CodeControls™ 3

User Interface Controls
for Windows

Sequiter® Software Inc.

© Copyright Sequiter Software Inc., 1988-1997. All rights reserved.

No part of this publication may be reproduced, or transmitted in any form or by any means without the written permission of Sequiter Software Inc. The software described by this publication is furnished under a license agreement, and may be used or copied only in accordance with the terms of that agreement.

The manual and associated software is sold with no warranties, either expressed or implied, regarding its merchantability or fitness for any particular purpose. The information in this manual is subject to change without notice and does not represent a commitment on the part of Sequiter Software Inc.

CodeBase® and Sequiter® are registered trademarks of Sequiter Software Inc.

CodeBase++™, CodeBasic™, CodeControls™ and CodeReporter™ are trademarks of Sequiter Software Inc.

All other brand and product names are trademarks or registered trademarks of their respective holders.

The use of Windows in this manual refers to Microsoft Windows.

Contents

1 Introduction

CodeControls is a set of Windows custom ActiveX controls that are designed to make your user interface tasks easier. CodeControls combines the functionality of standard Windows combo boxes, edit controls, buttons, sliders and list boxes with our unique data entry abilities.

CodeControls works with several popular Windows interface design tools, including Microsoft Visual C++ Developer Studio, Visual Basic, Delphi and Borland C++ Builder. Use your favorite visual design tool to interactively set CodeControls' properties such as font, 3D appearance, control color and mask template characters. You can manipulate the controls through their documented properties, methods and events.

With CodeControls, programmers can design formatted data entry applications without writing a single line of code. Use CodeControls' input masking feature to ensure that strings, dates and numbers are entered and stored in the correct format.

CodeControls are also "data aware". Our controls can be linked to data files through our master control. Data file information can be reflected in any control linked to a master control. Any changes made to the controls' contents are also made in the appropriate data file and vice versa. Since CodeControls expands upon the capabilities of standard Windows controls, you can use our combo box to automatically look up records or use a list box to browse a database. The master control also automates other database management capabilities such as skipping, seeking, appending records and much more.

Furthermore, CodeControls interacts seamlessly with CodeBase technology to provide dBASE functionality and file compatibility. CodeControls lets you access and change the data, index and memo files of dBASE, FoxPro and Clipper. This file compatibility also extends to multi-user platforms where CodeControls applications can share records and files with concurrently running dBASE, FoxPro, Clipper and CodeBase applications.

Requirements

CodeControls can be used with four different development environments under Windows 95 or Windows NT 4.0.

- Microsoft Visual Basic
- Microsoft Visual C++
- Borland C++ Builder
- Borland Delphi

The requirements for each environment are described below:

Microsoft Visual Basic

To use CodeControls with Visual Basic, you need CodeBase 6.3 (or greater) for Visual Basic. You should also have some knowledge of how to use ActiveX controls under Visual Basic.

Microsoft Visual C++ To use CodeControls with Visual C++, you need CodeBase 6.3 (or greater) for C or C++. You should also have some knowledge of how to use ActiveX controls under Visual C++.

Borland C++ Builder To use CodeControls with the Borland C++ Builder, you need CodeBase 6.3 (or greater) for C or C++. You should also have some knowledge of how to use ActiveX controls under C++ Builder.

Borland Delphi To use CodeControls with Borland Delphi, you need CodeBase 6.3 (or greater) for Delphi and some knowledge of how to program under Delphi.

In all four cases, some database knowledge is useful, but not essential. See the *CodeBase User's Guide* for information on database records, fields, index files and tags.

The hardware requirements for CodeControls depend more on the application you plan to write than on CodeControls itself. In general, any computer able to run Windows 95 or Windows NT should be able to run applications that utilize CodeControls.

Manual Conventions

This manual uses several conventions to distinguish between the various language constructs. These conventions are listed below:

- A CodeControls' control name is always shown highlighted, as follows: e.g. **CbMaster**
- Function parameters are displayed in italics: e.g. *setting*, *fileName*
- The name "CodeControls" refers to the actual CodeControls custom controls.
- There are four versions or the CodeBase User's Guide: *CodeBase User's Guide C Edition, C++ Edition, Visual Basic Edition and the Delphi Edition*. This manual will refer to the user's guide as simply "the *CodeBase User's Guide*". Similarly, there are four versions of the reference guide, which will be referred to as "the *CodeBase Reference Guide*".
- The various development tools that support CodeControls are referred to collectively as "application development tools". These tools include Visual Basic, Visual C++, Delphi and Borland C++ Builder.
- "CodeBase Technology Products" or "CodeBase" refer to CodeBase for C, C++, Visual Basic and Delphi collectively.

2 Control Description

OLE controls

CodeControls is made up of six custom ActiveX controls.

ActiveX controls are OLE controls, which can be used by any OLE container that supports OLE controls. OLE controls are accessed through their properties and methods and they can fire events. Methods are similar to the member functions of a C++ class and they can be called in the same way. Properties are similar to the member variables of a C++ class and they are exposed to allow the container and the control communicate. The control fires events to notify the container that an important action has occurred. OLE controls also support persistent properties. The values of the properties for a control are saved to a stream or file. This allows the property values to be read from storage every time a new instance of the control is created.

CodeControls provides six custom OLE controls:

1. The CodeControls **CbButton** Button Control
2. The CodeControls **CbCombo** Combo Box
3. The CodeControls **CbEdit** Edit Control
4. The CodeControls **CbList** List Box
5. The CodeControls **CbSlider** Slider Control
6. The CodeControls **CbMaster** Master Control

The first five controls listed above are similar in many ways to the standard button, combo, edit(text), list and slider controls provided by Windows. CodeControls, however, provides additional capabilities, the most important being data-aware capabilities and input/display masking.

Data Aware

CodeControls may be data aware in that each control may be linked to the information within a data file. The control then automatically displays the contents of the field. In addition, any changes made to the contents displayed by the control are automatically written back to disk.

Input Masking

Input masking ensures that the users of your application enter data according to a predefined "mask". This mask can contain certain characters which represent specific types of data. For example a mask of "999" only allows input of numeric values up to three digits in length.

Display Masking

Display masking ensures that information displayed in CodeControls is formatted in certain ways, according to the mask used for input purposes. Display masking is very versatile, and can perform such tasks as displaying dBASE date field information in customized date formats (e.g. 19930523 as May 23/93).

Master Control

The sixth control listed above, the CodeControls **CbMaster** Master Control, provides built-in data file management capabilities. Some of the more important abilities include: appending and deleting, searching and traversing a data file. The most impressive feature of the **CbMaster** control is that all of these tasks can be accomplished in an application without writing any program code.

Any number of CodeControls **CbButton**, **CbEdit**, **CbCombo**, **CbList** and **CbSlider** controls can be linked to a data file through a **CbMaster** control and thus have access to these capabilities. The process of linking individual CodeControls to a **CbMaster** control is called *binding*. Such a control is said to be *bound* to a **CbMaster** control.

CbButton

A **CbButton** control can be used to execute a particular action when the button is pressed. You can specify the action of the button by using the DataAction property, which can take on the value of a default action such as appending, skipping a record and so on, or you may customize the action. In addition, you may customize the look of the button. **CbButton** supplies two sets of bitmaps to match the default actions supplied by the button or you may use your own bitmaps or choose text to be placed on the button.

CbEdit

A **CbEdit** control is similar to the Windows standard edit control. As shown in Figure 2.1, **CbEdit** controls appear as a rectangle into which a user can enter and edit text.

Figure 2.1 **CbEdit** control.

In addition to all of the capabilities found in the standard Windows edit controls, a **CbEdit** control has two major features.

First, a **CbEdit** control can be bound to a field in a data file. The **CbEdit** displays the contents of this field for the control's current record. Any changes made to the contents of the control are also made to the appropriate record's field in the data file.

Second, the **CbEdit** control uses input and display masking to ensure information is formatted correctly.

See the "CodeControls Concepts" and "Programming CodeControls" chapter for more information on linking the **CbEdit** control (and other CodeControls) to data files.

CbList

A **CbList** control is similar to the Windows standard list box control. **CbList** controls display a list of text strings in a rectangular frame. The user can select an item from the list using either the mouse or the arrow keys. The entries in the **CbList** cannot be changed by the programmer when the **CbList** is bound to a **CbMaster** control.

Figure 2.2 **CbList** control.

A **CbList** control can be linked to a data file via a dBASE expression that is evaluated for each record in the file. This expression is commonly a field name but may also contain the dBASE functions supported by CodeBase. (See the "Appendix C: dBASE Expressions" chapter of the CodeBase *Reference Guide*). The results of this expression are displayed as entries in the list box.

Like the **CbEdit** control, the CodeControls **CbList** control also uses display masking capabilities to change the way that information is presented to the user.

For information on the effects of binding a **CbList** to a **CbMaster** control, see the "CodeControls Concepts" and the "Programming CodeControls" chapter.

NOTE

A **CbList** control linked to a data file can display many more entries than a standard list box can under Windows 95. A standard list box has a limit of 32,767 entries of data under Windows 95.

When linked to a data file, a **CbList** control can display up to 1 billion entries. (If not linked to a data file, the **CbList** control has the same limits as a standard list box.)

CbCombo

A **CbCombo** control acts much the same as a standard combo box control. It combines a list box with either a static, display only edit control, or a fully functional edit control into which the user can type.

This manual refers to the top section of the **CbCombo** control as the "edit portion of the control", but keep in mind that this may either be a fully functional edit control, or a display only control.

If the **CbCombo** control's edit portion is static, the control displays the list box's currently selected item (if there is one). If the edit portion is not static, the user can type in text and the **CbCombo** searches for matching text in the list portion.

A **CbCombo** control may be one of the following three types:

Simple

The Simple style combines a functional edit control with a list box. The list box portion is always visible, as shown in Figure 2.3 a.

Figure 2.3 a **CbCombo** with Simple style.

Drop Down

The Drop Down style is the same as the Simple style, except that the visibility of the list box portion is under the end-user's control. The list box is only visible if the user clicks on the control's down arrow.

As with the Simple style, the text in edit portion of the **CbCombo** may be edited by the user and a search of the list entries is performed.

This is the default style.

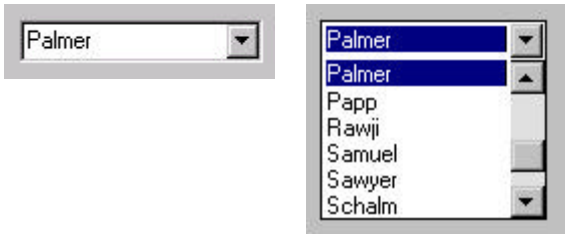


Figure 2.3 b **CbCombo** with Drop Down style.

Drop Down List

This style is similar to the Drop Down style, except that its edit portion is static and cannot be directly changed by the user.

As with the Drop Down style, the display of the Drop Down List style's list box portion is under the control of the user. When the **CbCombo** button is clicked, the list portion is made visible.

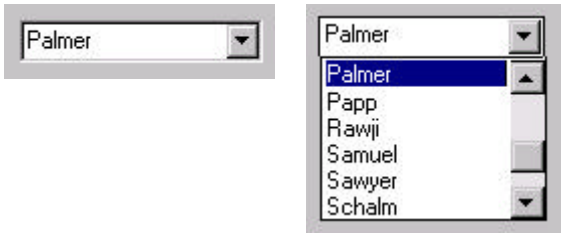


Figure 2.3 c **CbCombo** with Drop Down List style.

CbSlider

A **CbSlider** control is similar to the Windows scroll bar control and it can be oriented vertically or horizontally. A **CbSlider** control can be used in two ways. It can be used to show the position of the current record in the data file or it can be used for data input where the position of the thumb corresponds to a value to be entered into a record field. See the "CodeControls Concepts" and "Programming CodeControls" chapter for more information and examples with **CbSlider**.

CbMaster

A **CbMaster** control is a powerful tool that can be used to perform high level database functions. This control can be both linked to a data file and bound to other CodeControls. Using a **CbMaster** control, you can build a sophisticated database application with little or no coding.

A **CbMaster** control can display the current record number, the tag spin list, a slider to reposition the data file and up to fourteen buttons, each of which is discussed below:

Figure 2.4 **CbMaster** control.

Top: When the Top button is pressed, the first record in the data file becomes the current record.

Previous Page: A page is a constant number of records to skip. The size of a page is determined by the PageSize property. When the Previous Page button is pressed, the **CbMaster** skips to the record that is one page before the current record. The new record subsequently becomes the current record.

Previous: When the Previous button is pressed, the **CbMaster** skips one record backwards and that record becomes the current record.

Search: The Search button invokes the search dialog. This dialog, which is explained below, lets the user locate a particular record in the data file.

Next: When the Next button is pressed, the **CbMaster** skips one record forward and that record becomes the current record.

Next Page: A page is a constant number of records to skip. The size of a page is determined by the PageSize property. When the Next Page button is

pressed, the **CbMaster** skips to the record that is one page after the current record. The new record subsequently becomes the current record.

Bottom: When the Bottom button is pressed, the last record in the data file becomes the current record.

Append: When the Append button is clicked, a new record is appended to the data file. The AppendRecordType property determines what kind of record will be appended. If AppendRecordType is set to 0, a blank record with blank memos is appended to the data file. This enables the user to enter text over 'blank space'. If AppendRecordType is set to 1, then a copy of the current record is appended. In this case, if a record has a memo entry, it's contents are also appended. Finally, if AppendRecordType is set to 2, a copy of the current record is used as above except that any memo entries are blanked out.

The record is actually appended to the data file and the index file(s) are updated when the user moves off the appended record or when the form is unloaded. The user has the option of aborting the append before this point by pressing the Undo button.

Delete: Pressing the Delete button toggles the deletion flag for the current record. The display rectangle on this button turns red when the deletion flag is true. If the current record is not marked for deletion, the display rectangle is gray.

Pack: Pressing the Pack button physically deletes the records that have been marked as deleted. The data file must be opened exclusively for this button to be enabled.

Reindex: Pressing the Reindex button recreates all of the index files associated with the data file. The data file must be opened exclusively for this button to be enabled.

Flush: Pressing the Flush Record button will cause any changes made to the current record to be written to disk immediately.

Refresh: Pressing the Refresh button will reload the record buffer with information from the disk. Any changes to the record buffer will be lost.

Undo: Pressing the Undo button reverses any changes made to the current record. This option is available from the time that changes are made to the current record until the user moves off the current record. The button is disabled as soon as the Undo option becomes invalid.

If no tag has been selected, all positioning via a **CbMaster** control is done according to the file's natural order. If a tag has been selected, the tag ordering is used.

Search Dialog

When the **Search** button on the **CbMaster** control is clicked, the Search Dialog, shown in Figure 2.5, is displayed. The search dialog provides the user with an easy means of searching for a specific record in the data file using either a tag expression or a record number.

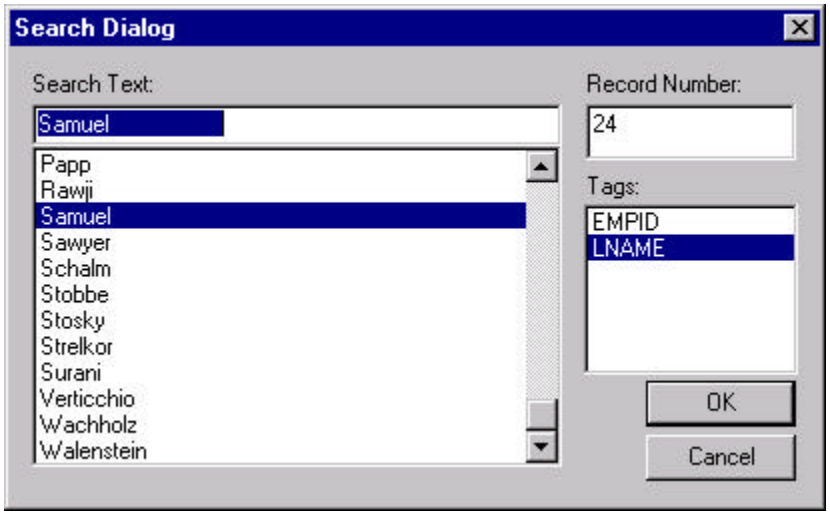


Figure 2.5 Search Dialog

Search Text

The search dialog initially uses the **CbMaster** selected tag. If the **CbMaster** is using natural ordering, the search dialog selects the first tag in the list. The user can scroll through the records in the tag or the user can type into the edit portion and the closest match from the list is scrolled to the top of the list. The user can then select a desired record by using the arrow keys or clicking an entry with the mouse.

NOTE

When the user types in the search key, the format must correspond to the format of the selected tag. For example, if the selected tag is based on a Date field, the search key must also be a date expression.

Tags

The highlighted entry in the "Tags" list box determines what is displayed in the "Search" control.

When a tag is chosen, the "Search" control displays a list of the tag's key values. The entries are displayed in the order according to the tag. For example, if the tag is based upon a field called "LAST_NAME", the contents of this field would be displayed for each record in the file and the entries would be ordered alphabetically.

Record Number

A specific record can also be selected by entering its record number in the "Record Number" edit control. If the entered value is does not exist in the file, the data file is not repositioned.

OK Once a record has been selected by one of the above methods, it can be made the current record by selecting the "OK" button or by pressing the ENTER key. The **CbMaster** is automatically repositioned and the search dialog tag becomes the **CbMaster's** selected tag. If the **CbMaster** was using natural ordering before the search dialog was invoked, then the **CbMaster** does not select the search dialog tag and it continues to use natural ordering.

CANCEL Selecting this button cancels any selection made in the "Search" dialog and returns to the main application. The original record position and selected tag for the **CbMaster** is restored.

3 CodeControls Concepts

This chapter describes some of the features of CodeControls, including input and display masking and 3D effects. In addition, the concepts and skills behind using CodeControls as data aware controls within your dialog boxes is discussed.

Property Pages

OLE custom controls can support multiple property pages. A property page is a graphical interface used to view and change the properties of the control at design time.

CodeControls custom OLE controls were designed with property pages to make building your application easier. Property pages are displayed in a dialog box at design time.

Follow these steps to display a property page under Visual Basic, Delphi and C++ Builder.

1. Right click a CodeControls custom control that has been placed on a form.
2. Click the Properties option from the menu.

Follow these steps if you want to display the property pages under Visual C++.

1. Right click a CodeControls custom control that has been placed on a dialog box.
2. Highlight the Control Object option from the menu. This causes another small menu to pop up.
3. Click the Properties option from the small menu.

Figure 3.1 shows a dialog box displaying the property pages for a **CbMaster** control.

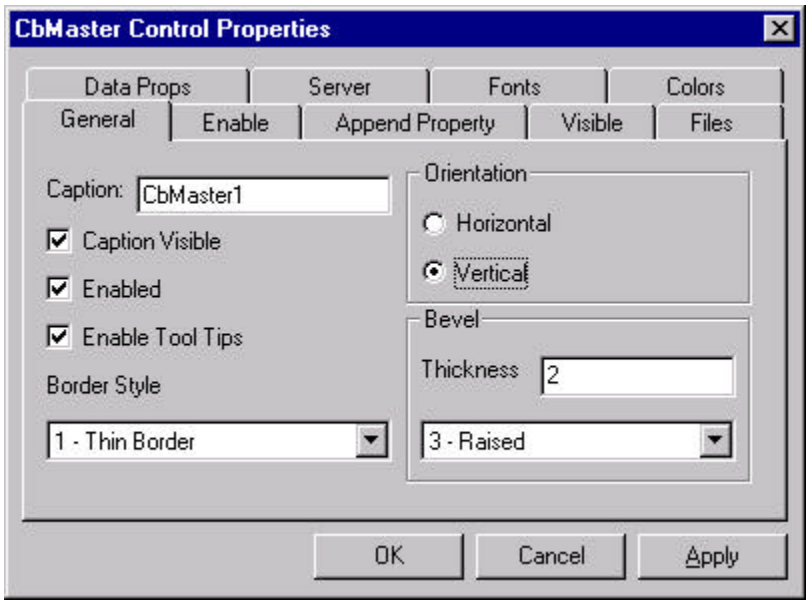


Figure 3.1

To switch to a different property page, click on the property page tab. When you switch to a different property page, the changes made to the previous page are applied to the properties.

The dialog box has three buttons, OK, CANCEL and APPLY. The Apply button is enabled when a property has been changed. When the Apply button is pressed, the changes on the current property page are applied to the properties. Pressing the OK button applies the changes on the current property page and closes the dialog box. The Cancel button, ignores any changes on the current property page and closes the dialog box.

All the controls come with the stock Font and Color property pages, so you can customize the look of CodeControls. Each CodeControls custom control also comes with set of property pages that were tailor made for that particular control.

Input and Display Masking

A *mask* is a string of special formatting characters that can be used to restrict the type and length of information displayed or entered into a control. Mask strings are also referred to as *templates*.

For example, a numeric mask would limit the display of information in a control to numeric data, while preventing a user from changing the contents of a control to anything but numeric data. The length of the mask would determine how many numeric characters could be displayed or entered into the control.

There are two properties associated with the CodeControls input/display masking capabilities; they are FormatType and Mask. The FormatType determines what type of information is stored in the control, while the Mask characters determine the actual format of the data.

Changing Formats

Generally, the formatting properties can be set at design time and run time. An exception to this rule is when the **CbList** or the list portion of the **CbCombo** is not linked to a data file. The formatting properties can not be changed at runtime because of efficiency considerations. If the formatting was changed at runtime, each entry would have to be removed from the list, reformatted and then reentered into the list. It is conceivable that the list could contain thousands of entries, in which case changing the formatting for every entry would be very time consuming.

The formatting can be changed at runtime when the **CbList** or list portion of the **CbCombo** is linked to a data file. This is because the **CbList** and the list portion of the **CbCombo** are implemented as a virtual list boxes. The data file can have millions of records, but only those records that can be displayed in the list box are loaded into memory. Since the entries in the list box are constantly being updated with new records as the user scrolls through the data file, the formatting can easily be changed at runtime.

See the "Properties" chapter for information on the FormatType and Mask properties.

Maximum Length

When masking occurs, the size of the mask string template overrides the explicit maximum length setting for **CbEdit** controls and the edit portion of **CbCombo** controls.

FormatType

CodeControls supports four settings for the FormatType property. The four settings supported FormatType are listed in Table 3.1.

Type	Value	Description
None (Normal)	0	No masking at all. Performs like the standard Windows control. This is the default setting.
Date	1	Used to edit/display date information only. Includes Standard dBASE Format and full month name format.
Generic Mask	2	Used to edit/display any type of information as defined by the input mask formatting characters.
Numeric	3	Used to edit/display numeric information only. Ignores the specified mask in favor of numeric properties. See Table 3.6 for information about the numeric properties.

Table 3.1 FormatType Settings

NOTE

If CodeControls are set for password protection and a FormatType is specified, the password protection aspect of the control is ignored.

Mask Characters

CodeControls uses different masking characters for the different format types. In general the mask contains a character representation for acceptable user entry key presses for each character position in the control. If the control is used for displaying information, this mask may then be used to convert the contents of the control into a more appealing format. For example, the information may be forced into upper case, or dates may be displayed in a preferred format and so on.

Property Pages

A property page provides the user with a graphical interface with which to modify and view the properties of a control at design time. Two property pages are provided with the **CbEdit**, **CbList** and the **CbCombo** controls to specify the formatting properties and numeric properties. These property pages make it easier for the programmer to specify the formatting for the control. The Formatting property page and Numeric Formatting property page for the **CbCombo** are illustrated below.

