the text on font selection:

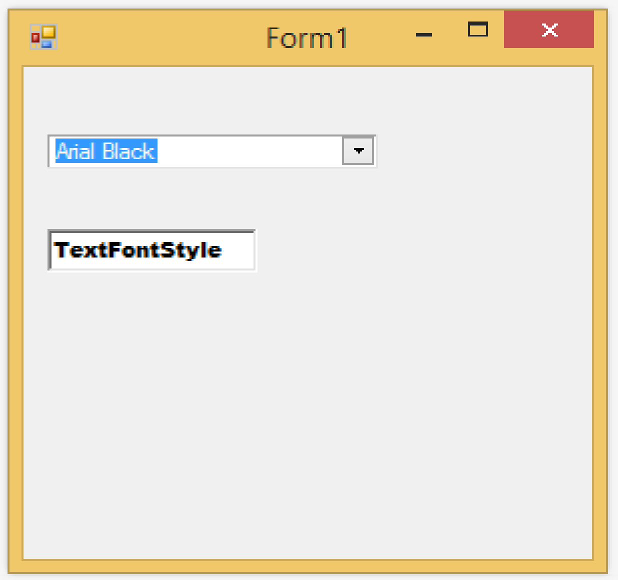
VB copyCode

```
Private Sub C1FontPicker1_ValueChanged(sender As Object, e As EventArgs) Handles
C1FontPicker1.ValueChanged
    RichTextBox1.SelectionFont = New Font(DirectCast(sender,
C1FontPicker).Value.ToString(), RichTextBox1.SelectionFont.Size,
RichTextBox1.SelectionFont.Style)
End Sub
```

C# copyCode

```
private void c1FontPicker1_ValueChanged(object sender, EventArgs e)
{
    richTextBox1.SelectionFont = new
Font(((C1FontPicker)sender).Value.ToString(), richTextBox1.SelectionFont.Size,
richTextBox1.SelectionFont.Style);
}
```

4. Run the application. Now you can select the font from **C1FontPicker** drop-down. The following image shows the output:



Applying Visual Styles to C1Font Picker

The smart tag on the top right corner of the C1FontPicker control lets users select the visual style of the **C1FontPicker**.

To Change the Visual Style using the Smart Tag

Complete the following steps:

1. Select the **C1FontPicker** control.
2. Click the smart tag (▶) to open the C1FontPicker Tasks menu.
3. Click the visual style drop-down arrow and select a visual style to apply on the **C1FontPicker** control.

To Change the Visual Style through Code

The code snippets refer the font picker application created in Working with C1FontPicker.

1. Add the **ComboBox** control to the form.
2. Add the following line of code to the application to get the visual styles in the ComboBox:

VB

```
InitializeComponent()
ComboBox2.DataSource = [Enum].GetValues(GetType(VisualStyle))
```

C#

```
InitializeComponent();
comboBox2.DataSource = Enum.GetValues(typeof(VisualStyle));
```

3. Add code to the **comboBox1_SelectedIndexChanged** event handler so that the visual style of C1FontPicker is updated on selecting an option from the combo box listing visual styles available:

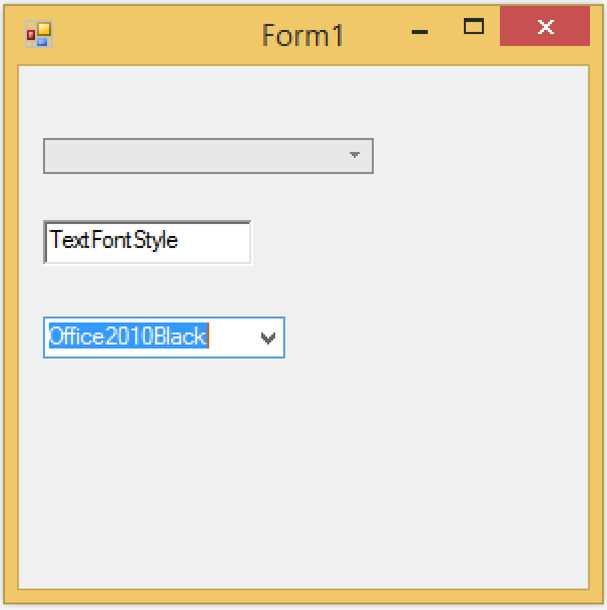
VB copyCode

```
Private Sub ComboBox2_SelectedIndexChanged(sender As Object, e As EventArgs)
Handles ComboBox2.SelectedIndexChanged
    C1FontPicker1.VisualStyle = DirectCast(ComboBox2.SelectedItem, VisualStyle)
End Sub
```

C# copyCode

```
private void comboBox2_SelectedIndexChanged(object sender, EventArgs e)
{
    c1FontPicker1.VisualStyle = (VisualStyle)comboBox2.SelectedItem;
}
```

4. Run the application. Now you can select the visual style from the drop down of the combo box and set the visual style of the **C1FontPicker**. The following image shows the output:



Data Binding

C1Input controls can function both in unbound and in bound mode. In bound mode, a control's Value is bound to a data source field.

C1Input controls support data binding to all .NET data sources. This includes ADO.NET data source objects such as DataTable, DataView and DataSet, and also DataObjects components such as C1ExpressTable, C1ExpressView, C1ExpressConnection, C1DataView, C1DataTableSource and C1DataSet.

For details about creating ADO.NET data source objects, please refer to the .NET Framework documentation.

For details about using DataObjects, see the C1DataObjects documentation included in the ComponentOne Studio Enterprise. DataObjects is a data framework, a part of ComponentOne Studio Enterprise, enhancing ADO.NET in many ways.

To bind a C1Input control to a date source field:

To bind a **C1Input** control to a data source field:

1. Assign the data source object to the control's DataSource property
2. Then assign a field of the data source object to the control’s DataField property.

   You can set the DataSource and DataField properties in code as well as in the designer. At design time, you can select the data source object and its field from the lists of available data sources and their fields provided by the DataSource/DataField property combo boxes.

3. Set the DataType property.

   After you bind the control to a data field, the DataType property is automatically set to the data type of the field it is bound to. In most cases, this setting is what you need. However, sometimes you need different types for the control and the field. For example, you might have a string field containing dates. In such cases, you can set the DataType property after binding and use the BindingFormatting/BindingParsing events to convert data to/from the control’s DataType when it is written from/saved to the data source.

The DataSource/DataField properties are used to bind the Value property of the control to a data source field. The Value property is the main property of a **C1Input** control. It holds and returns a value with of a specific data type. In addition, you have the freedom to bind other properties of the control to other data sources and fields, as in any other WinForms control, using the DataBindings property. This functionality is not **C1Input**-specific; it is supported by the .NET Framework for all controls.

> ⚠ **Caution:** Do not use properties other than the **Value** property under **DataBindings** to bind the control’s value to a field, use the DataSource and DataField properties instead (or use **Value** in **DataBindings**). For example, although it is possible to bind the **Text** property to a field, the result will not be the same as binding the **Value** property.

Value and Text: Displaying, Validating, and Updating Values

The Value property is the main property of a **C1Input** control and is responsible for returning and accepting a value with a specific DataType.

When the control is not in EditMode, the Text shown by the control will display its current Value (except when TextDetached property is set to **True**) in a properly formatted form, see Formatting Data for details.

If the control is not read-only, it automatically switches to EditMode when it receives input focus, provided that TextDetached property is set to **False**. Input focus refers to when the input window of the control receives focus. In edit mode, the control’s text is edited by the user while the Value remains unchanged until the editing ends. When the user attempts to leave the control, (for instance, move the input focus elsewhere), the Value property is updated with the Text entered by the user.

The process of updating the Value involves the following three main actions:

- **Parsing**

Since the Value property is typed (according to the control's current DataType), the process starts from the action of parsing the Text string, and converting it to the correct DataType. If necessary, the entered text can be validated at this point using string-based validation techniques such as pattern matching and regular expressions, see Validating Data for details.

- **Validation**

  Once a typed value is obtained, it passes the PostValidation check, where it can be matched against a list of predefined values, maximum and minimum values and intervals, or validated programmatically in the PostValidating event. After validation, the new value is assigned to the Value property, and then the Value property is updated to the data source.

- **Updating**

  Updating the Value normally occurs when the user tries to move the input focus out of the control or makes a mouse click outside of the control. However, it can also be triggered programmatically, by calling the UpdateValueWithCurrentText method at any time.

Setting the TextDetached property to **True** (it is **False** by default) forces the control into a special mode. If TextDetached property is set to **True**, the link between Value and Text is disabled, changing Value does not update Text, and changing Text does not update Value even when the control loses input focus. The Text property becomes independent of the Value property. This mode is useful when you want full programmatic control over updating the Text and Value.

Formatting Data

C1Input controls support a rich formatting model which enables developers to customize the appearance of a control's text in almost any way imaginable. The main function of formatting is to display a string Text representation of a typed or stored Value.

Formatting is controlled by the FormatType property, see Format Types for details. Its enumerated values define data will be formatted in the control. Some of the options correspond to .NET *standard format specifiers* for numeric and date-time types, for example, StandardNumber and LongDate, see Formatting Types in the .NET Framework documentation.

One FormatType option, CustomFormat, corresponds to the case of a *custom format specifier* as defined in the .NET Framework documentation, the specifier itself is determined by the CustomFormat property. For example, CustomFormat property is set to "##,###.###" produces numbers with at most five digits before and three digits after decimal point. See Custom Format Specifiers for details.

There is also a special FormatType option, UseEvent, which delegates the formatting to the Formatting event.

The ability to represent NULL values (System.DBNull) is controlled by the NullText and EmptyAsNull properties.

Sometimes you may find it useful to trim leading and/or trailing spaces when showing the formatted value. You can use the TrimStart and TrimEnd properties for that.

It is possible to specify two different formats, one for display (when the control is read-only or is not in the edit mode), and another for edit mode, see Value and Text: Displaying, Validating, and Updating Values to find more information about edit mode.

These two formatting modes are governed by the DisplayFormat and EditFormat properties. By default, both of them inherit from the control's properties. To assign specific FormatType, CustomFormat or other formatting property (see FormatInfo class) for a specific mode, expand the DisplayFormat or EditFormat nodes, and change the **(Inherit)** flags and set the desired sub-property.

Format Types

By default (FormatType property is set to DefaultFormat), the Text is obtained by applying the standard **ToString()** method of the current DataType to the typed Value (more exactly, if the type has a type converter, TypeConverter.ToString() is used). This conversion (as all others) uses the regional settings provided by the CultureInfo property.

Formatting is controlled by the FormatType property. Its enumerated values define the method of formatting values. Some of the options correspond to *standard format specifiers* for numeric and date-time types, for example, the StandardNumber and LongDate formats, for more information see Formatting Types in the .NET Framework documentation. One option, CustomFormat, corresponds to the case of a *custom format specifier* as defined in the .NET Framework documentation, the specifier itself is determined by the CustomFormat property. There is also an option delegating formatting to code in an event. The following table describes the list of available options:

Formatting Option	Description
DefaultFormat	Conversion using TypeConverter.ConvertToString().
UseEvent	Conversion performed by user code in the Formatting (or Parsing) event.
CustomFormat	Formatting uses the string assigned to the CustomFormat property. Parsing uses NumberStyle, DateTimeStyle, and CustomFormat properties.
GeneralNumber	The number is converted to the most compact decimal form, using fixed point or scientific notation.
Currency	The number is converted to a string that represents a currency amount.
FixedPoint	The number is converted to a string of the form "-ddd.ddd..." where each 'd' indicates a digit (0-9). The string starts with a minus sign if the number is negative. The numeric precision is given by the property NumberFormatInfo.NumberDecimalDigits of the specified culture.
StandardNumber	The number is converted to a string of the form "-d,ddd,ddd.ddd...", where each 'd' indicates a digit (0-9). The string starts with a minus sign if the number is negative. Thousand separators are inserted between each group of three digits to the left of the decimal point. The numeric precision is given by the property NumberFormatInfo.NumberDecimalDigits of the specified culture.
Percent	The number is converted to a string that represents a percent as defined by the NumberFormatInfo.PercentNegativePattern property or the NumberFormatInfo.PercentPositivePattern property. If the number is negative, the string produced is defined by the PercentNegativePattern and starts with a minus sign. The converted number is multiplied by 100 in order to be presented as a percentage. The default numeric precision given by NumberFormatInfo is used.
Scientific	The number is converted to a string of the form "-d.ddd...E+ddd" or "-d.ddd...e+ddd", where each 'd' indicates a digit (0-9). The string starts with a minus sign if the number is negative. One digit always precedes the decimal point. The exponent always consists of a plus or minus sign and a minimum of three digits. The exponent is padded with zeros to meet this minimum, if required.
RoundTrip	The round-trip specifier guarantees that a numeric value converted to a string will be parsed back into the same numeric value. This format is supported by floating-point types only.
Integer	Displays number as a string that contains the value of the number in Decimal (base 10) format. This format is supported for integral types only.
Hexadecimal	The number is converted to a string of hexadecimal digits. This format is supported for integral types only.
YesNo	Converts to Boolean and shows No for False, Yes for True.

TrueFalse	Converts to Boolean and shows True or False.
OnOff	Converts to Boolean and shows Off for False, On for True.
GeneralDate	General date/time pattern (short time).
LongDate	Displays a date according to specified CultureInfo's long date format.
MediumDate	Displays a date using the medium date format ("dd-MMM-yy").
ShortDate	Displays a date using specified CultureInfo's short date format.
LongTime	Displays a time using your locale's long time format; includes hours, minutes, seconds.
MediumTime	Displays time in 12-hour format using hours and minutes and the AM/PM designator ("hh:mm tt").
ShortTime	Displays a time using the 24-hour format, for example, 17:45.
LongDateShortTime	Displays the long date and short time according to specified CultureInfo's format.
LongDateLongTime	Displays the long date and long time according to specified CultureInfo's format.
ShortDateShortTime	Displays the short date and short time according to specified CultureInfo's format.
ShortDateLongTime	Displays the short date and long time according to specified CultureInfo's format.
MonthAndDay	Displays the month and the day of a date.
DateAndTimeGMT	Formats the date and time as Greenwich Mean Time (GMT).
DateTimeSortable	Formats the date and time as a sortable index.
GMTSortable	Formats the date and time as a GMT sortable index.
LongDateTimeGMT	Formats the date and time with the long date and long time as GMT.
YearAndMonth	Formats the date as the year and month.

Custom Format Specifiers

Setting FormatType to CustomFormat corresponds to the case of a *custom format specifier* as defined in the .NET Framework documentation, the specifier itself is determined by the CustomFormat property. In the CustomFormat string you define the format desired using special characters for numeric and date-time types as presented in the following tables. For details, see Custom Numeric Format Strings and Date and Time Format Strings in the .NET Framework documentation.

Numeric Custom Format Specifiers

The following table describes the custom numeric format specifiers:

Format Specifier	Name	Description
0	Zero placeholder	If the value being formatted has a digit in the position where the '0' appears in the format string, then that digit is copied to the output string. The position of the leftmost '0' before the decimal point and the rightmost '0' after

		the decimal point determines the range of digits that are always present in the output string.
#	Digit placeholder	If the value being formatted has a digit in the position where the '#' appears in the format string, then that digit is copied to the output string. Otherwise, nothing is stored in that position in the output string. Note that this specifier never displays the '0' character if it is not a significant digit, even if '0' is the only digit in the string. It will display the '0' character if it is a significant digit in the number being displayed.
.	Decimal point	The first '.' character in the format string determines the location of the decimal separator in the formatted value; any additional '.' characters are ignored. The actual character used as the decimal separator is determined by the NumberDecimalSeparator property of the NumberFormatInfo object that controls formatting.
,	Thousand separator and number scaling	The ',' character serves two purposes. First, if the format string contains a ',' character between two digit placeholders (0 or #) and to the left of the decimal point if one is present, then the output will have thousand separators inserted between each group of three digits to the left of the decimal separator. The actual character used as the decimal separator in the output string is determined by the NumberGroupSeparator property of the current NumberFormatInfo object that controls formatting. Second, if the format string contains one or more ',' characters immediately to the left of the decimal point, then the number will be divided by the number of ',' characters multiplied by 1000 before it is formatted. For example, the format string '0,,' will represent 100 million as simply 100. Use of the ',' character to indicate scaling does not include thousand separators in the formatted number. Thus, to scale a number by 1 million and insert thousand separators you would use the format string '#,##0,,'.
%	Percentage placeholder	The presence of a '%' character in a format string causes a number to be multiplied by 100 before it is formatted. The appropriate symbol is inserted in the number itself at the location where the '%' appears in the format

		string. The percent character used is dependent on the current NumberFormatInfo class.
E0 E+0 E-0 e0 e+0 e-0	Scientific notation	If any of the strings 'E', 'E+', 'E-', 'e', 'e+', or 'e-' are present in the format string and are followed immediately by at least one '0' character, then the number is formatted using scientific notation with an 'E' or 'e' inserted between the number and the exponent. The number of '0' characters following the scientific notation indicator determines the minimum number of digits to output for the exponent. The 'E+' and 'e+' formats indicate that a sign character (plus or minus) should always precede the exponent. The 'E', 'E-', 'e', or 'e-' formats indicate that a sign character should only precede negative exponents.
'ABC' "ABC"	Literal string	Characters enclosed in single or double quotes are copied to the output string literally, and do not affect formatting.
;	Section separator	The ';' character is used to separate sections for positive, negative, and zero numbers in the format string.
True|False	Boolean format	String representation for two Boolean values, True and False separated with '|', Strings "True" and "False" can be replaced with any other strings representing the two Boolean values.
Other	All other characters	All other characters are copied to the output string as literals in the position they appear.

Examples

The following table displays examples using the custom numeric format specifiers:

CustomFormat	Value	Output
#####	123	123
00000	123	00123
(###) ### - ####	1234567890	(123) 456 – 7890
#.##	1.2	1.2
0.00	1.2	1.20
00.00	1.2	01.20
#,#	1234567890	1,234,567,890
#,,	1234567890	1235

#,,,	1234567890	1
#,##0,,	1234567890	1,235
#0.##%	0.086	8.6%
0.###E+0	86000	8.6E+4
0.###E+000	86000	8.6E+004
0.###E-000	86000	8.6E004
[##-##-##]	123456	[12-34-56]
##;(##)	1234	1234
##;(##)	-1234	(1234)

Date-Time Custom Format Specifiers

The following table describes the custom date-time format specifiers:

Format Specifier	Description
d	Displays the current day of the month, measured as a number between 1 and 31, inclusive. If the day is a single digit only (1-9), then it is displayed as a single digit. Note that if the 'd' format specifier is used alone, without other custom format strings, it is interpreted as the standard short date pattern format specifier. If the 'd' format specifier is passed with other custom format specifiers, it is interpreted as a custom format specifier.
dd	Displays the current day of the month, measured as a number between 1 and 31, inclusive. If the day is a single digit only (1-9), it is formatted with a preceding 0 (01-09).
ddd	Displays the abbreviated name of the day for the specified DateTime object. If a specific valid format provider (a non-null object that implements IFormatProvider with the expected property) is not supplied, then the AbbreviatedDayNames property of the DateTimeFormat object and its current culture associated with the current thread is used. Otherwise, the AbbreviatedDayNames property from the specified format provider is used.
dddd (plus any number of additional "d" characters)	Displays the full name of the day for the specified DateTime object. If a specific valid format provider (a non-null object that implements IFormatProvider with the expected property) is not supplied, then the DayNames property of the DateTimeFormat object and its current culture associated with the current thread is used. Otherwise, the DayNames property from the specified format provider is used.
f	Displays seconds fractions represented in one digit. Note that if the 'f' format specifier is used alone, without other custom

	format strings, it is interpreted as the full (long date + short time) format specifier. If the 'f' format specifier is passed with other custom format specifiers, it is interpreted as a custom format specifier.
ff	Displays seconds fractions represented in two digits.
fff	Displays seconds fractions represented in three digits.
ffff	Displays seconds fractions represented in four digits.
fffff	Displays seconds fractions represented in five digits.
ffffff	Displays seconds fractions represented in six digits.
fffffff	Displays seconds fractions represented in seven digits.
g or gg (plus any number of additional "g" characters)	Displays the era (A.D. for example) for the specified DateTime object. If a specific valid format provider (a non-null object that implements IFormatProvider with the expected property) is not supplied, then the era is determined from the calendar associated with the DateTimeFormat object and its current culture associated with the current thread. Note that if the 'g' format specifier is used alone, without other custom format strings, it is interpreted as the standard general format specifier. If the 'g' format specifier is passed with other custom format specifiers, it is interpreted as a custom format specifier.
h	Displays the hour for the specified DateTime object in the range 1-12. The hour represents whole hours passed since either midnight (displayed as 12) or noon (also displayed as 12). If this format is used alone, then the same hour before or after noon is indistinguishable. If the hour is a single digit (1-9), it is displayed as a single digit. No rounding occurs when displaying the hour. For example, a DateTime of 5:43 returns 5.
hh, hh (plus any number of additional "h" characters)	Displays the hour for the specified DateTime object in the range 1-12. The hour represents whole hours passed since either midnight (displayed as 12) or noon (also displayed as 12). If this format is used alone, then the same hour before or after noon is indistinguishable. If the hour is a single digit (1-9), it is formatted with a preceding 0 (01-09).
H	Displays the hour for the specified DateTime object in the range 0-23. The hour represents whole hours passed since midnight (displayed as 0). If the hour is a single digit (0-9), it is displayed as a single digit.
HH, HH (plus any number of additional "H" characters)	Displays the hour for the specified DateTime object in the range 0-23. The hour represents whole hours passed since midnight (displayed as 0). If the hour is a single digit (0-9), it is formatted with a preceding 0 (01-09).
m	Displays the minute for the specified DateTime object in the range 0-59. The minute represents whole minutes passed

	since the last hour. If the minute is a single digit (0-9), it is displayed as a single digit. Note that if the 'm' format specifier is used alone, without other custom format strings, it is interpreted as the standard month day pattern format specifier. If the 'm' format specifier is passed with other custom format specifiers, it is interpreted as a custom format specifier.
mm, mm (plus any number of additional "m" characters)	Displays the minute for the specified DateTime object in the range 0-59. The minute represents whole minutes passed since the last hour. If the minute is a single digit (0-9), it is formatted with a preceding 0 (01-09).
M	Displays the current month, measured as a number between 1 and 12, inclusive. If the month is a single digit (1-9), it is displayed as a single digit. Note that if the 'M' format specifier is used alone, without other custom format strings, it is interpreted as the standard month day pattern format specifier. If the 'M' format specifier is passed with other custom format specifiers, it is interpreted as a custom format specifier.
MM	Displays the current month, measured as a number between 1 and 12, inclusive. If the month is a single digit (1-9), it is formatted with a preceding 0 (01-09).
MMM	Displays the abbreviated name of the month for the specified DateTime object. If a specific valid format provider (a non-null object that implements IFormatProvider with the expected property) is not supplied, the AbbreviatedMonthNames property of the DateTimeFormat object and its current culture associated with the current thread is used. Otherwise, the AbbreviatedMonthNames property from the specified format provider is used.
MMMM	Displays the full name of the month for the specified DateTime object. If a specific valid format provider (a non-null object that implements IFormatProvider with the expected property) is not supplied, then the MonthNames property of the DateTimeFormat object and its current culture associated with the current thread is used. Otherwise, the MonthNames property from the specified format provider is used.
s	Displays the seconds for the specified DateTime object in the range 0-59. The second represents whole seconds passed since the last minute. If the second is a single digit (0-9), it is displayed as a single digit only. Note that if the 's' format specifier is used alone, without other custom format strings, it is interpreted as the standard sortable date/time pattern format specifier. If the 's' format specifier is passed with other custom format specifiers, it is interpreted as a custom format specifier.

ss, ss (plus any number of additional "s" characters)	Displays the seconds for the specified DateTime object in the range 0-59. The second represents whole seconds passed since the last minute. If the second is a single digit (0-9), it is formatted with a preceding 0 (01-09).
t	Displays the first character of the A.M./P.M. designator for the specified DateTime object. If a specific valid format provider (a non-null object that implements IFormatProvider with the expected property) is not supplied, then the AMDesignator (or PMDesignator) property of the DateTimeFormat object and its current culture associated with the current thread is used. Otherwise, the AMDesignator (or PMDesignator) property from the specified IFormatProvider is used. If the total number of whole hours passed for the specified DateTime is less than 12, then the AMDesignator is used. Otherwise, the PMDesignator is used. Note that if the 't' format specifier is used alone, without other custom format strings, it is interpreted as the standard long time pattern format specifier. If the 't' format specifier is passed with other custom format specifiers, it is interpreted as a custom format specifier.
tt, tt (plus any number of additional "t" characters)	Displays the A.M./P.M. designator for the specified DateTime object. If a specific valid format provider (a non-null object that implements IFormatProvider with the expected property) is not supplied, then the AMDesignator (or PMDesignator) property of the DateTimeFormat object and its current culture associated with the current thread is used. Otherwise, the AMDesignator (or PMDesignator) property from the specified IFormatProvider is used. If the total number of whole hours passed for the specified DateTime is less than 12, then the AMDesignator is used. Otherwise, the PMDesignator is used.
y	Displays the year for the specified DateTime object as a maximum two-digit number. The first two digits of the year are omitted. If the year is a single digit (1-9), it is displayed as a single digit.
yy	Displays the year for the specified DateTime object as a maximum two-digit number. The first two digits of the year are omitted. If the year is a single digit (1-9), it is formatted with a preceding 0 (01-09).
yyyy	Displays the year for the specified DateTime object, including the century. If the year is less than four digits in length, then preceding zeros are appended as necessary to make the displayed year four digits long.
z	Displays the time zone offset for the system's current time zone in whole hours only. The offset is always displayed with a leading or trailing sign (zero is displayed as '+0'), indicating hours ahead of Greenwich mean time (+) or hours behind Greenwich mean time (-). The range of values is –12 to +13. If

	the offset is a single digit (0-9), it is displayed as a single digit with the appropriate leading sign. The setting for the time zone is specified as +X or –X where X is the offset in hours from GMT. The displayed offset is affected by daylight time.
zz	Displays the time zone offset for the system's current time zone in whole hours only. The offset is always displayed with a leading or trailing sign (zero is displayed as '+00'), indicating hours ahead of Greenwich mean time (+) or hours behind Greenwich mean time (-). The range of values is –12 to +13. If the offset is a single digit (0-9), it is formatted with a preceding 0 (01-09) with the appropriate leading sign. The setting for the time zone is specified as +X or –X where X is the offset in hours from GMT. The displayed offset is affected by daylight time.
zzz, zzz (plus any number of additional "z" characters)	Displays the time zone offset for the system's current time zone in hours and minutes. The offset is always displayed with a leading or trailing sign (zero is displayed as '+00:00'), indicating hours ahead of Greenwich mean time (+) or hours behind Greenwich mean time (-). The range of values is –12 to +13. If the offset is a single digit (0-9), it is formatted with a preceding 0 (01-09) with the appropriate leading sign. The setting for the time zone is specified as +X or –X where X is the offset in hours from GMT. The displayed offset is affected by daylight time.
:	Time separator.
/	Date separator.
"	Quoted string. Displays the literal value of any string between two quotation marks preceded by the escape character (/).
'	Quoted string. Displays the literal value of any string between two " ' " characters.
%*c*	Where *c* is a standard format character, displays the standard format pattern associated with the format character.
\c	Where *c* is any character, the escape character displays the next character as a literal. The escape character cannot be used to create an escape sequence (like "\n" for new line) in this context.
|	Section separator. Custom date-time format can contain multiple format strings separated with '|', This feature allows to specify multiple input formats for date-time values. Only the first format string is used to format value, convert date-time to string. Performing the inverse conversion (parsing) from string to date-time, a string will be recognized (parsed) if it satisfies one of the allowed formats.
Any other character	Other characters are written directly to the output string as literals.

Examples

The following table displays examples using the custom date-time format specifiers:

Format Specifiers	Current Culture	Time Zone	Output
d, M	en-US	GMT	12, 4
d, M	es-MX	GMT	12, 4
d MMMM	en-US	GMT	12 April
d MMMM	es-MX	GMT	12 Abril
dddd MMMM yy gg	en-US	GMT	Thursday April 01 A.D.
dddd MMMM yy gg	es-MX	GMT	Jueves Abril 01 DC
h , m: s	En-US	GMT	6 , 13: 12
hh,mm:ss	En-US	GMT	06,13:12
HH-mm-ss-tt	En-US	GMT	06-13-12-AM
hh:mm, G\MT z	En-US	GMT	05:13 GMT +0
hh:mm, G\MT z	En-US	GMT +10:00	05:13 GMT +10
hh:mm, G\MT zzz	En-US	GMT	05:13 GMT +00:00
hh:mm, G\MT zzz	En-US	GMT –9:00	05:13 GMT -09:00

Parsing (Updating) Data

Data modified by the end user in a **C1Input** control is converted from a string to a typed Value. Converting data from a string representation is called *parsing*. It is the opposite of formatting. Parsing is controlled by the ParseInfo property. The ParseInfo property provides access to the ParseInfo Class that contains sub-properties that control different aspects of parsing.

For the most part, you will probably be satisfied with the default parsing that is performed according to the format specification, as it is the inverse of formatting. By default, the same format property value is used for parsing as for formatting. However, you can change any of aspects of how the control parses, by expanding the ParseInfo property, changing the **(Inherit)** flags, and setting desired properties.

The ParseInfo class also contains two flag properties, NumberStyle and DateTimeStyle, which enable you to fine-tune parsing by allowing or disallowing white spaces and special characters in input strings for numeric and date-time data. For more information see the ParseInfo class in the reference section.

By setting the FormatType property to UseEvent, you can make your parsing entirely custom through writing code to handle the parsing action in the Parsing event.

> **Note:** Parsing is not performed in DateTimeInput and NumericInput modes. It is unnecessary, because in this case the content is already a typed value (date/time or number), so there is no need to parse a string to obtain the value.

Culture (Regional) Settings

Regional settings affect almost all aspects of **C1Input** functionality. Formatting, parsing, validating data and performing masked input all depend on cultural settings for string comparison, numeric and date time formats and special characters, such as decimal point character. See description of the CultureInfo class in .NET Framework documentation for details on culture-specific settings.

C1Input controls use the following properties to define **CultureInfo**:

- **Culture** property

  The Culture property defines what culture is used by the control. It is an integer ID with a list of all cultures available at design time. The default is **Current Culture**, which is the current culture used by the application containing the control.

- **CultureInfo** property

  The CultureInfo property contains all the settings, the CultureInfo object corresponding to the specified culture ID.

- **UserCultureOverride** property

  The Boolean UserCultureOverride property allows the culture settings to be overridden by the end user regional settings.

You can change any settings in the CultureInfo programmatically. To enable this, **C1Input** controls fire the CultureInfoSetup event at startup and whenever the Culture property is set. Handling this event you can fine-tune various CultureInfo settings. For example, you may want to set CultureInfo.DateTimeFormat.FirstDayOfWeek according to your application needs.

Another setting affecting most of **C1Input** functionality is the Boolean CaseSensitive property (although it is not culture-related). Case sensitivity is used in string comparisons. **C1Input** controls have a CaseSensitive property that defines the default case sensitivity for all operations. You can override this setting in most classes controlling particular functionality, such as ParseInfo, PreValidation, PostValidation, and so on.

Edit Mask

C1Input controls support masked input when you set the EditMask property to a mask string. If you define an edit mask, each character position in the control maps to either a special placeholder or a literal character. Literal characters, or literals, can give visual cues about the type of data being used. For example, the parentheses surrounding the area code of a telephone number and dashes are literals: (412)-123-4567. The edit mask prevents you from entering invalid characters into the control and provides other enhancements of the user interface.

To enable masked input, set the EditMask property to a mask string composed of placeholders and literals, see the table of available placeholders below. You can also define your own placeholders, using the CustomPlaceholders collection.

Although setting EditMask is enough in simple cases, there is also a MaskInfo property containing sub-properties controlling various important aspects of masked input. One of them is the CustomPlaceholders collection mentioned above. Some of the others are:

Property	Description
AutoTabWhenFilled	If **True**, focus automatically moves to the next control when the mask is filled. Default: **False**.
PromptChar	Character displayed on empty positions. Default: '_'.
SaveBlanks	If **True**, the stored text includes blank positions as StoredEmptyChar. Default: **False**.
SaveLiterals	If **True** (default), the stored text (StoredContent) includes literals.
ShowLiterals	Enumeration controlling the way in which literals appear while the user types. They can appear

	always, or never, or as the user reaches a literal while typing.
SkipOptional	If **True** (default), optional mask positions are automatically skipped until the first position allowing the typed character.
StoredEmptyChar	Character stored in empty mask positions. Default: '_'.

See the MaskInfo class for the complete list of mask-related properties.

If ShowLiterals property is set to FreeFormatEntry, optional mask positions can be completely omitted; there is no need to fill them with blank characters.

Mask characters (placeholders) used in **C1Input** are similar to those used in Microsoft Access and Microsoft ActiveX MaskedEdit control (and more placeholders can be defined using the using the CustomPlaceholders collection):

Placeholder	Description
#	Digit placeholder permits a numeric character or a plus or minus sign in this position (entry optional).
.	Decimal placeholder. The actual character used is the one specified as the decimal placeholder in your international settings. This character is treated as a literal for masking purposes.
,	Thousands separator. The actual character used is the one specified as the thousands separator in your international settings. This character is treated as a literal for masking purposes.
:	Time separator. The actual character used is the one specified as the time separator in your international settings. This character is treated as a literal for masking purposes.
/	Date separator. The actual character used is the one specified as the date separator in your international settings. This character is treated as a literal for masking purposes.
\	Treat the next character in the mask string as a literal. This allows you to include the #, &, A, ... characters in the mask. This character is treated as a literal for masking purposes.
&	Character placeholder (entry required). Any character is permitted.
>	Convert all the characters that follow to uppercase.
<	Convert all the characters that follow to lowercase.
~	Turns off the previous < or >.
!	Causes the optional characters that follow in the edit mask to display from right to left, rather than from left to right. So, blanks appear on the left.
^	Turns off the previous ! character. After ^, blanks appear on the right.
A	Alphanumeric character placeholder (entry required). For example: a – z, A – Z, or 0 – 9.
a	Alphanumeric character placeholder (entry optional).
0	Digit placeholder (entry required). For example: 0 – 9.
9	Digit placeholder (entry optional).

C	Character or space placeholder (entry optional). Any character is permitted.
L	Letter placeholder (entry required). For example: a – z or A – Z.
?	Letter placeholder (entry optional).
\n	New line literal. It is applicable when **Multiline** property is set to **True**.
"	All characters in a string enclosed in double quotes are considered as literals.
Literal	All other symbols are displayed as literals; that is, as themselves.

Example

The telephone number mentioned above, (412) 123-4567 can be represented with a mask EditMask set to **(000) 000-0000**.

Validating Data

C1Input controls support data validation both of the raw input string (PreValidation) and of the typed value entered by the user (PostValidation). See Value and Text: Displaying, Validating, and Updating Values for explanation of the validation process.

Input String Validation (PreValidation)

Input string validation is controlled by the PreValidation property. The PreValidation class allows you to specify validation rules either as wildcard pattern strings or regular expression strings. All rules (strings) are specified in the PatternString property. Multiple rules (sub-strings) are separated by the ItemSeparator ('|' by default).

The PreValidation property defines how the PatternString is interpreted.

Value	Description
ExactList	PatternString contains a list of possible values separated by ItemSeparator.
PreValidatingEvent	The PreValidating event is being used in validation.
Wildcards	PatternString contains a list of wildcard patterns separated by the ItemSeparator. The following characters are reserved in a pattern: ? (any single character), # (any single digit), * (zero or more characters), \ (escape). You can also define your own custom pattern characters using the PreValidation property.
RegexPattern	PatternString contains a regular expression.

Using the PreValidatingEvent option, you can perform input string validation in code, in the PreValidating event. For more information, see the event description.

If you use regular expressions, the RegexPattern option, there is also a RegexOptions property that is sometimes needed to set flags affecting regular expression functionality.

> **Note:** Input string validation (PreValidation) is not used in DateTimeInput and NumericInput modes. When DateTimeInput or NumericInput modes are active is Typed Value Validation (PostValidation) is performed.

Examples

The following examples describe how the Validation and PatternString properties are interpreted:

- Validation property set to ExactList, PatternString property set to red|green|blue: Input string must be one of the three permitted values, red, green or blue, possibly ignoring the case, if CaseSensitive is set to **False**.
- Validation property set to Wildcards, PatternString property set to (412)*: Input string must start with (412), possibly ignoring the case, if CaseSensitive is set to **False**.
- Validation property set to RegexPattern, PatternString set to [0-9]*: Input string contains one or more digits.

Typed Value Validation (PostValidation)

PostValidation allows you to validate the typed Value entered by the user.

The PostValidation class allows you to:

- Check that the value matches one of the values in a pre-defined value list specified in the Values property.
- Test the value to see if it is below a minimum or above a maximum, that is you can test to see if the value belongs to an interval.

  You can even specify multiple intervals of allowed values. The intervals are defined in the Intervals property where you specify minimum and maximum values for each interval, and also whether minimum and maximum values are used or ignored, and whether or not the inequality is strict (minimum/maximum value included).

- Exclude some values using the ValuesExcluded property.
- Perform validation programmatically in the PostValidating event.

To distinguish between declarative and programmatic validation, use the PostValidation property with two possible values: ValuesAndIntervals and PostValidatingEvent. Note that PostValidatingEvent disables automatic validation of values and intervals. Call the ValidateValuesAndIntervals method from the event code if you want to combine event code with values and intervals validation.

Editing Date and Time Values

C1TextBox supports a special editing mode called DateTimeInput mode that makes editing date and time values easier. This mode is enabled when DataType property is set to **DateTime** and DateTimeInput property is set to **True** (default). In the DateTimeInput mode, the currently selected date or time field, such as the year, month, date, and so on, is highlighted and edited separately. Formatted fields represented in string form, such as month or day of the week in LongDate format, can be typed as numbers on the keyboard, and their string representation is updated automatically. The UP ARROW/DOWN ARROW keys or mouse wheel can be used to increment/decrement the current field.

Additional properties controlling date-time input are:

Property	Description
MinShortYear	The minimum year that can be entered without leading zeros (when DateTimeInput is set to **True**). For example, if MinShortYear is set to **300**

	(default), entering 200 is not allowed (will be ignored), whereas 400 is interpreted as 0400 A.D. Regardless of this property value, entering 0200 will be interpreted as year 0200 A.D.
CurrentTimeZone	This property is **True** by default, which means that date-time values are invariant, not adjusted to time zones. If this property is set to **False**, the Text shown to the user and the underlying stored Value become different. The stored Value belongs to the time zone defined by the GMTOffset property specifying the offset in hours and minutes of the base time zone from Greenwich Mean Time. The Text shown to the user belongs to the local time zone defined by the user computer settings. Displaying values and parsing values entered by the user, **C1Input** adjusts it to the time zone difference.

To make editing date-time values even more convenient to the user, you can use the specialized C1DateEdit control. In addition to C1TextBox functionality, it supports a drop-down calendar and up/down buttons (speedbuttons) incrementing/decrementing the currently selected date-time field.

Editing Numeric Values

C1TextBox supports a special editing mode called NumericInput mode that makes editing numeric values easier. This mode is enabled when the NumericInput property is set to **True** (default) and DataType is one of numeric data types (Byte, UInt16, UInt32, UInt64, SByte, Int16, Int32, Int64, Decimal, Single, Double). In the NumericInput mode, numbers are edited in a calculator-like fashion. It accepts only digits, +/- sign, and, if data type and format allow, decimal point and exponent. Other characters, such as letters, are ignored. There are also special functional keys recognized in NumericInput mode for values of type Single and Double: F9 (change sign), F2 (negative infinity), F3 (positive infinity), F4 (NaN, "not a number").

To make editing numbers even more convenient to the user, you can use the specialized C1NumericEdit control. In addition to C1TextBox functionality, it supports a drop-down calculator and up/down buttons (speed buttons) incrementing/decrementing the value by the specified Increment.

Drop-Down and Increment Buttons

The specialized **C1Input** controls for date-time and numeric editing, C1DateEdit and C1NumericEdit controls, support drop-down and increment/decrement (up/down) buttons. Button visibility is controlled by the ShowDropDownButton and ShowUpDownButtons properties.

To control drop-down alignment and distance from the control you can use the DropDownAlign and GapHeight properties. To open/close drop-down programmatically, use the OpenDropDown and CloseDropDown methods. Opening/closing drop-down triggers events DropDownOpened and DropDownClosed. You can use the DropDownOpened event to adjust drop-down properties (mostly, calendar properties in Calendar) before the drop-down is shown to the user. You can check if the drop-down is open using the DroppedDown property.

C1DateEdit Control

C1DateEdit control supports up/down buttons and drop-down calendar.

The up/down buttons function if DateTimeInput property is set to **True**. They increment/decrement the currently selected field of the date-time value, see Editing Date and Time Values.

The drop-down calendar has the same object model as the standard **MonthCalendar** control (System.Windows.Forms.MonthCalendar) and almost the same appearance with additional buttons such as Clear, Today, and two year navigation buttons.

Button visibility is controlled by the properties ShowClearButton and ShowTodayButton. These properties and all other calendar properties are available both in the designer and in code in the Calendar object. If you want to change calendar properties programmatically when the calendar is opened, before it is shown to the user, use the DropDownOpened event.

C1NumericEdit Control

C1NumericEdit control supports up/down (spin) buttons and drop-down calculator.

Up/down buttons increment/decrement the **Value** by the amount specified in the Increment property (default: 1).

The drop-down calculator follows the standard Windows calculator model, allows the user to perform calculations without leaving the control.

Custom Drop-Down

C1Input includes a powerful custom drop-down functionality that allows you to create any drop-down editor you need, in addition to the standard calendar and calculator drop-downs provided by **C1Input**. Drop-down editors are created visually as forms in your project.

To create your own custom drop-down editor, use the C1DropDownControl. This control class derives from C1TextBox and adds custom drop-down functionality and up/down buttons.

To create a drop-down editor for your control:

1. Add a form to your project derived from C1.Win.C1Input.DropDownForm and select the form class name in the DropDownFormClassName property of your C1DropDownControl.

2. In your DropDownForm-derived form you can set the **Value** property of C1DropDownControl when necessary (use the **DropDownForm.OwnerControl** property to get the control object), or you can do it when the form is closing, in the PostChanges event.

See the DropDownForm class reference for the full description of available options for custom drop-down forms. Also see **Documents\ComponentOne Samples\WinForms** (installed by default) for common-use samples of the custom drop-down functionality.

If you need to create a custom control with drop-down functionality, this can be done by deriving a custom control class from C1DropDownControl and overriding its **C1DropDownControl.DefaultDropDownFormClassName** property.

Programmatic Formatting, Parsing, and Validation

If standard and custom format specifiers are not enough, you can format values in code in the Formatting event by setting the FormatType property to UseEvent, see Formatting Data. In your formatting code, you can use the standard **C1Input** formatting as a helper or for any other purposes, calling the Format method.

Parsing can also be done in event code, in the Parsing event, by setting the FormatType property to UseEvent. You can use standard **C1Input** parsing routines in your code if you need them, with the following ParseInfo methods: Parse, ParseFixed, ParseFloat, ParseInteger, ParseBoolean and ParseDateTime.

Some useful methods for edit mask management can be found in the MaskInfo class.

When you need to synchronize the Value property with the text currently entered by the user, call the UpdateValueWithCurrentText method. Normally, this synchronization is done automatically when the control loses focus, but in certain situations you may find necessary to call this method and force the Value update. Updating Value involves parsing the input text, validating, and updating the Value property, see Value and Text: Displaying, Validating, and Updating Values. You can also perform the first two phases, parsing and validation without changing the Value, using the methods ParseContent and CheckValidationResult.

Error Handling

Error handling is very important in data input forms. **C1Input** gives developers full control over various error conditions, such as the errors listed in the following topics.

Data Errors

WinForms data sources such as ADO.NET and C1DataObjects contain provisions for detecting logical errors in data, by the data source itself or by the programmer, setting the **RowError** property or calling the **SetColumnError** method (**SetFieldError** in C1DataObjects). You can show logical row and column errors in **C1Input** using System.Windows.Form.ErrorProvider component.

To show logical column errors, use the ErrorProvider component with **C1Input** controls as you would use it with any other controls.

To show row error in C1DbNavigator control, set the ErrorProvider property to an ErrorProvider component. Then C1DbNavigator will display error icon with RowError ToolTip text when there is an error in current row.

Incorrect Format in Displaying Data

Incorrect format is possible, although generally avoided in applications that data fetched from the database or another data source does not match the format or edit mask defined in a **C1Input** control. In such cases, the control cannot show its value properly formatted. Although **C1Input** controls have reasonable default behavior handling this situation, you may want to inform the user of invalid data. This is done using the ErrorProvider property (ErrorProvider in C1Label control). If you set this property to an ErrorProvider component, **C1Input** uses that ErrorProvider component to signal errors when it displays invalid data (data that can't be formatted for display in the control). It calls the ErrorProvider.SetControl method when such error is detected. Before doing that, **C1Input** fires the FormatError event where you can customize the error message (ErrorProvider ToolTip text) and perform other actions.

User Input Errors

When **C1Input** detects an error while parsing or validating input value, it fires the ValidationError event. Then, by default, it shows an error message. The default behavior can be changed and customized in different ways:

C1Input controls have an ErrorInfo property containing settings (properties of the ErrorInfo class) affecting error handling:

Property	Description
BeepOnError	If **True**, the control beeps signaling an error. Default: **False**.
CanLoseFocus	If **True**, the control is allowed to lose focus regardless of the error. This property is **False** by default, meaning that the control will stay in focus until the error is corrected.

	Note that setting ErrorAction to SetValueOnError or ResetValue allows the user to leave the control after error by resetting its value.
ErrorAction	Enumerated value that determines what action is performed on the control value when an error occurs. ErrorAction set to None (default) means that the **Value** is not changed, remains as it was before the unsuccessful value update. If ErrorAction is set to SetValueOnError, control's **Value** is set to the value specified in the ValueOnError property of the ErrorInfo class. If ErrorAction is set to ResetValue, control's **Value** is set to the last value the control had before it entered edit mode. Setting the ErrorAction property to ThrowException interrupts execution and throws an exception, ValidationException.
ErrorMessage	Error message shown in the standard message box and/or in the exception.
ErrorMessageCaption	The text to display in the title bar of the error message box.
ErrorProvider	Gets or sets an ErrorProvider object used to indicate error state of the control.
ShowErrorMessage	If **True** (default), the standard error message is shown.
ValueOnError	Value used to reset the control if ErrorAction is set to SetValueOnError.
ValueOnErrorIsDbNull	Boolean property used to set ValueOnError to **DBNull** (only necessary at design time).

In addition to that, ErrorInfo.ErrorMessage can be specified for particular actions: edit mask errors (MaskInfo.ErrorMessage), parsing (ParseInfo.ErrorMessage), pre- and post-validation (PreValidation.ErrorMessage, PostValidation.ErrorMessage). If a specialized error message is not specified in one of these sub-objects, the control's ErrorInfo.ErrorMessage takes effect. Note that you can use ErrorProvider icon to indicate the error, instead of showing a message box, if you set ErrorProvider to an ErrorProvider component and ShowErrorMessage to **False**.

The properties listed above, when set in the control's ErrorInfo object, affect all error handling in the control. When an error occurs, their values can be customized programmatically to handle that particular error. This is achieved by passing an ErrorInfo argument to the ValidationError event. The ErrorInfo argument passed to ValidationError is a copy of the control's ErrorInfo with all its properties. It is an independent copy, so you can change properties in the ErrorInfo event argument for the current error without affecting the overall control's ErrorInfo settings. By setting ErrorInfo properties in the ValidationError event you specify how to handle the error. For example, you can suppress the standard error message (and show your own message instead) by setting the ShowErrorMessage property to **False**, or you can change ValueOnError (and set ErrorAction to SetValueOnError), or change the ErrorMessage. Keep in mind that you must set the properties of the ErrorInfo argument passed to the ValidationError event, not the properties of the control's ErrorMessage.

After the ValidationError event, error handling proceeds as specified in the event's ErrorInfo argument. If ErrorAction is set to ThrowException, an exception is thrown (using the ErrorMessage text). If BeepOnError is **True**, the control beeps. If ShowErrorMessage is set to **True**, the standard error message box is shown (with ErrorMessage text). After that, the control's value may be changed if so specified by ErrorAction. Finally, moving focus to another control is either canceled or permitted, according to CanLoseFocus, if validation was triggered by an attempt to move focus out of the control. If ErrorProvider property is set to an ErrorProvider component, that component is used to show the error icon near the offending control, with ErrorMessage ToolTip (**ErrorProvider.SetError** is called).

If you perform parsing or validation programmatically, in event code, and exit the event with an error condition (set the event's **Succeeded** argument to **False**), you can describe the error and how it must be handled by setting the properties of an ErrorInfo argument passed to the event. Such argument is provided for the following events: PreValidating, Parsing and PostValidating. Its initial values are taken from the control's ErrorInfo property. This ErrorInfo argument that you change in the event is then passed to the ValidationError event where it can be further changed as described above.

Handling NULL and Empty Values

NULL values (DBNull) can be difficult to handle without appropriate tools. **C1Input** provides flexible rules for handling nulls allowing the programmer the ability to solve this problem in practically any circumstance.

Displaying NULL and Empty Values

When a control is not in edit mode or is read-only, null values are displayed according to the NullText property (which can be overridden in DisplayFormat). If the property EmptyAsNull is set to **True** (default: **False**), empty strings are also displayed with the same NullText string. The EmptyAsNull property can also be overridden in EmptyAsNull. In edit mode, the NullText and EmptyAsNull properties of the EditFormat object take effect, instead of those of DisplayFormat.

In edit mode with an active EditMask, the null value and the empty string are shown as an empty mask with literals in their places and prompt characters filling the rest.

When editing a date-time value with the DateTimeInput property set to **True**, the null value is represented by an empty control. When the user starts editing with a keystroke or mouse click, the control immediately turns to a non-null value, namely, to the last non-null value assigned to the control or to today's date.

In programmatic formatting (FormatType set to UseEvent), the Formatting event is only called for non-null values.

Entering NULL and Empty Values

Unless in edit mask mode or in date-time editing withDateTimeInput set to **True**, the user can enter a null value in one of the following ways:

- If the control text equals NullText, the resulting value is null. The effective NullText value here is determined by NullText. Comparison with NullText is case-sensitive or not depending on the **CaseSensitive** property.
- Clearing the control, entering an empty string results in null value if the C1TextBox.EmptyAsNull property is set to True (it is False by default).
- In programmatic parsing, DBNull value can be returned by the programmer in the **Parsing** event.

If the user enters a null value (either by entering an empty string or NullText, see above), input string validation and parsing are skipped, see Input string validation (PreValidation) and Parsing (Updating) Data. However, PostValidation is performed in this case as in all others, see Typed value validation (PostValidation).

Customizing C1Input's Appearance

C1Input is designed to make customization easy for you. You have endless possibilities in changing the default appearance for each C1Input control. C1Input provides numerous property styles for the input boxes, as well as built-in themes for Windows XP and Office 2007.

Visual Styles

C1Input has seven built-in visual styles: System, Office2007Blue, Office2007Black, Office2007Silver, Office2010Blue, Office2010Black, and Office2010Silver.

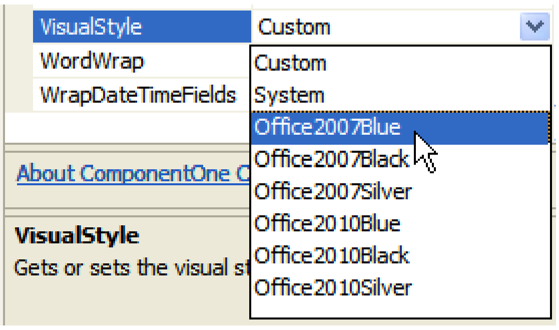
Setting the **VisualStyle** property on a C1Input control will control the gradients and borders used to paint C1TextBox, C1Label, C1DbNavigator, C1DropDownControl, C1DateEdit (including the drop down calendar), C1NumericEdit (including the drop down calculator), C1SplitButton, C1ComboBox,C1CheckBox, C1RangeSlider and C1Button.

To customize the appearance of a C1Input control using Visual Styles, set the **VisualStyle** property to **Custom**, **Office2007Black**, **Office2007Blue**, **Office2007Silver**, **System**, **Office2010Blue**, **Office2010Black**, or **Office2010Silver**. This property can be set either in the designer or in code. The following table describes each of the Visual Styles:

Visual Style	Description
Custom	No visual style (use styles and appearance properties as usual).
Office2007Black	Style matches Office2007 Black color scheme.
Office2007Blue	Style matches Office2007 Blue color scheme.
Office2007Silver	Style matches Office2007 Silver color scheme.
System	Style matches the current system settings.
Office2010Blue	Style matches Office2010 Blue color scheme.
Office2010Black	Style matches Office2010 Black color scheme.
Office2010Silver	Style matches Office2010 Silver color scheme.

Using the Designer

Locate the **VisualStyle** property in the Properties window and set it to **Custom**, **Office2007Black**, **Office2007Blue**, **Office2007Silver**, **System**, **Office2010Blue**, **Office2010Black**, or **Office2010Silver**. In this example, the **VisualStyle** property is set to **Office2007Blue** for a C1TextBox control.



Using the Code Editor

Add code to the **Form_Load** event to set the **VisualStyle** property of a control to **Custom**, **Office2007Black**, **Office2007Blue**, **Office2007Silver**, or **System**. The following code sets the **VisualStyle** property to **Office2007Blue** for a C1TextBox control:

To write code in Visual Basic

Visual Basic

```
Me.C1TextBox1.VisualStyle = C1.Win.C1Input.VisualStyle.Office2007Blue
```

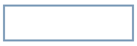
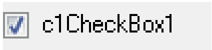
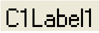
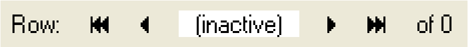
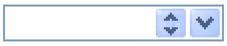
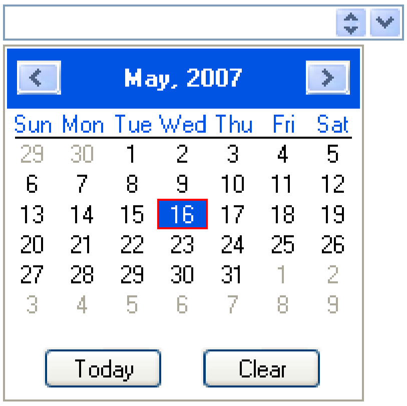
To write code in C#

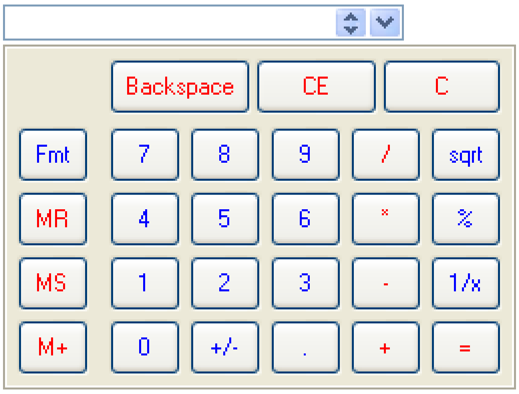
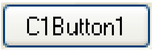
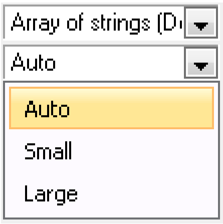
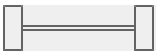
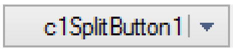
C#

```
this.c1TextBox1.VisualStyle = C1.Win.C1Input.VisualStyle.Office2007Blue;
```

Custom Visual Style

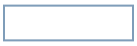
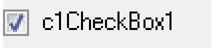
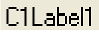
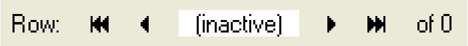
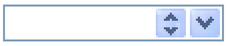
No visual style is applied.

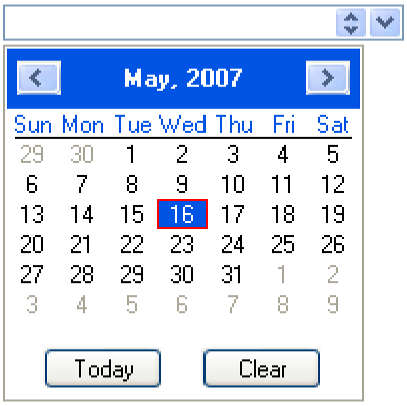
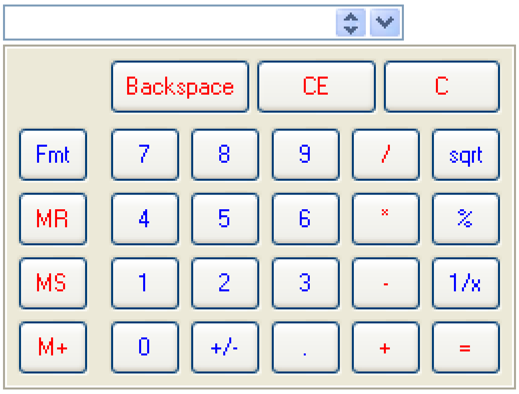
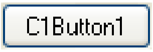
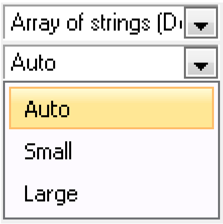
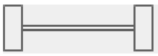
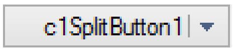
C1TextBox	
C1CheckBox	
C1Label	
C1DbNavigator	
C1DropDownControl	
C1DateEdit	 May, 2007 Sun Mon Tue Wed Thu Fri Sat 29 30 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 1 2 3 4 5 6 7 8 9 Today Clear
C1NumericEdit	

	 Backspace CE C Fmt 7 8 9 / sqrt MR 4 5 6 * % MS 1 2 3 - 1/x M+ 0 +/- . + =
C1Button	
C1ComboBox	Array of strings (D Auto Auto Small Large
C1RangeSlider	
C1SplitButton	

System Visual Style

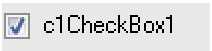
The current system settings.

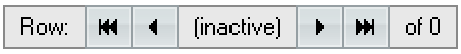
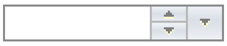
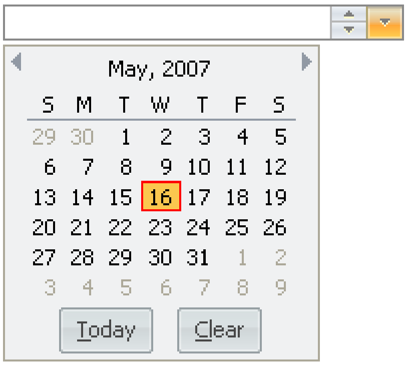
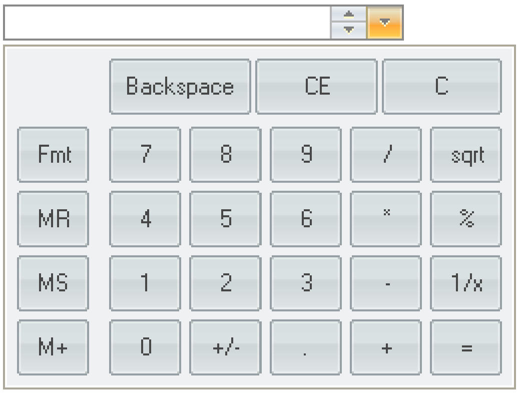
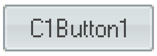
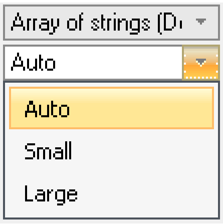
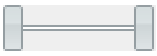
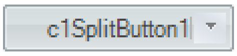
C1TextBox	
C1CheckBox	☑ c1CheckBox1
C1Label	
C1DbNavigator	Row: ⏮ ◀ (inactive) ▶ ⏭ of 0
C1DropDownControl	
C1DateEdit	

	 May, 2007 Sun Mon Tue Wed Thu Fri Sat 29 30 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 1 2 3 4 5 6 7 8 9 Today Clear
C1NumericEdit	 Backspace CE C Fmt 7 8 9 / sqrt MR 4 5 6 * % MS 1 2 3 - 1/x M+ 0 +/- . + =
C1Button	
C1ComboBox **Note:** For illustration purposes two comboboxes are shown.	
C1RangeSlider	
C1SplitButton	

Office2007Black Visual Style

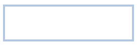
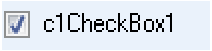
The Office 2007 Black color scheme.

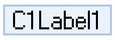
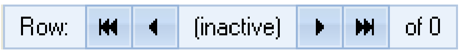
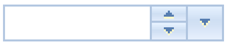
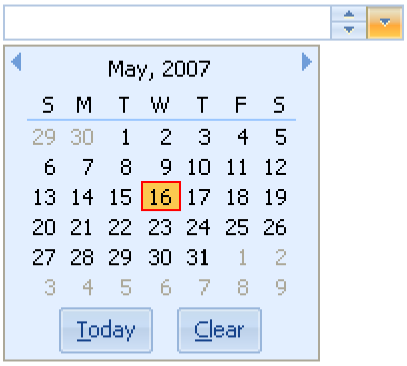
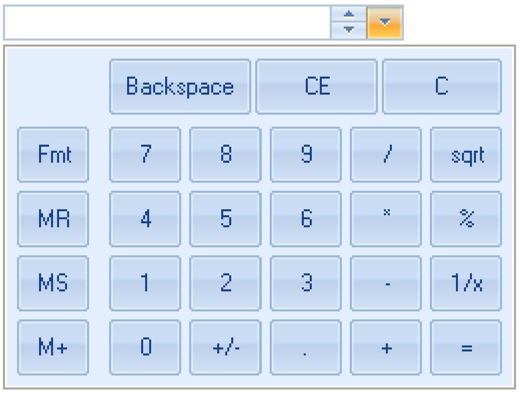
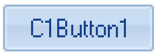
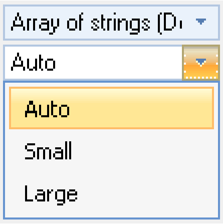
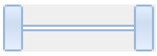
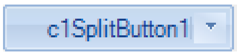
C1TextBox	
C1CheckBox	
C1Label	
C1DbNavigator	

	
C1DropDownControl	
C1DateEdit	
C1NumericEdit	 Backspace CE C Fmt 7 8 9 / sqrt MR 4 5 6 * % MS 1 2 3 - 1/x M+ 0 +/- . + =
C1Button	
C1ComboBox **Note:** For illustration purposes two comboboxes are shown.	
C1RangeSlider	
C1SplitButton	

Office2007Blue Visual Style

The Office 2007 Blue color scheme.

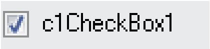
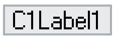
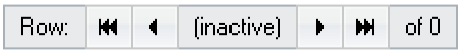
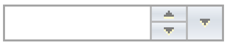
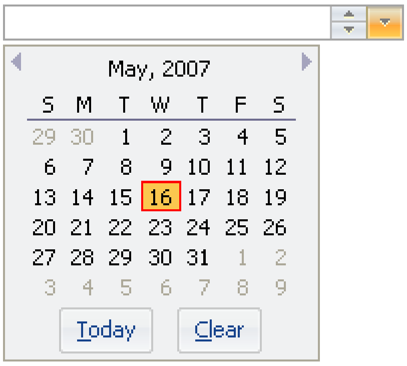
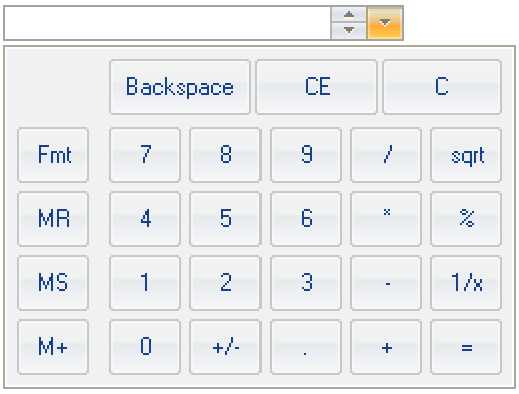
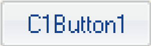
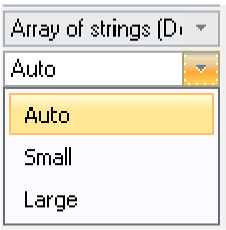
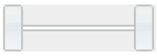
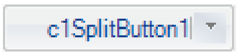
C1TextBox	
C1CheckBox	

C1Label	
C1DbNavigator	
C1DropDownControl	
C1DateEdit	 May, 2007 S M T W T F S 29 30 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 1 2 3 4 5 6 7 8 9 Today Clear
C1NumericEdit	 Backspace CE C Fmt 7 8 9 / sqrt MR 4 5 6 * % MS 1 2 3 - 1/x M+ 0 +/- . + =
C1Button	
C1ComboBox **Note:** For illustration purposes two comboboxes are shown.	
C1RangeSlider	
C1SplitButton	

Office2007Silver Visual Style

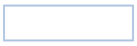
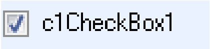
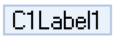
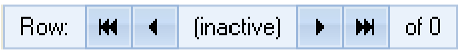
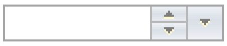
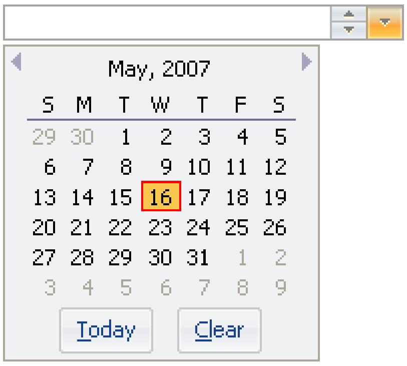
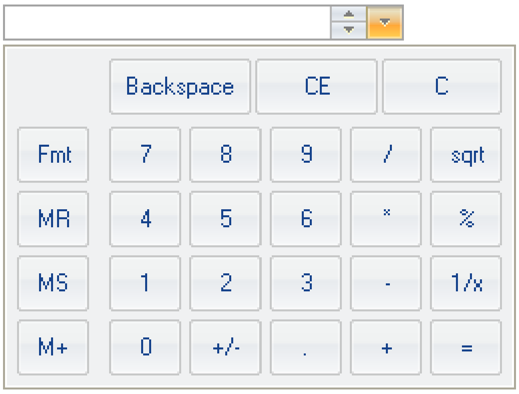
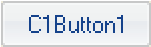
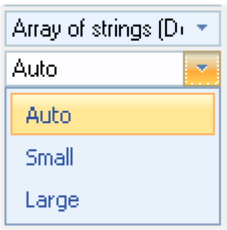
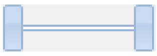
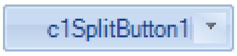
The Office 2007 Silver color scheme.

C1TextBox	

C1CheckBox	
C1Label	
C1DbNavigator	
C1DropDownControl	
C1DateEdit	 May, 2007 S M T W T F S 29 30 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 1 2 3 4 5 6 7 8 9 Today Clear
C1NumericEdit	 Backspace CE C Fmt 7 8 9 / sqrt MR 4 5 6 * % MS 1 2 3 - 1/x M+ 0 +/- . + =
C1Button	
C1ComboBox Note: For illustration purposes two comboboxes are shown.	
C1RangeSlider	
C1SplitButton	

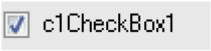
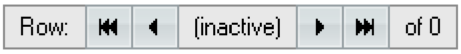
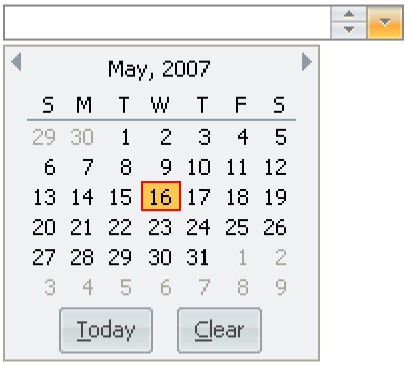
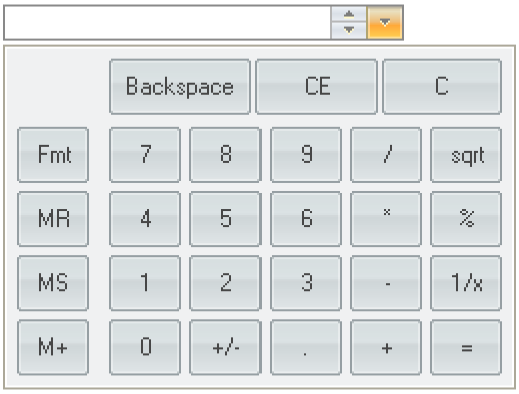
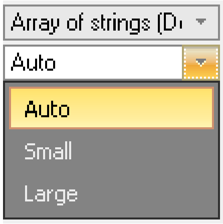
Office2010Blue Visual Style

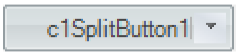
The Office 2010 Blue color scheme.

C1TextBox	
C1CheckBox	
C1Label	
C1DbNavigator	
C1DropDownControl	
C1DateEdit	 May, 2007 S M T W T F S 29 30 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 1 2 3 4 5 6 7 8 9 Today Clear
C1NumericEdit	 Backspace CE C Fmt 7 8 9 / sqrt MR 4 5 6 * % MS 1 2 3 - 1/x M+ 0 +/- . + =
C1Button	
C1ComboBox Note: For illustration purposes two comboboxes are shown.	
C1RangeSlider	
C1SplitButton	

Office2010Black Visual Style

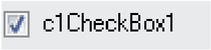
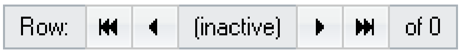
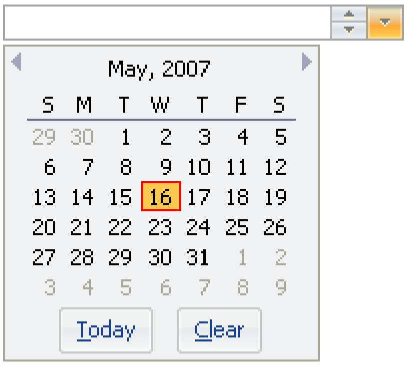
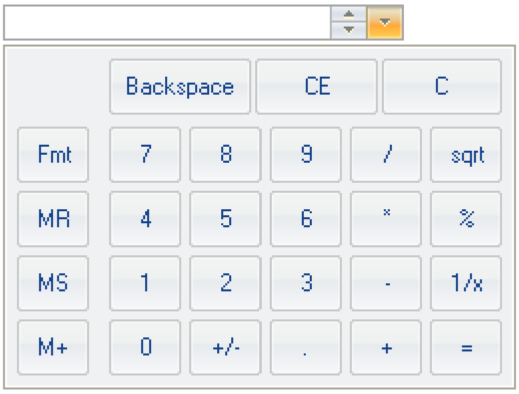
The Office 2010 Black color scheme.

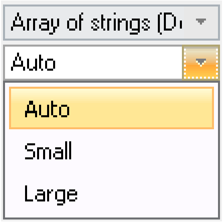
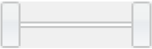
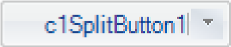
C1TextBox	
C1CheckBox	
C1Label	
C1DbNavigator	
C1DropDownControl	
C1DateEdit	
C1NumericEdit	 Backspace CE C Fmt 7 8 9 / sqrt MR 4 5 6 * % MS 1 2 3 - 1/x M+ 0 +/- . + =
C1Button	
C1ComboBox Note: For illustration purposes two comboboxes are shown.	
C1RangeSlider	

C1SplitButton	

Office2010Silver Visual Style

The Office 2010 Silver color scheme.

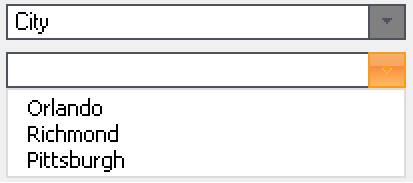
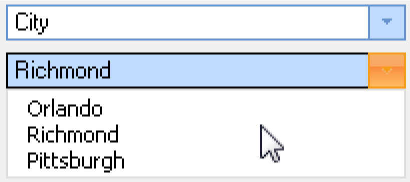
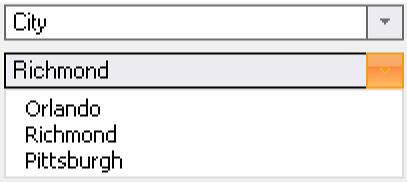
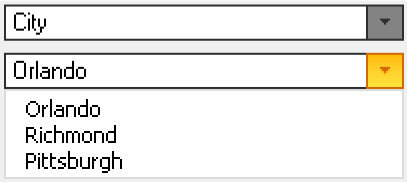
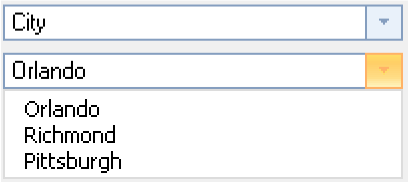
C1TextBox	
C1CheckBox	
C1Label	
C1DbNavigator	
C1DropDownControl	
C1DateEdit	 May, 2007 S M T W T F S 29 30 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 1 2 3 4 5 6 7 8 9 Today Clear
C1NumericEdit	 Backspace CE C Fmt 7 8 9 / sqrt MR 4 5 6 * % MS 1 2 3 - 1/x M+ 0 +/- . + =
C1Button	

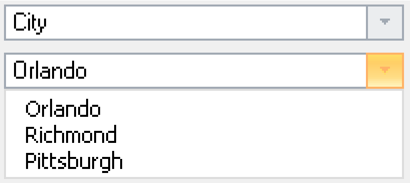
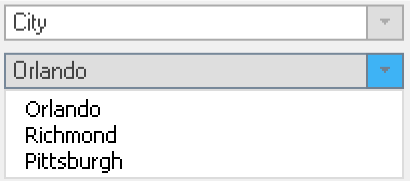
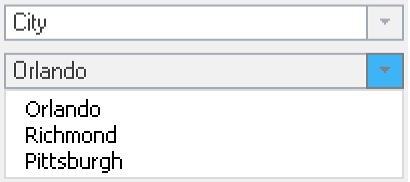
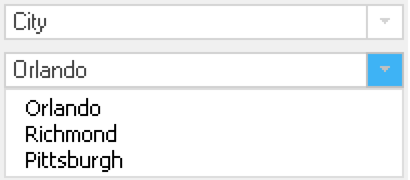
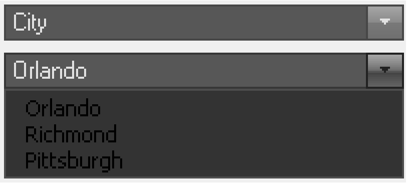
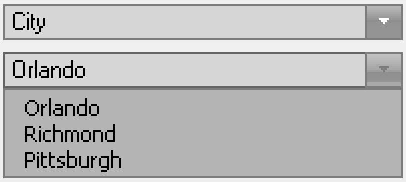
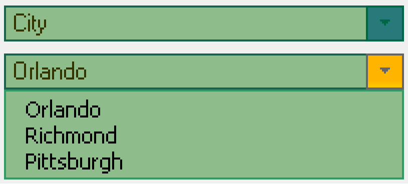
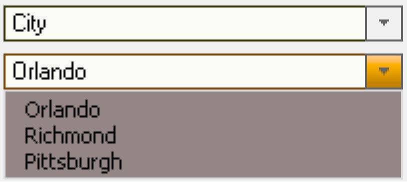
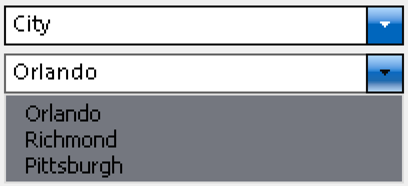
C1ComboBox Note: For illustration purposes two comboboxes are shown.	
C1RangeSlider	
C1SplitButton	

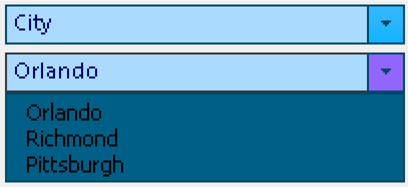
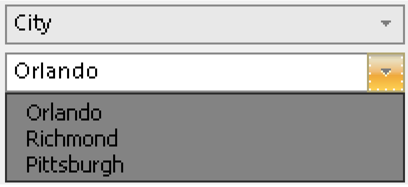
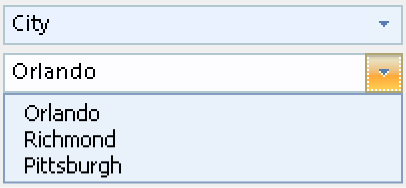
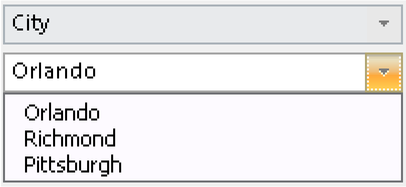
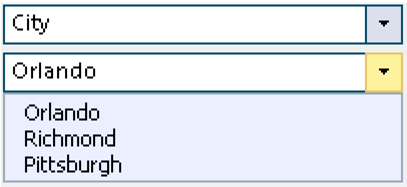
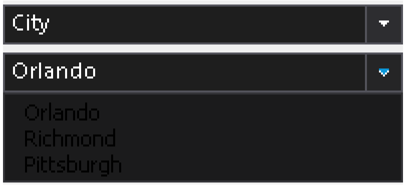
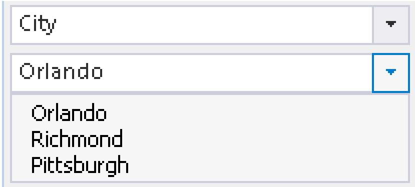
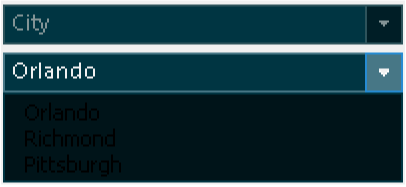
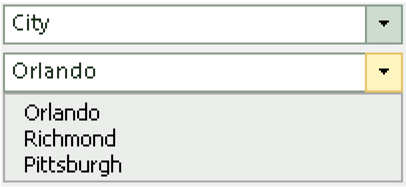
Themes

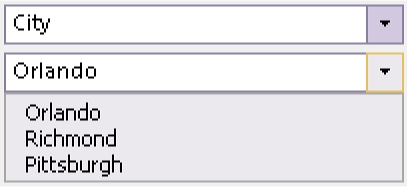
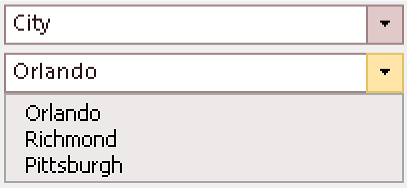
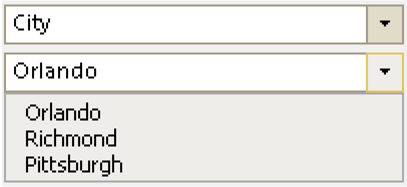
In addition to the Visual Styles you can use the C1ThemeController to apply other themes to the C1Input control. You could also create your own theme using the ThemeDesigner.