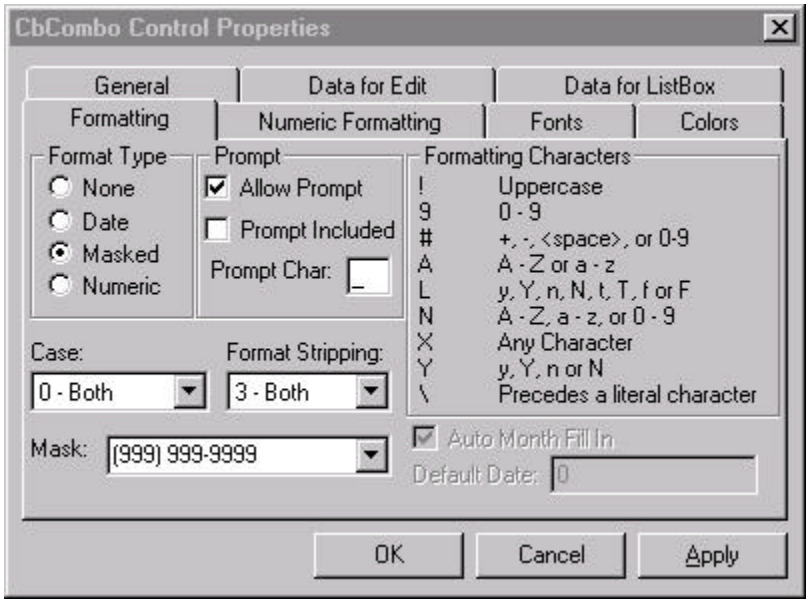


Figure 3.2 Formatting Property Page

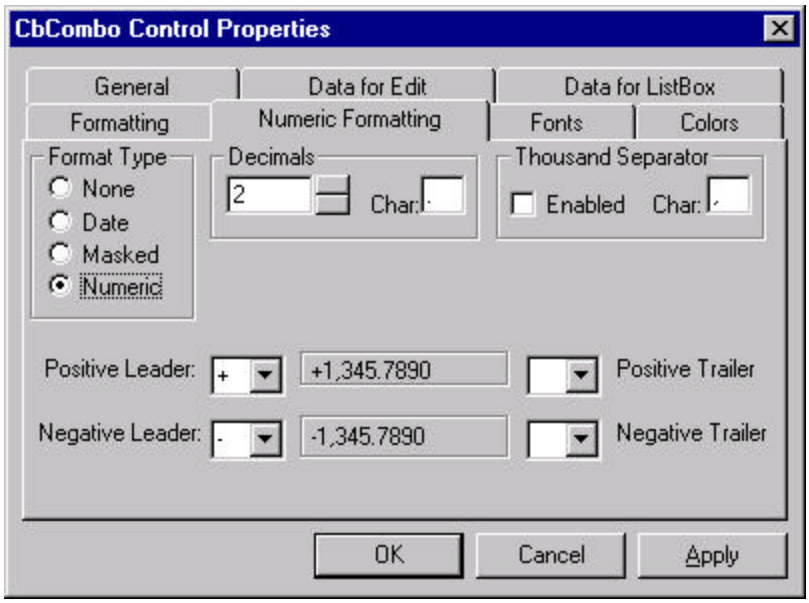


Figure 3.3 Numeric Formatting Property Page

Date Masks

Setting the FormatType to a Date restricts data in the control to valid date information. Dates can be displayed in a variety of ways. "Oct 23, 1960" could be displayed as 10/23/60, 1960/10/23, 19601023 , October 23, 1960, and so on.

The following table shows the formatting characters supported when the FormatType property is set to Date.

Mask Character	Meaning
C or c	Second digit of the century
CC or cc	First two digit of the century
YY or yy	First two digits of the year
DD or dd	Numeric representation of the day
MM or mm	Numeric representation of the month
MMM to MMMMMMMMM or mmm to mmmmmmmmm	Character representation of the month. The number of M's in the mask determines the number of letters of the month displayed.
\	When a backslash character '\' precedes any character, the character is treated as a literal, including any mask characters.

Table 3.2 Formatting characters for Date masks

Any characters in the mask template that are not one of the above special characters are displayed as they appear in the template. (For example the "/" in "MM/DD"). Any character that is preceded by a '\' backslash is also treated as a literal. A backslash literal is specified by two backslashes "\\". These literal characters are, from the user's perspective, permanent and may not be edited.

Date masks that use MMM to MMMMMMMMM for the month provide an extra feature: There is a property AutoMonthFillin which when set to true, automatically fills in the month string after the user enters in the first three characters. CodeControls automatically calculates the appropriate month and supplies the remaining characters. If AutoMonthFillin is set to false, the user must type in every character of the month string and CodeControls checks the spelling as the user types in the month. The language used for the month strings depends on the language used by computer running the application. Therefore if the German version of Windows 95 was installed, the application will use German month strings for the date formatting.

When using date masks, it is important to remember two things:

1. The first occurrence of C, Y, M or D in a date mask that is not preceded by a backslash '\' will be considered as a special masking character.
2. Any subsequent occurrences are treated as literal characters and are simply included in the result as they appear in the mask. In this case, the backslash '\' is not required to precede the mask character. Therefore you cannot have more than one century, year, month or day in your mask.

The following table illustrates some examples, and the result of applying these masks to the date September 30, 1987.

Example		Mask	Result
Date Masks:		MM/DD/YY	09/30/87
		MM/DD/CCYY	09/30/1987
		CCYYMMDD	19870930
		MMM 'YY	Sep '87
		MMMMMMMMM DD,YY	September 30, 1987
	*	Hello MMM DD/ YY	Hello Sep 30/87
	*	Giddy on MMM DD	Gi30ing on Sep DD
	*	Gi\d\dy on MMM DD	Giddy on Sep 30
	*	MMM DD it was giddy.	Sep 30 it was giddy.

Table 3.3 Date Mask Characters

Take particular note of the starred entries in Table 3.3. The first starred example shows that you can include other literal characters in your mask other than C, Y, M and D. The second mask may look a bit peculiar at first, but it illustrates the two points made above about date masks. The third and forth example in the previous table show the proper method for including literal characters that contain one or more of the special mask characters.

dBASE Date Format

If the control is bound to a date field, the format of the date is automatically stored in the data file in the 'CCYYMMDD' format. This format, which is consistent across xBase data files, describes the date with two characters each for the century, year, month and day. For example, September 30, 1987 is stored as "19870930" in a dBASE date field.

Regardless of which formatting characters are used in the control's mask, CodeControls will automatically convert the entered date to the 'CCYYMMDD' format required by xBase date fields if the Format Stripping property is set to Data File Stripping (1) or Text and Data File Stripping (3) (see the Retrieving Information section below).

Default Mask

When you set the FormatType to Date at run time, CodeControls automatically sets the Mask property to a default setting of "CCYY/MM/DD".

DefaultDate property

Sometimes it is desirable to have a date field automatically filled with a default date or to fill in parts of the date automatically. The DefaultDate property allows you to specify a default date. The DefaultDate is an OLE DATE data type. If blank dates or partial dates are allowed in the data file then set the DefaultDate property to 0.0, in which case, the date field is not automatically filled with a default date. If you want the current system date to be automatically assigned to a blank date field, then set DefaultDate to 1.0. If you want a particular date to always be used to fill in blank or partial dates, then set DefaultDate to any other valid date. See "Properties" chapter for more information.

Generic Masks

Setting the FormatType of the control to a Generic Mask (i.e. Alphanumeric) indicates that the control stores a general type of data that does not fit the Date or Numeric categories specifically, but still needs to be formatted.

An example requiring a generic mask is when a North American phone number needs to be displayed. Formatting information for a phone number usually includes area code parenthesis and a hyphen between the local dialing numbers.

Table 3.4 lists the formatting characters supported when using Generic type masking. In addition to performing masking, the characters marked with an asterisk permanently convert the entered value to the formatted value.

	Character	Meaning
*	!	Converts to upper case.
	9	Numeric data (0-9).
	#	Extended numeric data (+, -, <space>, or 0-9).
	A or a	Letters only (A-Z or a-z).
*	L or l	Logical data. (y, Y, n, N, t, T, f, or F) Converts to T or F.
	N or n	Alphanumeric data. (A-Z, a-z, 0-9).
	X or x	Any character
*	Y or y	Yes or No (y, Y, n, or N). Converts to Y and N.
	\	When a backslash character '\' precedes any character, the character is treated as a literal, including any mask characters.

Table 3.4 Formatting characters for Masked mask.

* Conversion characters

Any characters in the mask template that are not one of the above special characters are displayed as they appear in the template (For example the "-" in a phone number mask) and may not be edited or deleted. If you wish to treat a mask character as a literal, precede the character with a backslash '\'. A backslash literal is specified by two backslashes "\\".

Based on Table 3.4, Table 3.5 shows a variety of values that might be entered by a user in a Generically masked edit control, and the results that would actually be displayed by the control, based on the input mask.

Value Entered	Mask	Result
abc	!!!	ABC
123x4	99999	1234
y	Y	Y
y	L	T
ab123	XXXXX	ab123
a1b2c3	AAAAAA	abc
4035551212	(999) 999-9999	(403) 555-1212
abc	\AAAA\Y	AabcY
bbb	\A\aaa\Y	AabbY
abcdef	AA\\AA\\AA	ab\cd\ef

Table 3.5 Format masks and sample input

Numeric Masks

Unlike controls that use a Date or a Generic mask, the Numeric mask for a control is not actually set with the mask characters. Instead, you define the format for numeric data by setting a series of properties either at design time or at run time. Table 3.6 lists these properties:

Property	Description
DecimalChar	Specifies the character that separates the whole part from the fractional part of a number. The default is a period (".")
Decimals	Specifies the number of digits after the decimal point. The default is two.
PositiveLeader NegativeLeader	Specifies the prefix for positive and negative values. The default leader for positive values is a blank space (" "). The default leader for negative values is a negative sign ("-").
PositiveTrailer NegativeTrailer	Specifies the suffix for positive and negative values. The default trailer for both positive and negative values is a null string ("").
ThouSeparatorEnabled	Specifies whether a separator character should be displayed between hundreds and thousands, thousands and millions, etc. The default is set to false.
ThouSeparatorChar	Specifies the character to be displayed if Thousand Separators are used. The default is a comma (",").
NegativeColor	Specifies the color value for negative numbers. The default value is red. This property only applies to **CbEdit** controls.

Table 3.6 Numeric properties

Retrieving Information

Retrieving information from CodeControls is generally done in the same manner as retrieving information from any of the Windows standard controls. However, when masking occurs (see the Input and Display Masking section above), the application developer has two choices: retrieve all formatting characters stored in the mask (e.g. the parentheses and dash in a North American phone number), or omit the formatting characters, and just retrieve the raw data in the control.

The process of removing the formatting characters is called "stripping". Each type of mask (Date, Numeric, Generic) may be stripped when information is retrieved from the control.

When any date mask is stripped from a control, the date reverts to 'CCYYMMDD' format. Characters that are converted by the conversion mask characters in a generic mask are not returned to the originally typed characters when stripping occurs, but are left in their converted state. For example, if "abc" is entered into a control with a mask of "!!!", the value of the control will be "ABC", whether it is stripped or not.

There are three levels of stripping:

1. **Strip for Data File**. When information from a bound control is written to a data file, all formatting characters are removed. To maintain the integrity of date and numeric data file fields, it is important to strip out the formatting characters (such as commas).
2. **Strip for Clipboard** When information is copied from or pasted to the Windows clipboard, the formatting characters are stripped out. This option also effects the standard retrieval and storage of information from and to the control.
3. **Strip in All Cases**. All storage and retrieval of information within the control is stripped of any formatting characters.

Data Aware CodeControls

While CodeControls provides many advantages to the standard Windows controls, perhaps the most unique feature is its ability to automatically store and retrieve information to and from xBase data files. CodeControls can be used to create a sophisticated data entry system, with little or no coding on the part of the application developer.

Each control may be "bound" to a data file so that it automatically displays information from a record within the bound data file. Different controls within the same dialog may reference different records within the same data file or they can act in concert to display different information from the same record.

Not only do they display information, but with the **CbEdit** control and the edit portion of the **CbCombo** control, information entered by the end user is automatically written to the data file -- again without any coding on the part of the application developer.

CbMaster	After linking a **CbMaster** control to a data file, the **CbButton, CbEdit**, **CbList, CbCombo** and **CbSlider** controls can be bound to the **CbMaster**. Any database operations performed through the **CbMaster** control (i.e. using the **CbMaster** buttons) are reflected in all of the bound controls. By default, the **CbMaster** control traverses through the data file in its natural order. A new traversal order can be determined by selecting a tag for the master.
CbButton	The **CbButton** control can be linked to a data file and it can be used to execute individual actions corresponding to buttons on the **CbMaster** control. The **CbButton** control comes with a series of default actions, which include appending, repositioning the data file in various ways, flushing record changes, packing, reindexing, searching, deleting records and the ability to undo record changes.
CbEdit	The **CbEdit** control is the most straightforward of the CodeControls in that it may only be linked to a single field of a single data file. This provides a rapid method of allowing a user to input information directly into the data file. When this ability is combined with the input masking and the abilities of the **CbMaster** control, the **CbEdit** becomes a very useful tool.
CbList	The **CbList** can be used in two ways when it is bound to a **CbMaster**. The **CbList** can be used to reposition the data file or it can be used to assign data to a field. When a **CbList** control is used to reposition the data file, it becomes a powerful tool for quickly browsing through information in the entire data file. **CbList** controls are different from **CbEdit** controls in that they can reflect the information from any number of fields for every record within the linked data file. This is accomplished by providing a dBASE expression that describes the information to be retrieved and displayed from within the data file. This expression is evaluated for each record in the data file and the results are displayed in the list box. The expression is often simply a field name from the data file, but it may also include the dBASE operators and functions supported by CodeBase (see the *CodeBase Reference Guide* for more information on supported dBASE functions and expression syntax). The order of the entries displayed in the list box can be determined by selecting a tag for the **CbList** control.

NOTE

When using the Concatenate operators of dBASE (+,-) to display multiple fields, it is strongly recommended that a fixed width font be used so that the fields are vertically aligned.

With CbMaster

There is a close connection between a **CbList** control and its associated **CbMaster** control. In fact, when you select an item in the **CbList**, the selected record becomes the master control's current record, as though the master control itself was used to go that record.

Because of this fact, any other CodeControls linked to the same master as the **CbList** are also updated, to reflect the **CbMaster** control's new record. For example, if you click on an entry in the **CbList** control that corresponds to record number 30 in the data file, the master control's current record number also becomes 30, and any controls bound to the **CbMaster** are updated to reflect this new current record. See Figure 3.4 for an illustration of this concept.

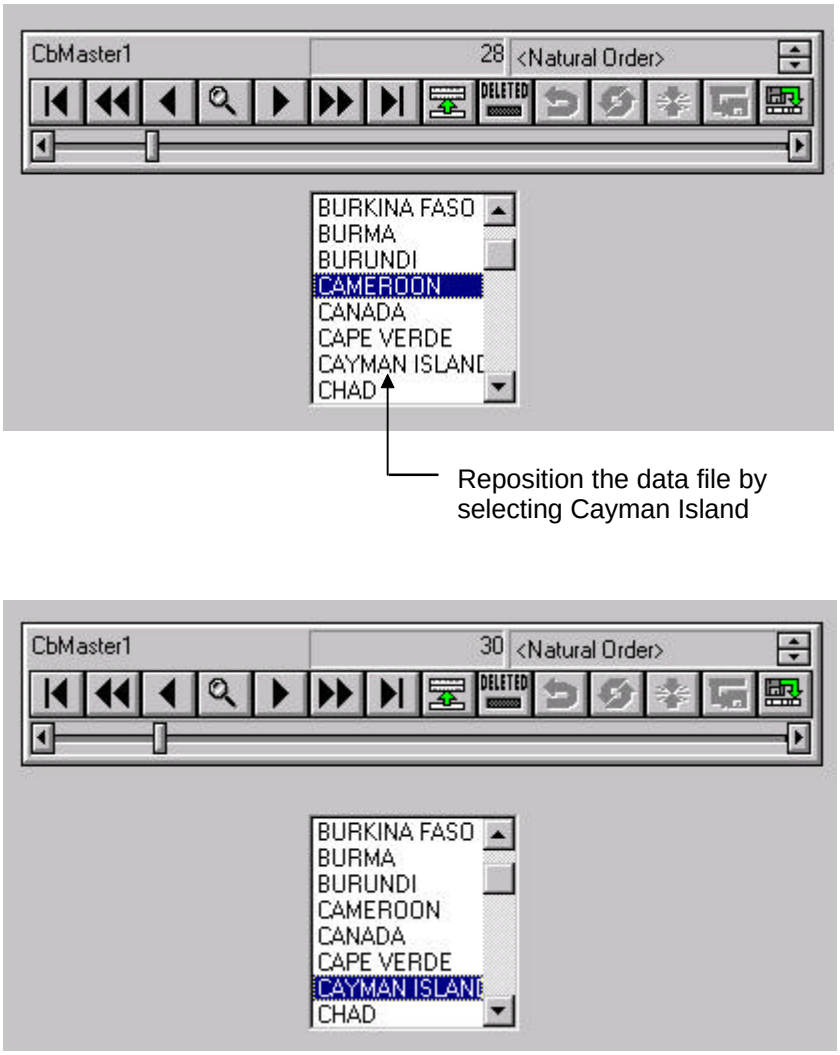


Figure 3.4 Now the data file is positioned to record 30.

This feature allows you to use a linked **CbList** as an easy-to-implement search tool for your applications. By binding other controls to the same master, you can present the end user with more detailed record information for any item selected in the list.

The second way a **CbList** may also be used is as a vehicle to hold valid data for a particular field. Bind the **CbList** to a field, set the DataAction property to Field Value (2) and fill the list with the data. Use the ListItems property at design time to specify the field data or use the AddItem method to add entries to the **CbList** at runtime. When an item is selected from the **CbList**, the item is entered into the field of the current record. See Figure 3.5 for an example. The change to the field becomes permanent when the record is flushed.

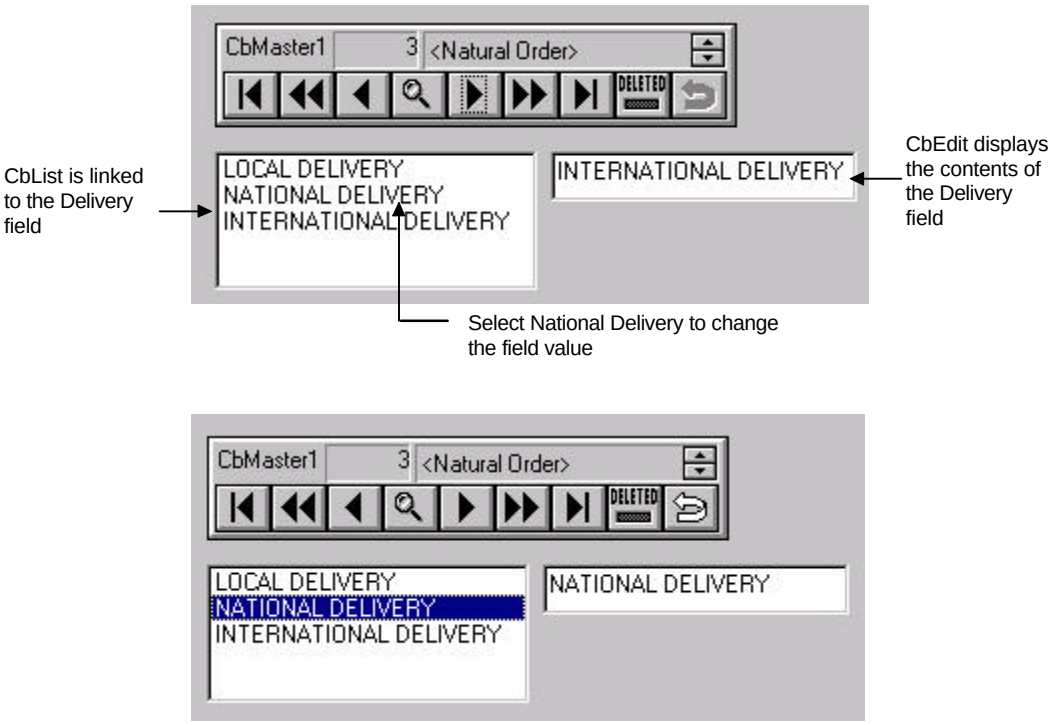


Figure 3.5 Now the field in record 3 has a new value of National Delivery.

CbSlider

A **CbSlider** control can be linked to a data file as well. A **CbSlider** may be used in two ways once it is linked to a data file.

The first way is to specify that the thumb on the slider represents the position of the current record in the data file. The positions on the slider represent all the records in the data file. For example, if the slider thumb is located in the middle of the slider and the data file has 70 records, the data file is positioned on 35th record, see Figure 3.6. Clicking on the slider will cause the record in that relative position in the data file to become the current record of the **CbMaster** control, see Figure 3.7. In this case the slider acts just like a Windows scroll bar.

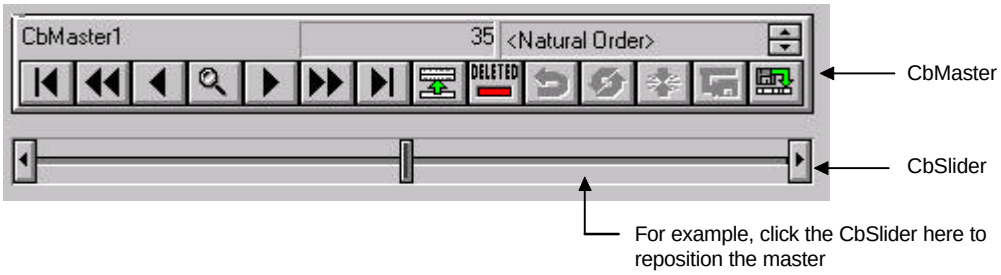


Figure 3.6

Figure 3.7 Now the data file is repositioned to record 52.

The slider can also be used for data entry into a data field. The positions on the slider can correspond to valid field values. For example, a field can have a value from 1 to 10. If value for this field in the first record is 5, then the slider thumb will be positioned in the middle slider to represent the value 5, see Figure 3.8. If the slider thumb is moved to a new position representing 7, then the value of the field is changed to 7, see Figure 3.9. The change to the field becomes permanent when the record is flushed.

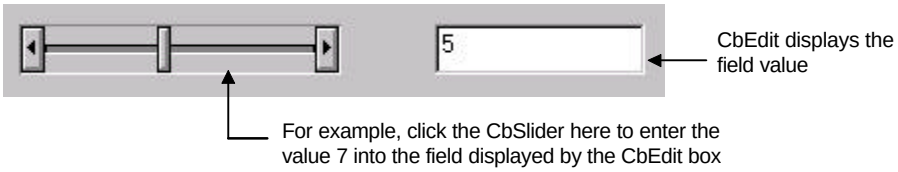


Figure 3.8

Figure 3.9

CbCombo

Both the edit portion and the list box portion of a **CbCombo** control can be associated with data files. The edit portion may be linked to a single field in the data file in the exact same manner as a **CbEdit** control. Any changes made to the contents of the control are also made to the bound field in the data file.

The list box portion also can be linked to the data file through a dBASE expression in the same manner as a **CbList** control. Like the **CbList** control, this expression is evaluated for each record in the data file and is used to fill the list box portion of the **CbCombo** control. The order of the entries in the list box portion can be determined by selecting a tag for the **CbCombo** control.

Figure 3.10 shows a graphical representation of the relationship between the edit and list box portions of a **CbCombo** and linked data files.