To customize the appearance of a C1Input control using Themes, add the **C1ThemeController** to your component tray and set the **Themes** property to any of the following predefined styles listed below:

Theme Name	Image
Office2007Black	
Office2007Blue	
Office2007Silver	
Office2010Black	
Office2010Blue	

Office2010Silver	 City Orlando Orlando Richmond Pittsburgh
Office2013DarkGray	 City Orlando Orlando Richmond Pittsburgh
Office2013LightGray	 City Orlando Orlando Richmond Pittsburgh
Office2013White	 City Orlando Orlando Richmond Pittsburgh
ExpressionDark	 City Orlando Orlando Richmond Pittsburgh
ExpressionLight	 City Orlando Orlando Richmond Pittsburgh
GreenHouse	 City Orlando Orlando Richmond Pittsburgh
RainerOrange	 City Orlando Orlando Richmond Pittsburgh
ShinyBlue	 City Orlando Orlando Richmond Pittsburgh

Violette	 City Orlando Orlando Richmond Pittsburgh
VisualStyleOffice2010Black	 City Orlando Orlando Richmond Pittsburgh
VisualStyleOffice2010Blue	 City Orlando Orlando Richmond Pittsburgh
VisualStyleOffice2010Silver	 City Orlando Orlando Richmond Pittsburgh
VS2013Blue	 City Orlando Orlando Richmond Pittsburgh
VS2013Dark	 City Orlando Orlando Richmond Pittsburgh
VS2013Light	 City Orlando Orlando Richmond Pittsburgh
VS2013DarkSolar	 City Orlando Orlando Richmond Pittsburgh
VS2013Green	 City Orlando Orlando Richmond Pittsburgh

VS2013Purple	
VS2013Red	
VS2013Tan	

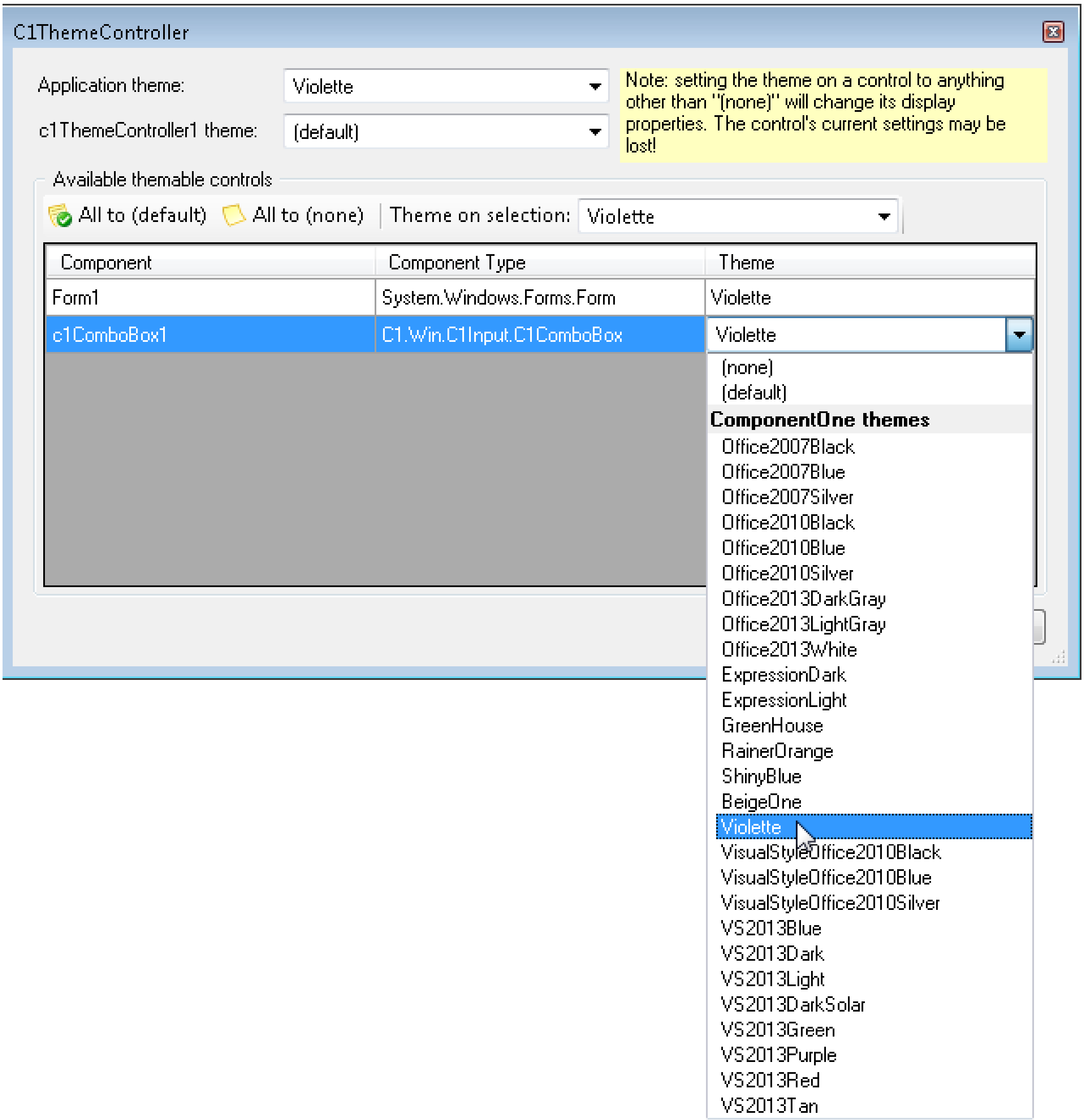
How to apply themes to the C1Input controls at Design Time

To use the C1ThemeController component with any of the C1Input controls, complete the following:

1. Add any of the C1Input controls, for example **C1ComboBox**, on the form at design time.
2. Add the **C1ThemeController** component to your component tray. The **C1ThemeController** dialog box appears.

   > If you use the C1ThemeController 2.0 component a C1ThemeController dialog box appears. The ThemeController dialog box is used to quickly to apply the theme to all themeable controls in the application, all themeable controls on the form, or different themes on different controls.

   The **C1ThemeController** dialog box lists all of the components that appears on your form. If you have themeable controls on your form before you add the C1ThemeController the dialog box lists all of the components on your form. Each control/component is initially set to none to prevent unintentional loss of property settings on those controls.
3. In the **C1ThemeController** dialog box click on the **Theme** dropdown button next to Form1 and c1ComboBox1 and select one of the predefined themes, for example, **Violette**.



C1ThemeController

Application theme: Violette

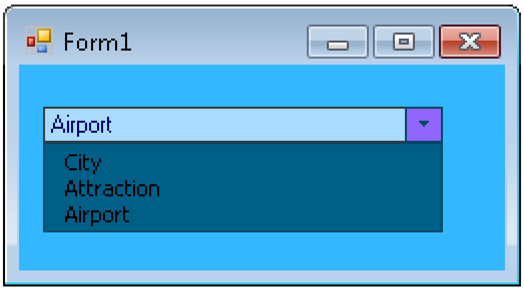
c1ThemeController1 theme: (default)

Note: setting the theme on a control to anything other than "(none)" will change its display properties. The control's current settings may be lost!

Available themable controls

Component	Component Type	Theme
Form1	System.Windows.Forms.Form	Violette
c1ComboBox1	C1.Win.C1Input.C1ComboBox	Violette

4. Click **OK** to save and close the **C1ThemeController** dialog box.
5. Run your project and observe that the **Violette** theme is applied to your **Form** and **C1Combobox**.



How to Apply Themes to the C1Input Controls Programatically

The following code shows how to programatically apply the built-in theme using the **RegisterTheme** and **SetTheme** methods:

To write code in Visual Basic

Visual Basic

```
'Register the theme file with the C1ThemeController
C1.Win.C1Themes.C1ThemeController.RegisterTheme("C:\Users\Documents\Visual Studio 2010\Projects\ThemesProject\ShinyBlue.c1theme")
'Apply it to a control, use the theme name, not the file name
Me.c1ThemeController1.SetTheme(c1ComboBox1, "ShinyBlue")
```

To write code in C#

C#

```
//Register the theme file with the C1ThemeController;

C1.Win.C1Themes.C1ThemeController.RegisterTheme(@"C:\Users\Documents\Visual Studio 2010\Projects\ThemesProject\ShinyBlue.c1theme");

//Apply it to a control and use the theme name
this.c1ThemeController1.SetTheme(c1ComboBox1, "ShinyBlue")
```

In addition to the predefined themes you can customize your theme using the **Themes** designer. For more information see the **Themes for WinForms** documentation.

C1Themes and the VisualStyle Property

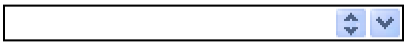
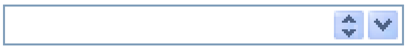
Many WinForms controls include a property called VisualStyle, of an enum type defined in the control assembly, but following a common naming pattern for the type and possible values. Typically, the enum type is called VisualStyle, with values such as Office2010Blue, Office2010Black, and so on. C1Themes are designed to provide a more powerful and flexible mechanism for adjusting the look of controls. Still there is obvious overlap between the two mechanisms, with the possibility of conflicts. The rules for dealing with this overlap are as follows:

- Theme sections for all C1 controls include a VisualStyle property that can be specified by the theme.
- By default and in all supplied themes, those properties are set to 'Custom' so that the VisualStyle does not interfere with applying other properties.
- Setting VisualStyle in a theme to anything other than custom sets the corresponding property on the target control and disables applying all other theme properties. (In the C1ThemeDesigner, this actually disables the rest of the theme tree for the control.)

While we recognize that backward compatibility or other considerations may require the use of VisualStyle rather than themes to customize the look of your application, we recommend that if possible you use C1Themes as they provide a more powerful and flexible mechanism for that. Support for visual styles in new controls will be phased out as themes will replace it.

Border Styles

The following border styles are available for **C1Input:**

Border Style	Preview
None	
FixedSingle	
Fixed3D (Default)	

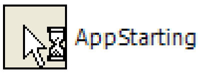
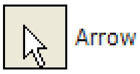
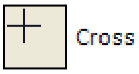
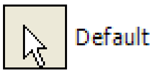
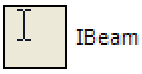
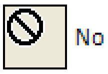
Border Color

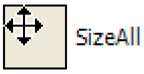
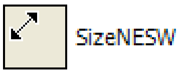
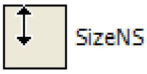
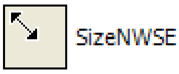
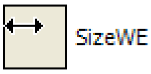
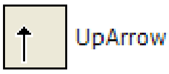
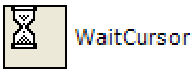
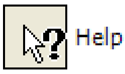
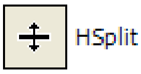
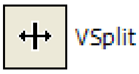
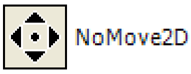
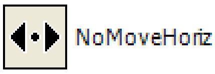
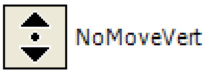
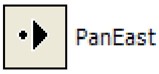
A border color can be applied to **C1DateEdit**, **C1CheckBox**, **C1Label**, **C1NumericEdit**, **C1DropDownControl**, and **C1TextBox** controls when the border style is set to **FixedSingle** and the value of the **BorderColor** property is specified.

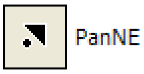
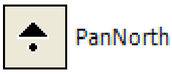
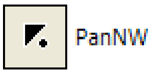
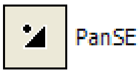
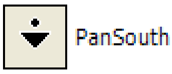
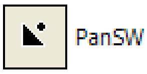
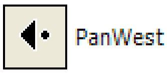
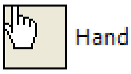
For more information on applying a border color to an applicable C1Input control see Displaying a Border Color for the C1Input controls.

Cursor Styles

You can customize how the cursor appears when the pointer moves over the control by specifying a value for the **Cursor** property. You can also customize how the cursor appears when the mouse is over the buttons on the applicable input controls using the **ButtonCursor** property. The cursor styles are applicable to all C1Input controls. The button cursor styles are applicable to the **C1DropDownControl**, **C1DbNavigator**, **C1DateEdit**, and **C1NumericEdit**. The cursor and button cursor styles appear like the following:

Button Cursor Style	Preview
AppStarting	
Arrow	
Cross	
Default	
IBeam	
No	

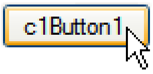
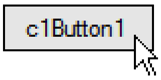
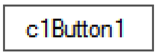
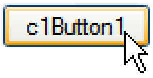
SizeAll	 SizeAll
SizeNESW	 SizeNESW
SizeNS	 SizeNS
SizeNWSE	 SizeNWSE
SizeWE	 SizeWE
UpArrow	 UpArrow
WaitCursor	 WaitCursor
Help	 Help
HSplit	 HSplit
VSplit	 VSplit
NoMove2D	 NoMove2D
NoMoveHoriz	 NoMoveHoriz
NoMoveVert	 NoMoveVert
PanEast	 PanEast
PanNE	

	 PanNE
PanNorth	 PanNorth
PanNW	 PanNW
PanSE	 PanSE
PanSouth	 PanSouth
PanSW	 PanSW
PanWest	 PanWest
Hand	 Hand

Flat Styles

C1Button provides different flat styles to choose from when you move the mouse over the button control and click it.

The **ButtonBase.FlatStyle** property includes the following values:

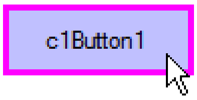
Flat Style	Preview
Standard	 c1Button1
Flat	 c1Button1
Popup	 c1Button1
System	 c1Button1

When the **ButtonBase.**FlatStyle property is set to "Flat" you can modify the border color, border thickness, hover backcolor, and mouse down backcolor using the **ButtonBase.FlatAppearance** property. The

ButtonBase.FlatAppearance property provides the following properties:

- **BorderColor** – Specifies the color of the border around the button.
- **BorderSize** – Specifies the size, in pixels, of the border around the button.
- **MouseDownBackColor** – Specifies the color of the client area of the button when the mouse is pressed within the bounds of the control.
- **MouseOverBackColor** – Specifies the color of the client area of the button when the mouse pointer is within the bounds of the control.

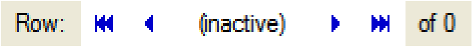
The following image represents a flat style C1Button with its **BorderColor**, **BorderSize**, and **MouseOverBackColor** properties set.



Button Color

In the **C1DBNavigator** control you can specify whether or not to show blue buttons using the ColorButtons property. You can also specify whether or not to show the colored buttons when hovering over the buttons using the ColorWhenHover property.

The following image shows how the colored buttons appear on the C1DBNavigator when the ColorButtons property is set to true.



Input for WinForms Task-Based Help

The task-based help section assumes that you are familiar with programming in the Visual Studio .NET environment, and know how to use **C1Input** controls in general. If you are a novice to the **Input for WinForms** product, please see the Input for WinFormsTutorials first.

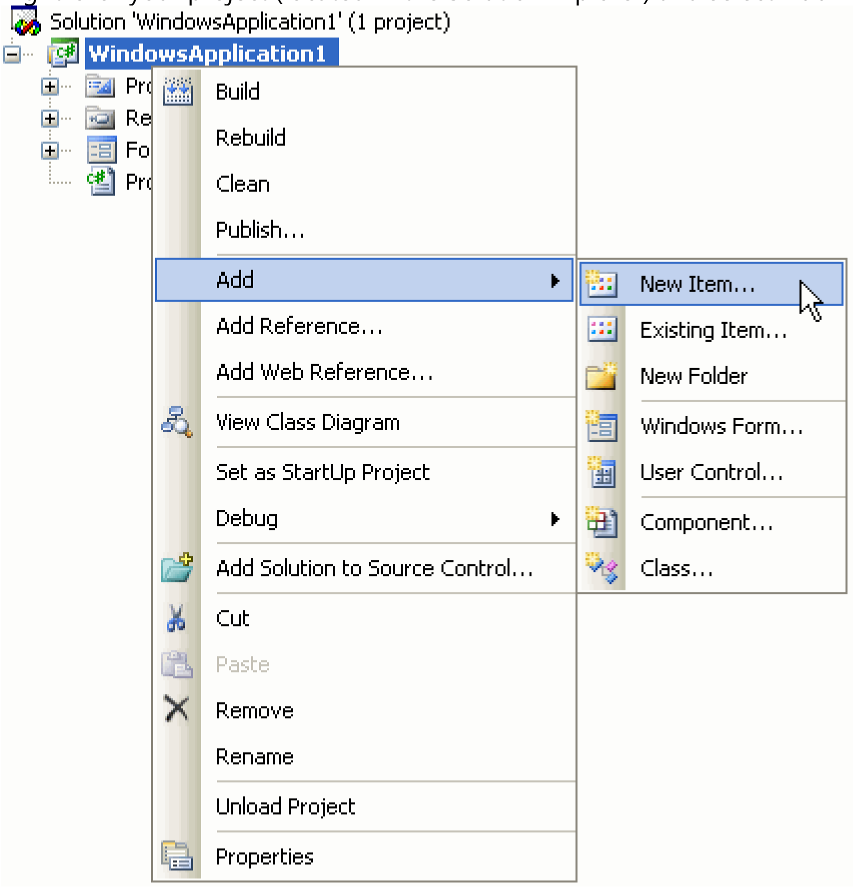
Each topic provides a solution for specific tasks using the **Input for WinForms** product. By following the steps outlined in each topic, you will be able to create projects using a variety of **Input for WinForms** features.

Each task-based help topic also assumes that you have created a new .NET project.

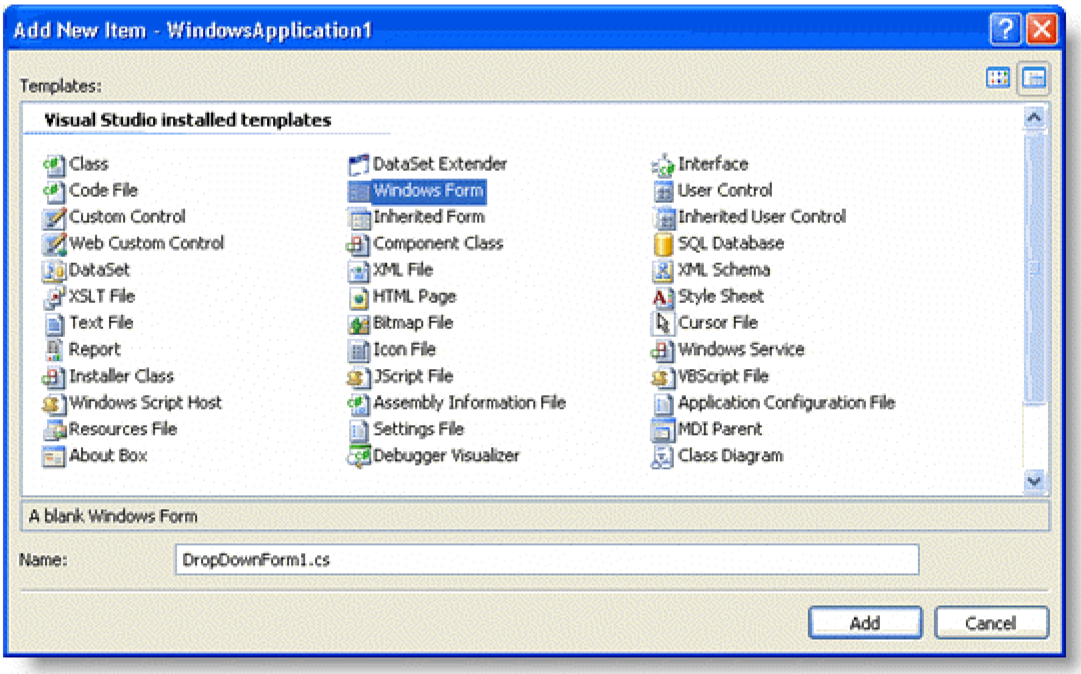
Adding a Drop-Down Form

To add a drop-down form to your project, complete the following steps:

1. Right-click your project (located in the Solution Explorer) and select **Add New Item** from the **Add** sub-menu.



2. In the **Add New Item** dialog box, select **Windows Form** from the list of **Templates** in the right pane. Then enter **DropDownForm1.cs** in the **Name** textbox.



A blank Windows Form

Name: DropDownForm1.cs

3. The next step is to replace the following class definition line(s) in the DropDownForm code:

To write code in Visual Basic

Visual Basic

```
Public Class DropDownForm1
    Inherits System.Windows.Forms.Form
```

To write code in C#

C#

```
Public Class DropDownForm1:System.Windows.Forms.Form
```

with:

To write code in Visual Basic

Visual Basic

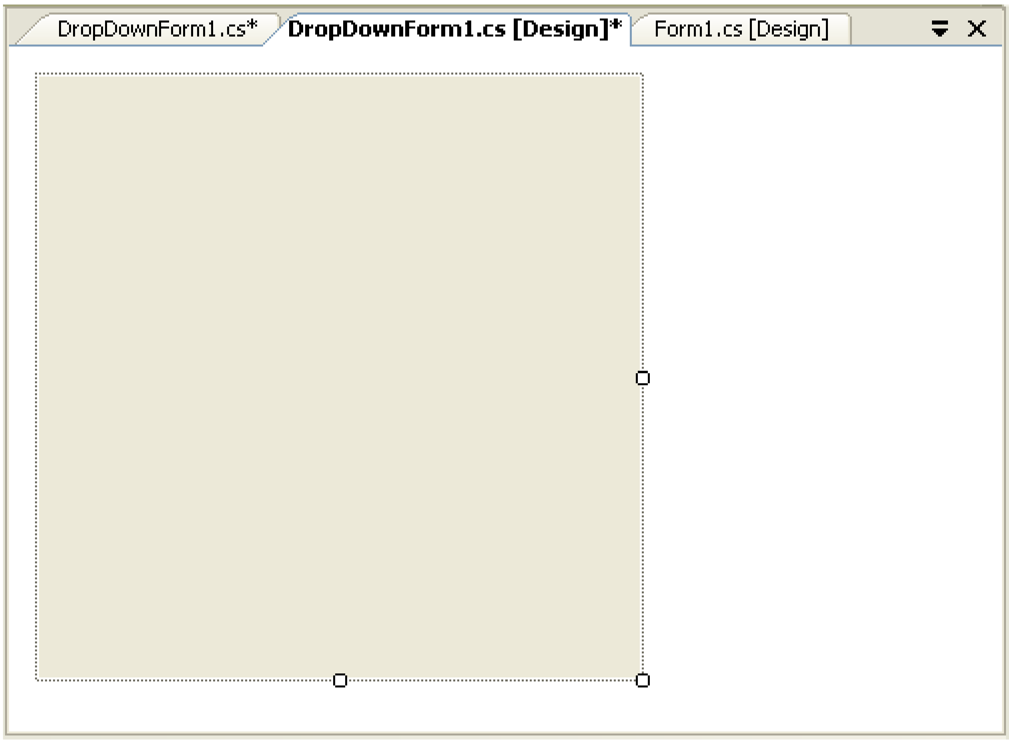
```
Public Class DropDownForm1
    Inherits C1.Win.C1Input.DropDownForm
```

To write code in C#

C#

```
Public Class DropDownForm1: C1.Win.C1Input.DropDownForm
```

The form should look like the image below before editing:



Changing the Navigation in the Navigator

To change the navigation in the Navigator, change Index like the following:

To write code in Visual Basic

Visual Basic

```
Private Sub c1DbNavigator1_BeforeAction(sender As Object, e As
C1.Win.C1Input.NavigatorBeforeActionEventArgs)
       If e.Button = C1.Win.C1Input.NavigatorButtonEnum.First Then
             ' Goto second record instead of the first
              e.Index = 1
       End If

        ' Go to the last row if user entered too large position
       If e.Button = C1.Win.C1Input.NavigatorButtonEnum.Position AndAlso e.Cancel
Then
              e.Cancel = False
       End If
End Sub
```

To write code in C#

C#

```
private void c1DbNavigator1_BeforeAction(object sender,
C1.Win.C1Input.NavigatorBeforeActionEventArgs e)
    {
              if (e.Button == C1.Win.C1Input.NavigatorButtonEnum.First)
        {
```

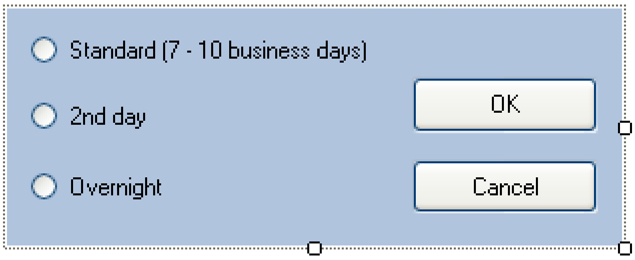
```
                  // Goto second record instead of the first
            e.Index = 1;
        }

               // Go to the last row if user entered too large position
        if (e.Button == C1.Win.C1Input.NavigatorButtonEnum.Position && e.Cancel)
        {
            e.Cancel = false;
        }
    }
```

Customizing the Drop-Down Editor

The drop-down form below includes option buttons and button controls for the user to make a selection from the C1DropDownControl.

The drop-down form appearance properties have been edited so that the form appears as below:



Select the form class name (for this example, WindowsApplication1.DropDownForm1) in the DropDownFormClassName property of your C1DropDownControl. Notice that when you run the project and select the drop-down arrow, the drop-down form now appears.

Enable the button controls on the drop-down form:

1. Set the **AcceptButton** and **CancelButton** properties of your DropDownForm1 to **button1** and **button2**, respectively.
2. Select the **OK** button and set its **DialogResult** property to **OK**. Similarly, select the **Cancel** button and set its **DialogResult** property to **Cancel**.
3. To make the drop-down form change the control text when it is closed after the user clicks an item, add the following event handler for the PostChanges event:

   To write code in Visual Basic

   Visual Basic

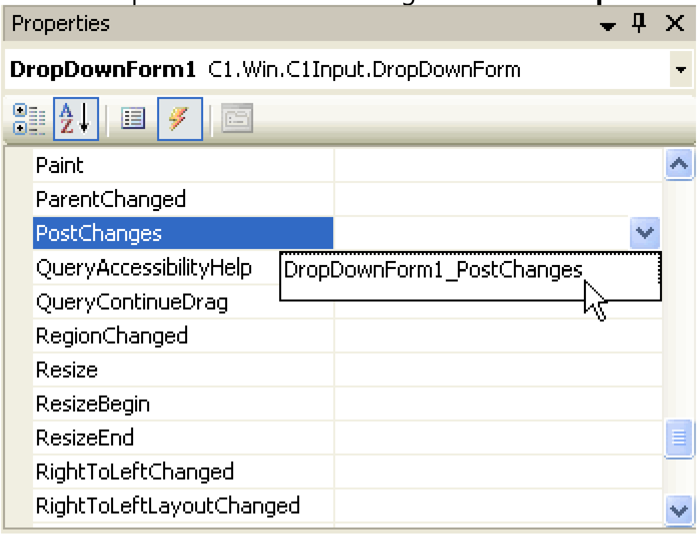
```
Private Sub DropDownForm1_PostChanges(sender As Object, e As System.EventArgs)
    If (MyBase.DialogResult = DialogResult.OK) Then
        Dim control1 As Control
        For Each control1 In MyBase.Controls
            If (TypeOf control1 is RadioButton AndAlso CType(control1,
RadioButton).Checked) Then
                MyBase.OwnerControl.Value = CType(control1, RadioButton).Text
            End If
        Next
    End If
End Sub
```

To write code in C#

C#

```
private void DropDownForm1_PostChanges(object sender, System.EventArgs e)
{
    if (DialogResult == DialogResult.OK)
    {
        foreach (Control control1 in Controls)
        {
            if (control1 as RadioButton != null &&
((RadioButton)control1).Checked)
            {
                OwnerControl.Value = ((RadioButton)control1).Text;
            }
        }
    }
}
```

4. At design time, select DropDownForm1 to view its properties in the Properties window, and then select the **Events** button ⚡ from the Properties toolbar.
5. Set the DropDownForm1.PostChanges event to **DropDownForm1_PostChanges**.



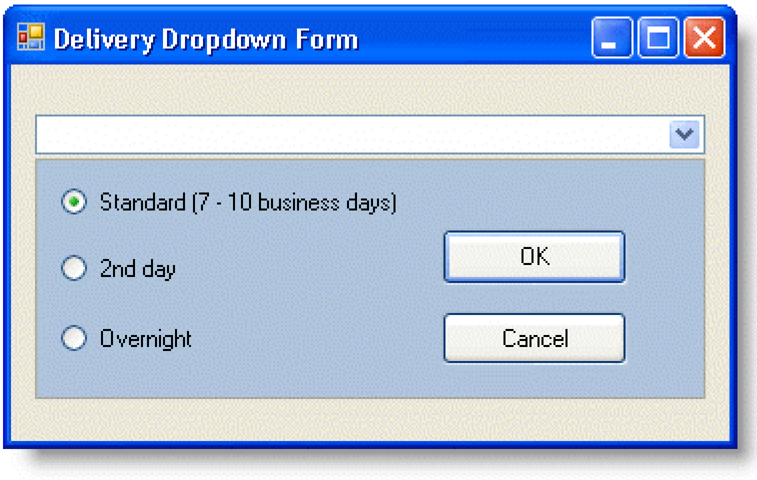
DropDownForm1 C1.Win.C1Input.DropDownForm

Paint
ParentChanged
PostChanges — DropDownForm1_PostChanges
QueryAccessibilityHelp
QueryContinueDrag
RegionChanged
Resize
ResizeBegin
ResizeEnd
RightToLeftChanged
RightToLeftLayoutChanged

6. To make the **OK** button (button1) receive focus when the form opens, set the DropDownForm1.FocusControl property to **button1**.
7. To have a check in the **Standard** option button, in design time select **radiobutton1** and set its **Checked** property to **True**.

This topic illustrates the following:

Your form should appear similar to the form below:

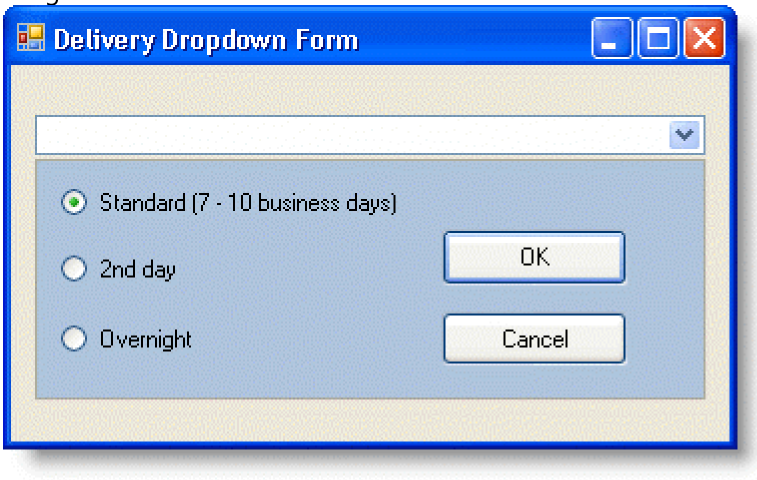


Customizing the C1DropDownControl

This topic shows how you can customize the **C1Input.C1DropDownControl**.

To make only the drop-down button visible:

1. Expand the VisibleButtons property node.
2. Set **UpDown** to **False**. Note that the **DropDown** default is set to **True**. The control should now look like the image below:



To make the width of the drop-down form equal to the width of the control:

1. Select the drop-down form.
2. Set Options.**AutoResize** to **True**.

Binding C1CheckBox

The following topics show how to bind C1CheckBox to a Boolean, String, and Integer field.

Binding C1CheckBox to a Boolean Field

To programmatically bind **C1CheckBox** to a Boolean field, use the following code:

To write code in Visual Basic

Visual Basic

```
C1CheckBox1.DataSource = dt
C1CheckBox1.DataField = "ColumnBoolean"
```

To write code in C#

C#

```
C1CheckBox1.DataSource = dt;
C1CheckBox1.DataField = "ColumnBoolean";
```

Binding C1CheckBox to a String Field

To programmatically bind **C1CheckBox** to a String field, use the following code:

To write code in Visual Basic

Visual Basic

```
c1CheckBox1.DataSource = dt
c1CheckBox1.DataField = "ColumnString"
c1CheckBox1.DataType = GetType(String)
' Use TranslateValues property to translate string values to/from the check box
states.
c1CheckBox1.TranslateValues.Checked = "Yes"
c1CheckBox1.TranslateValues.Unchecked = "No"
```

To write code in C#

C#

```
c1CheckBox1.DataSource = dt;
c1CheckBox1.DataField = "ColumnString";
c1CheckBox1.DataType = typeof(string);
// Use TranslateValues property to translate string values to/from the check box
states.
c1CheckBox1.TranslateValues.Checked = "Yes";
c1CheckBox1.TranslateValues.Unchecked = "No";
```

Binding C1CheckBox to an Integer Field

To programmatically bind **C1CheckBox** to an Integer field, use the following code:

To write code in Visual Basic

Visual Basic

```
c1CheckBox1.DataSource = dt
c1CheckBox1.DataField = "ColumnInt"
c1CheckBox1.DataType = GetType(Integer)
‘Use TranslateValues property to translate string values to/from the check box
states.
```

```
c1CheckBox1.TranslateValues.Checked = 1
c1CheckBox1.TranslateValues.Unchecked = 0
```

To write code in C#

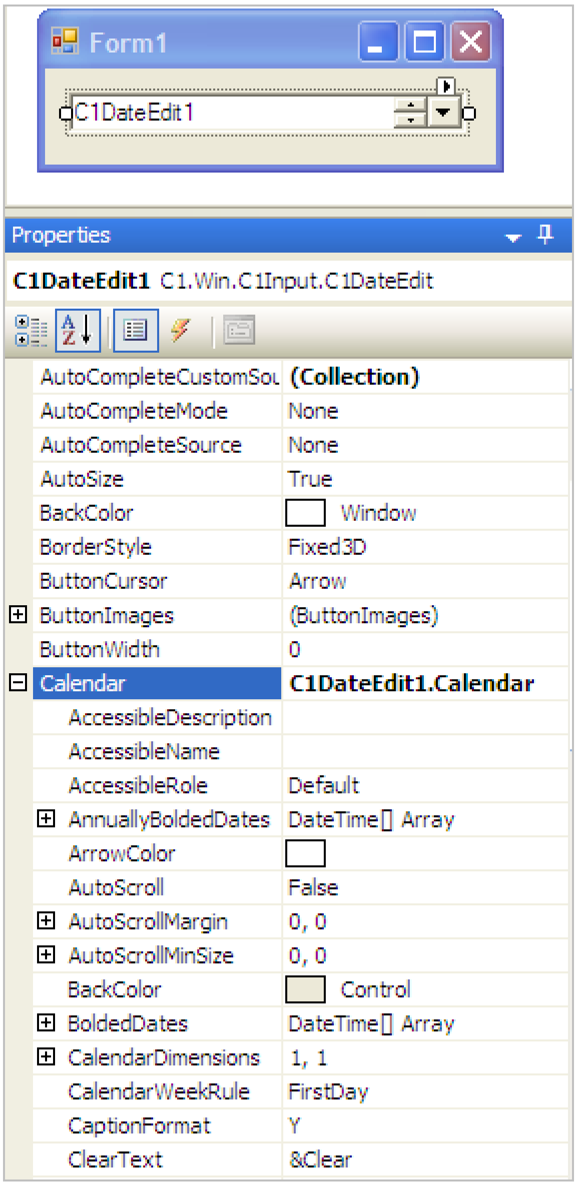
```
C#
c1CheckBox1.DataSource = dt;
c1CheckBox1.DataField = "ColumnInt";
c1CheckBox1.DataType = typeof(int);
// Use TranslateValues property to translate string values to/from the check box
states.
c1CheckBox1.TranslateValues.Checked = 1;
c1CheckBox1.TranslateValues.Unchecked = 0;
```

Setting the Calendar Drop-down

In previous versions of **Input for WinForms**, the Calendar feature in the **C1DateEdit** control allowed you to set the "Today" and "Clear" buttons by manipulating the **Calendar.UIString** property. In newer versions of the C1Input.DateEdit control, you can set these buttons by accessing the C1DateEdit properties menu.