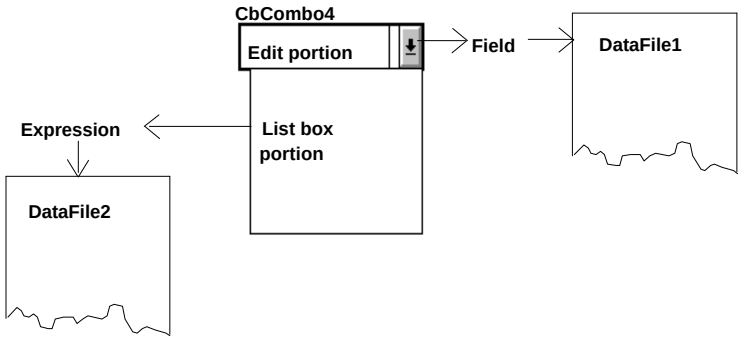


Figure 3.10 Relationship between edit and list box portion of **CbCombo** and two data files.

NOTE

> The edit and the list box portions of the **CbCombo** must be bound to different **CbMaster** controls.

The three different combinations of linking the edit and list box portions of the **CbCombo** control provide great flexibility.

1. When only the edit portion of the **CbCombo** control is linked to a data file, the application can use the list box portion to provide valid options for the edit portion. The selected option is entered into the data file via the edit portion. Items within the list box portion can be added to the **CbCombo** by using the ListItems property at design time or calling the AddItem method at runtime.
2. The second possibility is to have both the edit and list box portion linked to separate data files. This is perhaps the most useful combination, since the data file linked to the list box portion can be used as a look up table for the choices to be entered into the edit portion's data file field.
3. The third possibility is to have only the list box portion of the control linked to a data file. This technique, which is used by the **CbMaster**'s Search dialog, can be used to provide a quick method of performing incremental searches into the linked data file as the user types. This technique can also be used to simply present the user with a list of valid selections.

Searching

Searching a bound data file may be done in several ways:

1. Using the **CbMaster** Search button or Search method.
2. Using the **CbMaster's** Seek, SeekDouble, SeekN, SeekNext, SeekNextDouble or SeekNextN methods.
3. Using CodeBase functions and the **CbMaster** Go method.
4. Using a **CbList** or the list box portion of a **CbCombo** to locate the desired record.

Search Dialog

The Search Dialog, which is invoked by clicking the Search button or by calling the Search method, has already been explained in the "Control Description" chapter.

In general, this is a very easy way to give seeking capabilities to your users, since it requires no coding on your part.

CbMaster Seek Methods

The **CbMaster** provides six different methods that can be used for searching through the data file. These methods are Seek, SeekDouble, SeekN, SeekNext, SeekNextDouble or SeekNextN. The programmer passes a search key as a parameter to a seek method, which in turn searches for match in the currently selected tag. If match is found, the data file is automatically repositioned to matching record. Refer to the "Methods" chapter for more information on how to use these methods.

CodeBase Functions

Searches may also be performed using CodeBase functions. The first step is to get a **DATA4** pointer (**Data4** object) for the data file that is linked to a **CbMaster** control. This can be done by calling the CodeBase function **d4openClone() (Data4::openClone())**, which provides **DATA4** for a data file that needs to be opened more that once. The programmer can then develop his own search implementation. Once the appropriate record is found using CodeBase functions, it is important to update the display of the bound controls by calling the **CbMaster** Go method with the new record number. The **CbMaster** control goes to the new record and in turn sends update messages to its bound controls.

This technique provides the most programmatic freedom to the developer, but does require more coding than simply using the **Search** button on the **CbMaster** control.

CbList and CbCombo

CbList and **CbCombo** controls can also be used as search engines. **CbList** controls simply present the user with a list of records to select from using the mouse or the arrow keys. Selecting an item from the list repositions the data file.

CbCombo controls can provide a more sophisticated search engine. Users enter text in the edit portion; this text is used as a search key for items in the list portion.

The search behavior varies slightly depending on the style of the **CbCombo**. As discussed in the "Control Description" chapter, the **CbCombo** supports three different styles. Only the simple and drop down **CbCombo** styles support the search engine behavior. The drop down list can not support the search engine behavior because the edit portion is static.

When the user types into the edit portion of a simple **CbCombo**, an incremental search of the list is executed and the matching entry is scrolled to the top of the list. If there is no match, the first record in the selected tag is scrolled to the top. No automatic selecting is carried out in either case.

The drop down **CbCombo** behaves a little differently from the simple **CbCombo**. First the user types into a drop down **CbCombo** and then drops the list. If a matching item is found in the list, it is automatically selected. If the list is bound to a **CbMaster**, the new selection will cause the data file to be repositioned. When the user drops the list and there is no match, the first record in the selected tag is scrolled to the top of the list and nothing is selected.

The programmer may need to modify the search key before the search is executed. The LookUp event is fired just before the search is executed. This event may be intercepted and used to modify the search key. Refer to the "Events" chapter for more information on the LookUp event.

3D Effect

CodeControls provides the developer with a set of controls that can display a three dimensional or flat border. In addition, the **CbMaster** can be made to appear recessed into the surface of the dialog box or visually projected away from the dialog and towards the user.

This is done by providing a flexible and customizable beveled edge that surrounds the controls. By altering the "lighting" of the control by the perceived light source, a control can be visually inset or raised, giving it a three dimensional appearance.

The **CbEdit**, **CbCombo**, **CbList** and the **CbSlider** have a property called Appearance that can be used to give the control a 3D border or a flat single line border. If the value of Appearance is set to Flat (0) then the flat border is drawn. The 3D border is drawn if Appearance is set to 3D (1).

The **CbEdit** control has a property called BorderStyle that determines whether a border is drawn or not. If the BorderStyle is set to 0 then no border is drawn no matter the setting of Appearance property. If BorderStyle is set to 1, then the border is drawn according to the Appearance setting.

The **CbMaster** and **CbSlider** controls also have a property call BorderStyle, which determines whether a single line border is drawn around the control. If BorderStyle is set to 1, single line border is drawn around the control. If the BorderStyle is set to 0, then no single line border is drawn. The BorderStyle does not effect the appearance of the **CbMaster's** 3D border as determined by the BevelType property.

The appearance of the 3D border for the **CbMaster** control is controlled by two properties: BevelType and BevelThickness. The BevelType can have one of four values. If the value is set to No Bevel (0), then no 3D border is drawn and the only border drawn depends on the value of BorderStyle. If BevelType is set to 3D Bevel (1) then a regular 3D border is drawn. An inset 3D border is drawn when BevelType is set to Inset Bevel (2) and a raised border is drawn when BevelType is set to Raised Bevel (3).

The property BevelThickness only applies when the BevelStyle is set for the inset or raised 3D border. The BevelThickness determines the width in pixels of the 3D border. BevelThickness can have a value from 1 to 10.

The properties which alter the three dimensional appearance of the control are listed in Table 3.7. These properties are set at design time and some may be altered at runtime depending on the type of control.

The properties are represented graphically in Figure 3.13.

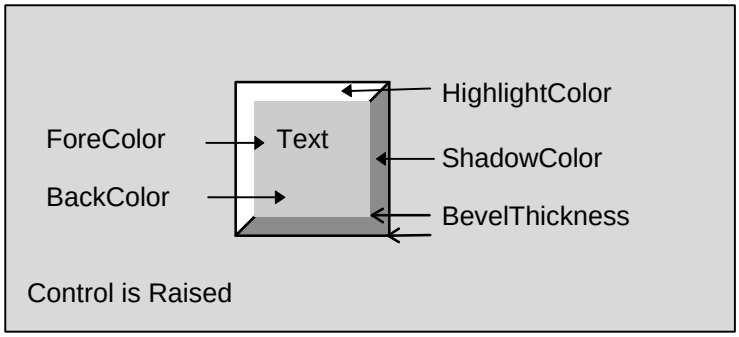


Figure 3.13 3D Characteristics

Property	Description
Appearance	This property determines whether a 3D border is drawn for the **CbEdit**, **CbCombo**, **CbList** and **CbSlider** controls.
BorderStyle	This property determines whether a single line border is drawn on a **CbMaster** or **CbSlider**. For the **CbEdit** , this property determines whether a border is drawn regardless of the Appearance setting.
BevelThickness	Determines the thickness of the **CbMaster** control's bevel. The default value is 2 pixels. See the "BevelType" section below for a description.
BevelType	This property determines what kind of 3D border appears on the **CbMaster**. The control can have no 3D border, a regular 3D border, an inset or raised 3D border. The default value is a raised border.
HighlightColor	Specifies the color of the light source that highlights the control. The HighlightColor can be visible in the upper or lower part of a control, depending on the whether the control is inset or raised. The **CbMaster**, **CbSlider** and **CbButton** have this property.
ShadowColor	Specifies the shadow color cast on the dark side of the control. The ShadowColor can be visible in the upper or lower part of a control, depending on the whether the control is inset or raised. The **CbMaster**, **CbSlider** and **CbButton** have this property.

Table 3.7 3D Properties

Bevel

The bevel is the angular ridge that surrounds the control and gives it its 3D appearance.

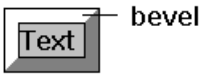


Figure 3.14 Bevel of a raised control.

Width The default width of CodeControls bevels is 2 pixels. This may be changed at design and run-time using the BevelThickness property.

Raised vs. Inset The bevel can also be changed so that the control appears to be either inset or raised with respect to your dialog box. This is accomplished internally by switching the highlight and shadow colors on the bevel. The BevelType property determines the kind of Bevel. The default BevelType is raised.

Outside Border The outside border is a Windows style that is specified by BorderStyle.

Changes made to the appearance of the bevel at run time take effect immediately.

Color

There are four parts of the control for which you can change the color: the control's background, highlight, shadow and text. Figures 3.15 and 3.16 below illustrate these four parts for an inset control and for a raised control.

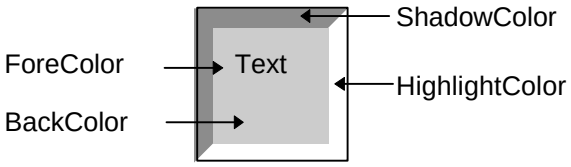


Figure 3.15 Inset control

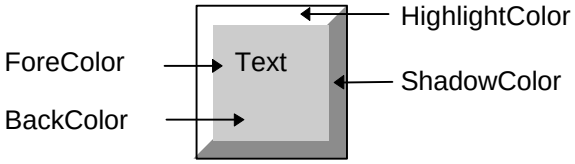


Figure 3.16 Raised control

These colors may be set at design time or run-time by setting the appropriate properties BackColor, ForeColor, HighlightColor and ShadowColor.

All color changes made at runtime take effect immediately.

Font

The actual font of the text displayed by the control is also under the control of the application developer. This may be done in a variety of ways depending upon the application development tools being used. In general, any font that can be displayed in Windows may be used with any CodeControls.

When changing the font for a control, however, it is up to the application developer to ensure that the font will fit within the control. A 36 point font, for example, should not be used with the average sized control.

Alignment

CodeControls offers three different ways in which the text within a control may be aligned in a **CbEdit** control. By default, all characters are entered from the left side of the control and proceed to the right as the user types in information.

The alignment of text in a **CbEdit** control is completely under the control of the application developer and may be either left, center, or right justified. The alignment is determined by the property Alignment and it can only be changed at design time.

4 Programming CodeControls

This chapter of the manual provides a general guide on how to program with CodeControls at design time and runtime. The instructions in this chapter can be used under any of the supported programming environments. If you have not done so already, you should read the "Control Description" and "CodeControls Concepts" chapters before proceeding.

Linking to a Data File

CodeControls are "data aware" in that they may be used to directly view and edit information within a data file (see the "CodeControls Concepts" chapter for a complete description of "data aware" controls).

In order to be data aware, however, they must be linked to the data file at design time or runtime. This section describes the techniques behind linking CodeControls to data files.

Linking the CbMaster

Before any other CodeControls may be linked to a data file, it is first necessary to add a **CbMaster** control to the container and link it to a data file.

Index Files

Most of the time a data file will have one or more index files associated with it. An index file provides the data file with tags, which are used to sort the records in a meaningful way. When a data file is being linked to the **CbMaster**, any associated index files should be specified as well. At runtime, the **CbMaster** will open its data file and then open each specified index file.

Once a **CbMaster** control has been added to the dialog, it may be linked to a data file at design time or at runtime using the following methods:

Design Time

The easiest way to link a **CbMaster** control to a data file at design time is to use its property pages. Right click the **CbMaster** control. Click the Properties option from the menu to pop up the property pages. Click the Files tab. This property page is used to specify the data file and any index files for the **CbMaster**, see Figure 4.1. Click the Browse button to pop up an Open dialog box. Choose a data file and click the OK button. The data file name is shown in the Database Name text box. Note that if you do not specify a path then CodeControls assumes that the data file is located in the current directory.

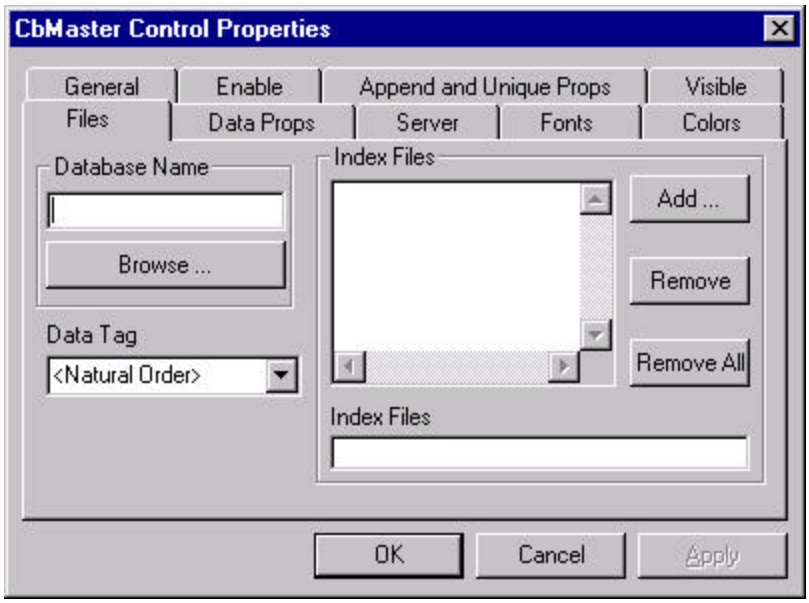


Figure 4.1

The process of choosing a new data file automatically clears the IndexFiles and DataTag properties. Now you can specify the index files that will be used by the **CbMaster** and the bound controls. Click the Add button and an Open dialog pops up. Select an index file and click the OK button. The index file is automatically added to the Index Files list and its tags are added to the Data Tag combo box. Add other index files in the same manner if desired. Select a tag from the Data Tag list if you want the **CbMaster** to use a particular tag at runtime otherwise natural order is used.

The **CbMaster** control is now linked to the data file. Click the OK button to close the property pages and to apply the changes to the properties.

Runtime

A **CbMaster** control may be linked to a data file at runtime by setting the DatabaseName property. Changing the DatabaseName property automatically clears the IndexFiles and DataTag properties. Always set the IndexFiles and DataTag properties after the DatabaseName is set. Once these properties are set, it is necessary to call the **CbMaster** RefreshFiles method to reset the bound controls to the new data file. See the "Properties" chapter for more information on DatabaseName, IndexFiles and DataTag.

WARNING!

When the RefreshFiles method is used to open a new data file, an error can occur if the field names and/or expressions used to link the controls are no longer valid for the new data file. This does not occur if DatabaseName has been set to a null string. Be sure to update the data specific properties discussed in Table 4.1, before RefreshFiles is called.

Client/Server

CodeControls supports using the CodeBase client/server configuration as well as the stand-alone version. In order to use the client/server configuration of CodeBase with CodeControls requires that the server must be running at design time as well as during runtime. This is because the **CbMaster** opens and closes files at design time to access field and tag information.

The first thing you should do is specify the server properties so that when the **CbMaster** opens a data file at design time, the **CbMaster** can connect to the server. Open the property pages for the **CbMaster** and select the Server tab. The ServerId property is the network identification for the server computer. By default the ServerId is set to "LOCALHOST". The default ServerId is only valid when the client and the server are on the same computer. Usually the client and server are on different computers and since it is unlikely that the server computer is called "LOCALHOST", the ServerId property will have to be changed to the correct name. If the server ProcessId has been changed for the server, then the ProcessId property for the **CbMaster** will also have to be updated. The UserName may be changed from the default value if desired. CodeBase transactions use the UserName to identify which client has made changes to the data file. The server UserName and Password verification is not currently supported but may be added to future versions. Refer to the CodeBase *Getting Started* manual for more information on these server settings.

CbMaster is linked to a data file under client/server in the same way as standalone applications. The programmer specifies the DatabaseName, IndexFiles and DataTag properties. The only difference is at design time. The Browse and Add buttons on the Files tab of the property pages are disabled. Under client/server, the location of the files must be specified relative to the location of the server. The Browse and Add buttons access files relative to the client computer. This can cause problems because the server and the client are usually on different computers and the data files are located on the server computer. The problem can be illustrated in the following way. Assume that the client and server are on different computers and the data files are in the C:\DATA directory of the server computer. Say that you wanted to link the C:\DATA\FILE.DBF file to the **CbMaster**. If the client accessed the data files on the server computer through the Browse button, the path would be set to Y:\DATA\FILE.DBF, where the Y is the network drive designation for the server computer. So using the Browse button does not make sense because the path should be C:\DATA\FILE.DBF. Therefore the programmer must type in the data file name and path relative to the location of the server. Similarly, the paths of the index files must also be typed into the Index Files text box on the property page. When the index files are typed in, the paths must be separated by semi-colons.

Applications

An application created with CodeControls can be used with either the stand-alone or client/server configurations CodeBase. The application need only be distributed with the appropriate CodeBase DLLs and CodeControls OCXs. The **CbMaster** can automatically tell whether the available CodeBase C4DLL.DLL is a stand-alone or client version. If the CbMaster detects the client version of the CodeBase C4DLL.DLL, then the **CbMaster** automatically tries to connect to the server. If the **CbMaster** detects the stand-alone version of CodeBase, the ServerId and other server properties are ignored and data files are manipulated directly through the CodeBase DLL. Therefore if you want an application to work with either client/server or stand-alone, the ServerId must set for every **CbMaster** in the application just in case the **CbMaster** detects the client version of the CodeBase DLL.

Binding Controls to the CbMaster

Once the dialog has a **CbMaster** control that is linked to a data file, the individual CodeControls within the dialog/form may be bound to the **CbMaster** control and through it, be linked to the data file.

A control may be bound to a **CbMaster** design time or at runtime. In general the first step is to bind the control to a specific **CbMaster**. The second step is to define the behavior of the control or specify what data file information is to be displayed. Some controls can be bound to a data field, while others may use a dBASE expression for displaying information.

Table 4.1 summarizes all of the data specific properties involved in binding controls.

Property	Applies To	Description
DatabaseName	CbMaster	Specifies the file name of the data file to which the CbMaster is linked.
DataSource	CbEdit, CbList, CbButton, CbSlider	Specifies the name of the CbMaster control to which the control is bound.
DataField	CbEdit, CbSlider	Specifies the name of the field to be linked to the control.
DataExpr	CbList	Specifies the dBASE expression* used to create entries for the list control.
DataTag	CbList, CbMaster	Specifies the tag that is used to order the entries.
DataAction	CbList, CbButton, CbSlider	Specifies the behavior of a bound control.
DataSourceList	CbCombo	Specifies the name of the CbMaster control to which the list portion of the combo box control is bound.
DataSourceEdit	CbCombo	Specifies the name of the CbMaster control to which the edit portion of the combo box control is bound.
DataFieldEdit	CbCombo	Specifies the name of the field to be linked to the edit portion of the combo box.

DataExprList	CbCombo	Specifies the dBASE expression* used to create entries for the list portion of the combo box.
DataTagList	CbCombo	Specifies the tag which is used to order the entries in the list portion of the combo box.

Table 4.1 Data Specific Properties

*For more information on dBASE expressions refer to the "Appendix C: dBASE Expressions" chapter in your CodeBase reference manual.

Binding at Design Time

The following describes how each control in CodeControls can be bound to a **CbMaster** control at design time. The easiest way to bind a control at design time is through the control's property pages.

CbButton

Open the property pages for the **CbButton** and click the Data tab. The Data property page will appear as illustrated in Figure 4.2.

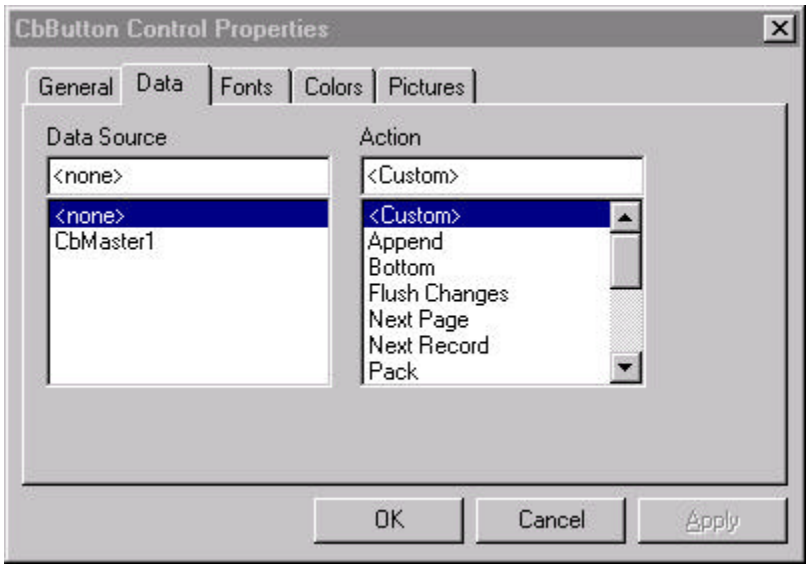


Figure 4.2

The Data Source combo box contains a list of all the **CbMaster** controls on the container. Select a **CbMaster** from the list. The next step for binding a **CbButton** to a **CbMaster** is choosing what action the button will have when it is clicked by a user. A list of potential actions is contained in the combo box labeled Action. The actions emulate much of the functionality of the **CbMaster** and includes positioning the data file in various ways, packing, refreshing, flushing, deleting records and so on. Select an action from the list and click the OK button to close the property pages and apply the changes to the properties. Now the **CbButton** is bound to a **CbMaster** control.

CbEdit

Open the property pages for the **CbEdit** and click the Data tab. The Data property page will appear as illustrated in Figure 4.3.

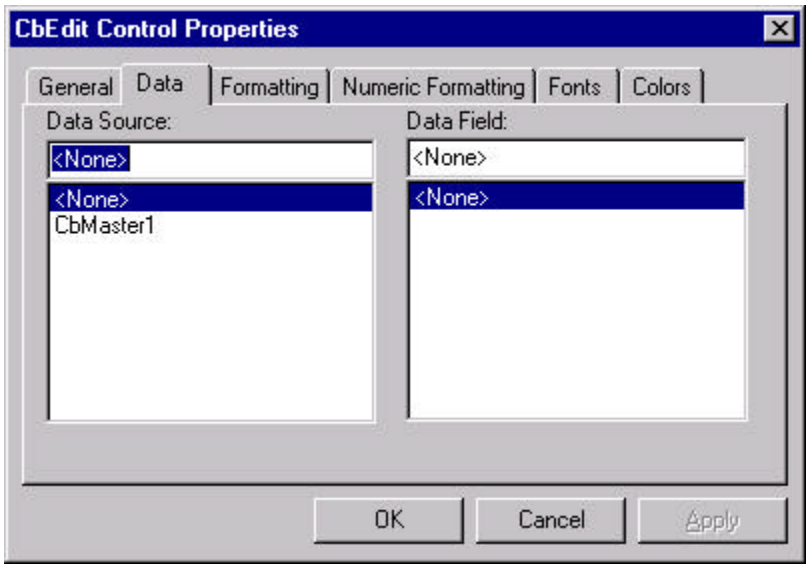


Figure 4.3

Select a **CbMaster** from the Data Source combo box. The next step is to chose what data is displayed in the edit box at runtime. When a **CbMaster** is selected, the Data Field combo box is automatically filled with field names. Select one of the field names from the list. Click the OK button to close the property pages and apply the changes to the properties. The **CbEdit** is now bound to a **CbMaster** control.