To set the "Today" and "Clear" buttons:

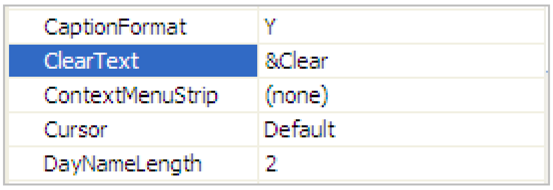
1. Add the **C1DateEdit** control to your form.
2. Select **C1.DateEdit1** from the Properties menu.
3. Locate **Calendar** in the left column and expand the **Calendar** property.



C1DateEdit1 C1.Win.C1Input.C1DateEdit

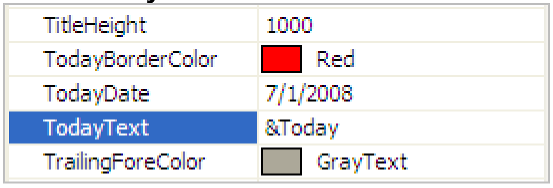
AutoCompleteCustomSou	**(Collection)**
AutoCompleteMode	None
AutoCompleteSource	None
AutoSize	True
BackColor	□ Window
BorderStyle	Fixed3D
ButtonCursor	Arrow
⊞ ButtonImages	(ButtonImages)
ButtonWidth	0
⊟ Calendar	**C1DateEdit1.Calendar**
AccessibleDescription	
AccessibleName	
AccessibleRole	Default
⊞ AnnuallyBoldedDates	DateTime[] Array
ArrowColor	□
AutoScroll	False
⊞ AutoScrollMargin	0, 0
⊞ AutoScrollMinSize	0, 0
BackColor	□ Control
⊞ BoldedDates	DateTime[] Array
⊞ CalendarDimensions	1, 1
CalendarWeekRule	FirstDay
CaptionFormat	Y
ClearText	&Clear

4. Locate **ClearText** in the left column and enter "&Reset" in the right column.



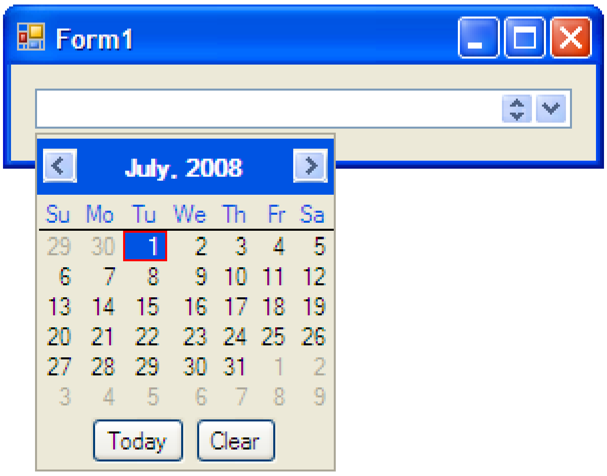
CaptionFormat	Y
ClearText	&Clear
ContextMenuStrip	(none)
Cursor	Default
DayNameLength	2

5. Locate **TodayText** in the left column and enter "&Now" in the right column.



TitleHeight	1000
TodayBorderColor	🟥 Red
TodayDate	7/1/2008
TodayText	&Today
TrailingForeColor	▨ GrayText

6. Press **F5** to compile and run the project.



Now when you open the C1DateEdit dropdown menu, the "Today" and "Clear" buttons are set.

Customizing Appearance Using Visual Styles

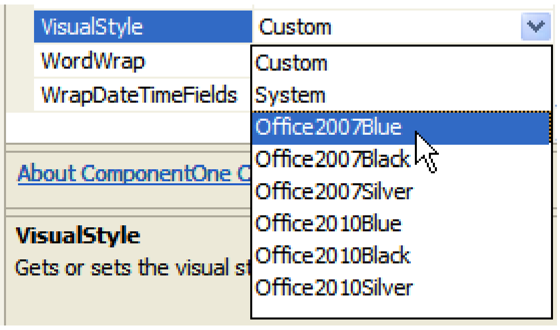
Setting the **VisualStyle** property on a **C1Input** control will control the gradients and borders used to paint C1TextBox, C1Label, C1DbNavigator, C1DropDownControl, C1DateEdit (including the drop down calendar), C1NumericEdit (including the drop down calculator), and C1Button.

To customize the appearance of a C1Input control using Visual Styles, set the **VisualStyle** property to **Custom**, **Office2007Black**, **Office2007Blue**, **Office2007Silver**, **System**, **Office2010Blue**, **Office2010Black**, or **Office2010Silver**. This property can be set either in the designer or in code. The following table describes each of the Visual Styles:

Visual Style	Description
Custom	No visual style (use styles and appearance properties as usual).
Office2007Black	Style matches Office2007 Black color scheme.
Office2007Blue	Style matches Office2007 Blue color scheme.
Office2007Silver	Style matches Office2007 Silver color scheme.
System	Style matches the current system settings.
Office2010Blue	Style matches Office2010 Blue color scheme.
Office2010Black	Style matches Office2010 Black color scheme.
Office2010Silver	Style matches Office2010 Silver color scheme.

Using the Designer

Locate the **VisualStyle** property in the Properties window and set it to **Custom**, **Office2007Black**, **Office2007Blue**, **Office2007Silver**, **System**, **Office2010Blue**, **Office2010Black**, or **Office2010Silver**. In this example, the **VisualStyle** property is set to **Office2007Blue** for a C1TextBox control.



Using the Code Editor

Add code to the **Form_Load** event to set the **VisualStyle** property to **Custom**, **Office2007Black**, **Office2007Blue**, **Office2007Silver**, or **System**. The following code sets the **VisualStyle** property to **Office2007Blue** for a C1TextBox control:

To write code in Visual Basic

Visual Basic

```
Me.C1TextBox1.VisualStyle = C1.Win.C1Input.VisualStyle.Office2007Blue
```

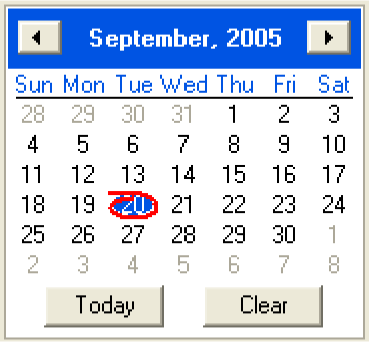
To write code in C#

C#

```
Me.C1TextBox1.VisualStyle = C1.Win.C1Input.VisualStyle.Office2007Blue
```

Customizing the C1DateEdit Control

This topic demonstrates how to customize the drop-down of a C1DateEdit control. By default, the drop-down appears like the image below:



Button Visibility

You can hide the **Clear** and **Today** buttons by performing the following tasks:

1. Select the **C1DateEdit** control.
2. In the Properties window, expand the **Calendar** property node.
3. Set the ShowClearButton and ShowTodayButton properties to **False**.

Format Display

The FormatType property allows you to edit the date displayed in the box.

By default, the date and time are displayed in the box. To only show the date, perform the following task:

1. Select the **C1DateEdit** control.
2. In the Properties window, set the FormatType property to **ShortDate**.

Displaying Clicked C1DropDown Buttons in a Text Box

To display clicked C1DropDownControl buttons in a text box, use the **C1DropDownControl.UpDownButtonClick** event. In this example, the output will be displayed in a textbox (TextBox1).

To write code in Visual Basic

Visual Basic

```
Private Sub C1DropDownControl1_UpDownButtonClick(ByVal sender As Object, ByVal e As
C1.Win.C1Input.UpDownButtonClickEventArgs) Handles
C1DropDownControl1.UpDownButtonClick
    If (e.Delta = 1) Then
        Me.TextBox1.AppendText("Up " & ControlChars.CrLf)
    ElseIf (e.Delta = -1) Then
        Me.TextBox1.AppendText("Down " & ControlChars.CrLf)
    End If
End Sub
```

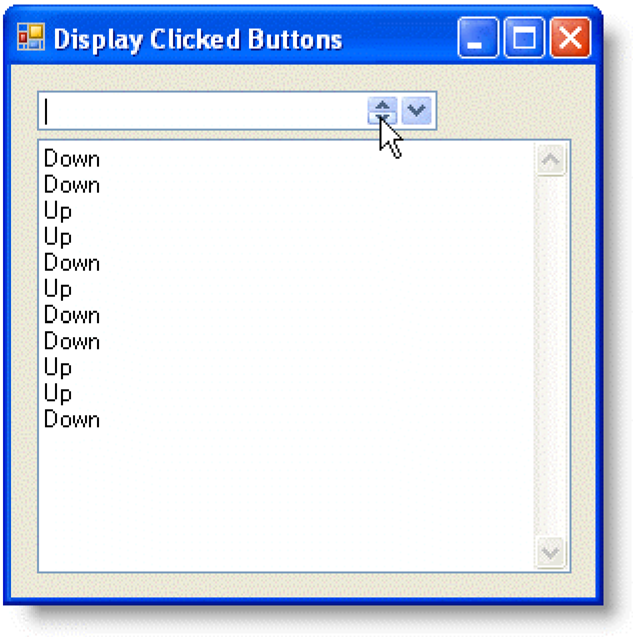
To write code in C#

C#

```
private void c1DropDownControl1_UpDownButtonClick(object sender,
C1.Win.C1Input.UpDownButtonClickEventArgs e)
{
    if ((e.Delta == 1))
    {
        this.textBox1.AppendText("Up\r\n");
    }
    else if ((e.Delta == -1))
    {
        this.textBox1.AppendText("Down\r\n");
    }
}
```

This topic illustrates the following:

When the **Up** or **Down** buttons are clicked in the **C1DropDownControl**, the words Up or Down will appear in a textbox to indicated which button was pressed.



Down
Down
Up
Up
Down
Up
Down
Down
Up
Up
Down

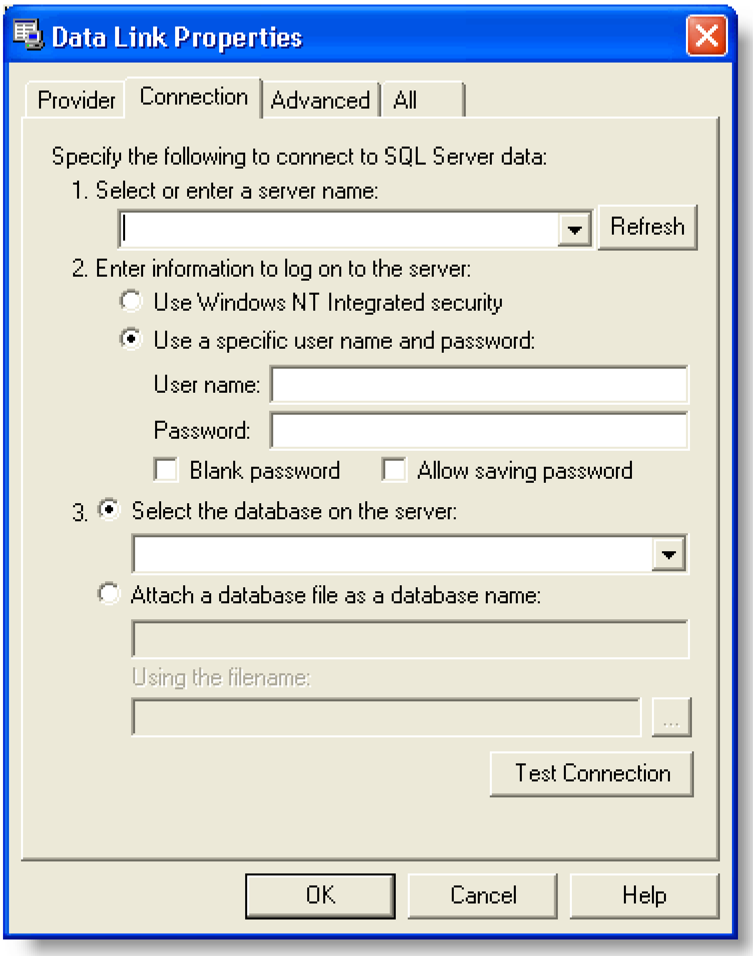
Working with a Database

The following topics demonstrate how to connect to a database and utilize **Input for WinForms** features once connected to a database.

Creating a New Connection

To create a new connection, complete the following steps:

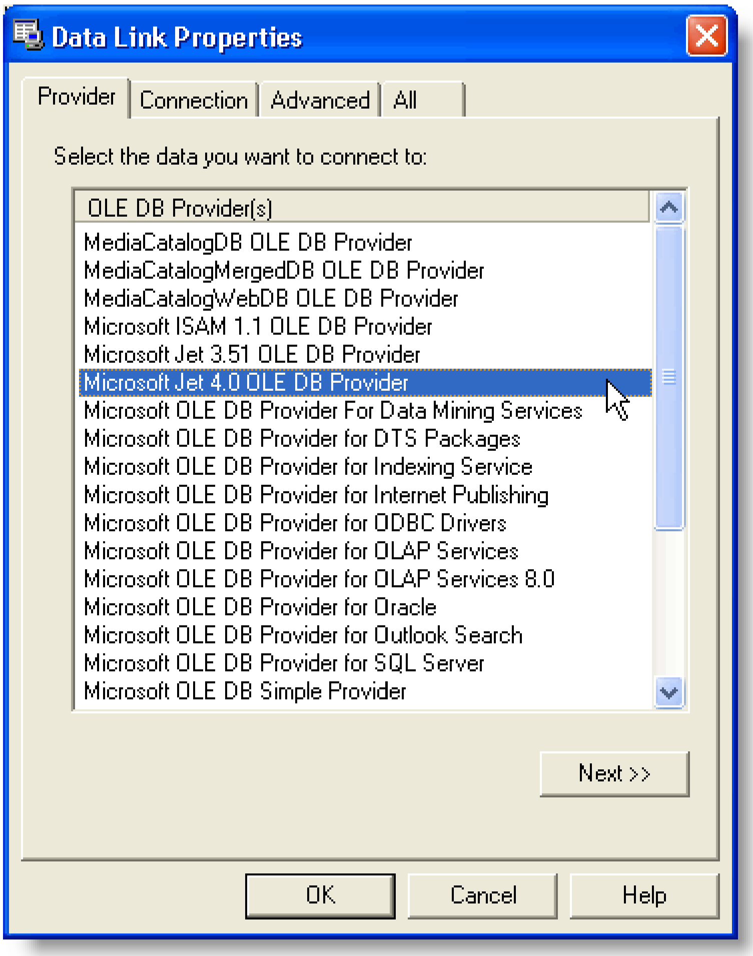
1. Select the C1ExpressConnection1.**ConnectionString** property from the Properties window and select **New Connection** from the drop-down box. The standard OLE DB **Data Link Properties** dialog box appears:


Data Link Properties
Provider
Connection
Advanced
All
Specify the following to connect to SQL Server data:
1. Select or enter a server name:
Refresh
2. Enter information to log on to the server:
○ Use Windows NT Integrated security
● Use a specific user name and password:
User name:
Password:
□ Blank password
□ Allow saving password
3. ● Select the database on the server:
○ Attach a database file as a database name:
Using the filename:
...
Test Connection
OK
Cancel
Help

2. Select the provider, the database and other necessary connection properties in the dialog box.

   In these tutorials, we use the standard MS Access Northwind sample database (C1NWind.mdb).

3. Select **Microsoft Jet 4.0 OLE DB Provider** in the **Provider** tab. Once you have chosen the data you can select the **Next** button.



Select the data you want to connect to:

OLE DB Provider(s)
MediaCatalogDB OLE DB Provider
MediaCatalogMergedDB OLE DB Provider
MediaCatalogWebDB OLE DB Provider
Microsoft ISAM 1.1 OLE DB Provider
Microsoft Jet 3.51 OLE DB Provider
Microsoft Jet 4.0 OLE DB Provider
Microsoft OLE DB Provider For Data Mining Services
Microsoft OLE DB Provider for DTS Packages
Microsoft OLE DB Provider for Indexing Service
Microsoft OLE DB Provider for Internet Publishing
Microsoft OLE DB Provider for ODBC Drivers
Microsoft OLE DB Provider for OLAP Services
Microsoft OLE DB Provider for OLAP Services 8.0
Microsoft OLE DB Provider for Oracle
Microsoft OLE DB Provider for Outlook Search
Microsoft OLE DB Provider for SQL Server
Microsoft OLE DB Simple Provider

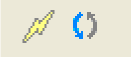
4. In the **Connection** tab, click on the **ellipsis** button to add the C1NWind.mdb database to the ConnectionString. The **Select Access Database** dialog box will appear. Enter the path, **Documents\ComponentOne Samples\Common\C1NWind.mdb** (or the location of your database) for the C1NWind.mdb database. Press **OK**

   That will close the **Data Link Properties** dialog box and put the connection string in the **ConnectionString** property.

Updating /Refreshing Data from the Database

This topic demonstrates how you can use the **Update/Refresh** button to send changes and re-fetch data from the database. Complete the following steps:

1. Add a C1DbNavigator control to your form and select the control to view its Properties window.
2. Expand the VisibleButtons property node and set **Update** and **Refresh** to **True**. Set the default navigation buttons (**Next**, **Previous**, **First**, **Last**) to **False**.
3. Set the PositionVisible property to **False**. As an option, you can set its ColorButtons property to **True** for the buttons to have color bitmaps:



4. To make the **Update** and **Refresh** buttons functional, add a **Click** event handler to the C1DbNavigator component. Enter the following code:

To write code in Visual Basic

Visual Basic

```
Private Sub c1DbNavigator1_UpdateData(sender As Object, e As System.EventArgs)
   c1ExpressConnection1.Update()
End Sub

Private Sub c1DbNavigator1_RefreshData(sender As Object, e As System.EventArgs)
   c1ExpressConnection1.Fill()
End Sub
```

To write code in C#

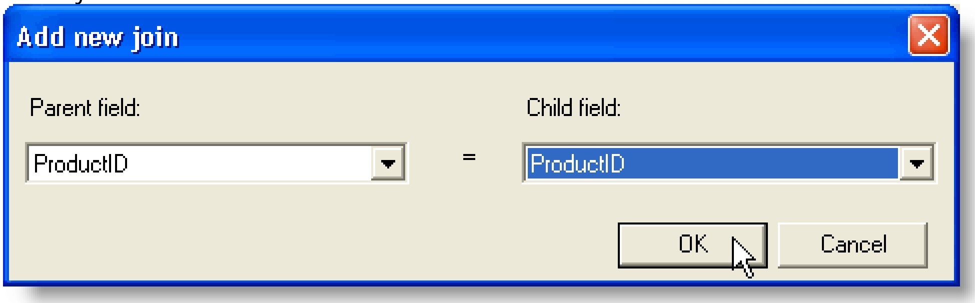
```
C#
private void c1DbNavigator1_UpdateData(object sender, System.EventArgs e)
{
    c1ExpressConnection1.Update();
}

private void c1DbNavigator1_RefreshData(object sender, System.EventArgs e)
{
    c1ExpressConnection1.Fill();
}
```

Creating a Master-Detail Relation

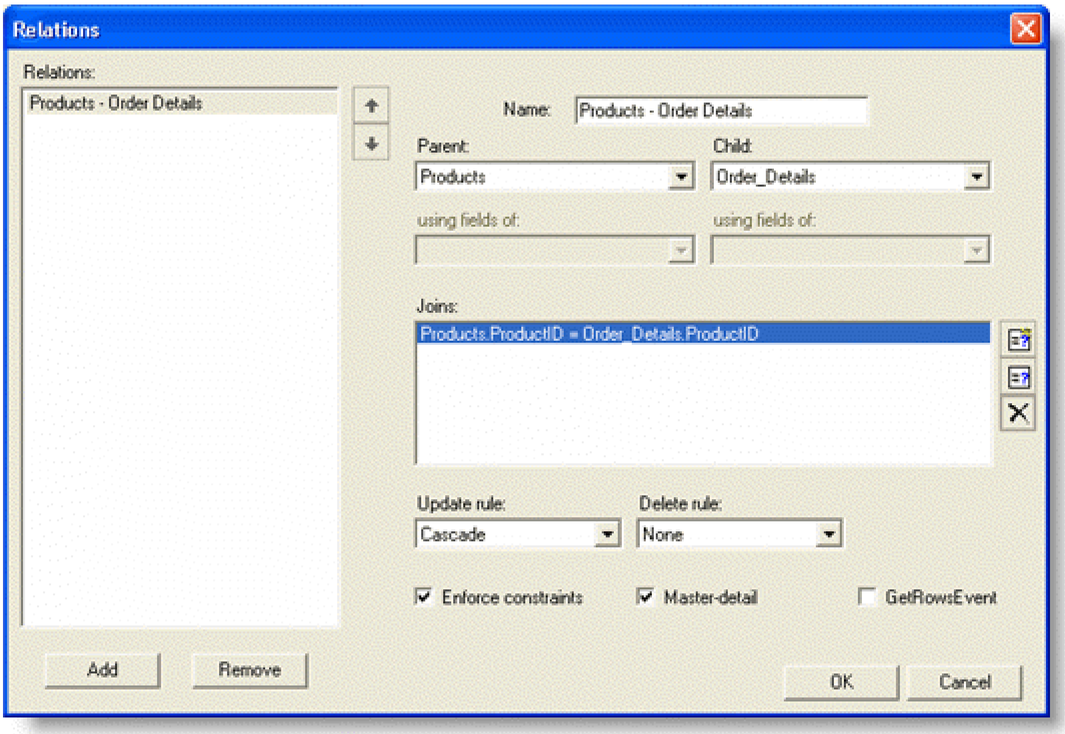
This topic shows how to create a master-detail relation between tables, C1ExpressTable1 (master) and C1ExpressTable2 (detail).

1. Select the C1ExpressionConnection1 component to view the Properties window. Press the **ellipsis** button in the **Relations** property of the C1ExpressionConnection1 control to open the **Relations** dialog box.
2. In the **Relations** dialog box, select **Products** for the Parent field, **Order Details** for the Child field, and then select the **Add join** button to open the **Add new join** dialog box.
3. Add a join with **ProductID** for both the Parent field and the Child field. Press **OK** to close the dialog box.



Parent field: ProductID = Child field: ProductID

The join should appear as below:

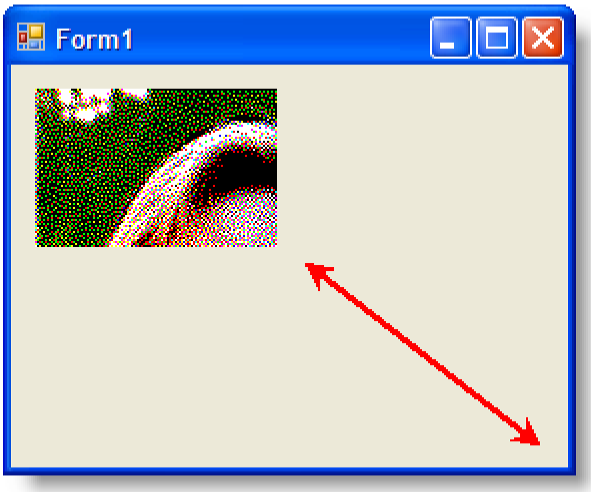


Relations:
Products - Order Details
Name: Products - Order Details
Parent: Products
Child: Order_Details
using fields of:
using fields of:
Joins:
Products.ProductID = Order_Details.ProductID
Update rule: Cascade
Delete rule: None
☑ Enforce constraints ☑ Master-detail □ GetRowsEvent

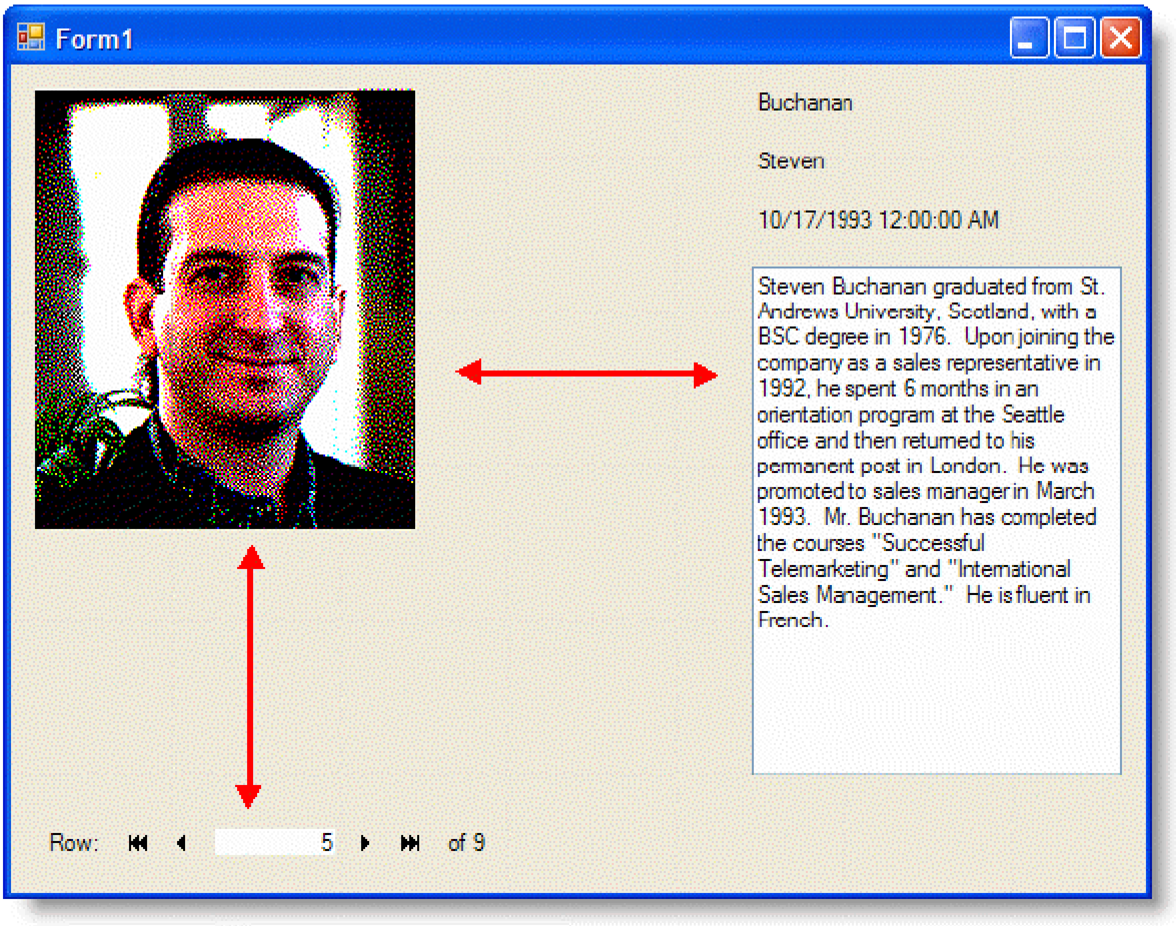
4. Press **OK** to close the **Relations** dialog box.

Customizing the PictureBox

Like most controls, you can easily increase or decrease the size of the C1.Input.PictureBox either using your mouse or through the C1.Input.PictureBox.Size property. But, depending on the dimensions of the picture files, an image might be cut off or might leave a large blank space showing within the PictureBox:



Or...



Buchanan

Steven

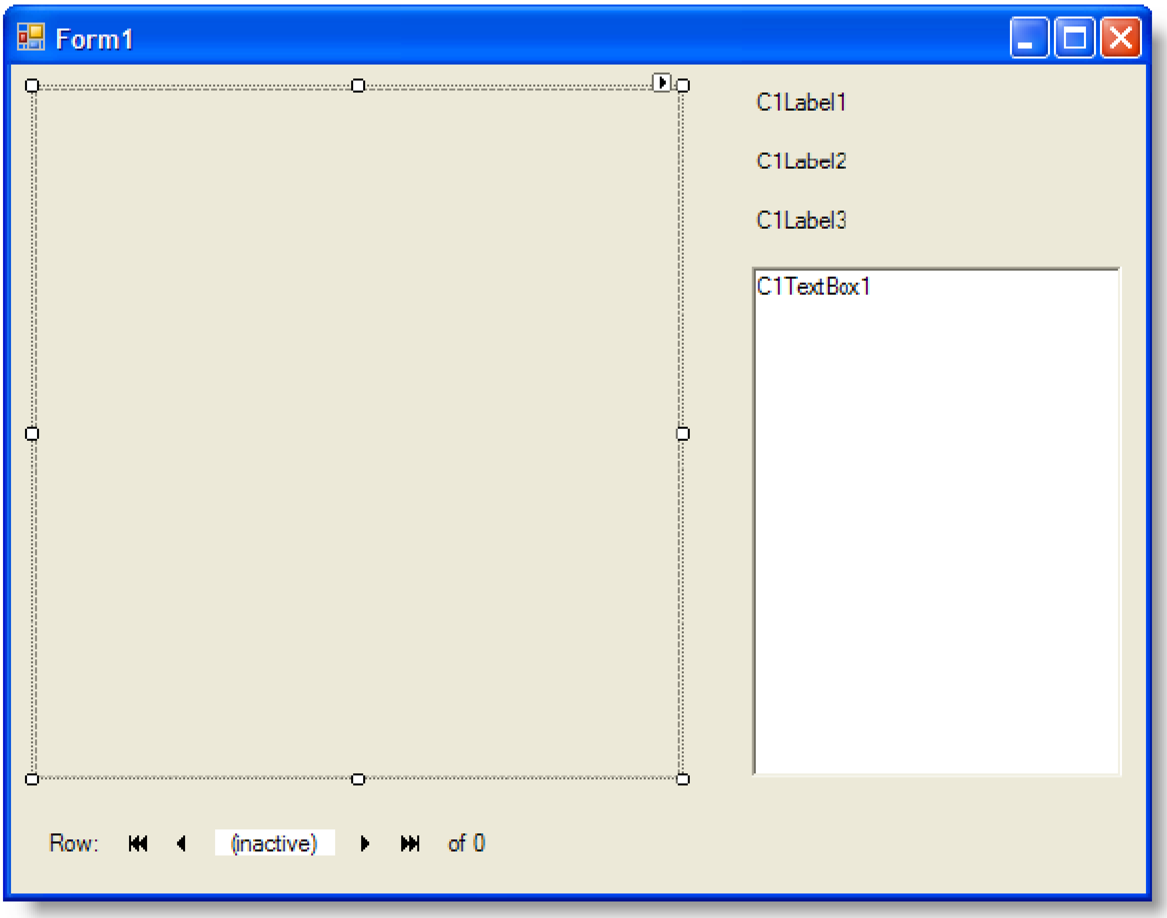
10/17/1993 12:00:00 AM

Steven Buchanan graduated from St. Andrews University, Scotland, with a BSC degree in 1976. Upon joining the company as a sales representative in 1992, he spent 6 months in an orientation program at the Seattle office and then returned to his permanent post in London. He was promoted to sales manager in March 1993. Mr. Buchanan has completed the courses "Successful Telemarketing" and "International Sales Management." He is fluent in French.

Perhaps you want your image to appear larger when the program runs. If you increase the size of the PictureBox control you still need to alter the Input.PictureBox.SizeMode for the picture to stretch to the controls boundaries.

To Expand the Image in the PictureBox

1. Create .NET project and add the following controls to your form.
    - C1ExpressTable1 (C1.Data.Express.C1ExpressTable)
    - C1Label1-3 (C1.Win.C1Input.C1Label)
    - C1PictureBox1 (C1.Win.C1Input.C1PictureBox)
    - C1TextBox1 (C1.Win.C1Input.C1TextBox)
    - C1DbNavigator1 (C1.Win.C1Input.C1DbNavigator)
2. Arrange the controls to resemble the form below:



3. Enter the following to the **C1.Data.Express.C1ExpressTable.ConnectionString** property:

   "**Provider=Microsoft.Jet.OLEDB.4.0;Data Source="Documents\ComponentOne Samples\Common\C1NWind.mdb**"

   Note: Step 3 assumes that you have the sample file, C1NWind.mdb, in the default location created upon installing the ComponentOne controls. If your database file is in a different location or you want to use a different database file, adjust this entry appropriately.

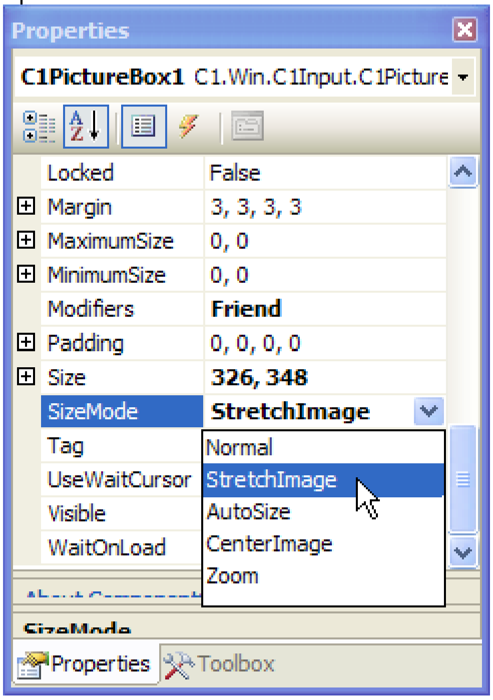
4. Using the Properties window, bind the remaining controls to the database:

Visual Style	Description
C1DbNavigator1.DataSource	C1ExpressTable1
C1Label1.DataSource	C1ExpressTable1
C1Label1.DataField	LastName
C1Label2.DataSource	C1ExpressTable1
C1Label2.DataField	FirstName
C1Label3.DataSource	C1ExpressTable1
C1Label3.DataField	HireDate
C1PictureBox1.DataSource	C1ExpressTable1
C1PictureBox1.DataField	Photo
C1TextBox1.DataSource	C1ExpressTable1

C1TextBox1.DataField	Notes

If you ran the program now, the images shown when cycling through the database would not fill the picture box and a large blank space would appear on the form. You need to stretch the image to the boundaries of the PictureBox.

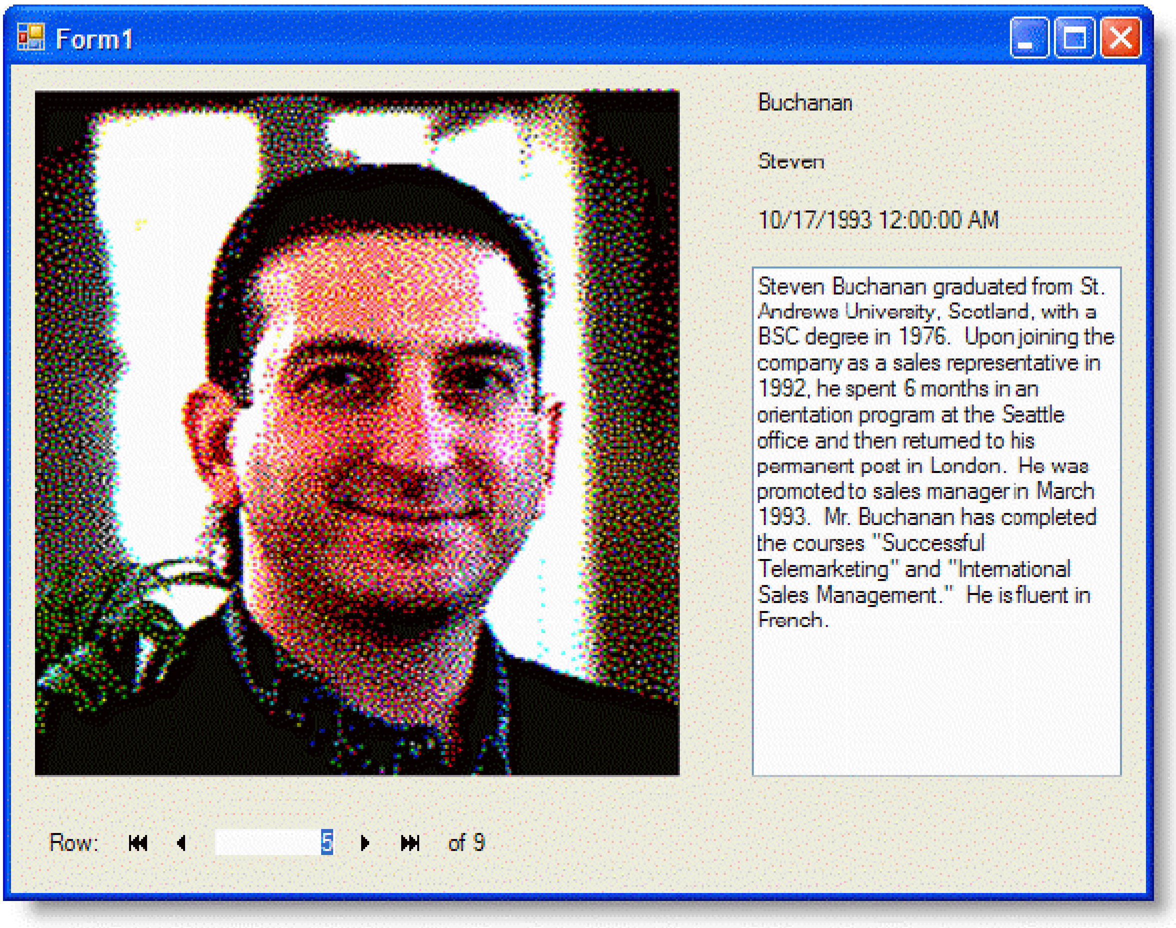
5. Change the C1PictureBox1.SizeMode property from **Normal** to **StretchImage**. Notice that there are 3 other options to choose from as well.