CbList

Open the property pages for the **CbList** and click the Data tab. The Data property page will appear as illustrated in Figure 4.4.

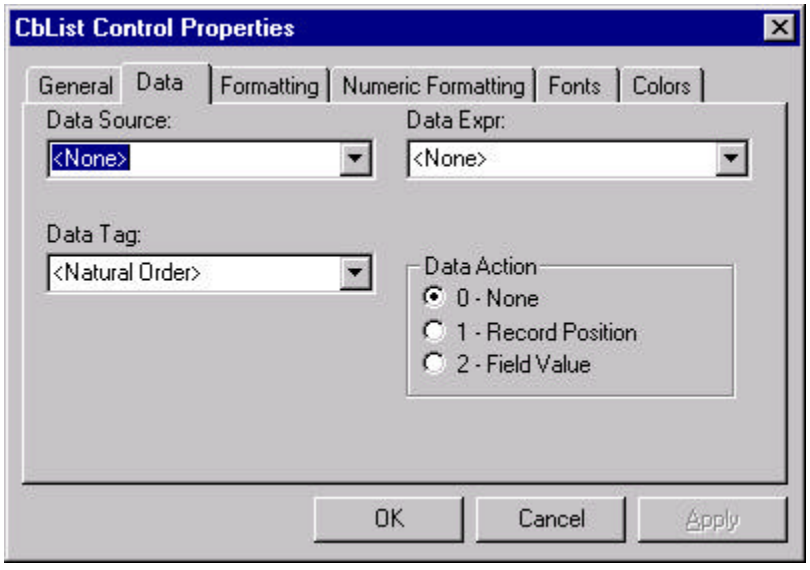


Figure 4.4

There are two ways a **CbList** may be bound to a **CbMaster** control. First, the **CbList** may be used as a means to reposition the data file. The second way is to fill the list box with potential field values and use the list box as a way to enter data into a particular field.

In both cases, a **CbMaster** must be chosen from the Data Source list. When a **CbMaster** has been selected, the Data Expression list will be automatically filled with field names.

If the **CbList** is used to reposition the data file, the list box is filled with data from each record in the data file. When the user selects a record from the **CbList**, the data file is automatically repositioned to that record. The value of the DataExpr property determines what data is displayed for each record in the **CbList**. The expression is often simply a field name, which can be selected directly from the Data Expr list on the property page. The user may also type in a dBASE expression, which can reflect information from any number of fields and include the dBASE operators and functions supported by CodeBase. The expression is evaluated for every record in the data file and displayed in the **CbList**. Click the radio button labeled Record Position to indicate that the **CbList** is to be used for repositioning the data file. The order of the records displayed in the **CbList** is determined by the DataTag property. The Data Tag list is automatically filled with valid tags when the **CbMaster** is chosen. If no tag is selected, the records are displayed in natural order.

The **CbList** can be bound to a field and used to store valid field values for that particular field. The programmer adds the field values to the **CbList** by using the AddItem method at runtime or by using the ListItems property at design time. When the user selects an item from the **CbList**, the value is automatically assigned to the bound field of the current record. Choose a **CbMaster** from the Data Source list and a field from the Data Expression list. In this case the DataExpression property must only specify a data field name. The DataTag property has no effect on contents of the **CbList**. Finally, be sure to click the Field Value radio button in the Data Action list.

CbCombo

The **CbCombo** has two property pages for binding the control, one for the edit portion and one for the list portion. The user may bind both the edit and the list portion of a **CbCombo**, but they must be bound to different **CbMaster** controls. The property pages are illustrated below in Figure 4.5 and Figure 4.6.

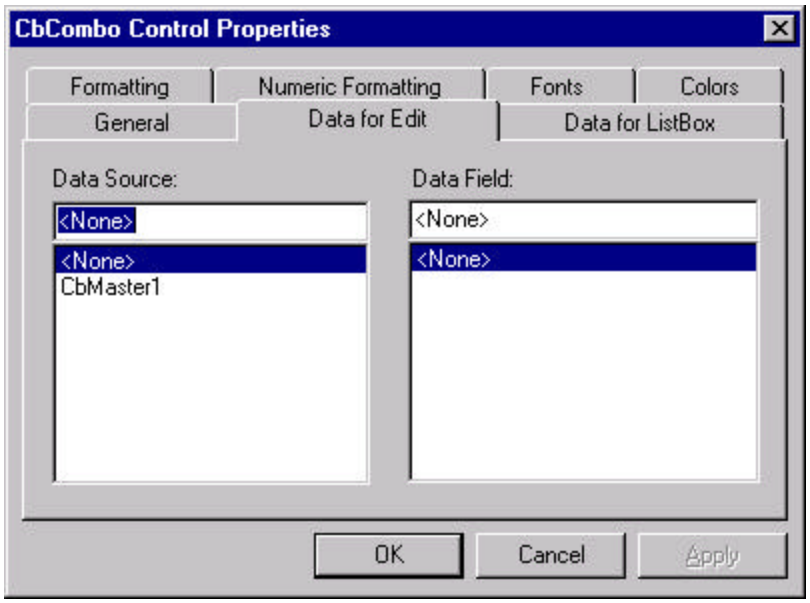


Figure 4.5

There are two main ways in which a **CbCombo** is used in database applications.

One way is to use the **CbCombo** to perform incremental searches and reposition the data file. In this case, only the list portion is linked to a **CbMaster** and the user types in the edit portion. As the user types, the list is searched for a matching string. When the user selects a record from the list, the data file is automatically repositioned. It is useful to specify a tag using the DataTagList property so that the records are ordered in a meaningful way. In addition, the tag aids in making the search quicker.

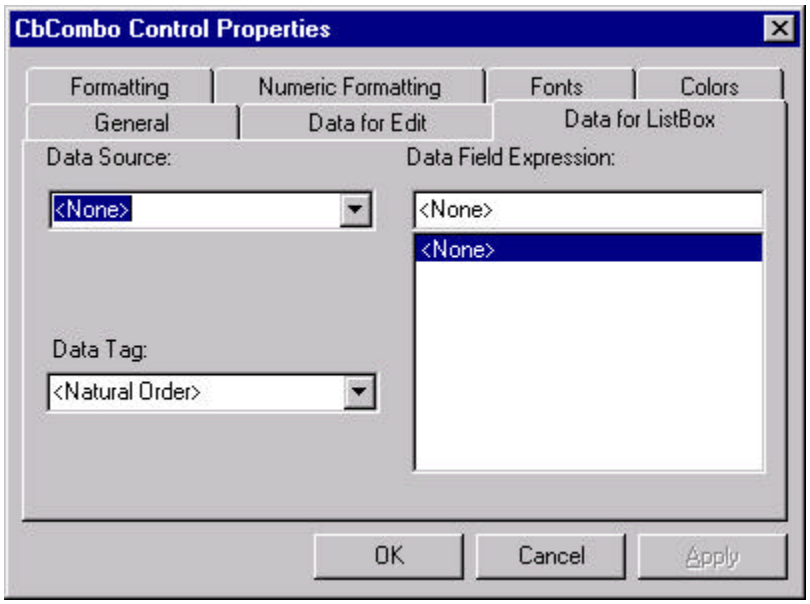


Figure 4.6

The second way to use the **CbCombo** is to assign field values to a data file. New field values will be assigned to the field bound to the edit portion of the **CbCombo**. The new field data will be supplied by the list portion of the **CbCombo**. There are two ways the list portion can supply the field data. The list portion can be bound to a different data file or the programmer can fill the list portion with valid field values using the AddItem method at runtime. In either case, when the an item is selected from the list, it is automatically assigned to the field bound to the edit. In addition, the user can type directly in the edit portion. As the user types, an incremental search will occur as discussed above.

Binding the edit portion of the **CbCombo** is done in the same manner as binding a field to a **CbEdit**. Choose a **CbMaster** from the Data Source list on the Data for Edit property page. The Data Field list is automatically filled with field names when a **CbMaster** is selected from the Data Source list. Choose a field from the Data Field list. Now the edit portion of the **CbCombo** is bound to a **CbMaster**.

Similarly, binding the list portion is the same as binding a **CbList** to a **CbMaster**. Choose a **CbMaster** from the Data Source list on the Data for List property page. The Data Expr list is automatically filled with field names and the Data Tag list is filled with tag names. You can choose a field from the Data Expr list or you can type in a dBASE expression. Select a tag from the Data Tag list if desired, otherwise natural ordering is used. Unlike the **CbList**, there is no DataAction property for the list portion of the **CbCombo**. It is not needed, since the **CbCombo** can simultaneously use the data file bound to the list for repositioning and for assigning data to the field bound to the edit.

If both the edit portion and the list portion of the **CbCombo** are to be bound then they must be bound to different **CbMaster** controls. The property pages will automatically update the Data Source lists for each portion of the **CbCombo**. For example, say that there are three **CbMasters** on the container: CbMaster1, CbMaster2 and CbMaster3. When the property pages for the **CbCombo** are initially opened, the Data Source lists for the edit and the list portions both list all three **CbMasters**. If CbMaster1 is selected as the Data Source for the edit portion, the CbMaster1 is removed from the Data Source list displayed for the list portion. Similarly, if CbMaster2 is bound to the list portion, the Data Source list for the edit portion will no longer list CbMaster2 as a potential DataSource.

CbSlider

Open the property pages for the **CbSlider** and click the Data tab. The Data property page will appear as illustrated in Figure 4.7.

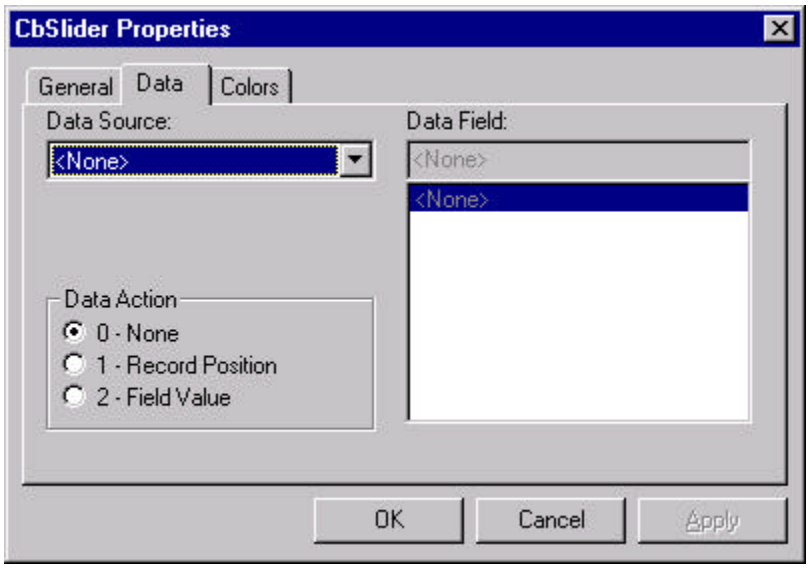


Figure 4.7

The functionality of the **CbSlider** is similar to that of the **CbList**, in that it can be used to reposition the data file and assign field values to the current record.

If the **CbSlider** is to be used for repositioning the data file then choose Reposition radio button from the Data Action list. Bind the **CbSlider** to a data file by selecting **CbMaster** from the Data Source list. The Data Field list is automatically filled with the field names associated with the data file linked the chosen **CbMaster**. In this case, a Data Field name need not be selected since it will have no effect and for this reason the Data Field list is disabled. The Data Field list is automatically disabled when the Reposition radio button is checked.

The **CbSlider** can be used to assigned numeric data to a field. Choose the Field Value radio button from the Data Action list, which automatically enables the Data Field list. Next, select a **CbMaster** from the Data Source list. Select a field name from the Data Field list. The **CbSlider** is now bound to a data field through the selected **CbMaster**. The position of the thumb on the slider corresponds to a number that lies between the MinValue and MaxValue properties. When the user clicks on the **CbSlider** at runtime, the value of the thumb position is automatically assigned to the bound field. Be sure to set the MinValue, MaxValue and ScrollUnit properties to reflect the desired values that can be assigned to the bound field.

Binding at Runtime

The following describes how each control in CodeControls can be bound to a **CbMaster** control at runtime. When the DataSource, DataSourceEdit or DataSourceList properties are changed at runtime, the control is automatically unbound from the current **CbMaster** and then bound to the new **CbMaster**.

CbButton

The first step in binding a **CbButton** at runtime is to set the DataSource property to the name of a **CbMaster**. The next step is specify the action of the **CbButton** through the DataAction property if necessary. Valid values for the DataAction property are listed in the "Properties" chapter.

The control can be unbound from a **CbMaster** by setting the DataSource property to an empty string ("").

CbEdit

The first step in binding a **CbEdit** at runtime is to set the DataSource property. Setting the DataSource property to a new value automatically sets the DataField property to an empty string. Therefore you must always set the DataField property after the DataSource is set to a new value. When the DataSource property is changed at runtime, the **CbEdit** automatically tries to connect to the new master. When control is linked to a valid **CbMaster** and the DataField property is changed, the **CbEdit** is automatically updated with the new field data for the current record.

The control can be unbound from a **CbMaster** by setting the DataSource property to an empty string (""). The programmer may still utilize the **CbEdit's** formatting capability even when it is not data aware. The **CbEdit** can be used in the same manner as a regular Window text control when it is not bound.

CbList

The first step in binding a **CbList** at runtime is to set the DataSource property. Setting the DataSource property to a new value automatically sets the DataExpr property to an empty string (""). Therefore you must always set the DataExpr property after the DataSource is set to a new value. When the DataSource property is changed at runtime, the **CbList** automatically tries to connect to the new master. When the DataExpr property is changed and the control is linked to a valid **CbMaster**, the **CbList** is automatically updated with the new data.

The DataAction property may also be changed at runtime. When this property is changed it automatically updates the contents of the **CbList** accordingly. If the **CbList** is currently bound to a valid **CbMaster** and the DataAction is set to Record Position(1) then the **CbList** is filled with data corresponding to each record according to the DataExpr property. If the DataAction is set to Field Value(2), the **CbList** DataExpr must specify a field and not a dBASE expression. The ListItems property may be used at design time to specify the field values. The AddItem method may be used at runtime to add field values to the **CbList**.

The DataTag property may also be changed at runtime. When the **CbList** is bound to a valid **CbMaster** and the **CbList's** DataTag property is changed, the contents of the **CbList** are automatically updated to reflect the new tag ordering. Note that the DataTag only has effect when the DataAction is set to Record Position (1).

CbCombo

There are two sets of data specific properties for the **CbCombo**, one set for the edit portion and one set for the list portion. Both the edit and the list portion may be bound to different **CbMasters** at runtime.

The first step in binding the edit portion is to set the DataSourceEdit property. Setting the DataSourceEdit to a new value automatically sets the DataFieldEdit property to an empty string. The next step is to set the DataFieldEdit property after the DataSourceEdit is set to a new value.

The first step in binding the list portion is to set the DataSourceList property to a new value. Setting the DataSourceList to a new value automatically sets the DataExprList to an empty string. Therefore you must always set the DataExprList property after the DataSourceList is set to a new value. When the DataExprList property is changed, the list automatically updated with the new data. Similarly, if the DataTagList is changed, the list is automatically updated to reflect the new tag ordering.

The **CbCombo** can be unbound by setting both the DataSourceEdit and the DataSourceList property to an empty string (""). The unbound **CbCombo** can then be used in the same manner as a Windows combo box.

CbSlider

The first step in binding a **CbSlider** at runtime is to set the DataSource property. Setting the DataSource property to a new value automatically sets the DataField property to an empty string. Therefore you must always set the DataField property after the DataSource is set to a new value. When the DataSource property is changed at runtime, the **CbSlider** automatically tries to connect to the new master.

The DataAction property may also be changed at runtime. When this property is changed, the **CbSlider** behavior changes accordingly. If the DataAction is set to Record Position (1) and the **CbSlider** is currently bound to a valid **CbMaster**, then the slider thumb is automatically positioned to reflect the location of the current record in the data file. If the DataAction is set to Field Value (2) and the **CbSlider** is bound to a valid data field, the slider thumb is automatically positioned to indicate the field value of the current record.

The **CbSlider** may be unbound by setting the DataSource property to an empty string ("").

'Unbinding' Controls

Controls bound to a **CbMaster** are automatically 'unbound' in the following circumstances:

- The form/dialog containing the **CbMaster** is unloaded.
- A new DataSource, DataSourceEdit, DataSourceList, property is specified.
- The bound controls' DataSource, DataSourceEdit, or DataSourceList properties are set to an empty string("").

WARNING!

Changing the DatabaseName property for a **CbMaster** control does not automatically unbind the controls bound to the **CbMaster**. Thus when the RefreshFiles method is used to update the data file link, an error will occur if the field names and/or expressions used to link the controls are no longer valid for the new data file. An error is not generated if the DatabaseName has been set to a null string. Similarly, calling CbMaster method Unlink does not automatically unbind the controls.

Closing the Data File

As the **CbMaster** controls are unloaded, they automatically close their associated data file and index files. If there are any pending updates that need to be flushed to disk, the **CbMaster** attempts to flush them. If another application has the data file locked and the **CbMaster** can not flush the changes, the **CbMaster** will continue to close the data file and any changes will be lost. In light of this behavior the programmer should ensure that there are no pending updates before an application is unloaded. The **CbMaster** must behave in this way because there is no way to abort the unloading of an application from the control level and thus keep the changes until the data file may be updated successfully.

The **CbMaster** control may be also be manually unlinked from its data file by using the Unlink method. The Unlink method does not unbind any bound controls. The **CbMaster** control may also be unlinked from a data file by setting the DatabaseName to an empty string and then calling RefreshFiles. The RefreshFiles method closes the current data file and then attempts to open the data file specified by DatabaseName. Since the DatabaseName is set the empty string the attempt to open the new data file fails, but does not generate an error.

WARNING!

Unlinking a CbMaster from its data file will not automatically unbind any bound controls. The bound controls must be explicitly unbound by changing their respective DataSource, DataSourceEdit or DataSourceList properties to a new value or to an empty string.

WARNING!

Do not close the **CbMaster**'s data file using CodeBase functions while it is linked to the data file. Doing so will lead to unpredictable results.

Ordering of Records

As mentioned in the "CodeControls Concepts" chapter, the **CbMaster** traverses the data file according to the selected tag. If there is no selected tag the **CbMaster** skips through the data file in natural order.

There are two ways a tag may be selected. The first way is to set the DataTag property to the name of the tag at design or runtime. The second way is to let the user choose the tag at runtime. The **CbMaster** can display a spin list at runtime that contains the tag names associated with the open index files. The user may select a tag name from the list and the data file is automatically traversed in the new order. When the user selects a tag from the spin list, the DataTag property is automatically updated with the new name.

Whenever the DataTag property is set, the **CbMaster's** Tag4 property is updated. The Tag4 property is read-only at runtime and returns a pointer to a **TAG4** structure.

Similarly, the order in which entries are displayed in a list box can be determined by setting the **CbList** DataTag or **CbCombo** DataTagList properties.

NOTE

Setting the DataTag property for a **CbMaster** control does not affect the DataTag and DataTagList properties of bound **CbList** and **CbCombo** controls.

Moving Through the Data File

This section describes the various ways in which a data file may be repositioned at runtime. The easiest way to reposition a data file linked to a **CbMaster** is to use the **CbMaster** itself. The **CbMaster** control has a graphical interface with buttons and a slider, which can be used interactively skip through the data file.

The **CbMaster** also provides a set of methods and properties that can be used programmatically for repositioning the data file. The **CbList**, **CbCombo**, **CbSlider** and **CbButton** controls have been designed so that when they are bound to a **CbMaster**, they can be used reposition the data file.

Any bound controls are automatically updated when any of the above methods are used to reposition the data file.

The programmer may also use CodeBase function calls to reposition the **CbMaster** data file directly. In this case, the bound controls are not updated automatically and the programmer should take this fact into account.

CbMaster Buttons

There are seven buttons that can be used to skip through the data file and they are described below.

The **CbMaster** has Top and Bottom buttons to skip directly to the top or bottom of the data file. There are Next and Previous buttons to skip forward and backward one record. The NextPage and PreviousPage buttons skip a set number of records forward and backward. The number of records to skip is determined by the PageSize property, which is set to ten by default. The **CbMaster** also has a Search button, which pops up a dialog box that can used to make incremental searches in the data file. See "Control Description" chapter for more information on the **CbMaster** buttons.

CbMaster Slider	The **CbMaster** can also display a slider, which can be used to reposition the data file. The slider works in the same way as a **CbSlider** with its DataAction set to Record Position (1). The position of the thumb on the slider represents the relative position of the current record in the data file or tag. When the user changes the position of the thumb by clicking on the slider or arrow buttons, the data file is repositioned to the record that occurs at that relative position in data file or tag.
CbButton	The **CbButton** control can be used to emulate any of the **CbMaster** buttons in functionality. When a **CbButton** is bound to a **CbMaster**, the behavior of the **CbButton** is specified by the DataAction property. The DataAction property can be set to emulate any of the repositioning buttons including the Search button. See the "Properties" chapter for more information on how to use the DataAction property. Thus, the **CbButton** can be a used as an alternative way of taking advantage of the **CbMaster** functionality without using a **CbMaster** directly.
CbMaster Methods	The following methods can be used at runtime to reposition the data file: Bottom, Go, Seek, SeekDouble, SeekN, SeekNext, SeekNextDouble, SeekNextN, Skip, Top. Refer to the "Methods" chapter for more details on how to use these methods.
CbMaster Properties	The RecNo and the Position properties can be used at runtime to reposition the data file. Refer to the "Properties" chapter for more details on how to use these properties.
CbList	One of the major ways a **CbList** can be used is as a tool for quickly browsing through information from the entire data file. When the **CbList** is bound to a **CbMaster**, the **CbList** can display information corresponding to each record in the data file according to a dBASE expression. If a tag is selected then only those records belonging to the tag are displayed in the **CbList**. The user can then scroll through data file using the **CbList**. When a user selects an item from the **CbList**, the data file is automatically repositioned to that record and any other bound controls are automatically updated.
CbSlider	One of the major ways a **CbSlider** can be used is as a tool for repositioning the data file. When the **CbSlider** is bound to a **CbMaster**, the position of the thumb on the scroll bar represents the position of the current record in the data file . When a user moves the thumb to a new position on the slider, the data file is positioned to the record that occurs in that relative position in the data file. For instance if the thumb is place in the middle of the slider and the data file has 100 records, the data file is positioned to the 50^{th} record. The other controls that are bound to the **CbMaster** are automatically updated.
CbCombo	One of the major ways a **CbCombo** can be used is as a search tool. When the list portion is bound to a **CbMaster**, the list can display information corresponding to each record in the data file according to a dBASE expression. As the user types into the edit portion, the **CbCombo** searches its list for matching data.

If the **CbCombo** has the simple style, the closest match is scrolled to the top of the list. In this case, a matching record is not automatically selected. The user must explicitly select a record from the list and then the data file is repositioned.

If the **CbCombo** has the drop down style and there is an exact match, the matching record is automatically selected when the user drops the list and the data file is automatically repositioned.

CodeBase Calls

A direct call to CodeBase repositioning function, such as d4top(), can be used to reposition the **CbMaster's** data file directly. To do this, you need to access the **CbMaster** control's **DATA4** structure pointer. This pointer is available at runtime through the Data4 property.

NOTE

The bound controls are not updated if you use a CodeBase function call to reposition the data file. Call a CbMaster method such as Go, to update the bound controls.

Reposition Event

After a control repositions the data file, it fires a Reposition event. The **CbList**, **CbCombo**, **CbMaster**, and **CbSlider** controls can fire a Reposition event.

The **CbList** and **CbSlider** only fire Reposition events when the DataAction property is set to Record Position (1) and the control is bound to a **CbMaster**. The **CbList** fires a Reposition event whenever the user selects a record with the mouse or arrow keys. The **CbList** will also fire a Reposition event if a list entry is selected using the ListIndex property or the SetSelected method. The **CbSlider** fires a Reposition event when the thumb is moved or clicked.

The **CbCombo** can fire Reposition events when the list portion is linked to a data file. A Reposition event is fired when the user selects a record from the list with the mouse or arrow keys. A Reposition event is also fired when the selection is changed with the ListIndex property. When the user types in a drop down **CbCombo**, the list is searched for a match. If an exact match exists and the user drops the list, the data file is automatically positioned to the record with the exact match and the Reposition event is fired.

The **CbMaster** also fires Reposition events. The event is fired when the user clicks one of the positioning buttons such as Top, Bottom and so on. If the user clicks on the **CbMaster** slider, a Reposition event is fired. The event is also fired when a repositioning method is called, such as Go, Skip, Seek and so on.

The programmer can intercept the Reposition event so that any operations that need to be done after the data file is repositioned can be completed.

Tags

The **CbList**, **CbCombo** and the **CbMaster** can use tags to order the records in the data file. Of course some tags may not have entries for every record in the data file. Since the **CbList**, **CbCombo** and **CbMaster** may all use different tags, it can be possible that the current record does not have an entry in one of the tags. When the current record does not have an entry in the tag used by the **CbList** or **CbCombo**, the selection is removed from the list so it does not reflect the current record. If the user selects a record from a **CbList** or **CbCombo** and the record does not have an entry in the **CbMaster's** selected tag, the **CbMaster** is not repositioned and the other bound controls are not updated.

Obtaining Values from Controls

Field values may be retrieved from **CbEdit** and **CbCombo** controls through their Text property. A particular item can be retrieved from a **CbList** or the list portion of the **CbCombo** through its List property. The thumb position on a **CbSlider** may represent a field value, which can be retrieved through the Value property. Values can also be retrieved directly from the **CbMaster** control by calling the GetField method. See the "Properties" chapter for more information on these properties.

Formatting

The **CbEdit, CbList** and **CbCombo** can format the field values according to a desired mask. See the "CodeControls Concepts" for more information on formatting data. A programmer may want retrieve the data from a masked control without the formatting characters. In this case, set the FormatStripping property of the control to Text Stripping (2) or Text & Data File Stripping (3), either of which will cause the Text property to retrieve the data without the formatting characters. If you want to retrieve the Text property with the formatting intact then set the FormatStripping property to Data File (1). This setting will strip the formatting characters from the data when it is assigned to the data file but formatting remains intact when the programmer accesses the Text property.

Modifying Records

Once you have linked the controls in your application to a data file, you may wish to modify the data in the file. There are four ways in which the current record of the **CbMaster** control can be modified:

1. Through user interaction with a bound control, such as a **CbEdit**, **CbList**, **CbCombo**, **CbSlider**
2. By assignment, to the Text property of the **CbEdit** and **CbCombo**.
3. By using the **CbMaster** SetField method.
4. Directly, through a CodeBase function call.

Direct Assignment

The first two techniques are straightforward; if the user makes changes to the value in a control, the current record also changes to reflect these modifications.

CbEdit

As the user types into the **CbEdit**, the current record is automatically updated with the new field value.

CbList

The **CbList** can be used to modify a field by binding the **CbList** to a field and filling the list with valid field values. When the user selects an item from the list, the bound field of the current record is automatically updated with the selected data. The **CbList** DataAction property must be set to Field Value (2), for this behavior.

CbSlider

The **CbSlider** can be used to modify a field, when the DataAction property is set the Field Value (2). When the users moves the slider thumb by clicking the slider or arrow buttons, the value of the relative position of the thumb on the slider is assigned to the field. The programmer must set the MinValue, MaxValue and the ScrollUnit properties so that the desired values are assigned to the field. For example, set the MinValue to 0 and the MaxValue to 10. If the user dragged the thumb half-way down the slider, the value 5 would be assigned to the bound field of the current record.

CbCombo

The **CbCombo** can be used to modify a field by binding the edit portion to a field. The user may modify the bound field in two ways. First the user may type in the edit portion of the **CbCombo**. As the user types, the field of the current record is automatically updated. Note that the user may only type in the edit portion of a **CbCombo** with the dropdown or simple style. The second way to modify the bound field is by selecting an item from the list portion. The selected data is automatically assigned to the field of the current record. Any of the three styles of **CbCombo** can be used to modify a field in this way.

Text property

Similarly, if the contents of a control are changed through program code by assigning a new value to the Text property, the current record is also modified to reflect these changes. Only the **CbEdit** and **CbCombo** controls have Text properties which can change the current record when they are bound to fields.

If the control is using formatting, the FormatStripping determines whether formatted data can be assigned to the Text property. If the FormatStripping is set to Data File stripping (1) or None (0), then formatted data can be assigned to the Text property. If the FormatStripping is set to Text stripping (2) or Data File and Text stripping (3), then raw data must be assigned to the Text property and data will be formatted internally.

CbMaster SetField Method

The programmer may change the field value by calling the SetField method of the **CbMaster**. The SetField method takes two parameters, the name of the field and the new field value contained in a Variant data type. When the SetField method is used to change a field value, the bound controls are automatically updated. See the "Methods" chapter for more information on how to use SetField.

CodeBase Calls

Changes to the data file may also be made through CodeBase function calls. When this is done, it is necessary to explicitly update the controls to reflect these changes.

At runtime, after the **CbMaster** control's data file has been opened, the **Data4** property contains a pointer to the data file's **DATA4** structure pointer. This is the same type of pointer returned by the **d4open** function (CodeBase C, Visual Basic and Delphi versions).

NOTE

CodeBase C++ users may convert this pointer to a **Data4** object by constructing a new object with

Data4::Data4(DATA4 *) .

You can use this pointer to directly modify the **CbMaster** control's data file, as well as modifying the contents of the current record.

When you modify a record directly, any controls bound to the **CbMaster** are not automatically updated to reflect these changes. The programmer can update the bound controls by calling a **CbMaster** method such as FlushRecord or Go.

Appending Records

Clicking the **Append** button on a **CbMaster** control causes a record to be appended to the data file, which the user can modify. The append can be aborted at this point by clicking the Undo button or calling the Undo method. The appended record is not actually flushed to disk until the data file is repositioned or the record is explicitly flushed. The appended record can be flushed by clicking the FlushRecord button or by calling the FlushRecord method.

AppendRecordType

The programmer may want to append a blank record or a copy of the current record, depending on the situation. To accommodate these different circumstances, the **CbMaster** provides the AppendRecordType property, which determines what kind of record will be appended. If AppendRecordType is set to Append Blank (0), the record buffer is blanked and this enables the user to enter text over the 'blank space'. If AppendRecordType is set to Append Duplicate with Memo (1), then the current record is appended. In this case, if a record has a memo entry, it's contents are also used in the append. Finally, if AppendRecordType is set to Append Duplicate No Memo (2), a copy of the current record is appended as above except that any memo entries are blanked out.

CodeBase Calls

A record may also be appended using CodeBase function calls with the **DATA4** pointer of the **CbMaster** control (from the Data4 property). Once an append is made in this manner, it would be necessary to update the bound controls calling a **CbMaster** method such as FlushRecord.

Deleting Records

The current record can be marked for deletion in three ways. The user can click the Deleted button on the **CbMaster** at runtime to toggle the deletion status of the current record. The programmer can use the Deleted property or the Delete method to toggle the deletion status of the current record.

CodeBase Calls

A record may also be deleted using CodeBase function calls with the **DATA4** pointer from the **CbMaster** control (from the Data4 property). Once a deletion is made in this manner, however, it may be necessary to update the bound controls calling the **CbMaster** FlushRecord method or a repositioning method.

NOTE

In both methods described above, the current record is only marked for deletion. It is not physically removed from the data file and can still displayed in the controls (the bar on the **CbMaster**'s Delete button becomes red when a record marked for deletion becomes the current record).

To prevent deleted records from being displayed, you can create and select a tag with a ".NOT. DELETED()" filter.

Packing

The records marked for deletion are physically removed from the data file when the data file is packed. The data file can be packed by clicking the **CbMaster** Pack button or by calling the Pack method.

Packing can only occur when the **CbMaster** has its data file opened exclusively. In fact, the Pack button and method are disabled when the data file is not opened exclusively. The data file can be opened exclusively by setting the AccessMode property to Deny All (2) at design time. If you want to open the data file exclusively at runtime, set AccessMode to DenyAll (2) and then call RefreshFiles to reopen the data file.

Reindexing

Index files can become out date if they remain closed while the data file is being modified or if there is power failure during an update. So it sometimes becomes necessary to recreate the index files associated with linked data file. Clicking the Reindex button on the **CbMaster** or calling the Reindex method will reindex the index files specified by the IndexFiles property.

Reindexing can only occur when the **CbMaster** has its data file opened exclusively. In fact, the Reindex button and method are disabled when the data file is not opened exclusively. The data file can be opened exclusively by setting the AccessMode property to Deny All (2) at design time. If you want to open the data file exclusively at runtime, set AccessMode to DenyAll (2) and then call RefreshFiles to reopen the data file.

Validate Event

When the current record is modified by the user of your application, you may wish to validate the changes before saving them to disk. The **CbMaster** control triggers a Validate event before many of the **CbMaster** actions are initiated. Intercepting this event allows the programmer to decide whether any changes should be flushed and whether a particular action should be allowed to continue.

Any of the following actions can trigger the event:

- The user clicks a positioning button (Top, Bottom, Next, NextPage, Previous, PreviousPage) , the Append button, the Search button, or when the user clicks the slider on the **CbMaster** control.
- The programmer calls the reposition methods (Top, Bottom, Skip, Go, Seek, SeekDouble, SeekN, SeekNext, SeekNextDouble, SeekNextN, Search) or the Append method.

- Changing the RecNo or Position properties of the **CbMaster**. Changing the RecNo property has the same effect as calling the Go method. Changing the Position property has the same effect as clicking the slider on the **CbMaster**.
- The form or dialog containing the **CbMaster** control is unloaded or whenever the **CbMaster** closes the data file, which occurs when the RefreshFiles and Unlink methods are called.
- The FlushRecord button is clicked or the FlushRecord method is called.

Before any of the above actions take place, the **CbMaster** attempts to flush the current record if it has been modified. The Validate event is fired just before the **CbMaster** attempts to flush any changes.

The programmer may want to cancel the action that triggered the Validate event in the first place. The action can be aborted, by intercepting the Validate event and setting the *Response* parameter to false.

Since the Validate event is fired before any changes are flushed, the programmer can intercept the Validate event and discard the changes before any flushing takes place. At this point, the programmer may want the **CbMaster** action to continue, in which case the *Response* parameter must remain true. On the other hand, if the *Response* parameter is set to false, the action is aborted.

Detecting Changes

The **CbMaster** control provides the RecordChanged property for detecting record changes. When any bound control modifies the record, the RecordChanged property is set to True. The RecordChanged property is read-only at runtime.

Saving Changes

When you know a record has changed and you want to write the changes to disk immediately, you can use a variety of methods to save the changes to disk.

The easiest way to save changes to disk is simply to let the **CbMaster** control do it for you. When a record is changed, the **CbMaster** control automatically writes the changes to disk when the user selects another action on the **CbMaster** control, such as moving to another record, appending a new record, or unloading the control's parent form.

If you need to update changes any earlier, you can call the **CbMaster**'s FlushRecord method. The **CbMaster** then attempts to write any changes to the current record to disk.

Canceling Changes

If the record has been changed and you want to discard the changes, then call either the **CbMaster** Undo or RefreshRecord method. These methods set the current record back to the original value and set the RecordChanged property to false.

The Undo method resets the current record from information stored in an internal buffer in memory whereas the RefreshRecord method re-reads the current record from disk. The RefreshRecord method can be useful in multi-user situations.

Result of Record Changes

Regardless of whether you let CodeControls automatically save changes for you, or whether your code performs this action through the properties and methods mentioned above, there are four possible outcomes to the saving (flushing) operation:

1. The operation succeeds.
2. An error occurs while flushing. In general, CodeBase will issue an error message directly.
3. A required lock cannot be placed. This outcome can occur when Optimistic locking is used by the **CbMaster**. The **CbMaster** attempts to lock a modified record before it is flushed to disk. If the lock attempt fails, the NoUpdate event is fired.
4. Flushing causes a duplicate key in a unique key index. If the index has a unique tag then the Unique property of **CbMaster** should be set to 1. This setting means that duplicate keys are not allowed in unique tags and if flushing a modified record would cause a duplicate key in a unique tag, the DuplicateKey event is fired. The DuplicateKey event is never fired if the Unique property is set to 0 and as a result, duplicate keys can be added to unique tags.

The NoUpdate and DuplicateKey events allow you to respond to these situations in whatever manner is appropriate to your application -- you can allow the default dialogs to appear, or you can modify the default behavior. See "Events" chapter for details on these events.

Locking

There are two kinds automatic locking methods provided by the **CbMaster** called Optimistic and Pessimistic locking.

The Optimistic locking occurs when a modified record is about to be flushed. The **CbMaster** tries to lock the record before it writes the record to disk. If the **CbMaster** fails to lock the record because it is locked by another application, the NoUpdate event is fired. If the **CbMaster** can not lock the record, it can not be flushed to disk.

Pessimistic locking occurs when the record is about to be modified. When the user attempts to change a record, the **CbMaster** tries to lock the record. If the **CbMaster** fails to lock the record, the NoEdit event is fired. If the **CbMaster** can not lock the record, the record can not be modified. The NoUpdate event will never be fired when Pessimistic locking is chosen, because if the record is modified, it will already be locked when it comes time to flush the changes.

See "Events" chapter for details on NoEdit and NoUpdate.

Error Event

When an error occurs during application execution, CodeControls triggers the Error event. All the controls can fire an Error event. The Error event can display a message box describing the error. If you don't want this default dialog response you can supply your own, or have none at all by intercepting the Error event and setting the *Response* parameter to false. In addition you can augment the event with your own code if you wish to perform other operations as well.

The Error event has three parameters. The first is *ErrorCode*, which is a long integer representing the error code. The second is *Description*, which is string describing the error. The third parameter is *Response*, which is an integer that can be modified to indicate whether default error message box should be issued. By default, the *Response* parameter is set to true, which means that the default error message box is displayed.

Generally, if the program does not recover from the Error event, then the control that fired the event becomes disabled. If the control becomes disabled because of an error, then the programmer must explicitly enable the control by setting the Enabled property to true, after the recovery is completed. If the code in the intercepted Error event causes the program to recover from the error then the control remains enabled. An infinite recursive loop may be generated if the code in the intercepted Error event causes another Error event to be fired. If this happens, the application will run out of stack space.

The following describes when the various controls issue an Error Event.

The Error event is fired by the **CbButton** control when the DataSource property is set to an invalid value.

The Error event is fired by the **CbCombo** control whenever the DataSourceEdit, DataSourceList, DataFieldEdit, DataExprList, or DataTagList properties are set to an invalid value. If the DataTagList is set to an invalid tag, the Error event is fired and the DataTagList is automatically reset to the previous valid tag. In this case the control remains enabled after the Error event is fired.

The **CbEdit** control fires the Error event whenever the DataSource or DataField properties are set to an invalid value.

The **CbList** controls fires the Error event whenever the DataSource, DataExpr or DataTag properties are set to an invalid value. If the DataTag is set to an invalid tag, the Error event is fired and the DataTag is automatically reset to the previous valid tag. In this case the control remains enabled after the Error event is fired.

The **CbMaster** control fires the Error event whenever the DatabaseName, IndexFiles or DataTag properties are set to invalid values. If the DataTag is set to an invalid tag, the Error event is fired and the DataTag is automatically reset to the previous valid tag. In this case, the control remains enabled after the Error event is fired.

The **CbSlider** control fires the Error event whenever the DataSource or DataField property is set to an invalid value.

Exceptions

Some of the controls will throw exceptions when their properties are set to invalid values. After the exception is thrown, the property reverts to the original value.

The **CbEdit**, **CbCombo** and **CbList** will throw exceptions if the Decimals or MaxLength properties are changed to invalid value. If numeric formatting has been enabled, the MaxLength property must be able to accommodate the number of decimals specified by the Decimals property, the decimal point and the negative sign. If the MaxLength property is changed to value that is too small, then an exception is thrown. If the user tries to set the Decimals property to a value that is too large to be accommodated by the current MaxLength, an exception is thrown.

If the DefaultDate property of the **CbEdit**, **CbList** or **CbCombo** is set to an invalid value or is null, an exception is thrown.

The **CbMaster** must create its own unique name when it is placed on a container that does not support the ambient name property. The containers under Delphi, VB and C++ Builder will automatically create a unique name for each instance of the control. VC++ dialog based containers do not generate unique names for each control instance. Since the **CbButton**, **CbCombo**, **CbEdit**, **CbList** and **CbSlider** need to have a list of available **CbMasters** on the container, the **CbMaster** generates its own unique name when it is created under VC++. The unique name may be changed at design time by setting the Name property. If the name is not unique among all the **CbMasters** on the dialog, an exception is thrown.

See "Properties" chapter for more details on the Decimals, MaxLength, DefaultDate and Name properties.

Customizing CbMaster

If you wish to alter, augment or replace the database management functionality built into the **CbMaster** control, you can do so using the ButtonClick event. When the user clicks one of the buttons in a **CbMaster** control, the ButtonClick event is fired. The ButtonClick event has a parameter called *Action*, which identifies which button was clicked.

By default, CodeControls automatically handles the event in the appropriate manner, based on the value of *Action*. For example, if *Action* equals Top (0), the **CbMaster** attempts to reposition the data file to the first logical record and updates any bound controls.

You also have the ability to augment or redefine the functionality of the **CbMaster** by performing other actions in addition to, or instead of the default actions defined by *Action*. For example, if *Action* equals Deleted (8), you may want to display a message box, prompting the user to confirm his action before continuing. If the user decides not to delete the record, the program sets *Action* to false. This notifies the **CbMaster** not to perform its default action, which in this case would be to delete the current record.

You cannot change *Action* to another non-zero value in order to change the type of button that was clicked. For example changing the value of *Action* from Top (0) to Bottom (6) will not cause the **CbMaster** control to move to the bottom record. If you change *Action* to any non-zero value, CodeControls simply ignores this value. If you require this type of functionality, you can achieve it by calling the appropriate **CbMaster** method, which will perform the database operation and then set the *Action* parameter to false.

Customizing the CbMaster Appearance

The **CbMaster** can show up to 14 different buttons, a spin list containing tag names, a slider, a caption and the current record number. Not all of the features and buttons may be required for a particular application. In fact, the programmer may want the **CbMaster** to be invisible at runtime. By setting the properties discussed below, the programmer can customize the appearance and functionality of the **CbMaster** control.

Each button has a *button*Visible property. The *button* part of the *button*Visible stands for the name of the button, which can be any one of the following: Append, Bottom, Deleted, Flush, Next, NextPage, Pack, Previous, PreviousPage, Refresh, Reindex, Search, Top and Undo. By default all of the buttons are visible and the *button*Visible properties are set to true. If a particular *button*Visible property is set to false, the button is not displayed. The programmer can customize the look of the **CbMaster** at design time and runtime by setting the *button*Visible properties.

Each button has a *button*Enabled property. The *button* part of the *button*Enabled stands for the name of the button as discussed for *button*Visible. By default, all of the buttons are enabled and the *button*Enabled properties are set to true. If the *button*Enabled property is set to false the button is disabled, but still visible. The **CbMaster** displays a special grayed out bitmap for the disabled button and the button does not receive any keyboard or mouse input. The *button*Enabled properties may be changed at design time or runtime.

The **CbMaster** can display a spin list containing tag names. When the user selects a tag from the list, the **CbMaster** traverses the data file according to the selected tag. The TagListEnabled property determines whether the tag list will be visible. By default, the TagListEnabled property is set to true and the tag list is displayed. Setting TagListEnabled to false, disables the tag list and it is no longer enabled and visible. The TagListEnabled property may be changed at design time or runtime.

Similarly, the **CbMaster** has a RecNoEnabled and SliderEnabled property which determine whether the current record number and slider are displayed . By default, each property is set to true and the record number and slider are displayed. These properties may be set at design time or runtime.

The **CbMaster** also has a CaptionVisible property which determines whether the caption is displayed. By default, CaptionVisible is set to true and the caption is displayed. CaptionVisible can by set at design time and runtime.

Finally the **CbMaster** has a Visibility property. This property determines whether the **CbMaster** is visible at runtime. When this property is set to false, the **CbMaster** is totally invisible at runtime, regardless of the settings of the properties discussed above. At design time the **CbMaster** remains visible for development purposes. The **CbMaster** retains its functionality even when it is invisible at runtime. The **CbMaster** still opens the data file, the methods are still functional, and other controls can still be bound to the **CbMaster**. Only the graphical interface is disabled when the **CbMaster** is invisible at runtime. The Visibility property may changed at design time and runtime. When the **CbMaster** becomes visible at runtime, it is displayed according to the properties discussed above.

Enabled Property

It is possible that situation may arise where you might want a control to be disabled. A disabled control is still visible but it can not receive mouse or keyboard input. Each CodeControls control has an Enabled property. When this property is set to true, the control is active and can receive mouse and keyboard input. The control is disabled by setting Enabled to false.

In general, a control is disabled after an Error event occurs. If the control is disabled due to an error, the programmer must explicitly enabled the control after the recovery has been completed.

The **CbMaster** control is automatically disabled when the Unlink method is called and is automatically enabled when the RefreshFiles method is called.

The **CbButton** is automatically enabled and disabled if the DataAction property is set to Flush (3), Pack (6), Refresh (9), Reindex (10), Deleted (12) or Undo (14). For example, if the **CbButton** is being used to emulate the Undo button of the **CbMaster** control, the **CbMaster** signals the **CbButton** as to whether it should be enabled. In this case, the Undo button is only enabled when the current record has been modified. The programmer can not enable the **CbButton** if it has received a message from the **CbMaster** that it should remain disabled. On the other hand if a **CbButton** has received a message to become enabled, the programmer may disable **CbButton** explicitly and subsequently enable it again.

Visibility Property

There are some situations that require a control to be invisible at runtime. The control is still functional while it is invisible and only its graphical interface has been disabled. The control can still be bound to a **CbMaster** and the properties and methods are still fully functional.

Each control has a Visibility property which is set to true by default. The control becomes invisible at runtime by setting Visibility to false. The Visibility property can be set at design time or runtime. The control is always visible at design time for development purposes.

NOTE

Visual Basic, Delphi and C++ Builder automatically provide a Visible property for each OLE control. Do NOT use this property if you want a control to be invisible at runtime. The default Visible property causes problems for the controls at runtime. Instead, use the Visibility property that is provided for each control in CodeControls.