C1PictureBox1 C1.Win.C1Input.C1Picture

Locked	False
Margin	3, 3, 3, 3
MaximumSize	0, 0
MinimumSize	0, 0
Modifiers	**Friend**
Padding	0, 0, 0, 0
Size	**326, 348**
SizeMode	**StretchImage**
Tag	
UseWaitCursor	
Visible	
WaitOnLoad	

6. Run the program and notice how large the employee photo appears.



Buchanan

Steven

10/17/1993 12:00:00 AM

Steven Buchanan graduated from St. Andrews University, Scotland, with a BSC degree in 1976. Upon joining the company as a sales representative in 1992, he spent 6 months in an orientation program at the Seattle office and then returned to his permanent post in London. He was promoted to sales manager in March 1993. Mr. Buchanan has completed the courses "Successful Telemarketing" and "International Sales Management." He is fluent in French.

Navigating the C1DateEdit Control

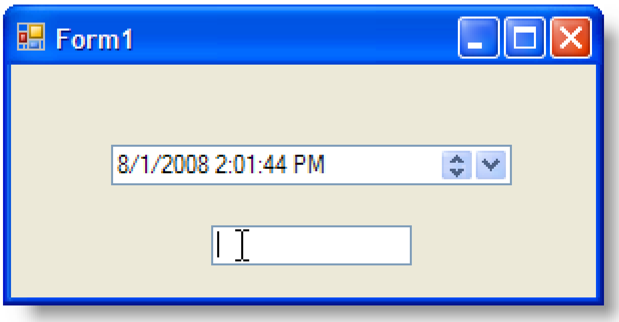
When end-users select the **C1DateEdit** control, then press the Enter or Tab to move to another control or select another control with the mouse, the current date is automatically entered into the **C1DateEdit** control.

To move to another control without the date being automatically entered, complete one of the following:

Using the Designer:

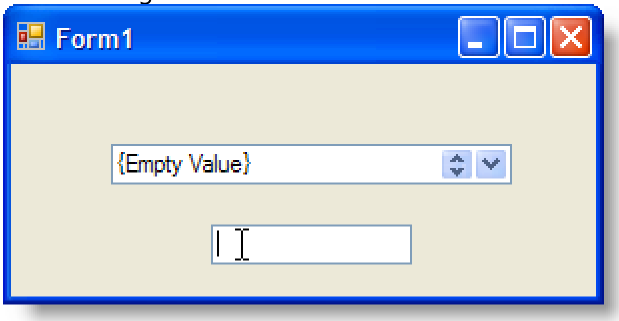
1. Create a new .NET project and place a **C1DateEdit** control and a **C1TextBox** control on your form.
2. To show end-users that a control has an empty value, in the C1DateEdit.NullText property, enter "{Empty Value}".

   If you ran the program at this point and selected the **C1DateEdit** control with either the keyboard or mouse, then try to select the **C1TextBox** control, today's date would automatically be entered into the C1DateEditr field and you would be unable to leave that field blank.



Notice that the cursor is in the **C1TextBox** but the current date remains in the C1DateEdit field.

3. Using the Properties window, change the C1DateEdit1.DateTimeInput property to **False** and change the C1DateEdit1.EmptyAsNull property to **True**.
4. Run the Program and click on the **C1DateEdit** control, then click on the **C1TextBox**.



Notice that even after switching to the **C1TextBox**, the C1DateEdit field remains empty.

Using the Code Editor:

1. Create a new .NET project and in the Solution Explorer add a reference to the C1Input control.
2. Add the following import statement to the code editor

To write code in Visual Basic

Visual Basic

```
Imports C1.Win.C1Input
```

To write code in C#

C#

```
using C1.Win.C1Input;
```

3. Add the C1DateEdit control and C1TextBox control to the Form_Load event.

To write code in Visual Basic

Visual Basic

```
Dim X As New C1DateEdit
Controls.Add(X)
X.Location = New Point(50, 40)
Dim Y As New C1TextBox
Controls.Add(Y)
```

```
Y.Location = New Point(100, 80)
```

To write code in C#

```
C#
C1DateEdit X = new C1DateEdit();
Controls.Add(X);
X.Location = new Point(50, 40);
C1TextBox Y = new C1TextBox();
Controls.Add(Y);
Y.Location = new Point(100, 80);
```

4. To show end-users that the control has an empty value, add the following code to the C1DateEdit entry.

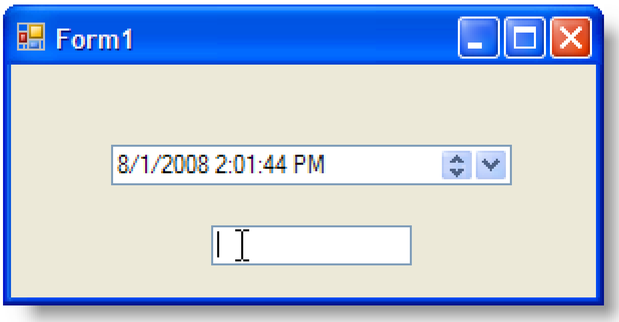
   To write code in Visual Basic

```
Visual Basic
X.NullText = "{Empty Value}"
```

   To write code in C#

```
C#
X.NullText = "{Empty Value}";
```

   If you ran the program at this point and selected the **C1DateEdit** control with either the keyboard or mouse, then try to select the **C1TextBox** control, today's date would automatically be entered into the **C1DateEdit** field and you would be unable to leave that field blank.



5. To preserve the "Empty Value" in the C1DateEdit field even after switching to another control, add the following code to the C1DateEdit entry.

   To write code in Visual Basic

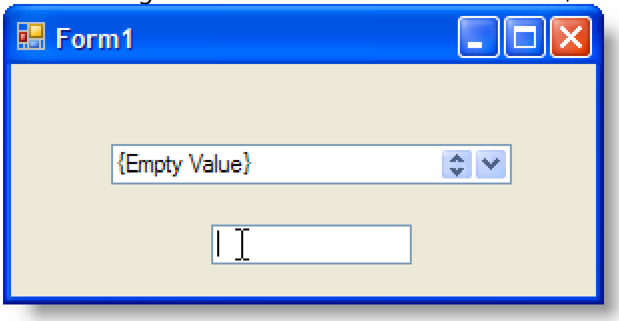
```
Visual Basic
X.DateTimeInput = False
X.EmptyAsNull = False
```

   To write code in C#

```
C#
X.DateTimeInput = False;
```

```
X.EmptyAsNull = False;
```

6. Run the Program and click on C1DateEdit control, then click on the C1TextBox.



Notice that even after switching to the **C1TextBox**, the **C1DateEdit** field remains empty.

Displaying a Border Color for the C1Input Controls

A border color can be applied to **C1DateEdit**, **C1Label**, **C1NumericEdit**, **C1DropDownControl**, and **C1TextBox** controls.

To create a border color for the C1DropDownControl at design time

1. Add a **C1DropDownControl** to your form.
2. Navigate to C1DropDownControl's properties window and set the **BorderStyle** property to "FixedSingle".
3. Change C1DropDownControl's **BorderColor** property to "Red".

To create a border color for the C1DropDownControl programmatically

Use the following code to add a border color to the C1DropDownControl:

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
        Dim dropdown As New C1DropDownControl
        Me.Controls.Add(dropdown)
        dropdown.BorderStyle = BorderStyle.FixedSingle
        dropdown.BorderColor = Color.Red
    End Sub
```

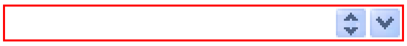
To write code in C#

C#

```
private void Form1_Load(object sender, EventArgs e)
        {
            C1DropDownControl dropdown = new C1DropDownControl();
            this.Controls.Add(dropdown);
            dropdown.BorderStyle = BorderStyle.FixedSingle;
            dropdown.BorderColor = Color.Red;
        }
```

✓ **This topic illustrates the following:**

A red border color with a fixed single border style is added to the **C1DropDownControl**.



Showing a Message Box when the Border Color Changes

You can use the **BorderColorChanged** event when the value of the **BorderColor** property changes.

To create a message box when the border color changes for C1TextBox, complete the following:

1. Add a **C1TextBox** control to your form.
2. Navigate to C1TextBox's properties window and change the **BorderStyle** property to "FixedSingle".
3. Add a **MouseClick** event to the **C1TextBox** control to change C1TextBox's border color to purple.

   To write code in Visual Basic

   Visual Basic

   ```
   Private Sub C1TextBox1_MouseClick(ByVal sender As System.Object, ByVal e As
   System.Windows.Forms.MouseEventArgs) Handles C1TextBox1.MouseClick
       C1TextBox1.BorderColor = Color.Purple
   End Sub
   ```

   To write code in C#

   C#

   ```
   private void c1TextBox1_MouseClick(object sender, MouseEventArgs e)
           {

               c1TextBox1.BorderColor = Color.Purple;
           }
   ```

4. Add a **BorderColorChanged** event to C1TextBox1 to show a message box that informs the user the border color has changed.

   To write code in Visual Basic

   Visual Basic

   ```
   Private Sub C1TextBox1_BorderColorChanged(ByVal sender As System.Object, ByVal e
   As System.EventArgs) Handles C1TextBox1.BorderColorChanged
      MessageBox.Show("The C1TextBox1 border color change to purple")
   End Sub
   ```

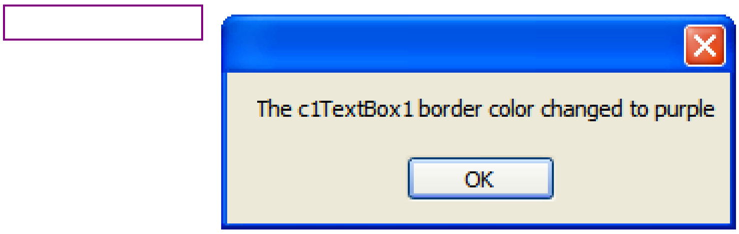
   To write code in C#

   C#

   ```
   private void c1TextBox1_BorderColorChanged(object sender, EventArgs e)
           {
               MessageBox.Show("The c1TextBox1 border color changed to purple");
           }
   ```

✓ **This topic illustrates the following:**

When you mouse click on the **C1TextBox** control the border color changes to purple. Once it changes to purple the **BorderColorChanged** event fires and a message box appears informing the user that the border color has changed.



The c1TextBox1 border color changed to purple

Set IME Mode

The **ImeMode** property can be used to set an Input Method Editor (IME) mode of the C1Input controls. An Input Method Editor is a program that lets users enter complex symbols or characters, like Japanese Kanji characters, into the input controls, using a basic keyboard.

The following table describes the values available for **ImeMode** property.

Value	Description
On	Indicates that the IME is on. Symbols or characters that are specific to Chinese or Japanese can be entered. Valid for Japanese, Simplified Chinese, and Traditional Chinese IME only.
Off	Indicates that the IME is off. The object behaves the same as English entry mode. Valid only for Japanese, Simplified Chinese and Traditional Chinese IME.
Disable	Indicates that the IME is disabled. It means that a user cannot turn the IME on from the keyboard as the IME window is hidden.
Hiragana	Hiragana Double Byte Characters. Valid only for the Japanese IME.
Katakana	Katakana Double Byte Characters. Valid only for the Japanese IME.
KatakanaHalf	Katakana Single Byte Characters. Valid only for the Japanese IME.
AlphaFull	Alphanumeric Double Byte Characters. Valid only for Korean and Japanese IME.
Alpha	Alphanumeric Single Byte Characters. Valid only for Korean and Japanese IME.
HangulFull	Hangul Double Byte Characters. Valid only for the Korean IME.
NoControl	None (Default).
Inherit	Indicated that the IME mode of the parent control is inherited.
Close	Indicates that the IME is closed. Valid only for the Chinese IME.
Hangul	Hangul Single Byte Characters. Valid only for the Korean IME.
OnHalf	Indicates that the IME is on HalfShape. Valid only for the Chinese IME.

Complete the following steps to change the IME mode for the C1Input control:

1. Create a new Windows Application project. Place a C1Input control (C1TextBox, C1ComboBox, C1DateEdit, C1DropDownControl or C1NumericEdit) on the form.

2. From the **Properties** window, set the **ImeMode** property, as per your requirement.

Move Focus

This feature makes navigating through the controls easier and allows you to add keyboard navigation support to the C1Input (C1TextBox, C1ComboBox, C1DateEdit, C1DropDownControl and C1NumericEdit) controls. The following properties enable you to move focus from or to the C1Input controls.

- ExitOnLastChar**:** Moves focus from the C1Input control either when the length of the text entered reaches the maximum length, as defined in the MaxLength property, or when the mask is filled. Its default value is **False**.
- ExitOnLeftRightKey**:** Moves focus from the C1Input control when the left or right arrow keys are pressed. Its default value is **None.**
- TabStop**:** Indicates whether the focus is moved to the C1Input control, from the control on the left, when the TAB key is pressed. Its default value is **True**.

The table below describes values and behavior of the above properties.

Property	Possible Value	Description
ExitOnLastChar	**True**	Enables moving focus from the C1Input control when the length of text entered reaches the maximum length, defined in the **MaxLength** property, or when the mask is filled.
	False	Disables moving focus from the C1Input control when the length of text entered reaches the maximum length, defined in the **MaxLength** property, or when the mask is filled.
ExitOnLeftRightKey	**None**	Disables moving from the C1Input control when the arrow keys are pressed.
	Left	Enables moving focus to the control on the left side of the C1Input control, when the left arrow key is pressed.
	Right	Enables moving focus to the control on the right side of the C1Input control, when the right arrow key is pressed.
	Both	Enables moving focus to the control on either left or right side of the C1Input control, when the respective key is pressed.
TabStop	**True**	Enables moving focus to the C1Input control, from the control on the left side of the C1Input control, when the TAB key is pressed.
	False	Disables moving focus to the C1Input control, from the control on the left side of the C1Input control, when the TAB key is pressed.

Complete the following steps to enable or disable this feature in the C1Input control:

1. Create a new Windows Application project. Place a C1Input control (C1TextBox, C1ComboBox, C1DateEdit, C1DropDownControl or C1NumericEdit) on the form.
2. From the **Properties** window, set either one or all of the following properties, as per your requirement.

- Set the **ExitOnLastChar** to true or false, to enable or disable moving focus from the control when the text entered reaches the maximum length.
- Set the **ExitOnLeftRightKey** to Left, Right or Both, to enable moving the focus from the control when the respective key is pressed.
- Set the **TabStop** to true or false, to enable or disable moving focus to the C1Input control, when the tab key is pressed.

Select Specific Calendar Type

The CalendarType property, present in the C1DateEdit and C1NumericEdit controls lets you select specific calendar types to use a non-default Calendar. The following calendar types are supported by C1DateEdit and C1NumericEdit controls:

- Default
- ChineseLuniSolarCalendar
- EastAsianLunisolarCalendar
- GregorianCalendar
- HebrewCalendar
- HijriCalendar
- JapaneseCalendar
- JapaneseLunisolarCalendar
- JulianCalendar
- KoreanCalendar
- KoreanLunisolarCalendar
- TaiwanCalendar
- TaiwanLunisolarCalendar
- ThaiBuddhistCalendar
- UmAlQuraCalendar

Complete the following steps to change the calendar type for the C1Input control:

1. Create a new Windows Application project. Place a C1Input control (C1DateEdit or C1NumericEdit) on the form.
2. From the **Properties** window, set the **CalendarType** property, as per your requirement.

Spin Up/Spin Down Programmatically

The **SpinUp** and **SpinDown** methods enable you to increase or decrease the values entered in the C1Input controls (C1ComboBox, C1DropDownControl, C1DateEdit and C1NumericInput) when the focus is not on the control.

In addition to the above methods, C1ComboBox and C1DateEdit contain an additional property **AllowSpinLoop** that, when set to true, loops through the items when the **SpinUp** or **SpinDown** method is invoked.

Following is an example to use the **SpinUp** method:

1. Create a new Windows Application project. Place a C1NumericEdit control on the form.
2. From the **Properties** window, set the **Value** property to 0.
3. Add a textbox control onto the form and double click the textbox to generate the **TextChanged** event, in code.
4. Add the following code to the textbox's **TextChanged** event.

To write code in Visual Basic

Visual Basic

```
Private Sub TextBox1_TextChanged(sender As Object, e As EventArgs) Handles
TextBox1.TextChanged
```

```
      c1NumericEdit1.SpinUp(1)
End Sub
```

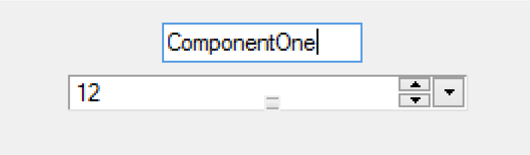
To write code in C#

```
C#
private void textBox1_TextChanged(object sender, EventArgs e)
{
      c1NumericEdit1.SpinUp(1);
}
```

5. Run the project.

What You've Accomplished

When you enter characters into the textbox control, the value in the C1NumericEdit control will increase by 1, each time a new character is entered, giving you the count of characters entered in the control.

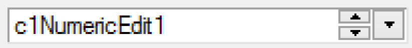

ComponentOne

12

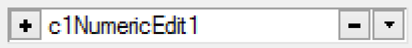
Change Up-Down Button Alignment

The UpDownButtonAlignment property lets you change the alignment of the Up and Down buttons present next to the DropDown button in C1DropDown, C1DateEdit and C1NumericEdit. This property can have the following values:

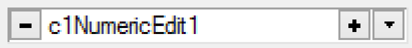
- **Default**: Both the buttons are placed on the right side, along with the drop-down button.


c1NumericEdit1

- **UpLeftDownRight**: The Up button is placed on the left side and the Down button is placed on the right side.


c1NumericEdit1

- **UpRightDownLeft**: The Up button is placed on the right side and the Down button is placed on the left side.


c1NumericEdit1

Complete the following steps to change the alignment of the Up and Down buttons:

1. Create a new Windows Application project. Place a C1Input control (C1DropDown, C1DateEdit or C1NumericEdit) on the form.
2. From the **Properties** window, set the **UpDownButtonAlignment** property, as per your requirement.

Input 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 the ComponentOne Studio.

Please refer to the pre-installed product samples through the following path:

Documents\ComponentOne Samples\WinForms

> **Note:** The Samples are also available at http://helpcentral.componentone.com/Samples.aspx.

Visual basic samples

The following Visual Basic samples are included with C1Input:

Sample	Description
CreditCardDropDown	Using a custom drop-down form. This sample uses the C1DropDownControl and DropDownForm controls.
FormattingInBinding	Demonstrates the use of two important events of C1TextBox, C1Label, and C1PictureBox controls.This sample uses the C1DateEdit, C1DbNavigator, C1Label, and C1PictureBox controls.
InheritedDropDown	Implements a fully functional font editing control based onC1DropDrownControl. The user can change font properties with the texteditor or open the drop-down portion to change font properties in a more convenient way. This sample uses the C1DropDownControl, C1TextBox, and DropDownForm controls.
LookUpSample	Shows how to create a custom drop-down control with names from a lookup table. The **UpDown** button scrolls the names, while the **ellipsis** (...) button allows to enter a new name.
NumPadDropDown1	Replaces the standard calculator in a C1NumericEdit control with the custom drop-down numpad. This sample uses the C1NumericEdit and DropDownForm controls.
NumPadDropDown2	Demonstrates the ability to enter numbers with the mouse or other pointing device. This sample uses the C1DbNavigator, C1NumericEdit, and DropDownForm controls.
ComboBoxFeatures	Demonstrates the basic features of C1ComboBox such as the difference between the Default and DropDownList styles, how to bind the dropdownlist to an enum, array of strings, and a bindingsource, and how to control the drop down form size using the MinimumSize and MaximumSize properties.
ComboBoxImages	Shows how to easily use images from the ImageList in the dropdown items of the C1ComboBox control.
ComboBoxItemModes	Demonstrates how to use the different values (Default, HtmlPattern, and Html) from the ItemMode property.
SplitButtons	This sample shows how to use the C1SplitButton control.
SplitButtonsBasic	This sample shows how to use the basic features of the C1SplitButton control such as how to use the Save and SaveAs actions.

C# samples

The following C# samples are included with C1Input:

Sample	Description
CreditCardDropDown	Using a custom drop-down form. This sample uses the C1DropDownControl and DropDownForm controls.
FormattingInBinding	Demonstrates the use of two important events of C1TextBox, C1Label, and C1PictureBox controls.This sample uses the C1DateEdit, C1DbNavigator, C1Label, and C1PictureBox controls.
InheritedDropDown	Implements a fully functional font editing control based onC1DropDrownControl. The user can change font properties with the texteditor or open the drop-down portion to change font properties in a more convenient way. This sample uses the C1DropDownControl, C1TextBox, and DropDownForm controls.
NumPadDropDown1	Replaces the standard calculator in a C1NumericEdit control with the custom drop-down numpad. This sample uses the C1NumericEdit and DropDownForm controls.
NumPadDropDown2	Demonstrates the ability to enter numbers with the mouse or other pointing device. This sample uses the C1DbNavigator, C1NumericEdit, and DropDownForm controls.
ComboBoxFeatures	Demonstrates the basic features of C1ComboBox such as the difference between the Default and DropDownList styles, how to bind the dropdownlist to an enum, array of strings, and a bindingsource, and how to control the drop down form size using the MinimumSize and MaximumSize properties.
ComboBoxImages	Shows how to easily use images from the ImageList in the dropdown items of the C1ComboBox control.
ComboBoxItemModes	Demonstrates how to use the different values (Default, HtmlPattern, and Html) from the ItemMode property.
SplitButtons	This sample shows how to use the C1SplitButton control.
SplitButtonsBasic	This sample shows how to use the basic features of the C1SplitButton control such as how to use the Save and SaveAs actions.

Input for WinForms Tutorials

The tutorials assume that you are familiar with programming in Visual Basic.NET, know what a DataSet is, and generally know how to use bound controls. The tutorials provide step-by-step instructions—no prior knowledge of C1Input is needed. By following the steps outlined in this chapter, you will be able to create projects demonstrating a variety of C1Input features, and get a good sense of what C1Input can do and how to do it.

The tutorials use an Access database, C1NWind.mdb. The database file C1NWind.mdb is in the **Common** subdirectory of the **ComponentOne Samples** program folder and the sample projects are in the **C1Input** subdirectory of the **ComponentOne Samples** installation directory.

If you have the **Northwind** database installed in a different location, you can change the connection string, or copy the C1NWind.mdb file to the required location.

> **Note:** If you are running the pre-built tutorial projects included in C1Input installation, please be aware that the projects have the sample database location hard coded in the connection string: **Documents\ComponentOne Samples\Common\C1NWind.mdb**

Binding C1Input Controls to a Data Source

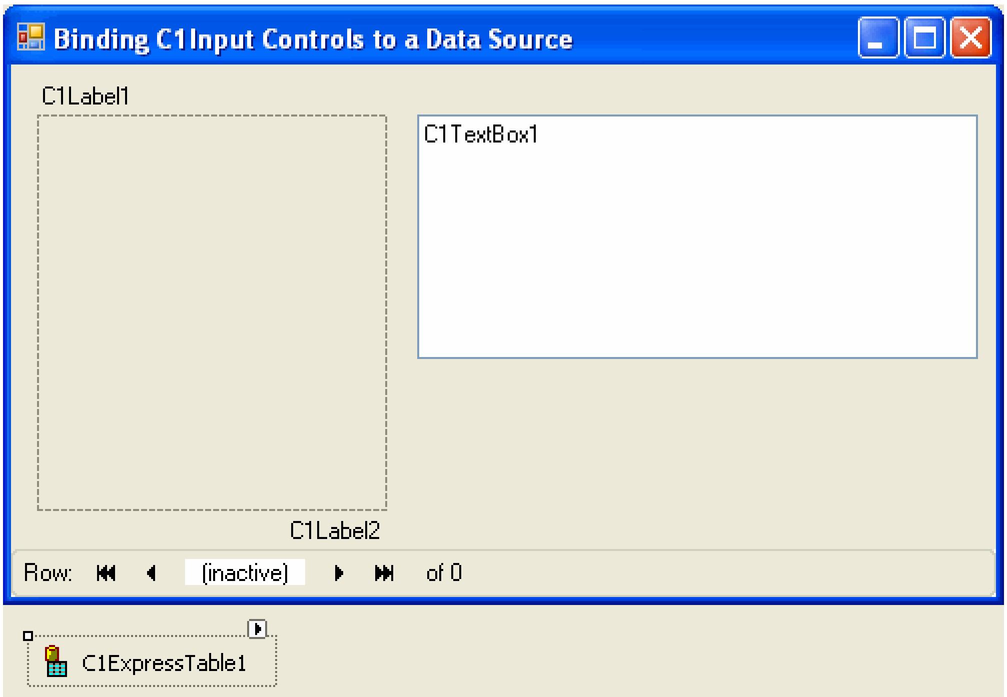
In this tutorial, you will see how easy it is to show database data in a form using C1Input components. Without any manual coding, you can format data, navigate data source rows, even display pictures.

C1Input controls support data binding to any .NET data source objects. Among them, you can use any standard ADO.NET object, such as DataTable, DataView or DataSet as your data source. The alternative and recommended way is to use C1DataObjects framework, a part of ComponentOne Studio Enterprise adding many enhancements to ADO.NET. We will use C1DataObjects Express Edition (C1DataExpress) data source in all our tutorials except this first one, since it is the easiest way to bind to data in a .NET application. If you are interested in ADO.NET versions of tutorial projects, they can be found in the Tutorials\ADO.NET subdirectory. This tutorial will show how to bind both to C1DataObjects and to ADO.NET.

Binding to a C1DataExpress Data Source

To bind to a C1DataExpress data source, complete the following steps:

1. Create a new Windows Application project.
2. Place the following components on the form as shown in the figure:
   - C1ExpressTable1 (C1.Data.Express.C1ExpressTable)
   - C1Label1-2 (C1.Win.C1Input.C1Label)
   - C1PictureBox1 (C1.Win.C1Input.C1PictureBox)
   - C1TextBox1 (C1.Win.C1Input.C1TextBox)
   - C1DbNavigator1 (C1.Win.C1Input.C1DbNavigator)



3. Select the **C1ExpressTable1** component, go to the Properties window, select the **ConnectionString** property drop-down arrow and select **New Connection**.

   The **Add Connection** dialog box opens.

4. Select the provider, the database, and other necessary connection properties in that dialog box.

   In these tutorials, we use the standard MS Access Northwind sample database (C1NWind.mdb). Add the following to the **ConnectionString** property: **Provider=Microsoft.Jet.OLEDB.4.0;Data Source="Documents\ComponentOne Samples\Common\C1NWind.mdb"**.

5. For the **C1ExpressTable1** component, open the **DbTableName** property box and select **Employees** from the **Database Table** list.

   Now, we will bind some **C1Input** controls to the data control. Suppose we want to show the first name, date of birth, notes, and photo for each employee. The first name and date of birth will be displayed in C1Label controls, notes in C1TextBox, and photo in C1PictureBox.

6. For the **C1Label1** control, go to the Properties window, select **C1ExpressTable1** from the drop-down list for the DataSource property.
7. Then select **FirstName** from the list for the DataField property.
8. Bind the **C1Label2** DataSource property to **C1ExpressTable1** and the **C1Label2** DataField property to **BirthDate** and **C1TextBox1** DataSource property to **C1ExpressTable1** and the **C1TextBox1** DataField property to **Notes**.
9. After binding the **C1Label2** control, you can notice that its DataType property is set to the field type, **DateTime**. Now, change the format for **C1Label2** so that it will not show time with the date of birth (showing time is the default). To change the format, select the FormatType property, open its combo box and select **MediumDate**.

   Next, bind the C1PictureBox control to the Photo field.

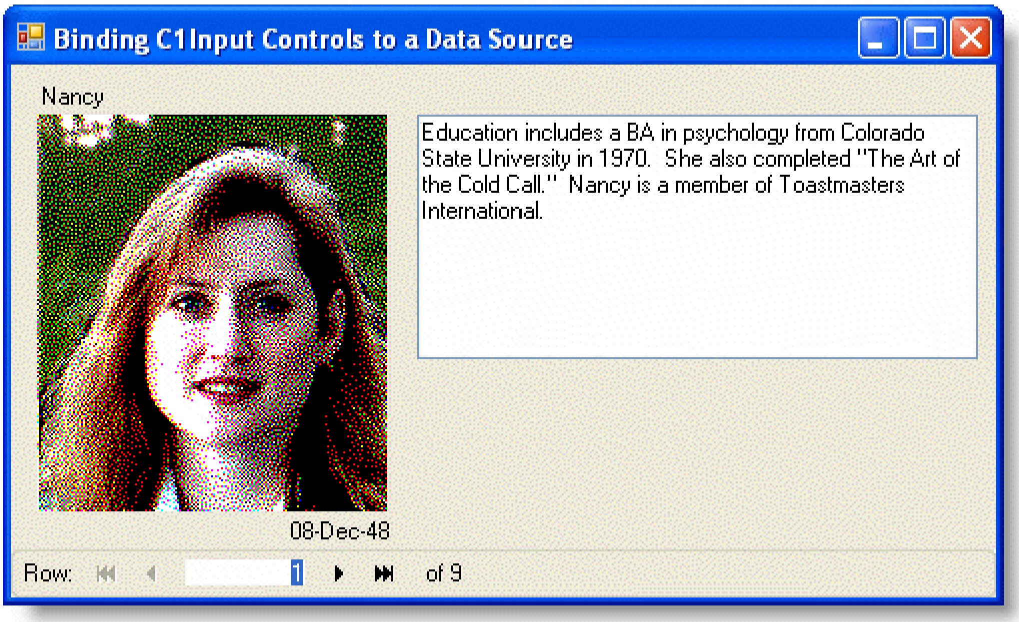
10. First select **C1ExpressTable** in the DataSource property combo box of the C1PictureBox control.
11. Open the DataField property combo box and select **Photo** from the list of available fields.
12. Set up the **C1DbNavigator1** control allowing the user to navigate through the data set. To bind the navigator control to the data source, select **C1ExpressTable1** in its DataSource property combo box.
13. To dock the navigator control to the bottom of the form and separate it from the rest of the form with a three-

dimensional line, set the following properties:

- **C1DbNavigator1.Dock = Bottom**
- **C1DbNavigator1.BorderStyle = Fixed3D**

Run the program and observe the following:

C1Input controls show data in Employees records. You can navigate between Employees records using the four VCR-style buttons of the navigator control. You can also go directly to a certain record by typing the record number in the navigator control record number edit field, or set the input focus to the record number field and use mouse wheel to scroll through the records.


Binding C1Input Controls to a Data Source

Nancy

Education includes a BA in psychology from Colorado State University in 1970. She also completed "The Art of the Cold Call." Nancy is a member of Toastmasters International.

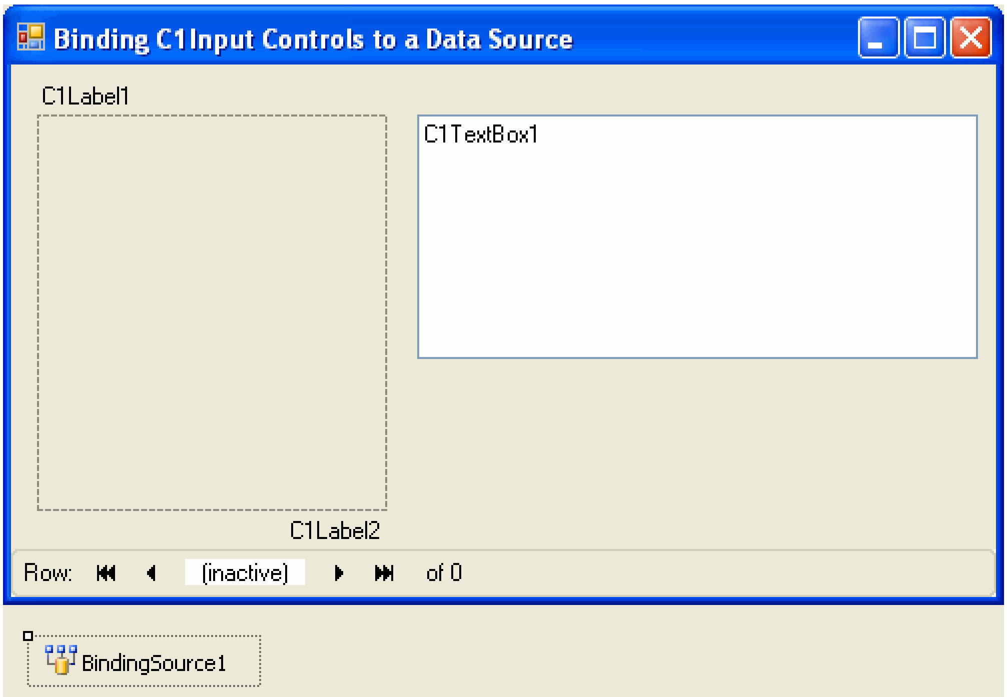
08-Dec-48

Row: 1 of 9

Binding to an ADO.NET Data Source

To bind to an ADO.NET data source, complete the following steps:

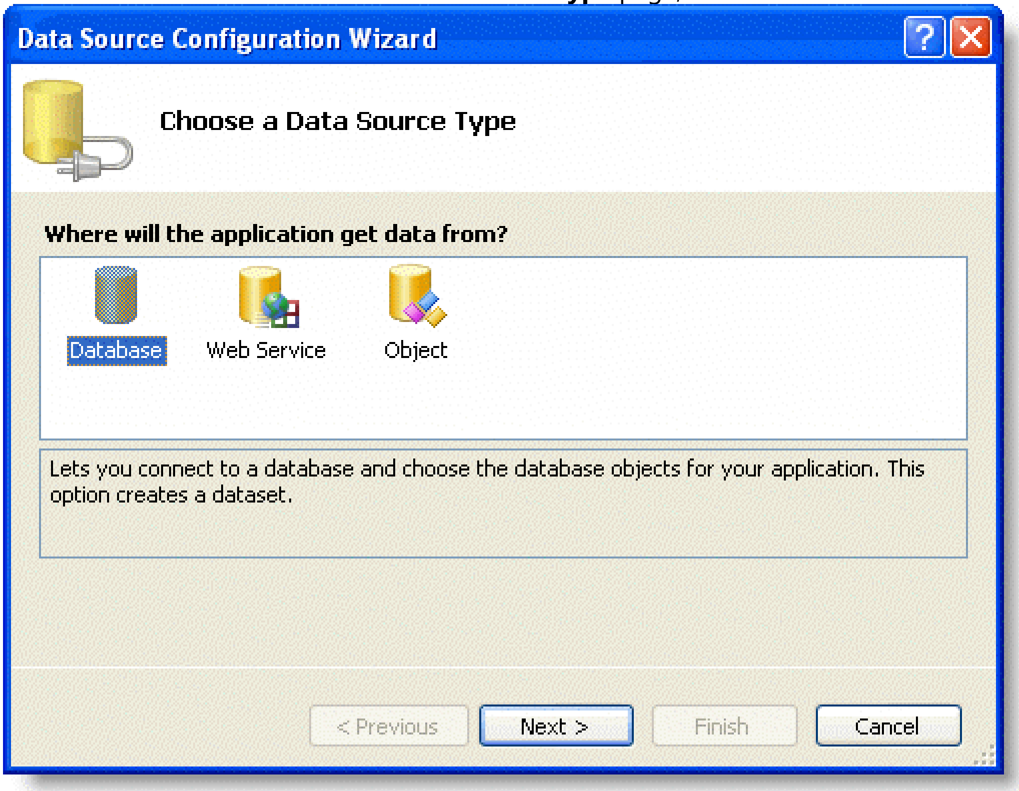
1. Create a new Windows Application project.
2. Place the following components on the form as shown in the figure:
    - C1Label1-2 (C1.Win.C1Input.C1Label)
    - C1PictureBox1 (C1.Win.C1Input.C1PictureBox)
    - C1TextBox1 (C1.Win.C1Input.C1TextBox)
    - C1DbNavigator1 (C1.Win.C1Input.C1DbNavigator)



3. Add a **BindingSource** component (located on the **Data** tab of the Toolbox) to the form.
4. In the Properties window, locate the **DataSource** property and expand the drop-down. Select **Add Project Data Source**.

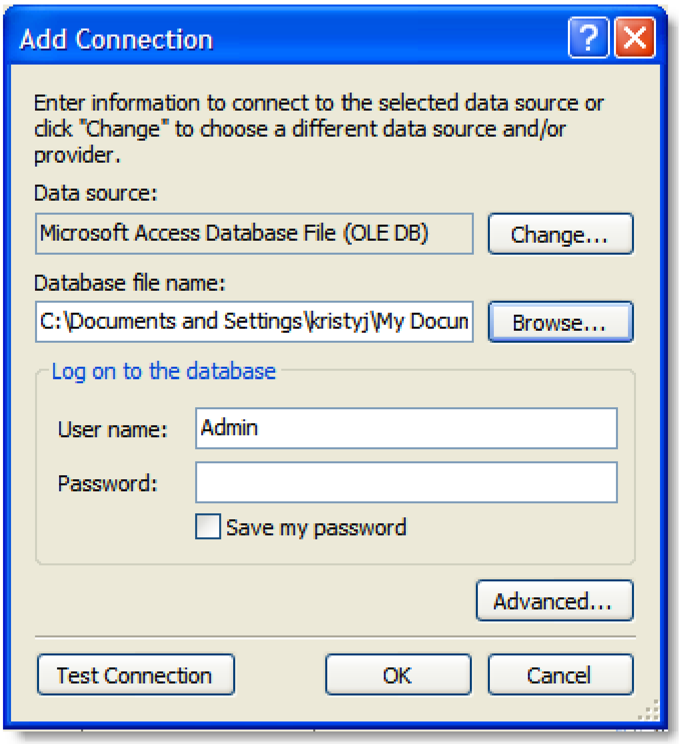
   The **DataSource Configuration Wizard** opens.

5. Select **Database** on the **Choose a Data Source Type** page, and click **Next**.


Data Source Configuration Wizard
Choose a Data Source Type
Where will the application get data from?
Database
Web Service
Object
< Previous
Next >
Finish
Cancel

Lets you connect to a database and choose the database objects for your application. This option creates a dataset.

6. Click the **New Connection** button to create a new connection or choose one from the drop-down list.
7. Click the **Browse** button to specify the location of the C1NWind.mdb database. 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.



Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source: Microsoft Access Database File (OLE DB)

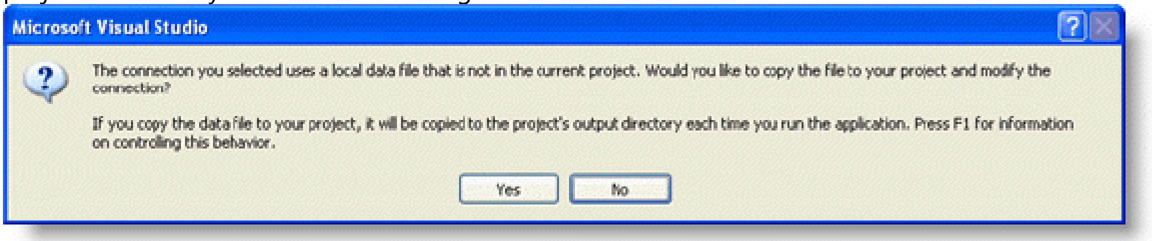
Database file name: C:\Documents and Settings\kristyj\My Docun

User name: Admin

Password:

□ Save my password

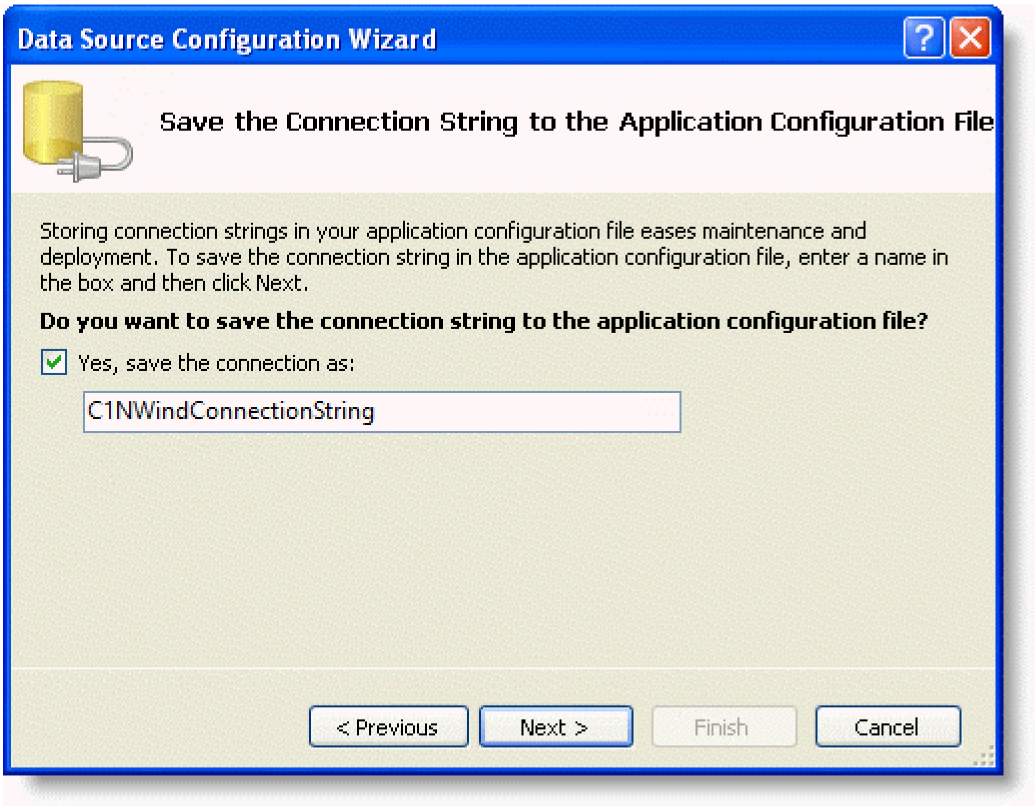
8. Click the **Next** button to continue. A dialog box will appear asking if you would like to add the data file to your project and modify the connection string. Click **No**.



The connection you selected uses a local data file that is not in the current project. Would you like to copy the file to your project and modify the connection?

If you copy the data file to your project, it will be copied to the project's output directory each time you run the application. Press F1 for information on controlling this behavior.

9. Save the connection string in the application configuration file by checking the **Yes, save the connection as** box and entering a name. Click the **Next** button to continue.



Save the Connection String to the Application Configuration File

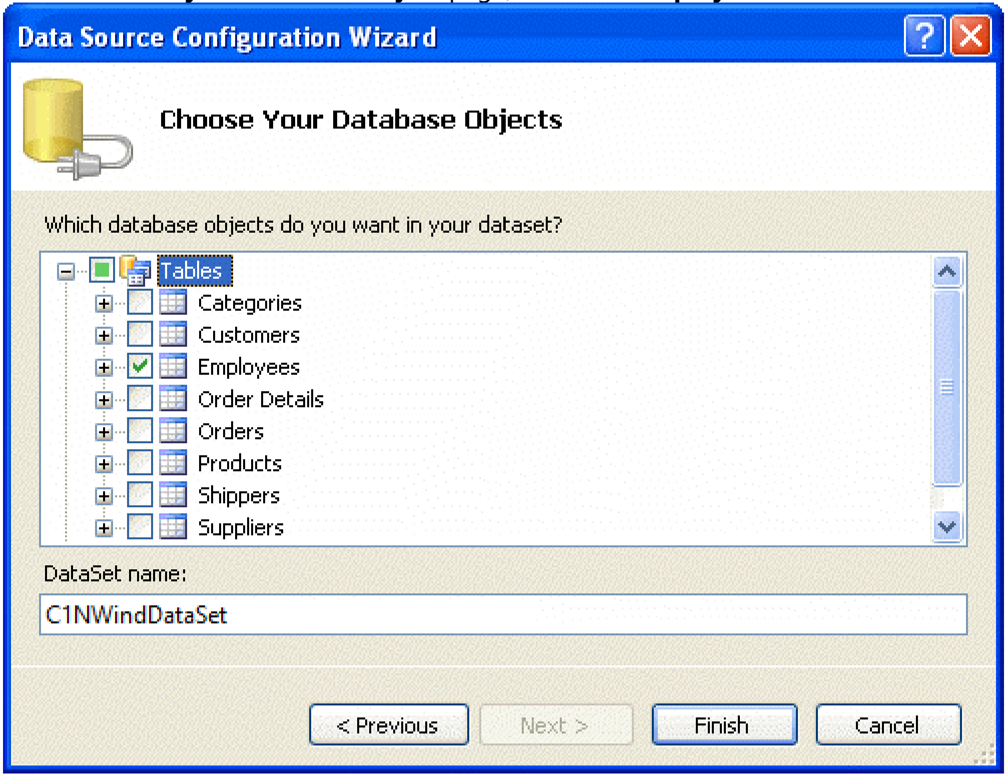
Storing connection strings in your application configuration file eases maintenance and deployment. To save the connection string in the application configuration file, enter a name in the box and then click Next.

Do you want to save the connection string to the application configuration file?

☑ Yes, save the connection as:

C1NWindConnectionString

10. On the **Choose your database object** page, select the **Employees** table. Click **Finish** to exit the wizard.



Choose Your Database Objects

Which database objects do you want in your dataset?

- ■ Tables
  - □ Categories
  - □ Customers
  - ☑ Employees
  - □ Order Details
  - □ Orders
  - □ Products
  - □ Shippers
  - □ Suppliers

DataSet name:

C1NWindDataSet

A DataSet and connection string are added to your project.

11. In the Properties window, set the **DataMember** property for BindingSource1 to **Employees**. A DataTableAdapter is added to the form and automatically adds the following code to the **Form_Load** event:

To write code in Visual Basic

Visual Basic