5 Properties

Properties

The following reference lists all of the properties that are unique to CodeControls.

CbMaster Properties				
AccessMode	DatabaseName	NameUnique	PreviousEnabled*	SearchVisible*
AppendEnabled*	DataTag	NextEnabled*	PreviousPageEnabled*	ServerId
AppendRecordType	Deleted	NextPageEnabled*	PreviousPageVisible*	ShadowColor
AppendVisible*	DeletedEnabled*	NextPageVisible*	PreviousVisible*	SliderEnabled
BackColor	DeletedVisible*	NextVisible*	ProcessId	Tag4
BevelThickness	Enabled	OptimisticLocking	ReadOnly	TagListEnabled
BevelType	EOFAction	OptimizeRead	RecNo	ToolTipsEnabled
BOFAction	FlushEnabled*	OptimizeWrite	RecNoEnabled	TopEnabled*
BorderStyle	FlushVisible*	Orientation	RecordChanged	TopVisible*
BottomEnabled*	Font	PackEnabled*	RefreshEnabled*	UndoEnabled*
BottomVisible*	ForeColor	PackVisible*	RefreshVisible*	UndoVisible*
Caption	HighlightColor	PageSize	ReindexEnabled*	Unique
CaptionVisible	hWnd	Password	ReindexVisible*	UserName
Data4	IndexFiles	Position	SearchEnabled*	Visibility

CbButton Properties				
AltBitmapDisabled	BitmapDown	DataAction	HighlightColor	UseAltBitmaps
AltBitmapDown	BitmapFocus	DataSource	hWnd	UseBitmaps
AltBitmapFocus	BitmapUp	DefaultBitmaps	ScaleButton	Visibility
AltBitmapUp	Caption	Enabled	ShadowColor	
BackColor	Caption3DText	Font	ToolTipEnabled	
BitmapDisabled	CaptionVisible	ForeColor	TooTipText	

CbEdit Properties				
Alignment	DecimalChar	hWnd	PositiveLeader	ThouSeparatorChar
AllowPrompt	Decimals	Locked	PositiveTrailer	ThouSeparatorEnabled
Appearance	DefaultDate	Mask	PromptChar	ToolTipEnabled
AutoMonthFillin	Enabled	MaxLength	PromptIncluded	ToolTipText
BackColor	Font	MultiLine	SelLength	Visibility
BorderStyle	ForeColor	NegativeColor	SetStart	
Case	FormatStripping	NegativeLeader	SelText	
DataField	FormatType	NegativeTrailer	ScrollBars	
DataSource	HideSelection	PasswordChar	Text	

CbList Properties				
Appearance	DecimalChar	hWnd	NegativeLeader	ThouSeparatorEnabled
BackColor	Decimals	List	NegativeTrailer	ToolTipEnabled
Case	DefaultDate	ListCount	PositiveLeader	ToolTipText
Columns	Enabled	ListIndex	PositiveTrailer	TopIndex
DataAction	Font	ListItems	SelCount	Visibility
DataExpr	ForeColor	Mask	Sorted	
DataSource	FormatStripping	MaxLength	Text	
DataTag	FormatType	MultiSelect	ThouSeparatorChar	

CbCombo Properties				
AllowPrompt	DataSourceList	FormatType	NegativeTrailer	Style
Appearance	DataTagList	hWnd	PositiveLeader	Text
AutoMonthFillin	DecimalChar	List	PositiveTrailer	ThouSeparatorChar
BackColor	Decimals	ListCount	PromptChar	ThouSeparatorEnabled
Case	DefaultDate	ListIndex	PromptIncluded	ToolTipEnabled
ContainerColor	Enabled	ListItems	SelLength	ToolTipText
DataFieldEdit	Font	Mask	SelStart	Visibility
DataExprList	ForeColor	MaxLength	SelText	
DataSourceEdit	FormatStripping	NegativeLeader	Sorted	

CbSlider Properties				
Appearance	DataField	HighlightColor	Orientation	ToolTipEnabled
BackColor	DataSource	hWnd	ScrollUnit	ToolTipText
BorderStyle	Enabled	MaxValue	ShadowColor	Value
DataAction	ForeColor	MinValue	ShowButtons	Visibility

Default Properties that are provided to OLE control by Visual Basic				
DragIcon	HelpContextID	Name	Tag	WhatsThisHelpID
DragMode	index	TabIndex	Top	Width
Height	Left	TagStop	Visible	

Default Properties that are provided to OLE control by Delphi and C++ Builder				
Cursor	HelpContext	ParentColor	ShowHint	Top
DragCursor	Hint	ParentFont	TabOrder	Visible
DragMode	Left	ParentShowHint	TabStop	Width
Height	Name	PopUpMenu	Tag	

Default Properties that are provided to OLE control by VC++				
Disabled	Visible	Group	Help Id	Tab Stop

* The descriptions for these properties are listed as follows: The property names ending in Visible are listed under *buttonName*Visible and the property names ending in Enabled are listed under *buttonName*Enabled.

The syntax given in the Usage statements illustrate how to use the properties in a program written under Visual Basic, Visual C++, Delphi or Borland C++ Builder. This is not the syntax that would appear in a header file. The italicized part of the Usage statements represent the name of a particular instance of the control.

Any part of the Usage statement in square brackets is optional. The *form* represents the name of the form where the control is located. The *CbCtrl* represents the name of a particular control on the form. You can retrieve the value of a property or you can set the property to a new value.

The Usage statements under Visual Basic, Delphi and C++ Builder are very similar. Let's use the Visual Basic Usage statement for the AccessMode property as an example:

VB Usage: [*form*].*CbCtrl*.**AccessMode**[= *setting%*]

The following example shows how to retrieve the value of the AccessMode property under Visual Basic.

```
Dim accessMode As Integer

accessMode = Form1.CbMaster1.AccessMode
```

The following example shows how to set the AccessMode property to a new value under Visual Basic.

```
Form1.CbMaster1.AccessMode = 2
```

Under Visual C++, two functions are provided for retrieving and setting properties. The m_*CbCtrl* represents the name given to the control using the Member Variables tab of the ClassWizard.

VC++ Usage: short *m_CbCtrl*.**GetAccessMode( )**
void *m_CbCtrl*.**SetAccessMode(** short *setting***)**

The following example shows how to retrieve the value of the AccessMode property under Visual C++.

```
short  accessMode;

accessMode = m_cbMaster1.GetAccessMode( );
```

The following example shows how to set the AccessMode property to a new value under Visual C++.

```
m_cbMaster1.SetAccessMode( 2 );
```

AccessMode Property

VB Usage: [*form*].*CbCtrl*.**AccessMode**[= *setting%*]

VC++ Usage: short *m_CbCtrl*.**GetAccessMode( )**
void *m_CbCtrl*.**SetAccessMode(** short *setting***)**

Delphi Usage: [*form*].*CbCtrl*.**AccessMode**[:= *setting:* short]

C++ Builder Usage: [*form*]->*CbCtrl*->**AccessMode**[= short *setting*]

Applicable To: **CbMaster**

Description: Sets or returns the access mode for the data file opened by the **CbMaster** control. AccessMode specifies what OTHER users will be able to do with the file. The ReadOnly property specifies what the CURRENT user will be able to do with the file.

AccessMode must be set to Deny All (2) whenever packing or reindexing files. If AccessMode is not set to Deny All (2) then the reindex and the pack buttons on the **CbMaster** will not be enabled nor will the Reindex or Pack methods have any effect.

Call RefreshFiles after changing the AccessMode property at runtime. The RefreshFiles closes the data file and then reopens the data file according to the new AccessMode setting.

AccessMode is read/write at design time and runtime.

The possible values for *setting* are:

Settings	Value	Description
Deny None Access	0	Open the data files in shared mode. Other users have read and write access. This is the default setting.
Deny Write Access	1	Open the data files in shared mode. The application may read and write to the files, but other users may open the file in read only mode and may not change the files.
Deny All Access	2	Open the data files exclusively. Other users may not open the files.

See Also: **ReadOnly** property, **Reindex, Pack, RefreshFiles** methods

Alignment Property

VB Usage: [*form*].*CbCtrl*.**Alignment**[= *setting%*]

VC++ Usage: short *m_CbCtrl*.**GetAlignment( )**
void *m_CbCtrl*.**SetAlignment(** short *setting***)**

Delphi Usage: [*form*].*CbCtrl*.**Alignment**[:= *setting:* short]

C++ Builder Usage: [*form*]->*CbCtrl*->**Alignment**[= short *setting*]

Applicable To: **CbEdit**

Description: Sets or returns the alignment of the text in a **CbEdit** control. As the user types into the **CbEdit** at runtime, the text is automatically justified according to this property.

This property is read-only at runtime and read/write at design time.

The possible values for *setting* are:

Settings	Value	Description
Left Justified	0	The text is left justified. This is the default setting.
Right Justified	1	The text is right justified.
Centered	2	The text is centered.

AllowPrompt Property

VB Usage: [*form*].*CbCtrl*.**AllowPrompt**[= {**TRUE** | **FALSE**}]

VC++ Usage: BOOL *m_CbCtrl*.**GetAllowPrompt( )**
void *m_CbCtrl*.**SetAllowPrompt(** BOOL *setting***)**

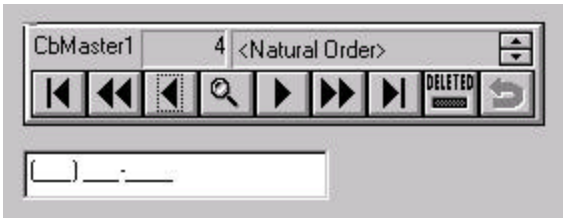
Delphi Usage: [*form*].*CbCtrl*.**AllowPrompt**[:= {**TRUE** | **FALSE**}]

C++ Builder Usage: [*form*]->*CbCtrl*->**AllowPrompt**[= {**TRUE** | **FALSE**}]

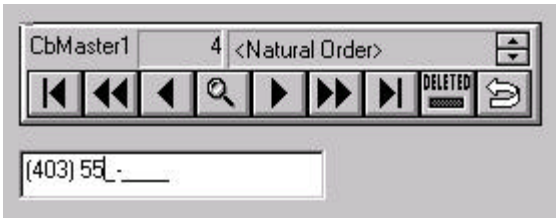
Applicable To: **CbEdit, CbCombo**

Description: This property is used when entering formatted data into a **CbEdit** or the edit portion of a **CbCombo**. If AllowPrompt is set to true, then the character specified by the PromptChar property is used as a place holder for any data to be entered by the user. The place holder characters are used to indicate how many characters should be entered into the field by the user. The default PromptChar is an underscore ("_") character. If the AllowPrompt property is set to false, then the PromptChar character is not displayed as place holder for the data.

The following example uses a character field to store a phone number with area code. The mask used by the **CbEdit** control is "(999)999-9999" where the 9's represent digits to be entered by the user. The AllowPrompt is set to true, so the PromptChar is used a place holder for empty field characters.



The phone number field for record 4 is blank and the characters to be entered by the user are represented by the underscores.



The phone number is being entered digit by digit and replacing the place holder characters.

AllowPrompt is read/write at design time and runtime.

See Also: **FormatType** property

AltBitmapDisabled Property

VB Usage: [*form*].*CbCtrl*.**AltBitmapDisabled**[= LoadPicture(*bitmapName*)]

VC++ Usage: CPicture *m_CbCtrl*.**GetAltBitmapDisabled( )**
void *m_CbCtrl*.**SetAltBitmapDisabled(** LPDISPATCH *newBitmap* **)**

Delphi Usage: [*form*].*CbCtrl*.**AltBitmapDisabled**[:= *newBitmap:* Variant]

C++ Builder Usage: [*form*]->*CbCtrl*->**AltBitmapDisabled**[= Variant *newBitmap*)]

Applicable To: **CbButton**

Description: Four bitmaps must be defined if the programmer wants to customize the appearance of the **CbButton**. The bitmaps must illustrate how the **CbButton** will look when it is in the up position without the focus, in the up position with the focus, in the down position and when it is disabled. There are two sets of four properties that can be used to customize the appearance of the **CbButton**. The first set of properties are BitmapDisabled, BitmapDown, BitmapFocus and BitmapUp, which specify the main custom bitmaps. The second set of properties are AltBitmapDisabled, AltBitmapDown, AltBitmapFocus and AltBitmapUp, which specify the alternative custom bitmaps.

The alternative custom bitmaps are used when the UseBitmaps and UseAltBitmaps properties are set to true. The DefaultBitmaps property must be set to 0, so that the custom bitmaps will be used. The AltBitmapDisabled property specifies the custom alternative bitmap for the disabled position of the **CbButton**.

This property is read/write at design time and runtime.

Visual Basic provides the LoadPicture function to load graphics files into a picture boxes, forms or image controls. The return value from this function can also be assigned to the **CbButton** AltBitmapDisabled property at runtime.

See Also: **AltBitmapDown, AltBitmapUp, AltBitmapFocus, UseBitmaps, UseAltBitmaps, DefaultBitmaps** properties

AltBitmapDown Property

VB Usage: [*form*].*CbCtrl*.**AltBitmapDown**[= LoadPicture(*bitmapName*)]

VC++ Usage: CPicture *m_CbCtrl*.**GetAltBitmapDown( )**
void *m_CbCtrl*.**SetAltBitmapDown(** LPDISPATCH *newBitmap* **)**

Delphi Usage: [*form*].*CbCtrl*.**AltBitmapDown**[:= *newBitmap:* Variant]

C++ Builder Usage: [*form*]->*CbCtrl*->**AltBitmapDown**[= Variant *newBitmap*)]

Applicable To: **CbButton**

Description: Four bitmaps must be defined if the programmer wants to customize the appearance of the **CbButton**. The bitmaps must illustrate how the **CbButton** will look when it is in the up position without the focus, in the up position with the focus, in the down position and when it is disabled. There are two sets of four properties that can be used to customize the appearance of the **CbButton**. The first set of properties are BitmapDisabled, BitmapDown, BitmapFocus and BitmapUp, which specify the main custom bitmaps. The second set of properties are AltBitmapDisabled, AltBitmapDown, AltBitmapFocus and AltBitmapUp, which specify the alternative custom bitmaps.

The alternative custom bitmaps are used when the UseBitmaps and UseAltBitmaps properties are set to true. The DefaultBitmaps property must be set to 0, so that the custom bitmaps will be used. The AltBitmapDown property specifies the custom alternative bitmap for the down position of the **CbButton**.

This property is read/write at design time and runtime.

Visual Basic provides the LoadPicture function to load graphics files into a picture boxes, forms or image controls. The return value from this function can also be assigned to the **CbButton** AltBitmapDown property at runtime.

See Also: **AltBitmapDisabled, AltBitmapUp, AltBitmapFocus, UseBitmaps, UseAltBitmaps, DefaultBitmaps** properties

AltBitmapFocus Property

VB Usage: [*form*].*CbCtrl*.**AltBitmapFocus**[= LoadPicture(*bitmapName*)]

VC++ Usage: CPicture *m_CbCtrl*.**GetAltBitmapFocus( )**
void *m_CbCtrl*.**SetAltBitmapFocus(** LPDISPATCH *newBitmap* **)**

Delphi Usage: [*form*].*CbCtrl*.**AltBitmapFocus**[:= *newBitmap:* Variant]

C++ Builder Usage: [*form*]->*CbCtrl*->**AltBitmapFocus**[= Variant *newBitmap*)]

Applicable To: **CbButton**

Description: Four bitmaps must be defined if the programmer wants to customize the appearance of the **CbButton**. The bitmaps must illustrate how the **CbButton** will look when it is in the up position without the focus, in the up position with the focus, in the down position and when it is disabled. There are two sets of four properties that can be used to customize the appearance of the **CbButton**. The first set of properties are BitmapDisabled, BitmapDown, BitmapFocus and BitmapUp, which specify the main custom bitmaps. The second set of properties are AltBitmapDisabled, AltBitmapDown, AltBitmapFocus and AltBitmapUp, which specify the alternative custom bitmaps.

The alternative custom bitmaps are used when the UseBitmaps and UseAltBitmaps properties are set to true. The DefaultBitmaps property must be set to 0, so that the custom bitmaps will be used. The AltBitmapFocus property specifies the custom alternative bitmap for the **CbButton** in the up position with the focus.

This property is read/write at design time and runtime.

Visual Basic provides the LoadPicture function to load graphics files into a picture boxes, forms or image controls. The return value from this function can also be assigned to the **CbButton** AltBitmapFocus property at runtime.

See Also: **AltBitmapDisabled, AltBitmapUp, AltBitmapDown, UseBitmaps, UseAltBitmaps, DefaultBitmaps** properties

AltBitmapUp Property

VB Usage: [*form*].*CbCtrl*.**AltBitmapUp**[= LoadPicture(*bitmapName*)]

VC++ Usage: CPicture *m_CbCtrl*.**GetAltBitmapUp( )**
void *m_CbCtrl*.**SetAltBitmapUp(** LPDISPATCH *newBitmap* **)**

Delphi Usage: [*form*].*CbCtrl*.**AltBitmapUp**[:= *newBitmap:* Variant]

C++ Builder Usage: [*form*]->*CbCtrl*->**AltBitmapUp**[= Variant *newBitmap*)]

Applicable To: **CbButton**

Description: Four bitmaps must be defined if the programmer wants to customize the appearance of the **CbButton**. The bitmaps must illustrate how the **CbButton** will look when it is in the up position without the focus, in the up position with the focus, in the down position and when it is disabled. There are two sets of four properties that can be used to customize the appearance of the **CbButton**. The first set of properties are BitmapDisabled, BitmapDown, BitmapFocus and BitmapUp, which specify the main custom bitmaps. The second set of properties are AltBitmapDisabled, AltBitmapDown, AltBitmapFocus and AltBitmapUp, which specify the alternative custom bitmaps.

The alternative custom bitmaps are used when the UseBitmaps and UseAltBitmaps properties are set to true. The DefaultBitmaps property must be set to 0, so that the custom bitmaps will be used. The AltBitmapUp property specifies the custom alternative bitmap for the **CbButton** in the up position without the focus.

This property is read/write at design time and runtime.

Visual Basic provides the LoadPicture function to load graphics files into a picture boxes, forms or image controls. The return value from this function can also be assigned to the **CbButton** AltBitmapUp property at runtime.

See Also: **AltBitmapDisabled, AltBitmapDown, AltBitmapFocus, UseBitmaps, UseAltBitmaps, DefaultBitmaps** properties

Appearance Property

VB Usage: [*form*].*CbCtrl*.**Appearance**[= *setting%*]

VC++ Usage: short *m_CbCtrl*.**GetAppearance( )**
void *m_CbCtrl*.**SetAppearance(** short *setting* **)**

Delphi Usage: [*form*].*CbCtrl*.**Appearance**[:= *setting:* short]

C++ Builder Usage: [*form*]->*CbCtrl*->**Appearance**[= short *setting*]

Applicable To: **CbEdit, CbList, CbCombo, CbSlider**

Description: The Appearance property is used to determine what kind of border is drawn around the control. If the value of Appearance is set to Flat (0) then a flat single line border is drawn. A three-dimensional border effect is drawn if Appearance is set to 3D (1).

The default value for the Appearance property is set to 3D (1).

This property is read/write at design time and read-only at run time for the **CbEdit**, **CbList** and the **CbCombo** controls. The Appearance property for the **CbSlider** is read/write at design and run time.

See Also: **BorderStyle** property

AppendRecordType Property

VB Usage: [*form.*]*CbCtrl*.**AppendRecordType**[= *setting%*]

VC++ Usage: short *m_CbCtrl*.**GetAppendRecordType( )**
void *m_CbCtrl*.**SetAppendRecordType(**short *setting***)**

Delphi Usage: [*form*].*CbCtrl*.**AppendRecordType**[:= *setting:* short]

C++ Builder Usage: [*form*]->*CbCtrl*->**AppendRecordType**[= short *setting*]

Applicable To: **CbMaster**

Description: A programmer may want to append a blank record or a copy of the current record, under different circumstances. AppendRecordType determines what kind of record will be appended.

The following table lists the AppendRecordType property settings:

Settings	Value	Description
Append Blank	0	Append a blank record and blank memo entries. This is the default setting.
Append Duplicate with Memo	1	Append a copy of the current record and memos.
Append Duplicate with No Memo	2	Append a copy of the current record and blank out any memo entries.

NOTE

The newly appended record is not actually written to disk until the data file is repositioned or has been explicitly flushed. The append may be aborted by calling the **CbMaster** Undo method or by clicking the Undo button, anytime before the appended record has been flushed.

See Also: **Append, Undo** method

AutoMonthFillin Property

VB Usage: [*form*].*CbCtrl*.**AutoMonthFillin**[= {**TRUE** | **FALSE**}]

VC++ Usage: BOOL *m_CbCtrl*.**GetAutoMonthFillin( )**
void *m_CbCtrl*.**SetAutoMonthFillin(** BOOL *setting***)**

Delphi Usage: [*form*].*CbCtrl*.**AutoMonthFillin**[:= {**TRUE** | **FALSE**}]

C++ Builder Usage: [*form*]->*CbCtrl*->**AutoMonthFillin**[= {**TRUE** | **FALSE**}]

Applicable To: **CbEdit, CbCombo**

Description: This property is used when entering formatted data into a Date field linked to a **CbEdit** or the edit portion of a **CbCombo**. AutoMonthFillin only has an effect when the Date mask specifies that the month be entered as a string (ie. the Date mask uses MMM to MMMMMMMMM for the month). When AutoMonthFillin is set to true, the month string is automatically filled in after the user enters in the first three characters. The control automatically calculates the appropriate month and supplies the remaining characters. If AutoMonthFillin is set to false, the user must type in every character of the month string. The control automatically checks the spelling as the user types in the month and it will only allow correct letters to be entered into the field.

The language of the month strings used by the Date formatting depends on the language support chosen when the Windows 95 or Windows NT was installed on the computer. For example, if the German version of Windows 95 is installed on the computer, the Date formatting will automatically display the months in German.

AutoMonthFillin is set to true, by default.

This property is read/write at design time and runtime.

See Also: **FormatType, Mask** properties

BackColor Property

VB Usage: [*form*].*CbCtrl*.**BackColor**[= *color&*]

VC++ Usage: OLE_COLOR *m_CbCtrl*.**GetBackColor( )**
void *m_CbCtrl*.**SetBackColor(** OLE_COLOR *color***)**

Delphi Usage: [*form*].*CbCtrl*.**BackColor**[:= *color:* TColor]

C++ Builder Usage: [*form*]->*CbCtrl*->**BackColor**[= TColor *color*]

Applicable To: **CbMaster, CbButton, CbCombo, CbEdit, CbList, CbSlider**

Description: Sets or returns the color used to paint the background color of the control.

Any valid RGB value may be used for this property. The actual number of colors available is limited by the monitor's display adapter and display driver.

The BackColor property for the **CbEdit**, **CbList** and the **CbCombo** is set to the Windows system background color, by default.

The BackColor property for the **CbMaster** and the **CbSlider** is set to the Windows system button face color, by default.

This property is read/write at design time and runtime.

See Also: **ForeColor** property

BevelThickness Property

VB Usage: [*form*].*CbCtrl*.**BevelThickness**[= *width%*]

VC++ Usage: short *m_CbCtrl*.**GetBevelThickness( )**
void *m_CbCtrl*.**SetBevelThickness(** short *width***)**

Delphi Usage: [*form*].*CbCtrl*.**BevelThickness**[:= *width:* short]

C++ Builder Usage: [*form*]->*CbCtrl*->**BevelThickness**[= short *width*]

Applicable To: **CbMaster**

Description: Sets or returns the bevel width, in pixels, for the **CbMaster** control. This property only applies when the BevelType property is set to Inset Bevel (2) or Raised Bevel (3).