```
Me.EmployeesTableAdapter.Fill(Me.NwindDataSet.Employees)
```

To write code in C#

C#

```
this.EmployeesTableAdapter.Fill(this.NwindDataSet.Employees);
```

Bind C1Input controls to the data source

Suppose we want to show the first name, date of birth, notes, and photo for each employee. The first name and date of birth will be displayed in a C1Label control, notes in C1TextBox, and photo in C1PictureBox. Binding the **C1Input** controls to the data source involves the following operations:

1. For the **C1Label1** control, go to the Properties window, select **NwindDataSet** from the drop-down list for the DataSource property.
2. Then select **Employees.FirstName** from the list for the DataField property.

   Note: To change the Label's **Size** property, you must set its **AutoSize** property to **False**.

3. As in the steps above, from the Properties window, bind the **C1Label2** DataSource property to **NwindDataSet** and the DataField property to **Employees.BirthDate**.
4. Bind the **C1TextBox1** DataSource property to **NwindDataSet** and the DataField property to **Employees.Notes**.

   After binding the controls, complete the following tasks:

   - Notice that the **C1Label2** DataType property is set to the field type, **DateTime**. Now, change the format for **C1Label2** so that it will not show time with the date of birth (showing time is the default). To change the format, select the FormatType property, open its combo box and select **MediumDate**.
   - Set the **C1TextBox1.Multiline** property to **True**.

   Next, bind the C1PictureBox control to the **Photo** field.

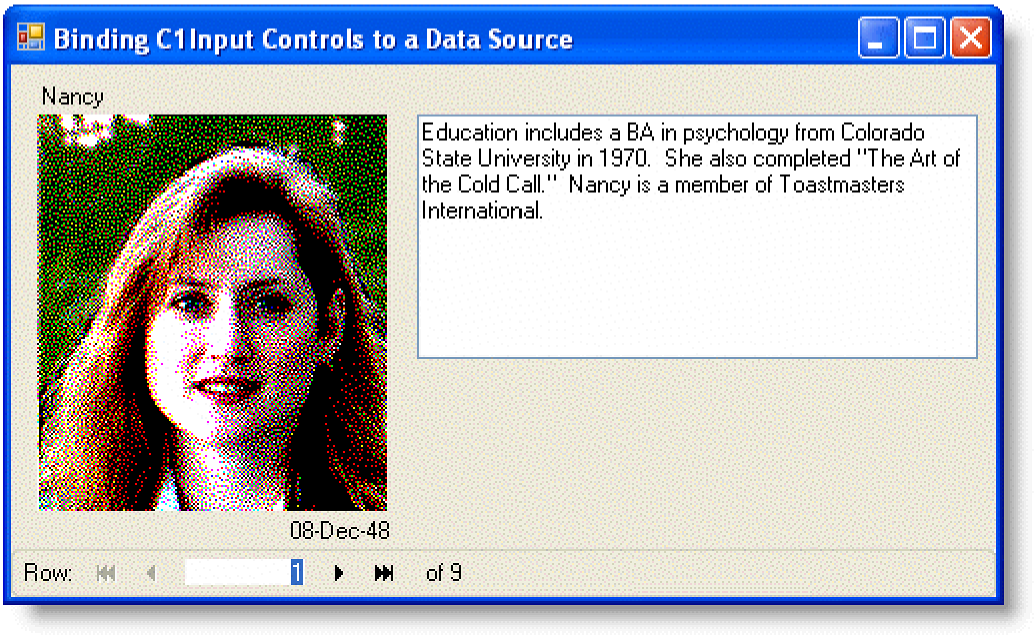
5. First select **NwindDataSet** in the DataSource property combo box of the C1PictureBox control.
6. Then open the DataField property combo box and select **Employees.Photo** from the list of available fields.

   Next, set up the **C1DbNavigator1** control allowing the user to navigate through the data set. To bind the navigator control to the data source:

7. First select **dataSet11** in its **DataSource** property combo box.
8. Then select **Employees** in its **DataMember** property combo box.
9. To dock the navigator control to the bottom of the form and separate it from the rest of the form with a three-dimensional line, set the following properties:
   - **C1DbNavigator1.Dock = Bottom**
   - **C1DbNavigator1.BorderStyle** = **Fixed3D**

Run the program and observe the following:

C1Input controls show data in Employees records. You can navigate between Employees records using the four DVD-style buttons of the navigator control. You can also go directly to a certain record by typing the record number in the navigator control record number edit field, or set the input focus to the record number field and use mouse wheel to scroll through the records.

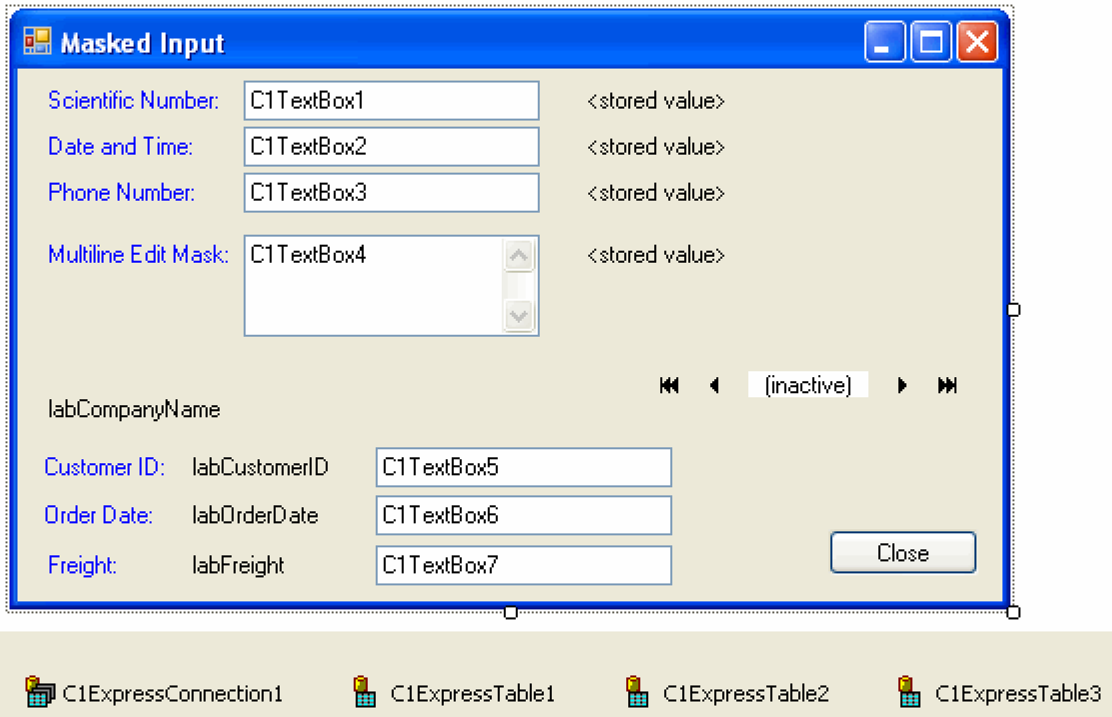


Education includes a BA in psychology from Colorado State University in 1970. She also completed "The Art of the Cold Call." Nancy is a member of Toastmasters International.

Masked Input

In this tutorial, you will learn how to use edit mask to facilitate and restrict user input.

1. Create a new Windows Application project. Place the following components on the form as shown in the figure:
   - C1ExpressConnection1 (C1.Data.Express.C1ExpressConnection)
   - C1ExpressTable1-3 (C1.Data.Express.C1ExpressTable)
   - Label1-4 (all of type System.Windows.Forms.Label)
   - C1TextBox1-7 (C1.Win.C1Input .C1TextBox)
   - C1Label1-4 (C1.Win.Input.C1Label)
   - C1DbNavigator1 (C1.Win.C1Input .C1DbNavigator)
   - Button1 (System.Windows.Forms.Button)



> **Note:** The labels in blue are all of type System.Windows.Forms.Label. The Text property for each label is as it appears on the form.

2. From the Properties window, set the following basic properties for the **Label**, C1Label, and **Button** controls:

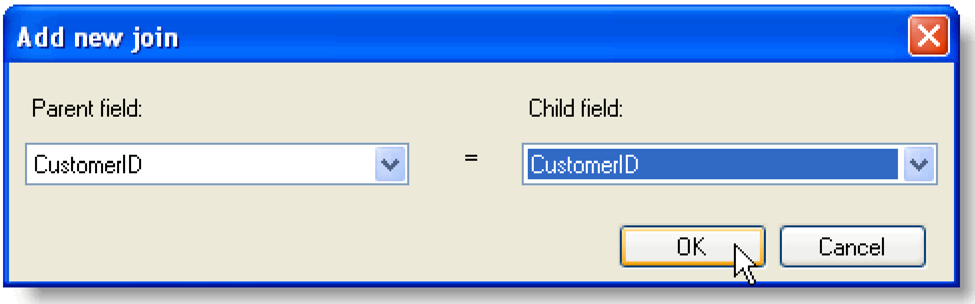
Control	Property	Value
Label1	Name	labStoredExpNumber
	Text	<stored value>
Label2	Name	labStoredDateTime
	Text	<stored value>
Label3	Name	labStoredPhone
	Text	<stored value>
Label4	Name	labStoredMultiline
	Text	<stored value>
C1Label1	Name	labCompanyName
	Text	labCompanyName
C1Label2	Name	labCustomerID
	Text	labCustomerID
C1Label3	Name	labOrderDate
	Text	labOrderDate
C1Label4	Name	labFreight
	Text	labFreight
Button1	Name	btnClose
	Text	Close

3. Select the **C1ExpressConnection1** component, go to the Properties window, open the **ConnectionString** property. Add the following to the **ConnectionString** property: **Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" Documents\ComponentOne Samples\Common "**.
4. Set the properties of C1ExpressTable1-3 as follows:

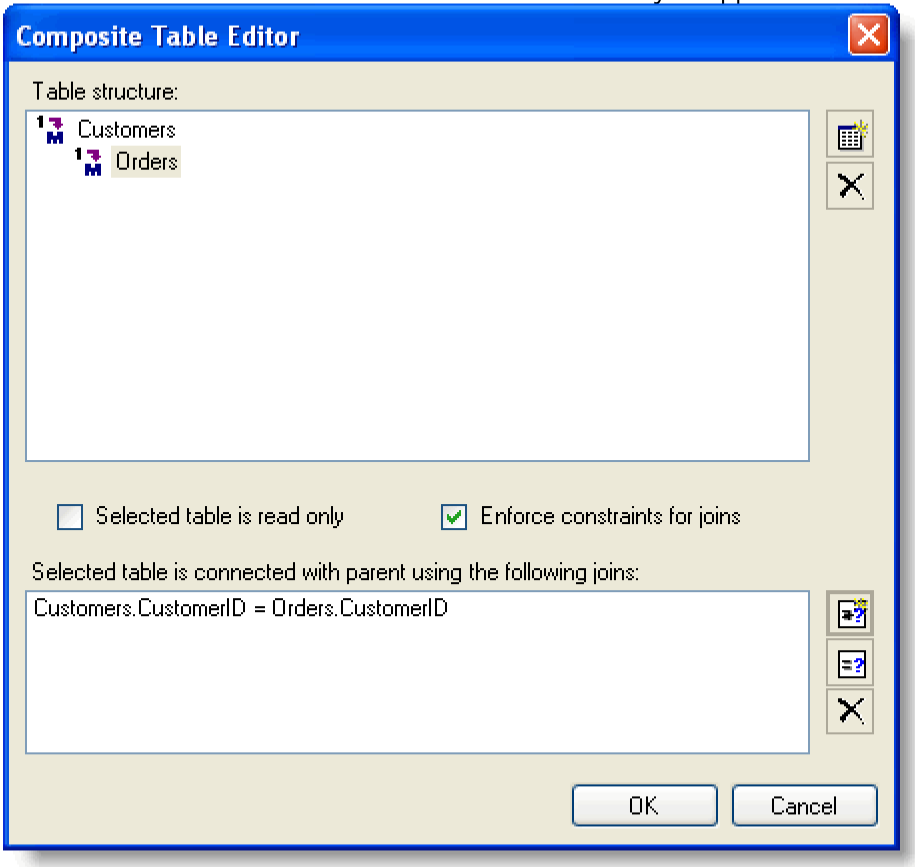
Control	Property	Value
C1ExpressTable1	ConnectionComponent	C1ExpressConnection1
	DbTableName	Customers
C1ExpressTable2	ConnectionComponent	C1ExpressConnection1
	DbTableName	Orders
C1ExpressTable3	ConnectionComponent	C1ExpressConnection1
	DbTableName	<Composite...> * See steps below.

* Selecting the <Composite...> value for the **C1ExpressTable3** opens the **Composite Table Editor**. In the editor, complete the following steps:

1. Select the **Add table** button to add **Customers** table.
2. Select the **Add table** button again to add **Orders** table related as One-to-Many (1-M).
3. Select the **Add join** button. The **Add new join** dialog box appears.
4. Select **CustomerID** for the Parent field and Child field.



5. Click **OK**.
6. The new **Customers.CustomerID – Orders.CustomerID** join appears in the window; click **OK**.



☐ Selected table is read only ☑ Enforce constraints for joins

Selected table is connected with parent using the following joins:

Customers.CustomerID = Orders.CustomerID

7. To bind controls to the data source, set the following properties:

> **Note:** If all of the fields in the CompositeTable are not appearing in the **DataField** drop-down, select C1ExpressTable3 and open the **C1ExpressTable Tasks** menu by clicking the smart tag (▸). In the **C1ExpressTable Tasks** menu, click **Retrieve Fields**. You may have to set the **DataSource** and **DataField** properties again.

Control	Property	Value
C1DbNavigator1	DataSource	C1ExpressConnection1
	DataMember	CompositeTable
labCompanyName	DataSource	C1ExpressConnection1

	DataField	CompositeTable.CompanyName
labCustomerID	DataSource	C1ExpressConnection1
	DataField	CompositeTable.CustomerID
labOrderDate	DataSource	C1ExpressConnection1
	DataField	CompositeTable.OrderDate
labFreight	DataSource	C1ExpressConnection1
	DataField	CompositeTable.Freight
C1TextBox5	DataSource	C1ExpressConnection1
	DataField	CompositeTable.CustomerID
C1TextBox6	DataSource	C1ExpressConnection1
	DataField	CompositeTable.OrderDate
C1TextBox7	DataSource	C1ExpressConnection1
	DataField	CompositeTable.Freight

8. The labels located to the right of the C1TextBox1-4 controls display the current Value property of the corresponding C1TextBox. To synchronize them with C1TextBox1-4, create event handlers **C1TextBox1(...4)_ValueChanged**:

   To write code in Visual Basic

   Visual Basic

```
Private Sub C1TextBox1_ValueChanged(ByVal sender As Object, ByVal e As
System.EventArgs) Handles C1TextBox1.ValueChanged
    Try
        labStoredExpNumber.Text = CType(Me.C1TextBox1.Value, String)
    Catch
        labStoredExpNumber.Text = ""
    End Try
End Sub

Private Sub C1TextBox2_ValueChanged(ByVal sender As Object, ByVal e As
System.EventArgs) Handles C1TextBox2.ValueChanged
    Try
        labStoredDateTime.Text = CType(Me.C1TextBox2.Value, String)
    Catch
        labStoredDateTime.Text = ""
    End Try
End Sub

Private Sub C1TextBox3_ValueChanged(ByVal sender As Object, ByVal e As
System.EventArgs) Handles C1TextBox3.ValueChanged
    Try
        labStoredPhone.Text = CType(Me.C1TextBox3.Value, String)
    Catch
        labStoredPhone.Text = ""
    End Try
End Sub

Private Sub C1TextBox4_ValueChanged(ByVal sender As Object, ByVal e As
System.EventArgs) Handles C1TextBox4.ValueChanged
    Try
        labStoredMultiline.Text = CType(Me.C1TextBox4.Value, String)
    Catch
        labStoredMultiline.Text = ""
    End Try
End Sub
```

To write code in C#

C#

```
private void c1TextBox1_ValueChanged(object sender, System.EventArgs e)
{
    try
    {
        labStoredExpNumber.Text = (string)this.c1TextBox1.Value;
    }
    catch
    {
        labStoredExpNumber.Text = "";
    }
}

private void c1TextBox2_ValueChanged(object sender, System.EventArgs e)
{
    try
    {
        labStoredDateTime.Text = (string)this.c1TextBox1.Value;
    }
    catch
    {
        labStoredDateTime.Text = "";
    }
}

private void c1TextBox3_ValueChanged(object sender, System.EventArgs e)
{
    try
    {
        labStoredPhone.Text = (string)this.c1TextBox1.Value;
    }
    catch
    {
        labStoredPhone.Text = "";
    }
}

private void c1TextBox4_ValueChanged(object sender, System.EventArgs e)
```

```
{
    try
    {
        labStoredMultiline.Text = (string)this.c1TextBox1.Value;
    }
    catch
    {
        labStoredMultiline.Text = "";
    }
}
```

9. **C1TextBox1** allows you to enter numbers with or without fractional part and with optional exponent. It uses the WhenNextStarted mode of showing mask literals, so the decimal point appears only when necessary, when the user starts entering the decimal part.

For C1TextBox1:

10. Set the EditMask property to **!###0.^999e#9**.

    Here '#' is an optional position for a digit or sign, '9' is an optional position for a digit, '0' is a required position for a digit, '!' is a special character specifying right justification for the following text, '^' cancels right justification mode, 'e' – a literal.

11. Then expand the MaskInfo property and set ShowLiterals to **WhenNextStarted**.

> **Note:** C1TextBox and C1NumericEdit support a special edit mode, NumericInput mode facilitating input of any numeric data type. It is usually more convenient for numeric input than edit mask. Try NumericInput before using an edit mask for numeric data.

12. **C1TextBox2** is used for entering a date/time value.

For C1TextBox2:

13. Set the EditMask property to **!90/90/9900 90:90 >PM**.

    There are two new special characters used in this mask: '>' causes the next characters to be converted to upper case, 'P' is a non-standard special character (custom placeholder) allowing entering either 'A' or 'P'.

14. To define a custom placeholder, expand the MaskInfo property and press the **ellipsis** button in the **CustomPlaceholders** property to open the **Collection Editor**. In the **Collection Editor**, add a new item. Set the properties of the newly added item:

Property	Value
Placeholder	P
LookupChars	AP

    Thus the letter 'P' represents a position in edit mask where the user can type 'A' or 'P'.

15. Now, we want to specify that entered date/time values are stored in the database in a compact form without literals. For example, the value "11/_8/2002 _1:42 PM" is stored as "11*82002*142P".

    To enable this storage format, set SaveBlanks to **True** and SaveLiterals to **False** (both properties are contained in MaskInfo). Also, change StoredEmptyChar from default '_' to '*' to store asterisk in blank positions. If you set SaveBlanks to **False**, SaveLiterals to **True**, blank positions will not be saved in the database, the above data will be saved as "11/8/2002 1:42PM".

> **Note:** C1TextBox and C1DateEdit support a special edit mode, DateTimeInput mode facilitating input of date/time data. It is usually more convenient for date/time input than edit mask. Try DateTimeInput before using an edit mask for date/time data.

Set up the **C1TextBox3** control for telephone number input with mask (999) 0099-00099. Here ‘9’ is an optional digit position, ‘0’ is a required digit position.

For C1TextBox3:

16. Set the EditMask property to **(999) 0099-00099**.
17. Expand the MaskInfo property and set AutoTabWhenFilled to **True**. This will automatically move the input focus to the next control once the user fills the last mask position.
18. The **C1TextBox4** control demonstrates multiline mask input.

    For the **C1TextBox4**:

    - Set its **Multiline** property to **True**.
    - Set its **ScrollBars** property to **Vertical**.
    - Set the EditMask property to the following:
      `"First Name:  "CCCCCCCCCCCCCCC\n"Last Name: "CCCCCCCCCCCCCCCCCCCC\n"`
    - `Date of Birth:  "!90/90/9900^\n"Work Status:  "CCCCCCCCCCCCCCC\n"Salary:`
    - `$"!######0.^99`

      Here ‘\n’ represents a line break.

    - Expand the MaskInfo property and set SaveBlanks to **True**.

    This ensures that all positions left blank by the user are saved as spaces. If SaveBlanks is set to **False**, optional positions not filled by the user will be ignored. Additionally, you can set SaveLiterals to **False**, which will prevent saving literal texts, so only the information typed by the user is saved.

19. All the previous controls were unbound. Let us also configure some data bound C1Label and C1TextBox controls. They are located at the bottom of the form. Their data binding properties were set on Step 4. Set their other properties as follows:

Control	Property	Value
labCustomerID	MaskInfo.EditMask	>LLLLL
C1TextBox5	EditMask	>LLLLL
labOrderDate	FormatType	CustomFormat
	CustomFormat	M/d/yyyy
	MaskInfo.EditMask	!90/90/0000
C1TextBox6	FormatType	CustomFormat
	CustomFormat	M/d/yyyy
	EditMask	!90/90/0000
labFreight	FormatType	CustomFormat
	CustomFormat	$ ####0.##
	MaskInfo.EditMask	$!99990.^99
C1TextBox7	FormatType	CustomFormat

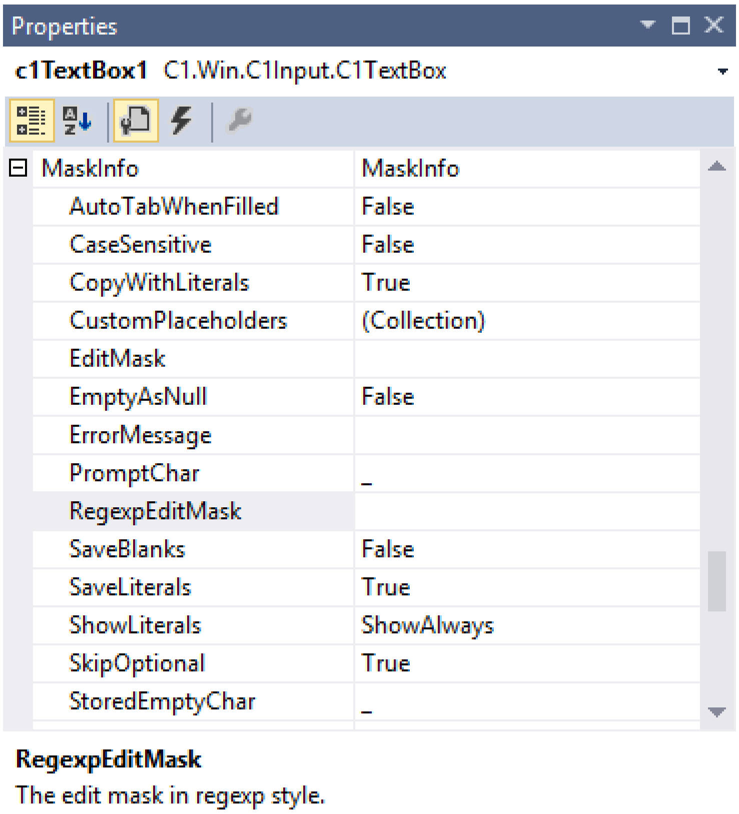
	CustomFormat	$ ####0.##
	EditMask	$!99990.^99

Run the program and observe the following:

- Mask characters appear when you move focus to a control (except c1TextBox1 that has ShowLiterals set to **WhenNextStarted**).
- In edit mode, empty mask positions are shown using PromptChar (default: '_'). When the control loses focus, these positions are hidden.
- Experiment typing in the controls and look at the stored values displayed in the labels.
- If you select the whole text and press Delete, the input control is cleared. Depending on the EmptyAsNull, this is interpreted either as DBNull value or as current edit cancellation, reverting to the previous value when the control loses focus. The same effect, canceling edit, has the ESC key.

Masked Input Using Regular Expressions

You can use regular expressions in masks for validating more complex input formats using RegexpEditMask. The Properties pane showing **RegexpEditMask** property for **c1TextBox** control is as shown:



Properties

c1TextBox1 C1.Win.C1Input.C1TextBox

⊟ MaskInfo	MaskInfo
AutoTabWhenFilled	False
CaseSensitive	False
CopyWithLiterals	True
CustomPlaceholders	(Collection)
EditMask	
EmptyAsNull	False
ErrorMessage	
PromptChar	_
RegexpEditMask	
SaveBlanks	False
SaveLiterals	True
ShowLiterals	ShowAlways
SkipOptional	True
StoredEmptyChar	_

RegexpEditMask
The edit mask in regexp style.

The keywords in regular expressions and their description is shown in the following table:

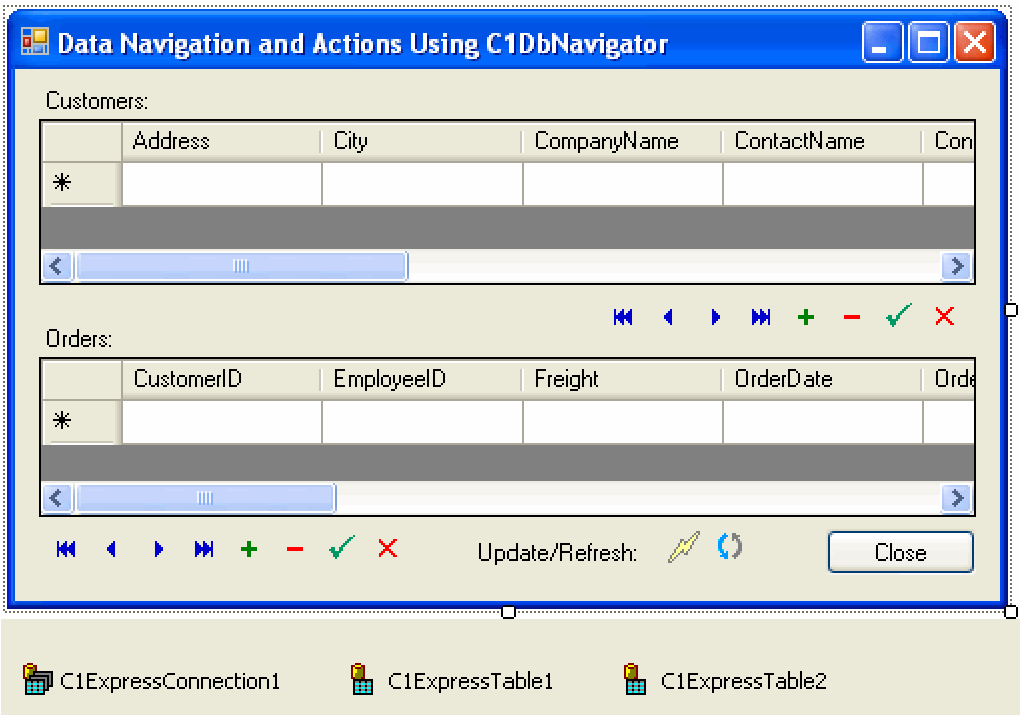
Keywords	Description
\A	Matches any upper case alphabet [A-Z].
\a	Matches any lower case alphabet [a-z].
\D	Matches any decimal digit [0-9].

Keywords	Description
\W	Matches any word character. It is same as [a-z A-Z 0-9].
\K	Matches SBCS Katakana.
\H	Matches all SBCS characters.
\ A	Matches any upper case DBCS alphabet [A-Z].
\ a	Matches any lower case DBCS alphabet [a-z].
\ D	Matches any DBCS decimal digit [0-9].
\ W	Matches any DBCS word character. It is same as [a-z A-Z 0-9].
\ K	Matches all DBCS Katakana.
\ J	Matches all Hiragana.
\ Z	Matches all DBCS characters.
\N	Matches all SBCS big Katakana.
\ N	Matches all DBCS big Katakana.
\ G	Matches DBCS big Hiragana.
\ T	Matches surrogate character.
[]	Matches a character subset.
[^]	To express an exclude subset.
-	Defines contiguous character range.
{}	Specifies the number of times to match.
*	Specifies zero or more matches. It is the short expression for {0,}.
+	Specifies one or more matches. It is the short expression for {1,}.
?	Specifies zero or one matches. It is the short expression for {0,1}.

Data Navigation and Actions Using C1DbNavigator

This tutorial demonstrates the main features of the C1DbNavigator control. It is primarily used to navigate through data source rows, but it can also be used to perform common actions on data such as adding rows, confirming changes, updating the data source, refreshing from the data source (canceling changes), and others.

1. Create a new Windows Application project.
2. Place the following components on the form as shown in the figure:
    - C1ExpressConnection (C1.Data.Express.C1ExpressConnection)
    - C1ExpressTable1-2 (C1.Data.Express.C1ExpressTable)
    - C1Label1-3 (C1.Win.C1Input.C1Label)
    - DataGridView1-2 (System.Windows.Forms.DataGridView)
    - C1DbNavigator1-3 (C1.Win.C1Input.C1DbNavigator)
    - Button1 (System.Windows.Forms.Button)



3. From the Properties window, set the following properties for the C1Label and **Button** controls:

Control	Property	Value
C1Label1	TextDetached	True
	Text	Customers:
C1Label2	TextDetached	True
	Text	Orders:
C1Label3	TextDetached	True
	Text	Update/Refresh:
Button1	Name	btnClose
	Text	Close

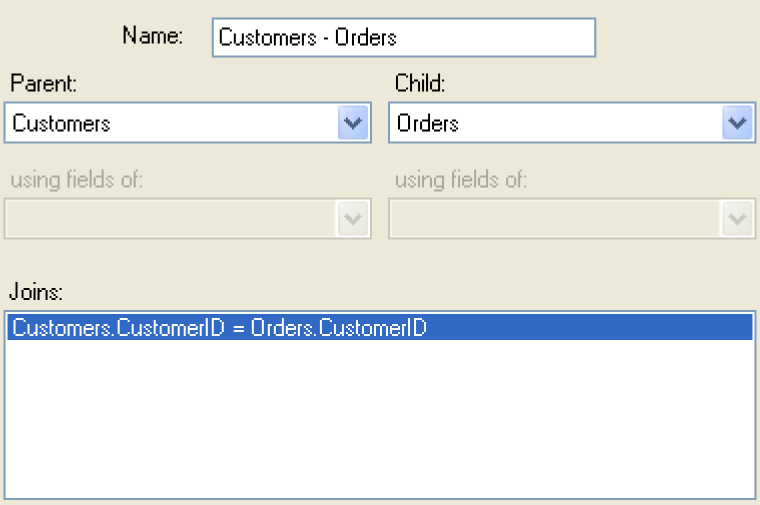
4. Select the C1ExpressConnection1 component, go to the Properties window, open the **ConnectionString** property. Add the following to the **ConnectionString** property: **Provider=Microsoft.Jet.OLEDB.4.0;Data Source="Documents\ComponentOne Samples\Common\C1NWind.mdb"**.
5. Set the properties of C1ExpressTable1-2 as follows:

Control	Property	Value
C1ExpressTable1	ConnectionComponent	C1ExpressConnection1
	DbTableName	Customers
C1ExpressTable2	ConnectionComponent	C1ExpressConnection1
	DbTableName	Orders

6. Also, you need to set the **AutoIncrement** property of the *OrderID* field in C1ExpressTable2, otherwise you will not be able to update data in the database: Select C1ExpressTable2, press the **ellipsis** button in the Properties window for the **Fields** property to open the **Fields** dialog box, select the *OrderID* field and set its **AutoIncrement** property to **ClientAndServer**. This is necessary because *OrderID* is an autoincrement field in the database, so it must be treated accordingly by C1DataExpress.

Create a master-detail relation between C1ExpressTable1 (master) and C1ExpressTable2 (detail):

7. Press the **ellipsis** button in the **Relations** property of the **C1ExpressionConnection1** control to open the **Relations** dialog box.
8. In the dialog box, select **Customers** for **Parent**, **Orders** for **Child**, add a join with CustomerID for both Parent field and Child field, press **OK** to close the **Relations** dialog box.



Name: Customers - Orders

Parent: Customers

Child: Orders

Joins:

Customers.CustomerID = Orders.CustomerID

9. To bind controls to the data source, set the following properties:

Control	Property	Value
DataGridView1	DataSource	C1ExpressConnection1
	DataMember	_Customers
C1DbNavigator1	DataSource	C1ExpressConnection1
	DataMember	_Customers
DataGridView2	DataSource	C1ExpressConnection1
	DataMember	_Customers.Customers – Orders
C1DbNavigator2	DataSource	C1ExpressConnection1
	DataMember	_Customers.Customers – Orders

10. Now **C1DbNavigator1** allows us to navigate the master table **Customers**, and the detail table **Orders**. However, we want some more functionality for our navigator control. We want the user to be able to perform common data action pressing navigator buttons. Using the VisibleButtons property, show the following buttons for C1DbNavigator1-2: **Add**, **Delete**, **Apply**, **Cancel**. By default, a navigator control also shows the current row number and row count. To save space, hide this information be setting the PositionVisible property to **False**.
11. The third navigator control, **C1DbNavigator3** is used to perform update and refresh for the whole data set (both tables **Customers** and **Orders**). To hide redundant navigator fields and buttons, set PositionVisible to **False** and select only two buttons, **Update** and **Refresh** in the VisibleButtons property. **Update** and **Refresh** buttons do not have built-in functionality, since their function may be different with different data sources. To make them work, you need to write code in event handlers. Create the following event handlers:

To write code in Visual Basic

Visual Basic