The larger the setting, the greater the control appears to be inset/raised off the form. Valid ranges for this property are from 0 to 10.

If an attempt is made to set this property to a value greater than 10, the property is automatically set to 10, by default. Similarly, if an attempt is made to set BevelThickness to a value less than 0, the property is automatically set to 0.

BevelThickness is set to 2, by default.

This property is read/write at design time and runtime.

See Also: **BevelType, BorderStyle** properties

BevelType Property

VB Usage: [*form*].*CbCtrl*.**BevelType**[= *setting%*]

VC++ Usage: short *m_CbCtrl*.**GetBevelType( )**
void *m_CbCtrl*.**SetBevelType(**short *setting***)**

Delphi Usage: [*form*].*CbCtrl*.**BevelType**[:= *setting:* short]

C++ Builder Usage: [*form*]->*CbCtrl*->**BevelType**[= short *setting*]

Applicable To: **CbMaster**

Description: Determines what kind of 3D border the **CbMaster** will display.

The following table lists the BevelType property settings:

Settings	Value	Description
No Bevel	0	The CbMaster does not have a 3D border.
3D Bevel	1	The CbMaster has a 3D border.
Inset Bevel	2	The CbMaster has an inset 3D border.
Raised Bevel	3	The CbMaster has a raised 3D border. This is the default setting.

This property is read/write at design time and runtime.

See Also: **BevelThickness, BorderStyle** properties

BitmapDisabled Property

VB Usage: [*form*].*CbCtrl*.**BitmapDisabled**[= LoadPicture(*bitmapName*)]

VC++ Usage: CPicture *m_CbCtrl*.**GetBitmapDisabled( )**
void *m_CbCtrl*.**SetBitmapDisabled(** LPDISPATCH *newBitmap* **)**

Delphi Usage: [*form*].*CbCtrl*.**BitmapDisabled**[:= *newBitmap:* Variant]

C++ Builder Usage: [*form*]->*CbCtrl*->**BitmapDisabled**[= Variant *newBitmap*)]

Applicable To: **CbButton**

Description: Four bitmaps must be defined if the programmer wants to customize the appearance of the **CbButton**. The bitmaps must illustrate how the **CbButton** will look when it is in the up position without the focus, in the up position with the focus, in the down position and when it is disabled. There are two sets of four properties that can be used to customize the appearance of the **CbButton**. The first set of properties are BitmapDisabled, BitmapDown, BitmapFocus and BitmapUp, which specify the main custom bitmaps. The second set of properties are AltBitmapDisabled, AltBitmapDown, AltBitmapFocus and AltBitmapUp, which specify the alternative custom bitmaps.

The main custom bitmaps are used when the UseBitmaps property is set to true and the UseAltBitmaps property is set to false. The DefaultBitmaps property must be set to 0, so that the custom bitmaps will be used. The BitmapDisabled property specifies the custom bitmap for the disabled position of the **CbButton**.

This property is read/write at design time and runtime.

Visual Basic provides the LoadPicture function to load graphics files into a picture boxes, forms or image controls. The return value from this function can also be assigned to the **CbButton** BitmapDisabled property at runtime.

See Also: **BitmapDown, BitmapUp, BitmapFocus, UseBitmaps, UseAltBitmaps, DefaultBitmaps** properties

BitmapDown Property

VB Usage: [*form*].*CbCtrl*.**BitmapDown**[= LoadPicture(*bitmapName*)]

VC++ Usage: CPicture *m_CbCtrl*.**GetBitmapDown( )**
void *m_CbCtrl*.**SetBitmapDown(** LPDISPATCH *newBitmap* **)**

Delphi Usage: [*form*].*CbCtrl*. **BitmapDown**[:= *newBitmap:* Variant]

C++ Builder Usage: [*form*]->*CbCtrl*-> **BitmapDown**[= Variant *newBitmap*)]

Applicable To: **CbButton**

Description: Four bitmaps must be defined if the programmer wants to customize the appearance of the **CbButton**. The bitmaps must illustrate how the **CbButton** will look when it is in the up position without the focus, in the up position with the focus, in the down position and when it is disabled. There are two sets of four properties that can be used to customize the appearance of the **CbButton**. The first set of properties are BitmapDisabled, BitmapDown, BitmapFocus and BitmapUp, which specify the main custom bitmaps. The second set of properties are AltBitmapDisabled, AltBitmapDown, AltBitmapFocus and AltBitmapUp, which specify the alternative custom bitmaps.

The main custom bitmaps are used when the UseBitmaps property is set to true and the UseAltBitmaps property is set to false. The DefaultBitmaps property must be set to 0, so that the custom bitmaps will be used. The BitmapDown property specifies the custom bitmap for the down position of the **CbButton**.

This property is read/write at design time and runtime.

Visual Basic provides the LoadPicture function to load graphics files into a picture boxes, forms or image controls. The return value from this function can also be assigned to the **CbButton** BitmapDown property at runtime.

See Also: **BitmapDisabled, BitmapUp, BitmapFocus, UseBitmaps, UseAltBitmaps, DefaultBitmaps** properties

BitmapFocus Property

VB Usage: [*form*].*CbCtrl*.**BitmapFocus**[= LoadPicture(*bitmapName*)]

VC++ Usage: CPicture *m_CbCtrl*.**GetBitmapFocus( )**
void *m_CbCtrl*.**SetBitmapFocus(** LPDISPATCH *newBitmap* **)**

Delphi Usage: [*form*].*CbCtrl*.**BitmapFocus**[:= *newBitmap:* Variant]

C++ Builder Usage: [*form*]->*CbCtrl*->**BitmapFocus**[= Variant *newBitmap*)]

Applicable To: **CbButton**

Description: Four bitmaps must be defined if the programmer wants to customize the appearance of the **CbButton**. The bitmaps must illustrate how the **CbButton** will look when it is in the up position without the focus, in the up position with the focus, in the down position and when it is disabled. There are two sets of four properties that can be used to customize the appearance of the **CbButton**. The first set of properties are BitmapDisabled, BitmapDown, BitmapFocus and BitmapUp, which specify the main custom bitmaps. The second set of properties are AltBitmapDisabled, AltBitmapDown, AltBitmapFocus and AltBitmapUp, which specify the alternative custom bitmaps.

The main custom bitmaps are used when the UseBitmaps property is set to true and the UseAltBitmaps property is set to false. The DefaultBitmaps property must be set to 0, so that the custom bitmaps will be used. The BitmapFocus property specifies the custom bitmap for the **CbButton** in the up position with the focus.

This property is read/write at design time and runtime.

Visual Basic provides the LoadPicture function to load graphics files into a picture boxes, forms or image controls. The return value from this function can also be assigned to the **CbButton** BitmapFocus property at runtime.

See Also: **BitmapDisabled, BitmapDown, BitmapUp, UseBitmaps, UseAltBitmaps, DefaultBitmaps** properties

BitmapUp Property

VB Usage: [*form*].*CbCtrl*.**BitmapUp**[= LoadPicture(*bitmapName*)]

VC++ Usage: CPicture *m_CbCtrl*.**GetBitmapUp( )**
void *m_CbCtrl*.**SetBitmapUp(** LPDISPATCH *newBitmap* **)**

Delphi Usage: [*form*].*CbCtrl*.**BitmapUp**[:= *newBitmap:* Variant]

C++ Builder Usage: [*form*]->*CbCtrl*->**BitmapUp**[= Variant *newBitmap*)]

Applicable To: **CbButton**

Description: Four bitmaps that must be defined if the programmer wants to customize the appearance of the **CbButton**. The bitmaps must illustrate how the **CbButton** will look when it is in the up position without the focus, in the up position with the focus, in the down position and when it is disabled. There are two sets of four properties that can be used to customize the appearance of the **CbButton**. The first set of properties are BitmapDisabled, BitmapDown, BitmapFocus and BitmapUp, which specify the main custom bitmaps. The second set of properties are AltBitmapDisabled, AltBitmapDown, AltBitmapFocus and AltBitmapUp, which specify the alternative custom bitmaps.

The main custom bitmaps are used when the UseBitmaps is set to true and the UseAltBitmaps property is set to false. The DefaultBitmaps property must be set to 0, so that the custom bitmaps will be used. The BitmapUp property specifies the custom bitmap for the **CbButton** in the up position without the focus.

This property is read/write at design time and runtime.

Visual Basic provides the LoadPicture function to load graphics files into a picture boxes, forms or image controls. The return value from this function can also be assigned to the **CbButton** BitmapUp property at runtime.

See Also: **BitmapDisabled, BitmapDown, BitmapFocus, UseBitmaps, UseAltBitmaps, DefaultBitmaps** properties

BOFAction Property

VB Usage: [*form*].*CbCtrl*.**BOFAction**[= *setting%*]

VC++ Usage: short *m_CbCtrl*.**GetBOFAction( )**
void *m_CbCtrl*.**SetBOFAction(** short *setting***)**

Delphi Usage: [*form*].*CbCtrl*.**BOFAction**[:= *setting:* short]

C++ Builder Usage: [*form*]->*CbCtrl*->**BOFAction**[= short *setting*]

Applicable To: **CbMaster**

Description: Sets or returns the action that takes place when the user attempts to skip before the first record in a data file or the selected tag.

Three different actions may take place when the user attempts to skip above the top of the data file or selected tag. The first action is to ignore the attempted skip and remain on the first record and thus remain positioned on a valid record. The second option is to set a beginning of file flag to be true. In this case there is no current record until the data file is repositioned to a valid record. The last option is to cause the **CbMaster** to append a record just as though the Append method had been called or the Append button had been pressed.

This property is read/write at design time and runtime.

BOFAction can have one of these values:

Settings	Value	Description
First Record	0	When attempting to skip above the first record, the first record remains the current record. This is the default setting.
BOF	1	A beginning of file flag is set to true and the data file is not positioned on a valid record until the data file is repositioned.

Append New Record	2	This setting causes the CbMaster to append a new record. The type of record appended depends on the AppendRecordType property.

See Also: **AppendRecordType, DataTag, EOFAction** properties**, Append** method

BorderStyle Property

VB Usage: [*form*].*CbCtrl*.**BorderStyle**[= *setting%*]

VC++ Usage: short *m_CbCtrl*.**GetBorderStyle( )**
void *m_CbCtrl*.**SetBorderStyle(** short *setting***)**

Delphi Usage: [*form*].*CbCtrl*.**BorderStyle**[:= *setting:* short]

C++ Builder Usage: [*form*]->*CbCtrl*->**BorderStyle**[= short *setting*]

Applicable To: **CbMaster, CbEdit, CbSlider**

Description: This property determines whether a border is drawn around the **CbEdit**. If their BorderStyle is set to 0 then no border is drawn regardless of the Appearance property setting. If BorderStyle is set to 1, then the border is drawn according to the setting of Appearance.

The **CbMaster** and **CbSlider** controls also have a property called BorderStyle, which determines whether a single line border is drawn around the control. If BorderStyle is set to 1, a single line border is drawn around the control. If the BorderStyle is set to 0, then the single line border is not drawn. The BorderStyle does not have an effect the appearance of the **CbMaster**'s 3D border as determined by the BevelType property.

BorderStyle for the **CbEdit** is read-only at run time and read/write at design time. The BorderStyle is set to 1, by default.

The **CbMaster** and **CbSlider** BorderStyle properties are read/write at design time and runtime. The **CbMaster** BorderStyle is set to 1, by default. The **CbSlider** BorderStyle is set to 0, by default.

See Also: **Appearance, BevelType** properties

*buttonName*Enabled Properties

VB Usage: [*form*].*CbCtrl*.***buttonName*Enabled**[= {**TRUE** | **FALSE**}]

VC++ Usage: BOOL *m_CbCtrl*.**Get*buttonName*Enabled( )**
void *m_CbCtrl*.**Set*buttonName*Enabled(** BOOL *setting***)**

Delphi Usage: [*form*].*CbCtrl*.***buttonName*Enabled**[:= {**TRUE** | **FALSE**}]

C++ Builder Usage: [*form*]->*CbCtrl*->***buttonName*Enabled**[= {**TRUE** | **FALSE**}]

Applicable To: **CbMaster**

Description: The *buttonName*Enabled property determines whether the button, specified by *buttonName*, is enabled on the **CbMaster** control. The possible values for *buttonName* are: Append, Bottom, Deleted, Flush, Next, NextPage, Pack, Previous, PreviousPage, Refresh, Reindex, Search, Top and Undo. For example the Top button on the **CbMaster** has a TopEnabled property.

The *buttonName*Enabled properties are set to true, by default, so all of the buttons on the **CbMaster** are enabled. If the *buttonName*Enabled property is set to false, then the button is disabled until the property is set back to true.

Enabled buttons can receive mouse and keyboard input. Disabled buttons are greyed out and can not received any mouse and keyboard input.

These properties are read/write at design time and runtime.

See Also: **RecNoEnabled, TagListEnabled, SliderEnabled** properties

*buttonName*Visible Properties

VB Usage: [*form*].*CbCtrl*.***buttonName*Visible**[= {**TRUE | FALSE**}]

VC++ Usage: BOOL *m_CbCtrl*.**Get*buttonName*Visible( )**
void *m_CbCtrl*.**Set*buttonName*Visible(** BOOL *setting***)**

Delphi Usage: [*form*].*CbCtrl*.***buttonName*Visible**[:= {**TRUE | FALSE**}]

C++ Builder Usage: [*form*]->*CbCtrl*->***buttonName*Visible**[= {**TRUE | FALSE**}]

Applicable To: **CbMaster**

Description: The *buttonName*Visible property determines whether the button specified by *buttonName*, is displayed within the **CbMaster** control. The possible values for *buttonName* are: Append, Bottom, Deleted, Flush, Next, NextPage, Pack, Previous, PreviousPage, Refresh, Reindex, Search, Top and Undo. For example, the Top button on the **CbMaster** has a TopVisible property.

The *buttonName*Visible properties are set to true, by default, so all of the buttons on the **CbMaster** are visible. If the *buttonName*Visible property of a button is set to false, then the button is not displayed on the **CbMaster**.

These properties are read/write at design time and runtime.

See Also: **RecNoEnabled, TagListEnabled, SliderEnabled** properties

Caption Property

VB Usage: [*form*].*CbCtrl*.**Caption**[= *newCaption$*]

VC++ Usage: CString *m_CbCtrl*.**GetCaption( )**
void *m_CbCtrl*.**SetCaption(** LPCTSTR *newCaption* **)**

Delphi Usage: [*form*].*CbCtrl*.**Caption**[:= *newCaption*: string]

C++ Builder Usage: [*form*]->*CbCtrl*->**Caption**[= AnsiString *newCaption*]

Applicable To: **CbMaster, CbButton**

Description: This property determines what text is displayed by the control. The text is only displayed when the CaptionVisible property is set to true.

Visual Basic, Delphi and C++ Builder provide a unique name for each instance of a control, which can be accessed through the Name property. The Caption property is initialized with this unique name, by default.

Visual C++ does not provide a unique name for each instance of a control. Each instance of the **CbMaster** creates its own unique name, which is used to initialized the Caption property. The **CbMaster's** unique name may be accessed through its NameUnique property. The **CbButton** Caption property is set to "CbButton" under Visual C++.

This property is read/write at design time and runtime.

See Also: **NameUnique, CaptionVisible, Caption3DText** properties

Caption3DText Property

VB Usage: [*form*].*CbCtrl*.**Caption3DText**[= {**TRUE** | **FALSE**}]

VC++ Usage: BOOL *m_CbCtrl*.**GetCaption3DText( )**
void *m_CbCtrl*.**SetCaption3DText(**BOOL *setting* **)**

Delphi Usage: [*form*].*CbCtrl*.**Caption3DText**[:= {**TRUE** | **FALSE**}]

C++ Builder Usage: [*form*]->*CbCtrl*->**Caption3DText**[= {**TRUE** | **FALSE**}]

Applicable To: **CbButton**

Description: This property determines whether the caption has a three dimensional appearance. When Caption3DText is set to true, the caption has three dimensional appearance. If this property is set to false, the caption is drawn as normal flat text.

The default value is set to true. This property only has an effect when the CaptionVisible property is set to true.

This property is read/write at design time and runtime.

See Also: **Caption, CaptionVisible** property

CaptionVisible Property

VB Usage: [*form*].*CbCtrl*.**CaptionVisible**[= {**TRUE** | **FALSE**}]

VC++ Usage: BOOL *m_CbCtrl*.**GetCaptionVisible( )**
void *m_CbCtrl*.**SetCaptionVisible(** BOOL *setting***)**

Delphi Usage: [*form*].*CbCtrl*.**CaptionVisible**[:= {**TRUE** | **FALSE**}]

C++ Builder Usage: [*form*]->*CbCtrl*->**CaptionVisible**[= {**TRUE** | **FALSE**}]

Applicable To: **CbMaster, CbButton**

Description: This property determines whether the caption is visible. By default, CaptionVisible is true, which means that the caption is displayed. If CaptionVisible is set to false, the caption is not displayed.

The CaptionVisible property for the **CbButton** should be set to false when the UseBitmaps property is set to true.

This property is read/write at design time and runtime.

See Also: **Caption, Caption3Dtext, UseBitmaps** property

Case Property

VB Usage: [*form*].*CbCtrl*.**Case**[= *setting%*]

VC++ Usage: short *m_CbCtrl*.**GetCase( )**
void *m_CbCtrl*.**SetCase(**short *setting***)**

Delphi Usage: [*form*].*CbCtrl*.**Case**[:= *setting:* short]

C++ Builder Usage: [*form*]->*CbCtrl*->**Case**[= short *setting*]

Applicable To: **CbEdit, CbList, CbCombo**

Description: This property is used when entering formatted data into a **CbEdit** or the edit portion of a **CbCombo**. The Case property is also used when displaying formatted data in the **CbList**. This property only has an effect when the FormatType is set to Date (1) or Masked (2).

The following table lists the Case property settings:

Settings	Value	Description
Both Cases	0	This setting allows the user to enter lower and upper case letters. No change is made to the case of the letters as they are entered from the keyboard or read from disk. This is the default value. Note that for the Date format, any month strings are automatically capitalized and the rest of the letters are converted lower case regardless of how the letters were entered. (ie. FEBRUARY appears as February)
Upper Case	1	This setting changes any lower case letters, entered from the keyboard or read from disk, to upper case letters.
Lower Case	2	This setting changes any upper case letters, entered or read from disk, to lower case. This setting does not effect the mask character "!". If "!" is used in a mask string, the character is automatically converted to upper case, regardless of the Case property.

NOTE

The Case property is read-only at runtime for the CbList and CbCombo when they are NOT linked to a data file. The reasons are explained in the "CodeControls Concepts" chapter. Otherwise, Case is a read/write property at design time and runtime.

See Also: **FormatType, Mask** property

Columns Property

VB Usage: [*form*].*CbCtrl*.**Columns**[= *setting%*]

VC++ Usage: short *m_CbCtrl*.**GetColumns( )**
void *m_CbCtrl*.**SetColumns(** short *setting***)**

Delphi Usage: [*form*].*CbCtrl*.**Columns**[:= *setting:* short]

C++ Builder Usage: [*form*]->*CbCtrl*->**Columns**[= short *setting*]

Applicable To: **CbList**

Description: This property determines how many columns should be displayed by a **CbList**. When the Columns property is set to 1 or more, a horizontal scroll bar is displayed instead of a vertical scroll bar. When the user clicks the horizontal scroll bar, the **CbList** displays the next set of columns.

This property only has an effect when the **CbList** is not bound to a **CbMaster** control. By default, Columns is set to 0, so the **CbList** does not use a horizontal scroll bar and only displays one column.

Columns is read-only at run time and read/write at design time.

ContainerColor Property

VB Usage: [*form*].*CbCtrl*.**ContainerColor**[= *color&*]

VC++ Usage: OLE_COLOR *m_CbCtrl*.**GetContainerColor( )**
void *m_CbCtrl*.**SetContainerColor(** OLE_COLOR *color***)**

Delphi Usage: [*form*].*CbCtrl*.**ContainerColor**[:= *setting:* TColor]

C++ Builder Usage: [*form*]->*CbCtrl*->**ContainerColor**[= TColor *setting*]

Applicable To: **CbCombo**

Description: Currently, there is a **CbCombo** painting problem when the Style property is set to Simple (1). The **CbCombo** has the integrated height style, which means that Windows automatically draws the **CbCombo** such that partial items are not displayed. Unfortunately, Windows leaves an unpainted gap at the bottom of the **CbCombo**.

The workaround involves painting the gap with the background color of the container. Unfortunately, Delphi and C++ Builder containers do not support the ambient back color property, which would allow the **CbCombo** to access the background color of its container. As a result, the **CbCombo** needs to use the ContainerColor property to supply the background color of the container. By default, the ContainerColor is set to the Windows system button face color, which is also default color for the container. Therefore, the programmer need only change the ContainerColor property when the container background color is changed from the default.

The ContainerColor property is automatically updated with the background color of the container under Visual Basic and Visual C++, since these containers support the ambient back color property. The programmer need not worry about the ContainerColor property under these environments.

Any valid RGB value may be used for this property. The actual number of colors available is limited by the monitor's display adapter and display driver.

ContainerColor is read/write at design time and runtime.

Data4 Property

VB Usage: [*form*].*CbCtrl*.**Data4**

VC++ Usage: long *m_CbCtrl*.**GetData4( )**

Delphi Usage: [*form*].*CbCtrl*.**Data4**

C++ Builder Usage: [*form*]->*CbCtrl*->**Data4**

Applicable To: **CbMaster**

Description: This property returns the **DATA4** pointer associated with the data file linked to the **CbMaster** control. CodeBase functions can use this pointer to manipulate the **CbMaster's** data file directly.

For example, the value of this property could be passed to CodeBase's **d4check** function to check to see whether the index files are valid.

The controls bound to the **CbMaster** will not be automatically updated if you use the CodeBase functions to reposition or modified the data file. Use a **CbMaster** method such as Go or FlushRecord to update the bound controls.

C/C++ compiler users must explicitly cast this pointer to a **DATA4** pointer before it may be used. CodeBase for C++ users should construct a **Data4** object using the constructor that uses a **DATA4** pointer.

This property is read only at run time.

See Also: **DATA4** structure / **Data4** object in the *CodeBase User's Guide*

DataAction Property

VB Usage: [*form*].*CbCtrl*.**DataAction**[= *setting%*]

VC++ Usage: short *m_CbCtrl*.**GetDataAction( )**
void *m_CbCtrl*.**SetDataAction(** short *setting* **)**

Delphi Usage: [*form*].*CbCtrl*.**DataAction**[:= *setting:* short]

C++ Builder Usage: [*form*]->*CbCtrl*->**DataAction**[= short *setting*]

Applicable To: **CbButton, CbList, CbSlider**

Description: **CbButton**

The DataAction property for the **CbButton** control determines what kind of action is executed when the button is pressed. This property only applies when the **CbButton** is linked to a data file. When the **CbButton** is bound to a **CbMaster**, the actions are executed on the data file linked to the **CbMaster**.

Possible values for DataAction:

Settings	Value	Description
Custom	0	When the DataAction is set to this value, no action is executed by the CbButton. This value allows the programmer to customize the action, by intercepting button events and writing the appropriate code to be executed. This is the default setting.
Append	1	When the CbButton is pressed it signals the CbMaster to append a record.
Bottom	2	When the CbButton is pressed it signals the CbMaster to go to the bottom of the data file according to the master's selected tag.
Flush Changes	3	When the CbButton is pressed it signals the CbMaster to flush the current record.
Next Page	4	When the CbButton is pressed it signals the CbMaster to skip one page forward according to the master's selected tag.
Next Record	5	When the CbButton is pressed it signals the CbMaster to skip to the next record according to the master's selected tag..