```
Private Sub C1DbNavigator3_UpdateData(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles C1DbNavigator3.UpdateData
    Me.C1ExpressConnection1.Update()
```

```
End Sub

Private Sub C1DbNavigator3_RefreshData(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles C1DbNavigator3.RefreshData
    Me.C1ExpressConnection1.Fill()
End Sub
```

To write code in C#

C#

```
private void c1DbNavigator3_UpdateData(object sender, System.EventArgs e)
{
    this.c1ExpressConnection1.Update();
}

private void c1DbNavigator3_RefreshData(object sender, System.EventArgs e)
{
    this.c1ExpressConnection1.Fill();
}
```

12. Navigator buttons can be monochrome, color (ColorButtons property set to **True**) or changing their color when hovered over (ColorWhenHover property set to **True**). They can also be three-dimensional (standard) or flat depending on the ButtonStyle property. Set the following properties:

Control	Property	Value
C1DbNavigator1	ButtonStyle	Standard
	C1DbNavigator1.ColorButtons	True
C1DbNavigator2	ButtonStyle	Standard
	ColorButtons	True
C1DbNavigator3	ButtonStyle	Standard
	ColorButtons	True

Using properties UIStrings and ButtonToolTips, it is also possible to change the text displayed in the navigator and to change button ToolTips. For instance, we could specify the first button ToolTip as "Select the first customer" instead of the default "First record". These properties can also be used to localize your application.

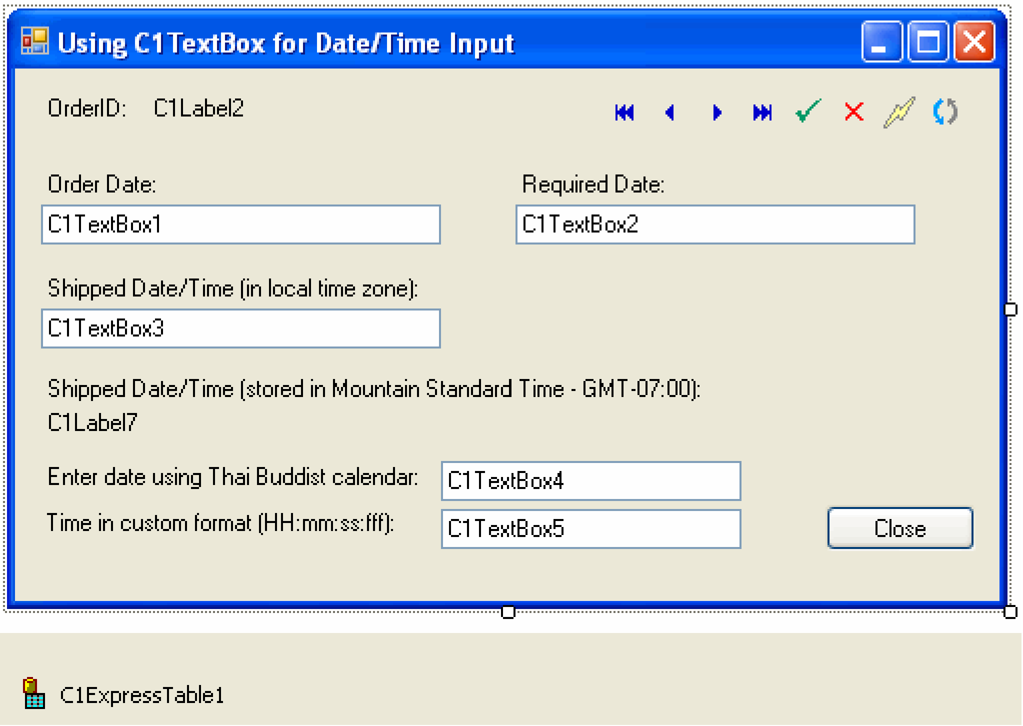
Run the program and observe the following:

- By pressing the navigator buttons, you can move through the rows. If you press the **Next** button and hold it for a certain time (determined by the MoveDelayFirst property), the current position will change automatically with frequency determined by the MoveDelayNext property.
- You can jump to any row number by typing the number in the navigator position area in C1DbNavigator1. You can also use arrow keys and mouse wheel in that area to move to next or previous rows. HOME and END keys move to the first and last row. PAGE UP and PAGE DOWN keys or mouse wheel with CTRL key pressed to page through the rowset.
- If you change the value in a grid cell and press **Cancel**, the modified value will revert to the original value.
- You can add new rows with the **Add** button and delete rows with the **Delete** button.
- You can send changes to the database with **Update** button and re-fetch data from the database (discarding the changes you may have made) with the **Refresh** button.

Using C1TextBox for Date-Time Input

When editing date-time data with the property DateTimeInput set to **True**, the C1TextBox and C1DateEdit controls work in a special mode. In this mode, instead of editing date-time values as regular strings, they are divided into separate fields for month, day, hour, and so on. It looks similar to the standard DateTimePicker control. However, C1TextBox and C1DateEdit support more formats than the standard DateTimePicker control and have many additional features, such as time zone adjustment, culture-dependent date formatting, and so on.

1. Create a new Windows Application project. Place the following components on the form as shown in the figure:
    - C1ExpressTable1 (C1.Data.Express.C1ExpressTable)
    - C1Label1-9 (C1.Win.C1Input.C1Label)
    - C1TextBox1-5 (C1.Win.C1Input.C1TextBox)
    - C1DbNavigator1 (C1.Win.C1Input.C1DbNavigator)
    - Button1 (System.Windows.Forms.Button)


Using C1TextBox for Date/Time Input
OrderID:
C1Label2
Order Date:
C1TextBox1
Required Date:
C1TextBox2
Shipped Date/Time (in local time zone):
C1TextBox3
Shipped Date/Time (stored in Mountain Standard Time - GMT-07:00):
C1Label7
Enter date using Thai Buddist calendar:
C1TextBox4
Time in custom format (HH:mm:ss:fff):
C1TextBox5
Close
C1ExpressTable1

2. From the Properties window, set the following properties for the C1Label and **Button** controls:

Control	Property	Value
C1Label1	TextDetached	True
	Text	OrderID:
C1Label3	TextDetached	True
	Text	Order Date:
C1Label4	TextDetached	True
	Text	Required Date:
C1Label5	TextDetached	True
	Text	Shipped Date/Time (in local time zone):
C1Labe6	TextDetached	True

	Text	Shipped Date/Time (stored in Mountain Standard Time - GMT-07:00):
C1Label8	TextDetached	True
	Text	Enter date using Thai Buddist calendar:
C1Label9	TextDetached	True
	Text	Time in custom format (HH:mm:ss:fff):
Button1	Name	btnClose
	Text	Close

3. Add the following to the **ConnectionString** property of the C1ExpressTable1 control: **Provider=Microsoft.Jet.OLEDB.4.0;Data Source="Documents\ComponentOne Samples\Common\C1NWind.mdb"**.
4. For the **C1ExpressTable1** component, open the **DbTableName** property combo box and select **Orders** from the database table list.

   Also, you need to set the **AutoIncrement** property of the *OrderID* field of C1ExpressTable1, otherwise you will not be able to update data in the database.

5. Select **C1ExpressTable1**, click the **ellipsis** button in the Properties window for the **Fields** property to open the **Fields** dialog box, select the *OrderID* field and set its **AutoIncrement** property to **ClientAndServer**. This is necessary because *OrderID* is an autoincrement field in the database, so it must be treated accordingly by C1DataExpress.
6. Bind controls to the data source by setting the following properties:

Control	Property	Value
C1DbNavigator1	DataSource	C1ExpressTable1
C1Label2	DataSource	C1ExpressTable1
	DataField	OrderID
C1TextBox1	DataSource	C1ExpressTable1
	DataField	OrderDate
C1TextBox2	DataSource	C1ExpressTable1
	DataField	RequiredDate
C1TextBox3	DataSource	C1ExpressTable1
	DataField	ShippedDate
C1Label7	DataSource	C1ExpressTable1
	DataField	ShippedDate

   Set up the navigator control **C1DbNavigator1**:

7. Set its PositionVisible property to **False** and ColorButtons property to **True**.
8. Expanding the VisibleButtons property, set the following flags to **True**: **First**, **Previous**, **Next**, **Last**, **Apply**, **Cancel**, **Update**, and **Refresh**.
9. Create the following event handlers (see the Data Navigation and Actions Using C1DbNavigator tutorial for details):

   To write code in Visual Basic

Visual Basic

```
Private Sub C1DbNavigator1_RefreshData(ByVal sender As Object, ByVal e As
System.EventArgs) Handles C1DbNavigator1.RefreshData
    C1ExpressTable1.ExpressConnection.Fill()
End Sub

Private Sub C1DbNavigator1_UpdateData(ByVal sender As Object, ByVal e As
System.EventArgs) Handles C1DbNavigator1.UpdateData
    C1ExpressTable1.ExpressConnection.Update()
End Sub
```

To write code in C#

C#

```
private void C1DbNavigator1_RefreshData(object sender, System.EventArgs e)
{
    C1ExpressTable1.ExpressConnection.Fill();
}

private void C1DbNavigator1_UpdateData(object sender, System.EventArgs e)
{
    C1ExpressTable1.ExpressConnection.Update();
}
```

10. In **C1TextBox1**, we will represent the OrderDate in a full format, with full day of week, day, full month name, and year. To specify that, set the **FormatType** property to **LongDate**.

    Since we leave the DateTimeInput property with its default value **True**, we will get date-time editing with separate fields for day of week, month, day, and year at run time. In the **C1TextBox2** control, set the **FormatType** to **ShortDate**.

    LongDate and ShortDate are examples of standard .NET formats. Their actual representation depends on the regional settings controlled by the Culture property.

11. The **C1TextBox3** control demonstrates how you can represent a date in different formats depending on whether it is in edit or display mode (whether it has the input focus). We will make it LongDateShortTime when the control does not have input focus, and in edit mode, when it has input focus, we will use custom format with time zone. To begin with the custom format, set the following properties:

Control	Property	Value
C1TextBox3	FormatType	CustomFormat
	CustomFormat	MM/dd/yyyy h:mm tt zzz

This format is the control's default format (where tt is the AM/PM designator and zzz is the time zone representation). We can override it, and other format-related properties, for display and/or edit mode using properties DisplayFormat and EditFormat. Expand the DisplayFormat property, then expand its sub-property (Inherit) and select **False** for the **FormatType** flag. That will break the inheritance from control for the FormatType property and allow us to change it. Set this property, and also set the FormatType property for **C1Label7** at this time to the following:

Control	Property	Value
C1TextBox3	DisplayFormat.FormatType	LongDateShortTime

C1Label7	FormatType	LongDateShortTime

12. Specify how NULL values (DBNull) are displayed in **C1TextBox3** and in **C1Label7**. Set the following properties:

Control	Property	Value
C1TextBox3	NullText	(not shipped yet)
	EmptyAsNull	True
C1Label7	NullText	(null)

By default, clearing the control (select all and pressing Delete) reverts it to the value it had before editing once the control loses focus. We changed this behavior setting the EmptyAsNull property to **True**, now clearing the control sets its value to DBNull.

13. The **C1TextBox3** control also demonstrates the time zone adjustment feature. It is useful in situations where information is entered into the database by operators located in different time zones. Then it is convenient to show and edit dates for each operator in his or her local time, but store the entered dates in the database adjusted to a single, unified time zone. Suppose the database server is located in Mountain Time zone (7 hours to the West from GMT, Greenwich Mean Time).

For the **C1TextBox3** control:

14. To enable time zone adjustment, set the control's CurrentTimeZone property to **False**. This will make the GMTOffset property modifiable.
15. Set the GMTOffset property to **–07:00 (Mountain Standard Time)**. Now C1TextBox3 converts all values to the local time before displaying them, and vice-versa, converts values entered by the user to the Mountain Time zone.
16. The next control, **C1TextBox4** demonstrates how date input can be localized for any culture supported in .NET. Set the Culture property to **Thai (Thailand)** and FormatType to **LongDate**. Then date format, including month names and year number, will be defined by the Thai Buddhist calendar.
17. The last control, **C1TextBox5** allows you to enter precise time with milliseconds. Set the FormatType property to **CustomFormat** and the CustomFormat property to **HH:mm:ss.fff**. Set the DataType property to DateTime.

Since this control is not data bound, to start editing from a non-empty value, set the **Value** property to some date-time value.

Run the program and observe the following:

- By setting the DateTimeInput property to **True**, you can edit date-time fields, such as year, month, day separately. You can type on the keyboard (months are entered numerically even if month name is shown) or use arrow keys or mouse wheel to advance a field. The AM/PM designator can be changed typing the first letter.
- When the C1TextBox3 control takes focus, the date format in it changes. You can specify the time zone in which the date is entered. By default, it is your local time zone. When you enter a date in C1TextBox3 and move focus away from the control, the C1Label8 control below shows the actual value stored in the database (and in the control's Value property). It is adjusted to the Mountain Time zone.

Data Validation

The main feature of C1TextBox that distinguishes it from the standard TextBox control is that it works with typed data. When the user enters something in a C1TextBox control (or its descendant, C1DateEdit or C1NumericEdit), the input string undergoes several transition phases before it becomes a typed value of the Value property. The input string goes through the following transition phases:

1. **Edit mask parsing**

   The first phase is edit mask parsing (if an edit mask is active, see the Masked Input tutorial), extracting the stored content string out of the masked string displayed in the control.

2. **PreValidation**

   The next phase is PreValidation of the input string. This validation is always performed over a string value, regardless of the DataType, since at this time the input string is not yet converted to a typed value (not yet parsed).

3. **Parsing**

   The next phase after pre-validation is parsing, that is, conversion to the required data type specified in the DataType property.

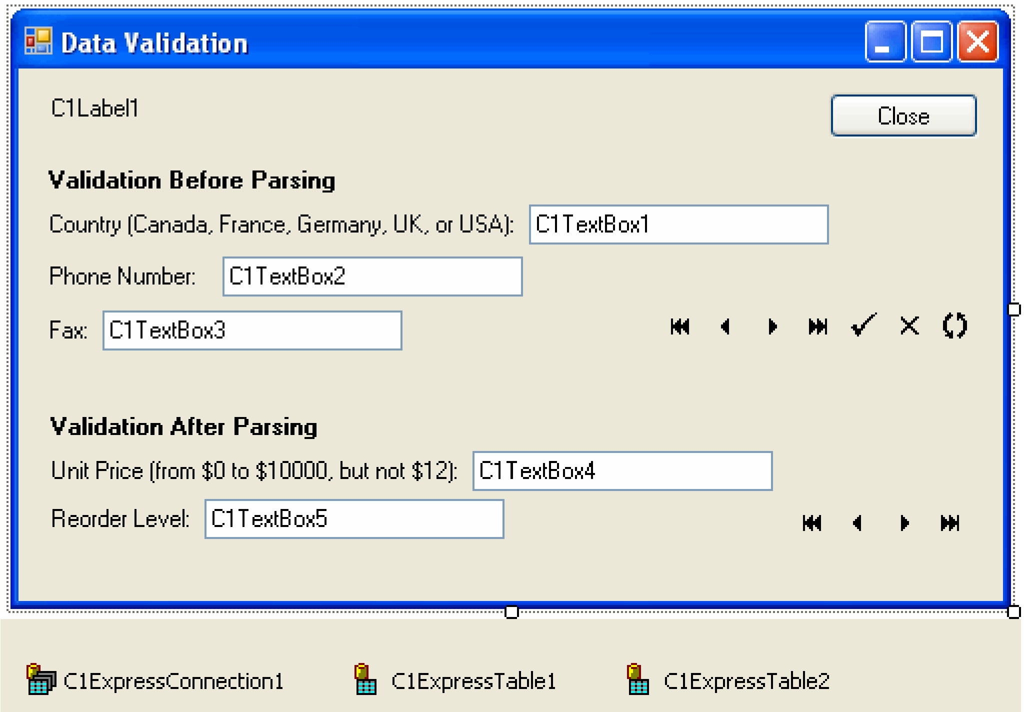
4. **PostValidation**

The last phase before modifying the Value property is PostValidation that is performed over the typed value obtained by parsing. If the value satisfies the PostValidation conditions, it is assigned to the Value property.

Of course, all these phases are optional, except parsing (and that is optional too, if DataType is set to **String**), they only occur if you specify validation conditions in corresponding properties.

This tutorial demonstrates pre-validation and post-validation, that is, how you can perform validation logic before and after parsing the input string.

1. Create a new Windows Application project. Place the following components on the form as shown in the figure:
   - C1ExpressConnection (C1.Data.Express.C1ExpressConnection)
   - C1ExpressTable1-2 (C1.Data.Express.C1ExpressTable)
   - C1Label1-8 (C1.Win.C1Input.C1Label)
   - C1TextBox1-5
   - C1DbNavigator1-2 (C1.Win.C1Input.C1DbNavigator)
   - Button1 (System.Windows.Forms.Button)



2. From the Properties window, set the following properties for the C1Label and **Button** controls:

Control	Property	Value
C1Label2	TextDetached	True
	Text	Validation Before Parsing
C1Label3	TextDetached	True
	Text	Country (Canada, France, Germany, UK, or USA):
C1Label4	TextDetached	True
	Text	Phone Number:
C1Label5	TextDetached	True
	Text	Fax:
C1Labe6	TextDetached	True
	Text	Validation After Parsing
C1Label7	TextDetached	True
	Text	Unit Price (from $0 to $10000, but not $12):
C1Label8	TextDetached	True
	Text	Reorder Level:
Button1	Name	btnClose
	Text	Close

3. Select the **C1ExpressConnection1** component, go to the Properties window, open the **ConnectionString** property combo box. Add the following to the **ConnectionString** property: **Provider=Microsoft.Jet.OLEDB.4.0;Data Source="Documents\ComponentOne Samples\Common\C1NWind.mdb"**.
4. Set the properties of C1ExpressTable1-2 as follows:

Control	Property	Value
C1ExpressTable1	ConnectionComponent	C1ExpressConnection1
	DbTableName	Customers
C1ExpressTable2	ConnectionComponent	C1ExpressConnection1
	DbTableName	Products

5. To bind controls to the data source, set the following properties:

Control	Property	Value
C1DbNavigator1	DataSource	C1ExpressTable1
C1DbNavigator2	DataSource	C1ExpressTable2
C1Label1	DataSource	C1ExpressTable1
	DataField	CompanyName
C1TextBox1	DataSource	C1ExpressTable1
	DataField	Country

C1TextBox2	DataSource	C1ExpressTable1
	DataField	Phone
C1TextBox3	DataSource	C1ExpressTable1
	DataField	Fax
C1TextBox4	DataSource	C1ExpressTable2
	DataField	UnitPrice
C1TextBox5	DataSource	C1ExpressTable2
	DataField	UnitsInStock

6. Set up the navigator controls. For both C1DbNavigator1 and C1DbNavigator2 set the PositionVisible property to **False**. In C1DbNavigator1 show two additional buttons using the VisibleButtons property: **Cancel** and **Apply**.
7. The top part of the form contains controls demonstrating pre-validation, the bottom part demonstrates post-validation. The first editable control, **C1TextBox1** allows you to enter a country name from a list of allowed countries.

   For the **C1TextBox1** control:

8. Expand the PreValidation property of that control and set its PatternString sub-property to **Canada, France, Germany, UK, USA**.
9. The separator string dividing the list to separate items is specified in the ItemSeparator property. Although we could use the default "|", set **ItemSeparator** to ", " (comma and space) for better readability.

   The fact that PatternString represents an exact list of values to match the input string against is indicated by the value of the PreValidation.Validation property set to **ExactList**. This is default, so we do not have to change it.

10. The next control, **C1TextBox2** validates user input using wildcard patterns. The input string is a telephone number. Different countries have different conventional phone numbers formats. We allow input strings in any of the three formats, for US, France and Germany. In this case, the PatternString contains three items, each item a wildcard pattern. An input string satisfies the condition if it matches one of the patterns. Expand the PreValidation property and set its sub-properties to the following:

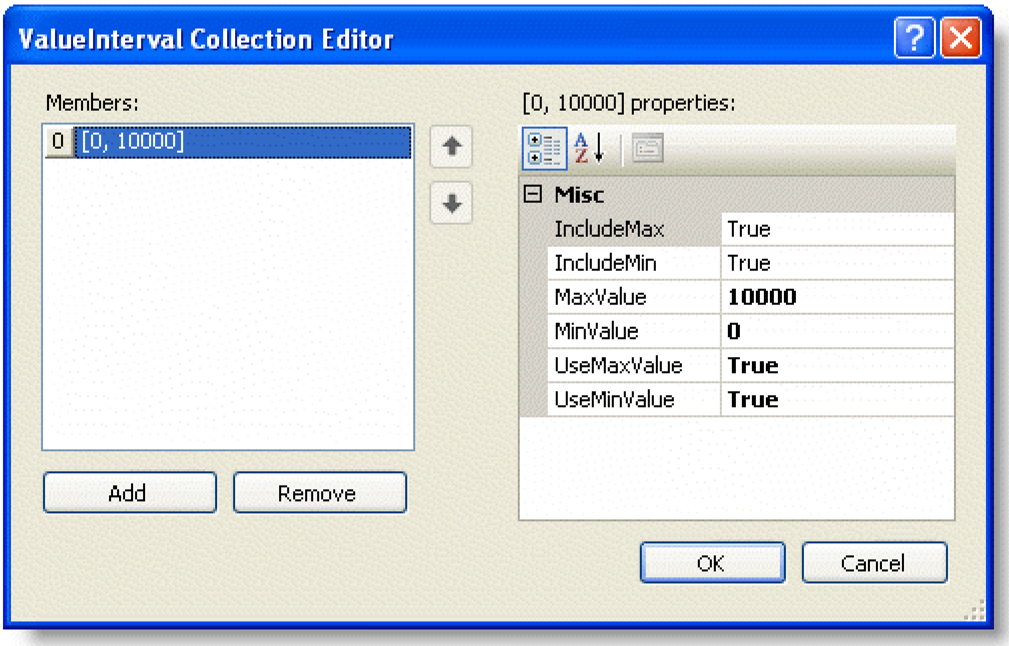
Control	Property	Value
C1TextBox2	PreValidation.Validation	Wildcards
	PreValidation.PatternString	(*) ###-####|##.##.##.##|####-######

11. Wildcard pattern validation demonstrated in the previous step is often not accurate enough. For instance, you cannot enter three digits instead of four in the third pattern. This kind of validation is better performed with regular expressions. .NET regular expressions are a very powerful means of string validation. The **C1TextBox3** control contains a fax number and string validation is specified with the following property settings:

Control	Property	Value
C1TextBox3	PreValidation.Validation	RegexPattern
	PreValidation.PatternString	(\(\d+\))?(\d+-\d+|(\d\d.){3}\d\d)

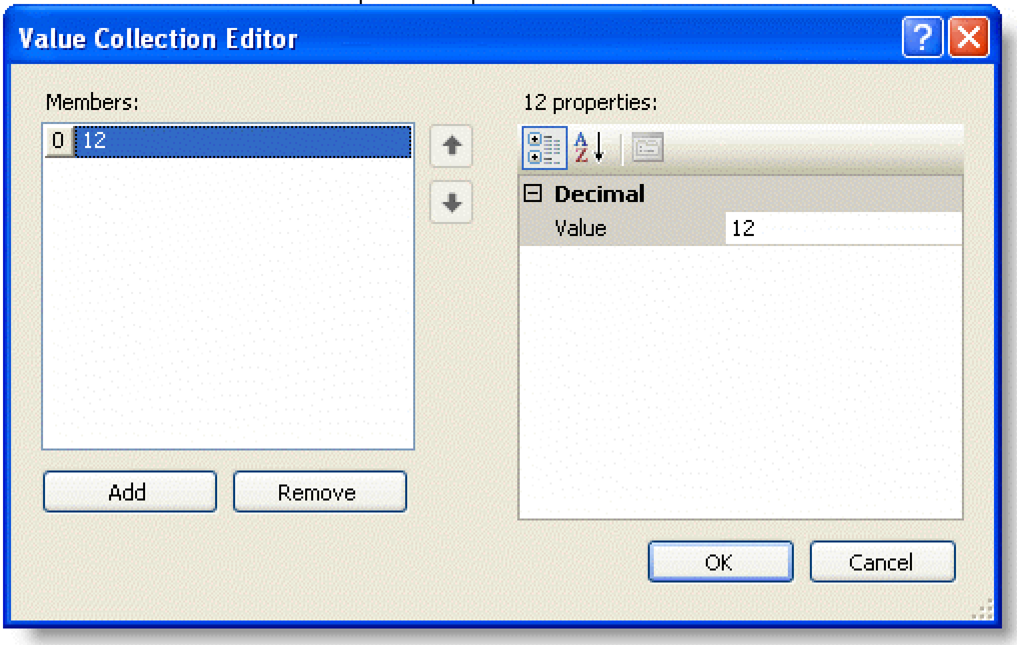
The **C1TextBox4** control validates the typed Value for the UnitPrice field (post-validation). The validation condition is that the price must be between 0 and 10,000 (0 and 10,000 included), and the price of $12.00 is not allowed. To specify such constraints:

12. Expand the PostValidation property and press the **ellipsis** button of the Intervals sub-property to open the **ValueInterval Collection Editor** dialog box.
13. In this dialog box, you can specify one or more intervals. The input value must belong to one of them. Add an interval and set its MinValue to **0** and MaxValue to **10000**.



IncludeMax	True
IncludeMin	True
MaxValue	**10000**
MinValue	**0**
UseMaxValue	**True**
UseMinValue	**True**

14. Press **OK** to close the dialog box.
15. Now specify the excluded value, 12.00. Press the **ellipsis** button of the ValuesExcluded property, add **12** in the **Value Collection Editor** that opens and press **OK** to close the editor.



Value	12

16. The **C1TextBox4** control also demonstrates another important feature unrelated to validation. Normally, we want to display a currency value in a Currency format (with dollar sign), but edit it as a simple number (no dollar sign). This is easy to do in C1Input. Just recall from the Using C1TextBox for Date-Time Input tutorial that you can specify different formats for display and edit mode. Set the FormatType to **Currency**.
17. In **C1TextBox5** control validation is performed programmatically, in the **PostValidating** event code. Set the Validation property to **PostValidatingEvent** and create the following event handler for the PostValidating

event:

To write code in Visual Basic

Visual Basic

```
Private Sub C1TextBox5_PostValidating(ByVal sender As Object, ByVal e As
C1.Win.C1Input.PostValidationEventArgs) Handles C1TextBox5.PostValidating
    If (CType(e.Value, Int16) < 0) Then
        e.ErrorInfo.ErrorMessage = "Value cannot be less than zero."
    ElseIf (CType(e.Value, Int16) > 5000) Then
        e.ErrorInfo.ErrorMessage = "Value cannot be greater than 5000."
    Else
        Return
    End If
    e.Succeeded = False
End Sub
```

To write code in C#

C#

```
private void C1TextBox5_PostValidating(object sender,
C1.Win.C1Input.PostValidationEventArgs e)
{
    if ((Int16)e.Value < 0)
        e.ErrorInfo.ErrorMessage = "Value cannot be less than zero.";
    else if ((Int16)e.Value > 5000)
        e.ErrorInfo.ErrorMessage = "Value cannot be greater than 5000.";
    else
        return;
    e.Succeeded = false;
}
```

This code verifies that the value lies between 0 and 5000. In case of an error, it sets the e.ErrorMessage property that makes the specified message text appear in the error message after the event. Setting e.Suceeded to False indicates that the code has detected a validation error.

Run the program and observe the following:

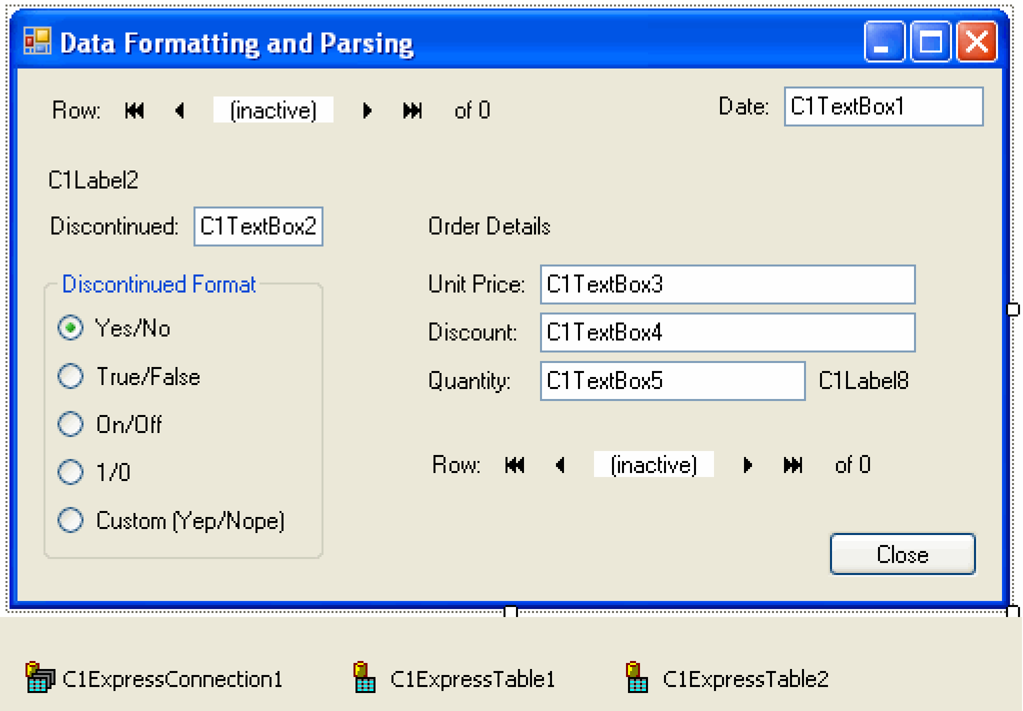
When any one of the validation conditions are not satisfied, an attempt to leave the control with incorrect value shows an error message box with caption "C1Input Validation Error" and a brief error description. In the C1TextBox5 control, the error message is one of the two specified in the code.

Data Formatting and Parsing

This tutorial demonstrates some options in formatting and parsing data.

1. Create a new Windows Application project. Place the following components on the form as shown in the figure:
    - C1ExpressConnection (C1.Data.Express.C1ExpressConnection)
    - C1ExpressTable1-2 (C1.Data.Express.C1ExpressTable)
    - C1Label1-8 (C1.Win.C1Input.C1Label)
    - C1TextBox1-5 (C1.Win.C1Input.C1TextBox)

- C1DbNavigator1-2 (C1.Win.C1Input.C1DbNavigator)
- GroupBox1 (System.Windows.Forms.GroupBox)
- RadioButton1-5 (all of the type System.Windows.Forms.RadioButton)
- Button1 (System.Windows.Forms.Button)



2. From the Properties window, set the following properties for the C1Label, **RadioButton**, and **Button** controls:

Control	Property	Value
C1Label1	TextDetached	True
	Text	Date:
C1Label3	TextDetached	True
	Text	Discontinued:
C1Label4	TextDetached	True
	Text	Order Details
C1Label5	TextDetached	True
	Text	Unit Price:
C1Labe6	TextDetached	True
	Text	Discount:
C1Label7	TextDetached	True
	Text	Quantity:
RadioButton1	Name	rbYesNo
	Text	Yes/No
	Checked	True
RadioButton2	Name	rbTrueFalse

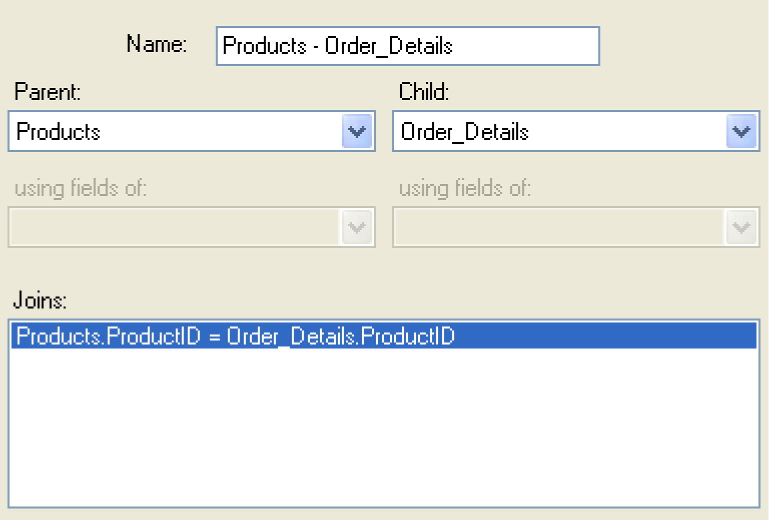
Control	Property	Value
	Text	True/False
RadioButton3	Name	rbOnOff
	Text	On/Off
RadioButton4	Name	rbOneZero
	Text	1/0
RadioButton5	Name	rbCustom
	Text	Custom (Yep/Nope)
Button1	Name	btnClose
	Text	Close

3. Select the C1ExpressConnection1 component, go to the Properties window, open the **ConnectionString** property combo box. Add the following to the **ConnectionString** property: **Provider=Microsoft.Jet.OLEDB.4.0;Data Source="Documents\ComponentOne Samples\Common \C1NWind.mdb."**.
4. Set the properties of C1ExpressTable1-2 as follows:

Control	Property	Value
C1ExpressTable1	ConnectionComponent	C1ExpressConnection1
	DbTableName	Products
C1ExpressTable2	ConnectionComponent	C1ExpressConnection1
	DbTableName	Order Details

Now, create a master-detail relation between C1ExpressTable1 (master) and C1ExpressTable2 (detail):

5. Click the **ellipsis** button in the **Relations** property of the C1ExpressionConnection1 control to open the **Relations** dialog box.
6. In the dialog box, select **Products** for Parent, **Order Details** for Child, add a join with ProductID for both Parent field and Child field, and press **OK** to close the **Relations** dialog box.



Name: Products - Order_Details

Parent: Products

Child: Order_Details

Joins:

Products.ProductID = Order_Details.ProductID

7. To bind controls to the data source, set the following properties:

Control	Property	Value
C1DbNavigator1	DataSource	C1ExpressConnection1
	DataMember	_Products
C1DbNavigator2	DataSource	C1ExpressConnection1
	DataMember	_Products.Products - Order_Details
C1Label2	DataSource	C1ExpressConnection1
	DataField	_Products.ProductName
C1TextBox2	DataSource	C1ExpressConnection1
	DataField	_Products.Discontinued
C1TextBox3	DataSource	C1ExpressConnection1
	DataField	_Products.Products - Order_Details.UnitPrice
C1TextBox4	DataSource	C1ExpressConnection1
	DataField	_Products.Products - Order_Details.Discount
C1TextBox5	DataSource	C1ExpressConnection1
	DataField	_Products.Products - Order_Details.Quantity
C1Label8	DataSource	C1ExpressConnection1
	DataField	_Products.Products - Order_Details.Quantity

8. For the **C1TextBox2** control, bound to the Discontinued field, set the FormatType property to **YesNo**. This is a Boolean format showing "Yes" for **True** and "No" for **False**.

   This will be the default format of the **C1TextBox2** control, but we will also make provisions for a custom format, setting the CustomFormat property of the **C1TextBox2** control to **Yep|Nope**. This custom format will be enabled when the FormatType property is set to **CustomFormat**, which will be done with option buttons in the next step.

9. To switch the *Discontinued* field format between various Boolean formats, use the radio buttons. Assign the following event handlers to the radio buttons' **Click** events:

   To write code in Visual Basic

   Visual Basic

```
Private Sub rbYesNo_Click(ByVal sender  As Object, ByVal e As System.EventArgs)
Handles rbYesNo.Click
    Me.C1TextBox2.FormatType = C1.Win.C1Input.FormatTypeEnum.YesNo
End Sub

Private Sub rbTrueFalse_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles rbTrueFalse.Click
    Me.C1TextBox2.FormatType = C1.Win.C1Input.FormatTypeEnum.TrueFalse
```

```
End Sub

Private Sub rbOnOff_Click(ByVal sender As Object, ByVal e As System.EventArgs)
Handles rbOnOff.Click
    Me.C1TextBox2.FormatType = C1.Win.C1Input.FormatTypeEnum.OnOff
End Sub

Private Sub rbOneZero_Click(ByVal sender As Object, ByVal e As System.EventArgs)
Handles rbOneZero.Click
    Me.C1TextBox2.FormatType = C1.Win.C1Input.FormatTypeEnum.Integer
End Sub

Private Sub rbCustom_Click(ByVal sender As Object, ByVal e As System.EventArgs)
Handles rbCustom.Click
    Me.C1TextBox2.FormatType = C1.Win.C1Input.FormatTypeEnum.CustomFormat
End Sub
```

To write code in C#

C#

```
private void radioButton_Click(object sender, System.EventArgs e)
{
    switch (((RadioButton)sender).Name)
    {
        case "rbYesNo":
            this.c1TextBox2.FormatType = C1.Win.C1Input.FormatTypeEnum.YesNo;
            break;
        case "rbTrueFalse":
            this.c1TextBox2.FormatType =
C1.Win.C1Input.FormatTypeEnum.TrueFalse;
            break;
        case "rbOnOff":
            this.c1TextBox2.FormatType = C1.Win.C1Input.FormatTypeEnum.OnOff;
            break;
        case "rbOneZero":
            this.c1TextBox2.FormatType = C1.Win.C1Input.FormatTypeEnum.Integer;
            break;
        case "rbCustom":
            this.c1TextBox2.FormatType =
                C1.Win.C1Input.FormatTypeEnum.CustomFormat;
            break;
    }
}
```

10. For **C1TextBox3** bound to the *UnitPrice* field, set the FormatType to **Currency**. The currency format with dollar sign is used both in display and edit mode. For C1TextBox4 bound to *Discount*, set the FormatType to **Percent** (values are displayed and entered times 100 with percent sign). Also, for **C1TextBox5** bound to *Quantity*, set the FormatType to **Hexadecimal**, so it will be represented in hexadecimal form (think of it as a kind of low-tech encryption; we, of course, need this rather unusual format for purely demonstrational purposes).
11. Now, to decipher the hexadecimal *Quantity*, we want to convert it to the normal decimal representation and display the decimal number in **C1Label8** with suffix “(base 10)”. There is no standard or custom format for such representation, but we can define our own formatting in code, in the Formatting event. If we set the

FormatType to **UseEvent**, our code can perform whatever formatting we need. Create the following event handler:

To write code in Visual Basic

Visual Basic

```
Private Sub C1Label8_Formatting(ByVal sender As Object, ByVal e As
C1.Win.C1Input.FormatEventArgs) Handles C1Label8.Formatting
    e.Text = e.Value.ToString() + " (base10)"
End Sub
```

To write code in C#

C#

```
private void C1Label8_Formatting(object sender, C1.Win.C1Input.FormatEventArgs
e)
{
    e.Text = e.Value.ToString() + " (base10)";
}
```

In the **C1TextBox1** control we will show how you can allow entering dates in multiple formats. Complete the following tasks:

12. Since this control is unbound, you need to set its DataType property manually; therefore, set the DataType property to **DateTime**.
13. Initialize its value with the current date in code, in 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
    Me.C1TextBox1.Value = DateTime.Now
End Sub
```

To write code in C#

C#

```
private void Form1_Load(object sender, System.EventArgs e)
{
    this.c1TextBox1.Value = DateTime.Now;
}
```

14. The DateTimeInput mode (see the Using C1TextBox for Date-Time Input tutorial) only works with a single date format, so we need to turn it off by setting the DateTimeInput property to **False** for the **C1TextBox1** control. Set the custom format as follows:

Control	Property	Value
C1TextBox1	DateTimeInput	False
	FormatType	CustomFormat
	CustomFormat	MM/dd/yyyy|d-

Control	Property	Value
		MMM-yyyy|d.M.yy

This is a list of allowed formats separated with '**|**'. Parsing an input string, all formats will be tried until a matching one is found. Formatting values for display, we obviously need a single format, so only the first format will be used for formatting.

We also want the control to display the date in LongDate format when not in focus.

15. Expand the DisplayFormat property, and set FormatType to **LongDate**.

    Now the control shows its date value in long format when not in focus, in the first short format when in focus, and the user can type data in any of the three allowed sort formats.

Run the program and observe the following:

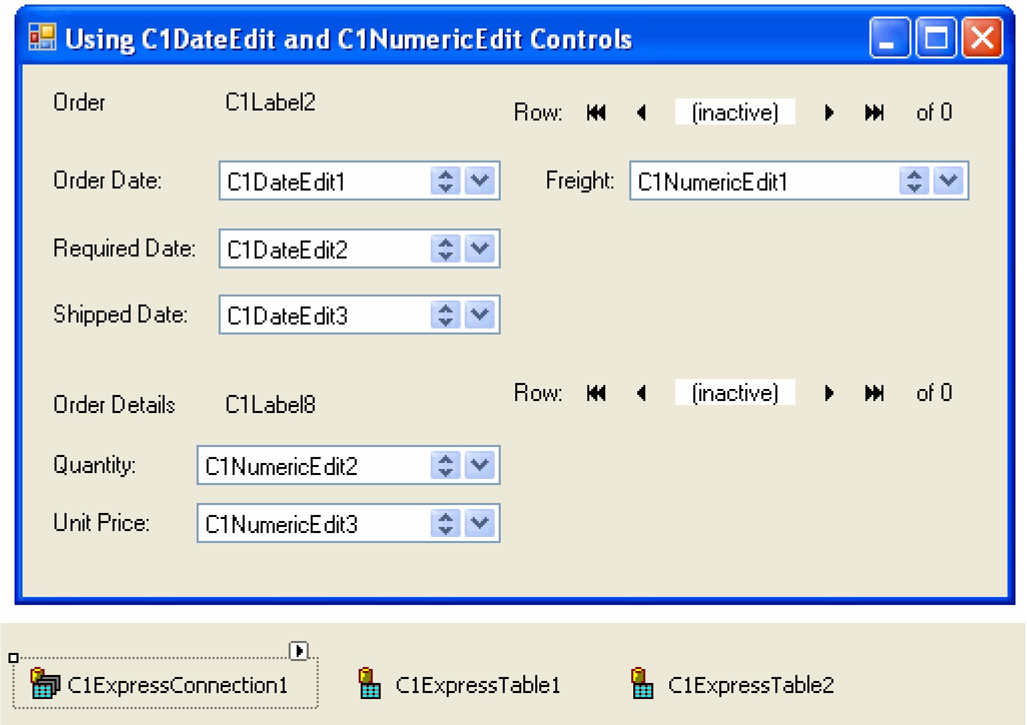
- By selecting different option buttons you can see how Discontinued is shown in different formats. The currently selected format also determines how Discontinued values can be entered by the user. Note also that you can enter Boolean values in abbreviated form; just the initial letter is enough for all formats except On/Off where you need to enter the first two letters. Using the 1/0 format, entering any number except 0 results in a **True** value, which corresponds to the standard .NET conversion from integer to Boolean.
- In the UnitPrice field you can enter or omit the dollar sign. Negative numbers can be entered both with minus sign and in parentheses, according to financial conventions.
- The Discount field allows you to enter the optional percent sign. When you enter a value in percent, it is automatically divided by 100 when it is saved in the data source.
- You can enter the date in the control at the bottom in any of the three allowed formats, as it is most convenient to you. For instance June 5, 2002 can be entered as 06/05/2002, or 5-Jun-2002, or 5.6.02.

Using C1DateEdit and C1NumericEdit Controls

The **C1DateEdit** control provides enhanced date-time editing capabilities. It derives from **C1TextBox**, so it supports DateTimeInput with separate fields for year, month, date, and so on, and all options for formatting and parsing date-time values. In addition to that, it enables the user to select date in a drop-down calendar, and also to increment/decrement date fields in DateTimeInput mode with up/down buttons.

The **C1NumericEdit** control is specialized for numeric input. It also derives from **C1TextBox**, which enables it with formatting, parsing and validation functionality. In addition to that, it allows the user to increment/decrement the value by a specified amount using the up/down buttons, and to use a drop-down calculator to calculate values.

1. Create a new Windows Application project. Place the following components on the form as shown in the figure:
    - C1ExpressConnection (C1.Data.Express.C1ExpressConnection)
    - C1ExpressTable1-2 (C1.Data.Express.C1ExpressTable)
    - C1Label1-10 (C1.Win.C1Input.C1Label)
    - C1DateEdit1-3 (C1.Win.C1Input.C1DateEdit)
    - C1NumericEdit1-3 (C1.Win.C1Input.C1NumericEdit)
    - C1DbNavigator1-2 (C1.Win.C1Input.C1DbNavigator)



2. From the Properties window, set the following properties for the C1Label controls:

Control	Property	Value
C1Label1	TextDetached	True
	Text	Order
C1Label3	TextDetached	True
	Text	Order Date:
C1Label4	TextDetached	True
	Text	Freight:
C1Label5	TextDetached	True
	Text	Required Date:
C1Labe6	TextDetached	True
	Text	Shipped Date:
C1Label7	TextDetached	True
	Text	Order Details
C1Label9	Name	True
	Text	Quantity:
C1Label10	Name	True
	Text	Unit Price:

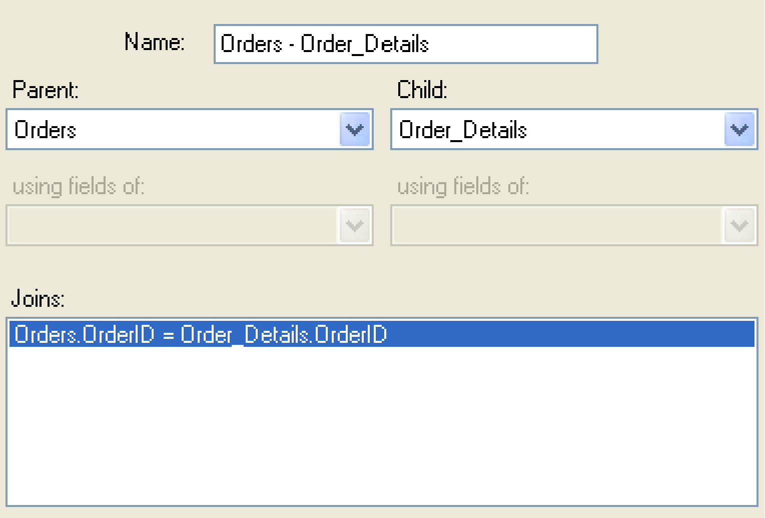
3. Select the C1ExpressConnection1 component, go to the Properties window, open the **ConnectionString** property combo box. Add the following to the **ConnectionString** property: **Provider=Microsoft.Jet.OLEDB.4.0;Data Source="Documents\ComponentOne Samples\Common\C1NWind.mdb"**.

4. Set the properties of C1ExpressTable1-2 as follows:

Control	Property	Value
C1ExpressTable1	ConnectionComponent	C1ExpressConnection1
	DbTableName	Orders
C1ExpressTable2	ConnectionComponent	C1ExpressConnection1
	DbTableName	Order Details

Create a master-detail relation between C1ExpressTable1 (master) and C1ExpressTable2 (detail):

5. Press the **ellipsis** button in the **Relations** property of the C1ExpressionConnection1 control to open the **Relations** dialog box.
6. In the dialog box, select **Orders** for Parent, **Order Details** for Child, add a join with OrderID for both Parent field and Child field, press **OK** to close the **Relations** dialog box.



Name: Orders - Order_Details

Parent: Orders

Child: Order_Details

Joins:

Orders.OrderID = Order_Details.OrderID

7. To bind controls to the data source, set the following properties:

Control	Property	Value
C1DbNavigator1	DataSource	C1ExpressConnection1
	DataMember	_Orders
C1DbNavigator2	DataSource	C1ExpressConnection1
	DataMember	_Orders.Orders - Order Details
C1Label2	DataSource	C1ExpressConnection1
	DataField	_Orders.OrderID
C1DateEdit1	DataSource	C1ExpressConnection1
	DataField	_Orders.OrderDate
C1DateEdit2	DataSource	C1ExpressConnection1
	DataField	_Orders.RequiredDate
C1DateEdit3	DataSource	C1ExpressConnection1
	DataField	_Orders.ShippedDate
C1NumericEdit1	DataSource	C1ExpressConnection1

Control	Property	Value
	DataField	_Orders.Freight
C1Label8	DataSource	C1ExpressConnection1
	DataField	_Orders.Orders - Order Details.ProductID
C1NumericEdit2	DataSource	C1ExpressConnection1
	DataField	_Orders.Orders - Order Details.Quantity
C1NumericEdit3	DataSource	C1ExpressConnection1
	DataField	_Orders.Orders - Order Details.UnitPrice

Set up the *OrderDate* field (control **C1DateEdit1**) for a simple date input with a drop-down calendar as the main means of entering a date.

8. Turn off the DateTimeInput mode setting the DateTimeInput property to **False**.
9. Make the calendar center-aligned relative to the control by setting the DropDownAlign property to **Center**.

The next control, **C1DateEdit2** (*RequiredDate* field) will serve as a regular date-time input control, in DateTimeInput mode, without a drop-down calendar.

10. Set the VisibleButtons property to **UpDown**.
11. To use the long date format, set the FormatType property to **LongDate**.

The *ShippedDate* field (control **C1DateEdit3**) will contain dates in MediumDate format , with up/down buttons and a drop-down calendar. Set its FormatType property to **MediumDate**.

12. Since ShippedDate can be NULL (DBNull), set the EmptyAsNull property to **True**. Then clearing the control contents will be equivalent to entering null value.
13. Suppose we want certain days to appear bold in the drop-down calendar. To do that, expand the Calendar property, select the calendar property **MonthlyBoldedDates** and add two dates to the array using the **Collection Editor** dialog box.

Now we will set up controls for numeric input. The **C1NumericEdit1** control is bound to the *Freight* field. To set up the **C1NumericEdit1** control:

14. Set its FormatType to **Currency**.
15. Then hide the up/down buttons since it does not make much sense for them to be used in a currency value; set the VisibleButtons property to **UpDown**.
16. Set the **TextAlign** property to **Right**.
17. Also set the DropDownAlign property to **Right** to make the drop-down calculator right-aligned as well.

The **C1NumericEdit2** control is bound to the *Quantity* field.

18. Since the *Quantity* field does not need a drop-down calculator, hide it by setting the VisibleButtons property to **UpDown**.
19. Since *Quantity* is an integer number, set the FormatType property to **Integer**.
20. Set the increment for up/down buttons to a certain value you prefer, for example, to **3**.
21. You can also change the control's appearance if you feel inclined to, for example, setting the **BorderStyle** property to **FixedSingle**.

The last field (**C1NumericEdit3**) allows the user to enter a currency value *UnitPrice* using a drop-down calculator.

22. Set the FormatType property to **Currency**.
23. As in the **C1NumericEdit2** control, set the **BorderStyle** property to **FixedSingle**, just to see that you have all the usual appearance options at your disposal.
24. Hide the up/down buttons (unnecessary for a currency field) by setting the VisibleButtons property to **UpDown**.
25. To center the drop-down calendar relative to the control, set the DropDownAlign property to **Center**.

Run the program and observe the following:

- The *OrderDate* field allows you to enter a date from the keyboard, or open the drop-down calendar and select a date there with the mouse or arrow keys. In the drop-down calendar you can see the buttons for changing the month and year. At the bottom of the calendar you see the **Clear** and **Today** button. The **Clear** button sets the Value to DBNull, and the **Today** button sets it to the today's date (that can be overridden in the Calendar.TodayDate property).
- Select a date field in *RequiredDate* and press the up/down buttons. Notice that you can enter a date without using keyboard.
- In the drop-down calendar in *ShippedDate*, you can see two bolded dates.
- Opening the drop-down calculator in the *Freight* and *Quantity* fields, you can see different button styles in the calculator.

Custom Drop-Down Form

C1DropDownControl is a control derived from C1TextBox, so it supports all its formatting, validation, and other features. Like other two C1TextBox-derived controls, it also supports up/down (spin) and drop-down buttons. However, unlike those specialized controls, C1DropDownControl allows you to attach your own logic to the up/down buttons and your own drop-down form/editor to the drop-down button.

In this tutorial we will create a form that can be used to show a list of MRU (most recently used) values. Note that although we will use this form in only one control, the same form without modification can be used in any number of controls in your projects.

> **Note:** Before opening Form1 at design time, you need to compile the tutorial project. Otherwise a "class not found" error will appear. This is because the drop-down form class needs to be compiled to become available in the **DropDownFormClass** property of a C1DropDownControl.

To create a custom drop-down form, complete the following steps:

1. Create a new Windows Application project. Place a C1DropDownControl (C1DropDownControl1) on the form.

   Now, add a form derived from drop-down form to your project.

2. From the Solution Explorer, right-click the project and select **Add | New Item** from the menu.
3. In the **Add New Item** dialog box, select **Windows Form** from the list of **Templates** in the right pane.
4. Then enter **MRUDropDown.vb** for Visual Basic (**MRUDropDown.cs** for C#) in the **Name** box and click **Add**.
5. The next step is to replace the following class definition line(s) in the MRUDropDown form code. Select **View | Code** from the menu and replace the code below:

   To write code in Visual Basic

   Visual Basic

```
Public Class MRUDropDown
    Inherits System.Windows.Forms.Form
```

To write code in C#

C#

```
public partial class MRUDropDown : System.Windows.Forms.Form
```

with:

To write code in Visual Basic

Visual Basic

```
Public Class MRUDropDown
    Inherits C1.Win.C1Input.DropDownForm
```

To write code in C#

C#

```
public partial class MRUDropDown : C1.Win.C1Input.DropDownForm
```

6. From the Properties window, expand the **Options** property node of the MRUDropDown form.
7. Set Option.**Focusable** to **False** and Option.**AutoResize** to **True**.

   The **AutoResize** option will make the width of the drop-down always equal the width of the control, and **Focusable** option set to **False** is needed because we do not want the drop-down form to take input focus. Instead, we want focus to remain in the control so the user can type in the control and select from the drop-down at the same time.

8. Place a ListBox control (**ListBox1**) on the drop-down form. To make the list box occupy the whole drop-down area, set the following ListBox1 properties:

Control	Property	Value
ListBox1	Dock	Fill
	IntegralHeight	False
	BorderStyle	None

9. To make the form open automatically when the user starts typing in the control, use the OwnerControlTextChanged event, add the following event handler:

   To write code in Visual Basic

Visual Basic

```
Private Sub MRUDropDown_OwnerControlTextChanged(ByVal sender As Object, ByVal e
As System.EventArgs) Handles MyBase.OwnerControlTextChanged
    OwnerControl.OpenDropDown()
    ListBox1.SelectedIndex = listBox1.FindString(OwnerControl.Text)
End Sub
```

To write code in C#

C#

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

```
{
    OwnerControl.OpenDropDown();
    listBox1.SelectedIndex = listBox1.FindString(OwnerControl.Text);
}
```

The second line selects the list box item corresponding to the current control text, if such item already exists.

10. To enable visual feedback while the user moves the mouse inside the list box, add the following event handler:

To write code in Visual Basic

Visual Basic

```
Private Sub ListBox1_MouseMove(ByVal sender As Object,ByVal e As
System.Windows.Forms.MouseEventArgs) Handles ListBox1.MouseMove
    ListBox1.SelectedIndex = ListBox1.IndexFromPoint(e.X, e.Y)
End Sub
```

To write code in C#

C#

```
private void listBox1_MouseMove(object sender,
System.Windows.Forms.MouseEventArgs e)
{
    listBox1.SelectedIndex = listBox1.IndexFromPoint(e.X, e.Y);
}
```

11. To enable the user to navigate the drop-down list box with the UP ARROW and DOWN ARROW keys and delete items with CTRL+DEL key, add the following event handler:

To write code in Visual Basic

Visual Basic

```
Private Sub MRUDropDown_KeyDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.KeyEventArgs) Handles MyBase.KeyDown
    If e.Modifiers = Keys.None And e.KeyCode = Keys.Up Then
        If ListBox1.SelectedIndex > 0 Then
            ListBox1.SelectedIndex = listBox1.SelectedIndex - 1
            OwnerControl.Text = listBox1.Text
            OwnerControl.SelectAll()
        End If
        e.Handled = True
    End If
    If e.Modifiers = Keys.None And e.KeyCode = Keys.Down Then
        If ListBox1.SelectedIndex < listBox1.Items.Count - 1 Then
            ListBox1.SelectedIndex = listBox1.SelectedIndex + 1
            OwnerControl.Text = listBox1.Text
            OwnerControl.SelectAll()
        End If
        e.Handled = True
    End If
    If e.Modifiers = Keys.Control And e.KeyCode = Keys.Delete And
ListBox1.SelectedIndex >= 0 Then
```

```
        ListBox1.Items.RemoveAt(listBox1.SelectedIndex)
        e.Handled = True
    End If
End Sub
```

To write code in C#

C#

```
private void MRUDropDown_KeyDown(object sender,
System.Windows.Forms.KeyEventArgs e)
{
    if (e.Modifiers == Keys.None && e.KeyCode == Keys.Up)
    {
        if (listBox1.SelectedIndex > 0)
        {
            listBox1.SelectedIndex--;
            OwnerControl.Text = listBox1.Text;
            OwnerControl.SelectAll();
        }
        e.Handled = true;
    }
    if (e.Modifiers == Keys.None && e.KeyCode == Keys.Down)
    {
        if (listBox1.SelectedIndex < listBox1.Items.Count - 1)
        {
            listBox1.SelectedIndex++;
            OwnerControl.Text = listBox1.Text;
            OwnerControl.SelectAll();
        }
        e.Handled = true;
    }
    if (e.Modifiers == Keys.Control && e.KeyCode == Keys.Delete &&
listBox1.SelectedIndex >= 0)
    {
        listBox1.Items.RemoveAt(listBox1.SelectedIndex);
        e.Handled = true;
    }
}
```

12. To select an item and close the drop-down form when the user clicks a list box item, add the following event handler:

To write code in Visual Basic

Visual Basic

```
Private Sub ListBox1_MouseDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles ListBox1.MouseDown
    ListBox1.SelectedIndex = listBox1.IndexFromPoint(e.X, e.Y)
    CloseDropDown(True)
End Sub
```

To write code in C#

C#

```
private void listBox1_MouseDown(object sender,
System.Windows.Forms.MouseEventArgs e)
{
    listBox1.SelectedIndex = listBox1.IndexFromPoint(e.X, e.Y);
    CloseDropDown(true);
}
```

13. To make the drop-down form actually change the control text when it is closed after the user clicks an item, add the following event handler for the PostChanges event:

    To write code in Visual Basic

    Visual Basic

    ```
    Private Sub MRUDropDown_PostChanges(ByVal sender As Object, ByVal e As
    System.EventArgs) Handles MyBase.PostChanges
        If ListBox1.SelectedIndex >= 0 Then
            OwnerControl.Value = listBox1.Text
        ElseIf ListBox1.FindStringExact(OwnerControl.Text) < 0 Then
            ListBox1.Items.Add(OwnerControl.Text)
        End If
    End Sub
    ```

    To write code in C#

    C#

    ```
    private void MRUDropDown_PostChanges(object sender, System.EventArgs e)
    {
        if (listBox1.SelectedIndex >= 0)
            OwnerControl.Value = listBox1.Text;
        else if (listBox1.FindStringExact(OwnerControl.Text) < 0)
            listBox1.Items.Add(OwnerControl.Text);
    }
    ```

    Now the drop-down form is ready and we can use it in the **C1DropDownControl1** in Form1.

14. Open Form1, select **C1DropDownContro1**, go to the Properties window and select the **DropDownFormClassName** property. This property allows you to select a drop-down form-derived form from your project.
15. Select **<Project Name>.MRUDropDown** from the combo box. This is all you need to attach the drop-down form to a control. If needed, you can attach this form to any number of controls the same way.

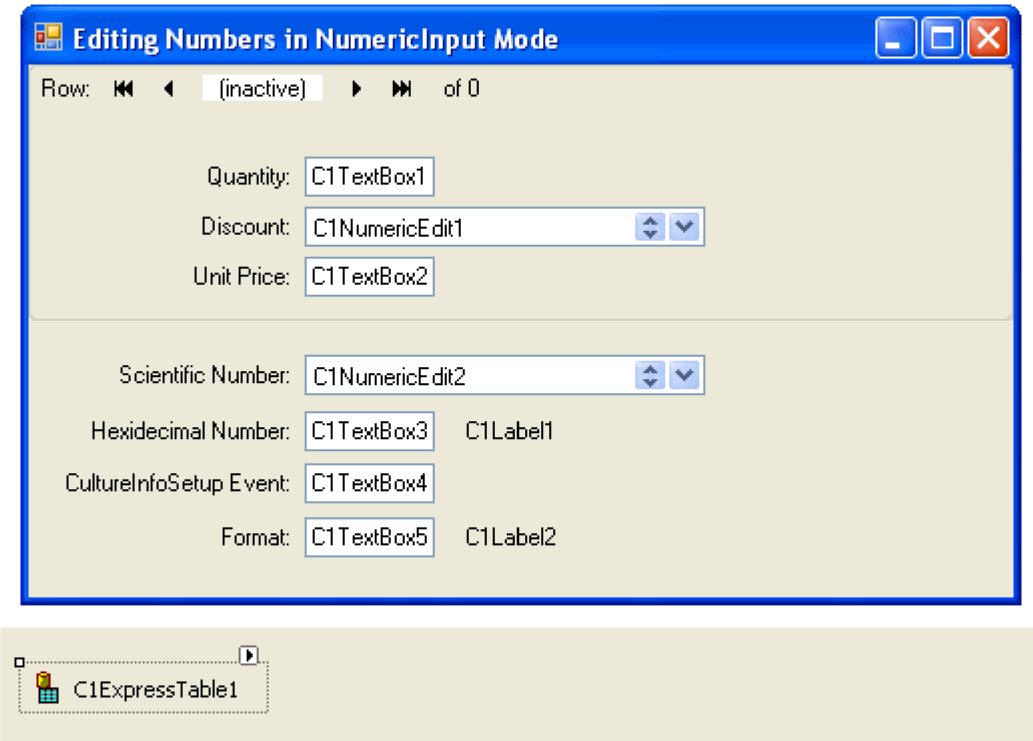
Run the program and observe the following:

- When you start typing, it automatically opens the drop-down list. The drop-down can also be opened with the drop-down button.
- After you type something in the control and press ENTER, the next time you open the drop-down you see the typed item in the list.
- You can navigate the list with the mouse and with UP/DOWN keys, although the list drop-down does not have input focus, the focus remains in the control input area.
- Clicking a drop-down item with the mouse changes the control text.

Editing Numbers in NumericInput Mode

In this tutorial, you will learn how to use NumericInput mode for editing numeric data. C1TextBox and C1NumericEdit controls have a NumericInput property that is **True** by default. If the NumericInput property is set to **True** and DataType is a numeric type, the control functions in numeric mode facilitating typing of digits, decimal point and other numeric characters.

1. Create a new Windows Application project. Place the following components on the form as shown in the figure:
    - C1ExpressTable1 (C1.Data.Express.C1ExpressTable)
    - Label1-7 (System.Windows.Forms.Label)
    - C1DbNavigator1 (C1.Win.C1Input.C1DbNavigator)
    - C1TextBox1-5 (C1.Win.C1Input.C1TextBox)
    - C1Label1-2 (C1.Win.Input.C1Label)
    - C1NumericEdit1-2 (C1.Win.C1Input.C1NumericEdit)



2. From the Properties window, set the following properties for the **Label** and C1DbNavigator controls:

Control	Property	Value
Label1	Text	Quantity:
Label2	Text	Discount:
Label3	Text	Unit Price:
Label4	Text	Scientific Number:
Label5	Text	Hexidecimal Number:
Label6	Text	CultureInfoSetup Event:
Label7	Text	Format:

Control	Property	Value
C1DbNavigator1	Dock	Top

3. Select the **C1ExpressTable1** component, go to the Properties window, open the **ConnectionString** property. Add the following to the **ConnectionString** property: **Provider=Microsoft.Jet.OLEDB.4.0;Data Source="Documents\ComponentOne Samples\Common\C1NWind.mdb"**.
4. For the **C1ExpressTable1** component, open the **DbTableName** property combo box and select **Order Details** from the database table list.
5. To bind controls to the data source, set the following properties:

Control	Property	Value
C1DbNavigator1	DataSource	C1ExpressTable1
C1TextBox1	DataSource	C1ExpressTable1
	DataField	Quantity
C1NumericEdit1	DataSource	C1ExpressTable1
	DataField	Discount
C1TextBox2	DataSource	C1ExpressTable1
	DataField	UnitPrice

6. Adjust some appearance properties of the controls:
   - Set **BorderStyle** to **FixedSingle** for the controls: C1Label1-2, C1TextBox1-5, C1NumericEdit1-2.
   - Expand the VisibleButtons property of C1DbNavigator1 and make two additional buttons visible: **Apply** and **Cancel**.
   - Expand the **UIStrings** property of C1DbNavigator1 and change **Row:** to **Order Item:**.
   - In **C1NumericEdit1**, set the VisibleButtons property to **UpDown** (this control will have only up/down buttons, no drop-down). And vice versa, in C1NumericEdit2 we want to have only drop-down calculator, no up/down buttons, so we set its VisibleButtons property to **DropDown**.
7. **C1TextBox1** represents an integer *Quantity* field. As an integer, it contains only digits, without decimal point and exponent. To specify integer format, set the FormatType to **Integer**.
8. **C1NumericEdit1** is used for editing a *Discount* field that is represented as a percentage value. This is specified by setting FormatType to **Percent**. Also, specify the increment to be used to increase/decrease the value by a fixed amount using the up/down buttons: Increment to **0.0005**. That will make the control's up/down button increment/decrement the value by 0.05%.
9. **C1TextBox2** is used to display and edit a *UnitPrice* field of Decimal type. To show it in currency format, with dollar sign, set FormatType to **Currency**. In this case, the currency sign shown in the value is defined by the current culture setting controlled by Culture property. If Culture is set to **(Current Culture)**, current system regional settings are used.

   There are several unbound (not bound to a data source) controls in the bottom part of the form. **C1NumericEdit2** edits numbers in scientific format, with decimal point and exponent. This control also includes a drop-down calculator.

10. Set DataType to **Decimal** (the default).
11. Set FormatType to **Scientific**. By default, a number in scientific format contains six digits after decimal point and three digits in the exponent. If you need a non-default format, it can be defined in the CustomFormat property.
12. The **C1TextBox3** control edits numbers in hexadecimal format. Set the DataType property to **Int16** (16-bit integer number).
13. Then set the FormatType property to **Hexadecimal**. In this mode, the user will be able to type only digits 0-9, letters A-F, and minus sign (or change the sign of the number pressing F9).
14. The C1Label control (**C1Label1**) adjacent to **C1TextBox3** shows the entered number in the usual decimal format. To connect it to the number, set C1Label1.DataType to **Int16** and add an event handler for the

ValueChanged event of the **C1TextBox3** control:

To write code in Visual Basic

Visual Basic

```
Private Sub C1TextBox3_ValueChanged( ByVal sender As Object, ByVal e As
System.EventArgs) Handles C1TextBox3.ValueChanged
    Try
        C1Label1.Value = C1TextBox3.Value
    Catch
    End Try
End Sub
```

To write code in C#

C#

```
private void c1TextBox3_ValueChanged(object sender, System.EventArgs e)
{
    try
    {
        c1Label1.Value = c1TextBox3.Value;
    }
    catch
    {
    }
}
```

The **C1TextBox4** control demonstrates how you can change number format by changing culture settings used in formatting. Set the DataType property to **Int32**

15. The set the FormatType property to **StandardNumber**. A number in this format has a sign, thousands separators and decimal point. Settings used for this formatting, such as the sign, thousand separator, and decimal point characters, are defined by the current culture settings controlled by the Culture property.
16. You can specify non-default culture settings handling the CultureInfoSetup event.

For example, we can show negative numbers with the word “minus” instead of the usual “-“. We will also suppress the fractional part of the number, and use “|” as thousands separator. We will also change the group sizes (so they are no longer thousands), make the first (rightmost) group contain one digit, second – two digits, third – three digits, and all the rest (higher) digits make a single group. This is done by the following event handler:

To write code in Visual Basic

Visual Basic

```
Private Sub C1TextBox4_CultureInfoSetup(ByVal sender As Object, ByVal e As
C1.Win.C1Input.CultureInfoSetupEventArgs) Handles C1TextBox4.CultureInfoSetup
    Dim ci As System.Globalization.CultureInfo = e.CultureInfo
    ci.NumberFormat.NegativeSign = " minus "
    ci.NumberFormat.NumberGroupSeparator = "|"
    ci.NumberFormat.NumberGroupSizes = New Integer() {1, 2, 3, 0}
    ci.NumberFormat.NumberDecimalDigits = 0
End Sub
```

To write code in C#

C#

```
private void c1TextBox4_CultureInfoSetup(object sender,
C1.Win.C1Input.CultureInfoSetupEventArgs e)
{
    System.Globalization.CultureInfo ci = e.CultureInfo;
    ci.NumberFormat.NegativeSign = " minus ";
    ci.NumberFormat.NumberGroupSeparator = "|";
    ci.NumberFormat.NumberGroupSizes = new int[] {1, 2, 3, 0};
    ci.NumberFormat.NumberDecimalDigits = 0;
}
```

C1TextBox5 shows how to specify a custom, non-standard number format.

17. Set DataType to **Decimal**, FormatType to **CustomFormat** and CustomFormat to "#,##0,;(###0,);zero".

    In this format, three colon-separated parts define three different formats used, correspondingly, for positive, negative numbers and zero. Character '0' denotes a required digit, character '#' specifies an optional digit, and ',' is the thousands separator. Thousands separator on the right means the number is stored as times 1000 the entered number (per separator).

18. To see the number that is stored when we enter a number in **C1TextBox5**, we use a C1Label control (**C1Label2**) adjacent to **C1TextBox5**. Set C1Label2.DataType to **Decimal** and add an event handler for the ValueChanged event of the **C1TextBox5** control:

    To write code in Visual Basic

    Visual Basic

    ```
    Private Sub C1TextBox5_ValueChanged(ByVal sender As Object, ByVal e As
    System.EventArgs) Handles C1TextBox5.ValueChanged
        C1Label2.Value = C1TextBox5.Value
    End Sub
    ```

    To write code in C#

    C#

    ```
    private void c1TextBox5_ValueChanged(object sender, System.EventArgs e)
    {
        c1Label2.Value = c1TextBox5.Value;
    }
    ```

Run the program and observe the following:

- The user cannot enter arbitrary text in NumericInput mode. For example, if you type a letter, it is ignored. Only characters allowed by the specified format are permitted.
- Using properties FormatType and CustomFormat, you can specify any standard or custom format supported in .NET (for details see Numeric Format Strings in the .NET documentation). All these formats are used both for display and editing of numeric values.
- Changing culture settings, you can control literal characters used in formatting and various other format settings.