Pack	6	When the CbButton is pressed it signals the CbMaster to pack the data file.
Previous Page	7	When the CbButton is pressed it signals the CbMaster to skip one page backward according to the master's selected tag.
Previous Record	8	When the CbButton is pressed it signals the CbMaster to skip one record backward according to the master's selected tag.
Refresh Record	9	When the CbButton is pressed it signals the CbMaster to reload the current record from disk.
Reindex	10	When the CbButton is pressed it signals the CbMaster to reindex the data file.
Search	11	When the CbButton is pressed it signals the CbMaster to open the search dialog.
Toggle Deletion Status	12	When the CbButton is pressed it signals the CbMaster to toggle the deletion status of the current record.
Top	13	When the CbButton is pressed it signals the CbMaster to go to the top of the data file according to the master's selected tag.
Undo	14	When the CbButton is pressed it signals the CbMaster to undo the changes made to the current record.

CbList and **CbSlider**

The DataAction properties for the **CbList** and **CbSlider** controls are different from the **CbButton** DataAction property. The DataAction property only applies when the **CbList** and **CbSlider** controls are linked to a data file. If no action is desired, set DataAction to None (0). The default value is set to None (0) for both the **CbList** and **CbSlider**.

This property is read/write at design time and runtime for both the **CbList** and **CbSlider**.

CbList When the DataAction property is set to Record Position (1), then the **CbList** is used to reposition the data file and indicate the current position of the data file. Each entry in the **CbList** corresponds to a record in the selected tag, which is specified by the DataTag property. If there is no selected tag then the **CbList** has an entry for every record in the data file and the records are displayed in natural order. The expression specified by the DataExpr property is evaluated for each record in the selected tag and the results are displayed in the **CbList**. When the user selects an item from the **CbList**, the data file is automatically repositioned to that record and the other bound controls are updated. If the current record does not have an entry in the **CbList's** selected tag, the selection is removed from the **CbList** and it does not indicate the current record position. If the user selects a record from the **CbList** and that record does not have an entry in the **CbMaster's** selected tag, the **CbMaster** is not repositioned and the other bound controls are not updated.

If the DataAction is set to Field Value (2), then the **CbList** can be used to assign data to the bound field of the current record. In this case, the programmer must explicitly fill the **CbList** with valid field values by using the ListItems property at design time or the AddItem method at runtime. When the user selects one of the strings from the **CbList**, that value is entered into the field of the current record and the other bound controls are updated. When the DataAction is set to Field Value (2), the DataExpr must be set to the name of a field, not a complicated dBASE expression. The DataTag property has no effect on this particular **CbList** behavior.

CbSlider

When the DataAction property is set to Record Position (1), then the **CbSlider** is used to reposition the data file and indicate the current position of the data file. The thumb on the **CbSlider** represents the relative position of the current record in the data file. If the position of the thumb is changed, the data file is repositioned and the other bound controls are updated. In this case, the DataField property has no effect on the **CbSlider's** behavior.

If the DataAction is set to Field Value (2), then the **CbSlider** can be used to assign data to the bound field of the current record. The DataField property must specify a field name for this behavior. The positions along the slider correspond to data values for the field. For example, the range for the slider could be set from 1 to 10. If the thumb is dragged to the middle of the slider, a value of 5 will be entered into the field of the current record and the other bound controls will be updated automatically.

See the "**CbSlider**" and "**CbList**" sections under "Data Aware Controls" in the "CodeControls Concepts" chapter for illustrations. Also refer to the "Programming CodeControls" chapter for information on how to bind controls to data files.

See Also: **DataSource, DataField, DataExpr, DataTag** properties

DatabaseName Property

VB Usage: [*form*].*CbCtrl*.**DatabaseName**[= *fileName$*]

VC++ Usage: CString *m_CbCtrl*.**GetDatabaseName( )**
void *m_CbCtrl*.**SetDatabaseName(** LPCTSTR *fileName***)**

Delphi Usage: [*form*].*CbCtrl*.**DatabaseName**[:= *fileName:* string]

C++ Builder Usage: [*form*]->*CbCtrl*->**DatabaseName**[= AnsiString *fileName*]

Applicable To: **CbMaster**

Description: This property specifies the name of the data file associated with the **CbMaster** control. Any of the fields in the specified data file may be bound to other CodeControls.

When the DatabaseName property is changed to a new value, the IndexFiles and DataTag properties are automatically set to an empty string (""). Therefore, set the IndexFiles and DataTag properties after the DatabaseName has been set. Call RefreshFiles after setting the DatabaseName, IndexFiles and DataTag properties at runtime. RefreshFiles closes the old files and opens the new data file and its index files.

Setting the DatabaseName property at runtime to an invalid file name triggers an Error event, after the RefreshFiles method is used to make the change take effect.

NOTE

If you change the DatabaseName property at runtime, you may also have to change the DataField properties of any bound controls whose field names do not match those of the newly specified data file. Failure to do so will result in an error when the RefreshFiles method is used to make the change take effect.

Specifying the data file can be done automatically by using one of the **CbMaster** property pages at design time. Open the property pages for the **CbMaster** and choose the Files tab. Press the Browse button and an Open dialog box appears. Select the appropriate data file and click Open. Now the name of the new data file, complete with its path, is added to the DatabaseName property.

NOTE

When you are using the client/server configuration of CodeBase with CodeControls, the Browse button is not enabled on the Files property page. Under client/server, the data file name must be typed in manually on the property page. See the "Client/Server" section of the "Programming CodeControls" chapter for more information.

This property is set to an empty string (""), by default.

This property is read/write at design and runtime.

See Also: **IndexFiles, DataTag** properties**, Error** Event

DataExpr Property

VB Usage: [*form*].*CbCtrl*.**DataExpr**[= *expr$*]

VC++ Usage: CString *m_CbCtrl*.**GetDataExpr( )**
void *m_CbCtrl*.**SetDataExpr(**LPCTSTR *expr***)**

Delphi Usage: [*form*].*CbCtrl*.**DataExpr**[:= *expr:* string]

C++ Builder Usage: [*form*]->*CbCtrl*->**DataExpr**[= AnsiString *expr*]

Applicable To: **CbList**

Description: Sets or returns the dBASE expression associated with a **CbList** control. This expression is evaluated for each record in the **CbMaster** control's data file to determine the contents of the list. This property is used in conjunction with the DataSource and DataTag properties.

Although the dBASE expression can be as simple as a single field name, it can also include dBASE operators and functions. For example, in a data file that includes the fields L_NAME and F_NAME, the dBASE expression "UPPER(L_NAME + F_NAME)" could be used to display an upper case combination of the two fields for each entry in the list. Refer to the "Appendix C: dBASE Expressions" chapter in the *CodeBase Reference Guide* for more information about dBASE expressions.

When you change the DataExpr property at run time, the **CbList** attempts to evaluate the new dBASE expression, and if successful, repopulates the control's list based on the new expression.

If an error occurs however, an Error event is triggered and the list is left empty and the control is disabled. The control can be subsequently enabled by specifying a correct field expression and setting the Enabled property to true.

The values of the parameters sent by the Error event indicate the type of error that occurred.

The DataExpr property is automatically set to an empty string ("") when the DataSource property is changed to a new value. Therefore, set the DataExpr property after the DataSource has been set.

DataExpr is set to "<None>", by default.

This property is read/write at design time and runtime.

See Also: **DataSource, DataTag, Enabled** properties, **Error** event.

DataField Property

VB Usage: [*form*].*CbCtrl*.**DataField**[= *fieldName$*]

VC++ Usage: CString *m_CbCtrl*.**GetDataField( )**
void *m_CbCtrl*.**SetDataField(** LPCTSTR *fieldName***)**

Delphi Usage: [*form*].*CbCtrl*.**DataField**[:= *fieldName:* string]

C++ Builder Usage: [*form*]->*CbCtrl*->**DataField**[= AnsiString *fieldName*]

Applicable To: **CbEdit, CbSlider**

Description: This property sets or returns the name of the database field bound to the **CbEdit** or **CbSlider** control. This property is used in conjunction with DataSource property to specify the link between the control and the **CbMaster**.

Setting this property to an invalid field name, causes an Error event to be fired and the control is disabled. The control can be subsequently enabled by specifying a correct field name and setting the Enabled property to true.

The DataField property is automatically set to an empty string ("") when the DataSource property is changed to a new value. Therefore, set the DataField property after the DataSource has been set.

DataField is set to "<None>", by default.

This property is read/write at design time and runtime.

See Also: **DataSource**, **Enabled** Property, **Error** Event

DataFieldEdit Property

VB Usage: [*form*].*CbCtrl*.**DataFieldEdit**[= *fieldName$*]

VC++ Usage: CString *m_CbCtrl*.**GetDataFieldEdit( )**
void *m_CbCtrl*.**SetDataFieldEdit(** LPCTSTR *fieldName***)**

Delphi Usage: [*form*].*CbCtrl*.**DataFieldEdit**[:= *fieldName:* string]

C++ Builder Usage: [*form*]->*CbCtrl*->**DataFieldEdit**[= AnsiString *fieldName*]

Applicable To: **CbCombo**

Description: This property sets or returns the name of the database field bound to the edit portion of the **CbCombo** control. This property is used in conjunction with DataSourceEdit property to specify the link between the edit portion of the **CbCombo** control and the **CbMaster**.

Setting this property to an invalid field name, causes an Error event to be fired and the **CbCombo** becomes disabled. The control can be subsequently enabled by specifying a correct field name and setting the Enabled property to true.

The DataFieldEdit property is automatically set to an empty string ("") when the DataSourceEdit property is changed to a new value. Therefore, set the DataFieldEdit property after the DataSourceEdit has been set.

DataFieldEdit is set to "<None>", by default.

This property is read/write at design time and runtime.

See Also: **DataSourceEdit, Enabled** Property, **Error** Event

DataExprList Property

VB Usage: [*form*].*CbCtrl*.**DataExprList**[= *expr$*]

VC++ Usage: CString *m_CbCtrl*.**GetDataExprList( )**
void *m_CbCtrl*.**SetDataExprList(**LPCTSTR *expr***)**

Delphi Usage: [*form*].*CbCtrl*.**DataExprList**[:= *expr:* string]

C++ Builder Usage: [*form*]->*CbCtrl*->**DataExprList**[= AnsiString *expr*]

Applicable To: **CbCombo**

Description: This property sets or returns the dBASE expression associated with the list portion of the **CbCombo** control. This expression is evaluated for each record in the **CbMaster**'s data file to determine the contents of the list. This property is used in conjunction with the DataSourceList and DataTagList properties.

Although the dBASE expression can be as simple as a single field name, it can also include dBASE operators and functions. For example, in a data file that includes the fields L_NAME and F_NAME, the dBASE expression "UPPER(L_NAME + F_NAME)" could be used to display an upper case combination of the two fields for each entry in the list. Refer to the "Appendix C: dBASE Expressions" chapter in the *CodeBase Reference Guide* for more information on dBASE expressions.

When you change the DataExprList property at run time, CodeControls attempts to evaluate the new dBASE expression, and if successful, repopulates the control's list based on the new expression.

Setting this property to an invalid field name, causes an Error event to be fired and the list is left empty and the **CbCombo** is disabled. The control can be subsequently enabled by specifying a correct field expression and setting the Enabled property to true.

The values of the parameters sent by the Error event indicate the type of error that occurred.

The DataExprList property is automatically set to an empty string ("") when the DataSourceList property is changed to a new value. Therefore, set the DataExprList property after the DataSourceList has been set.

DataExprList is set to "<None>", by default.

This property is read/write at design time and runtime.

See Also: **DataSourceList**, **DataTagList, Enabled** properties, **LookUp** and **Error** events

DataSource Property

VB Usage: [*form*].*CbCtrl*.**DataSource[=** *sourceName$*]

VC++ Usage: CString *m_CbCtrl*.**GetDataSource( )**
void *m_CbCtrl*.**SetDataSource(** LPCTSTR *sourceName***)**

Delphi Usage: [*form*].*CbCtrl*.**DataSource**[:= *sourceName:* string]

C++ Builder Usage: [*form*]->*CbCtrl*->**DataSource**[= AnsiString *sourceName*]

Applicable To: **CbButton**, **CbEdit, CbList, CbSlider**

Description: This property sets or returns the name of the **CbMaster** control to which the **CbButton**, **CbEdit, CbList, CbSlider** controls are linked.

If you change the DataSource property at run time, the control is unlinked from its original **CbMaster** and then linked to the new **CbMaster**. The data in the control is updated according to the current position of data file linked to the new **CbMaster**.

When the DataSource property is changed for the **CbEdit** and **CbSlider**, the DataField property is automatically set to an empty string (""). Similarly, the DataExpr and the DataTag properties of the **CbList** are automatically blanked out when the DataSource property is changed. Therefore, you must set the DataField, DataExpr and DataTag properties after the DataSource property has been set.

Setting this property to an invalid **CbMaster** name, causes an Error event to be fired and the control is disabled. The control can be subsequently enabled by specifying a correct data source and setting the Enabled property to true. Setting *sourceName* to an empty string ("") unlinks the control from the **CbMaster**, but does not generate an Error event.

DataSource is set to "<None>", by default.

This property is read/write at design time and runtime.

See Also: **DataField, DataExpr, DataAction, DataTag** properties**, Error** Event

DataSourceEdit Property

VB Usage: [*form*].*CbCtrl*.**DataSourceEdit[=** *sourceName$*]

VC++ Usage: CString *m_CbCtrl*.**GetDataSourceEdit( )**
void *m_CbCtrl*.**SetDataSourceEdit(** LPCTSTR *sourceName***)**

Delphi Usage: [*form*].*CbCtrl*.**DataSourceEdit**[:= *sourceName:* string]

C++ Builder Usage: [*form*]->*CbCtrl*->**DataSourceEdit**[= AnsiString *sourceName*]

Applicable To: **CbCombo**

Description: This property sets or returns the name of the **CbMaster** control linked to edit portion of the **CbCombo**.

If you change the DataSourceEdit property at run time, the edit portion of the **CbCombo** is unlinked from its original **CbMaster** and then the edit portion is linked to the new **CbMaster**. The **CbCombo** is updated according to the current position of data file linked to the new **CbMaster**.

Setting this property to an invalid **CbMaster** name, causes an Error event to be fired and the control is disabled. The control can be subsequently enabled by specifying a correct data source and setting the Enabled property to true. Setting *sourceName* to an empty string ("") unlinks the control from the **CbMaster**, but does not generate an Error event.

When the DataSourceEdit property is changed, the DataFieldEdit property is automatically set to an empty string (""). Therefore, you must set the DataFieldEdit property after the DataSourceEdit property has been set.

DataSourceEdit is set to "<None>", by default.

This property is read/write at design time and runtime.

See Also: **DataFieldEdit, Enabled** properties**, Error** Event

DataSourceList Property

VB Usage: [*form*].*CbCtrl*.**DataSourceList[** = *sourceName$*]

VC++ Usage: CString *m_CbCtrl*.**GetDataSourceList( )**
void *m_CbCtrl*.**SetDataSourceList(** LPCTSTR *sourceName***)**

Delphi Usage: [*form*].*CbCtrl*.**DataSourceList**[:= *sourceName:* string]

C++ Builder Usage: [*form*]->*CbCtrl*->**DataSourceList**[= AnsiString *sourceName*]

Applicable To: **CbCombo**

Description: This property sets or returns the name of the **CbMaster** control linked to the list portion of the **CbCombo**.

If you change the DataSourceList property at run time, the list portion of the **CbCombo** is unlinked from its original **CbMaster** and then the list portion is linked to the new **CbMaster**. The **CbCombo** is updated according to the current position of data file linked to the new **CbMaster**.

Setting this property to an invalid **CbMaster** name, causes an Error event to be fired and the control is disabled. The control can be subsequently enabled by specifying a correct data source and setting the Enabled property to true. Setting *sourceName* to an empty string ("") unlinks the control from the **CbMaster**, but does not generate an Error event.

When the DataSourceList property is changed, the DataExprList and DataTagList properties are automatically set to empty strings (""). Therefore, you must set the DataExprList and DataTagList properties after the DataSourceList property has been set.

DataSourceEdit is set to "<None>", by default.

This property is read/write at design time and runtime.

See Also: **DataExprList, DataTagList, Enabled** properties**, Error** Event

DataTag Property

VB Usage: [*form*].*CbCtrl*.**DataTag**[= *tagName$*]

VC++ Usage: CString *m_CbCtrl*.**GetDataTag( )**
void *m_CbCtrl*.**SetDataTag(** LPCTSTR *tagName***)**

Delphi Usage: [*form*].*CbCtrl*.**DataTag**[:= *tagName:* string]

C++ Builder Usage: [*form*]->*CbCtrl*->**DataTag**[= AnsiString *tagName*]

Applicable To: **CbMaster, CbList**

Description: This property sets or returns the name of the index tag to be used for ordering of records/entries in the **CbMaster** and **CbList** controls.

You can specify the name of any tag from any index file associated with the **CbMaster** control's data file. The **CbMaster** automatically opens all of the index files specified in the IndexFiles property, including non-production indexes.

If this property is not set at design time or is set to a null string at run time, the records/entries are accessed and displayed in record number (natural) order.

When you change this property, the Tag4 property of the **CbMaster** is automatically updated to reflect the **TAG4** pointer of the newly selected tag.

Changing DataTag to an illegal tag name causes an Error event for the control, and causes the current tag to revert to its previous setting. The control remains enabled.

The DataTag property of the **CbMaster** is automatically set to an empty string ("") when the DatabaseName property is changed to a new value. Similarly, the DataTag property of the **CbList** is automatically set to an empty string ("") when the DataSource property is changed. Therefore, set the DataTag property after the DatabaseName or DataSource has been set.

DataTag is set to "<Natural Order>", by default.

This property is read/write at design time and runtime.

For more information on production index files and CGP files, refer to the *CodeBase User's Guide*.

See Also: **DataSource**, **DataExpr**, **Tag4** properties

DataTagList Property

VB Usage: [*form*].*CbCtrl*.**DataTagList**[= *tagName$*]

VC++ Usage: CString *m_CbCtrl*.**GetDataTagList( )**
void *m_CbCtrl*.**SetDataTagList(** LPCTSTR *tagName***)**

Delphi Usage: [*form*].*CbCtrl*.**DataTagList**[:= *tagName:* string]

C++ Builder Usage: [*form*]->*CbCtrl*->**DataTagList**[= AnsiString *tagName*]

Applicable To: **CbCombo**

Description: This property sets or returns the name of the index tag to be use for ordering of records/entries in the list portion of the **CbCombo** control.

You can specify the name of any tag from any index file associated with the **CbMaster** control's data file.

If this property is not set at design time, or is set to a null string at run time, the records/entries are accessed and displayed in record number (natural) ordering.

Changing DataTagList property to an illegal tag name causes an Error event for the control, and causes the current tag to revert to its previous setting. The control remains enabled.

The DataTagList property is automatically set to an empty string ("") when the DataSourceList property is changed to a new value. Therefore, set the DataTagList property after the DataSourceList property has been set.

DataTagList is set to "<Natural Order>", by default.

This property is read/write at design time and runtime.

For more information on production index files and CGP files, refer to the *CodeBase User's Guide*.

See Also: **DataSourceList**, **DataExprList** properties

DecimalChar Property

VB Usage: [*form*].*CbCtrl*.**DecimalChar**[= *newChar$*]

VC++ Usage: CString *m_CbCtrl*.**GetDecimalChar( )**
void *m_CbCtrl*.**SetDecimalChar(** LPCTSTR *newChar***)**

Delphi Usage: [*form*].*CbCtrl*.**DecimalChar**[:= *newChar:* string]

C++ Builder Usage: [*form*]->*CbCtrl*->**DecimalChar**[= AnsiString *newChar*]

Applicable To: **CbEdit, CbList, CbCombo**

Description: Sets or returns the character to be used as the separator between the whole and fractional portion of a number. The default character is a period (".").

This property only applies when the FormatType property for the control is set to Numeric (3).

NOTE

The DecimalChar property is read-only at runtime for the CbList and CbCombo when they are NOT linked to a data file. The reasons are explained in the "CodeControls Concepts" chapter. Otherwise, DecimalChar is a read/write property at design time and runtime.

See Also: **FormatType** property

Decimals Property

VB Usage: [*form*].*CbCtrl*.**Decimals**[= *setting%*]

VC++ Usage: short *m_CbCtrl*.**GetDecimals( )**
void *m_CbCtrl***SetDecimals(** short *setting* **)**

Delphi Usage: [*form*].*CbCtrl*.**Decimals**[:= *setting:* short]

C++ Builder Usage: [*form*]->*CbCtrl*->**Decimals**[= short *setting*]

Applicable To: **CbEdit, CbList, CbCombo**

Description: Sets or returns the number of decimal characters for a numeric entry. The Decimals property only applies when the FormatType property for the control is set to Numeric (3).

The default value for this property is 2 and the maximum value is 10.

The value of Decimals must be less than or equal to the MaxLength property minus 2. The MaxLength must be long enough to accommodate a negative sign, the decimal point and the number of decimal places specified by Decimals. If the Decimals property is set to value that is greater than MaxLength minus 2, an exception is thrown and the previous value of Decimals is used.

If the Decimals is set with a negative value, 0 is automatically used, by default. If a number larger than 10 is specified, then 10 is used automatically instead.

If the control is linked to a data file the number of decimals is set to the number of decimals in the field.

NOTE

The Decimals property is read-only at runtime for the CbList when it is NOT linked to a data file. The reasons are explained in the "CodeControls Concepts" chapter. Otherwise, Decimals is a read/write property at design time and runtime.

See Also: **FormatType, MaxLength** properties

DefaultBitmaps Property

VB Usage: [*form*].*CbCtrl*.**DefaultBitmaps**[= *setting%*]

VC++ Usage: short *m_CbCtrl*.**GetDefaultBitmaps( )**
void *m_CbCtrl*.**SetDefaultBitmaps(** short *setting* **)**

Delphi Usage: [*form*].*CbCtrl*.**DefaultBitmaps**[:= *setting:* short]

C++ Builder Usage: [*form*]->*CbCtrl*->**DefaultBitmaps**[= short *setting*]

Applicable To: **CbButton**

Description: This property can be used to specify which bitmaps are displayed by the **CbButton** control. This property is used in conjunction with the UseBitmaps and UseAltBitmaps property. Make sure that the UseBitmaps property is set to true if you want the **CbButton** to display custom or default bitmaps.

CodeControls supplies two sets of default bitmaps to match the actions of the DataAction property. The bitmaps specified by DefaultBitmaps will only be used if the UseBitmaps property is set to true. CodeControls also supplies a alternative set of bitmaps to match the DataAction property. The alternative set of bitmaps is used when both the UseAltBitmaps and the UseBitmaps properties are set to true.

Set the DefaultBitmaps to Custom (0) and UseBitmaps to true, if you want to use your own bitmaps to decorate the **CbButton**. The custom bitmaps are specified by setting the BitmapDisabled, BitmapDown, BitmapFocus and BitmapUp properties. If the UseAltBitmaps is set to true then the AltBitmapDisabled, AltBitmapDown, AltBitmapFocus and AltBitmapUp properties are used to specify the custom bitmaps.

This property is read/write at design time and runtime.

Possible values for DefaultBitmaps:

Settings	Value	Description
Custom	0	Use the custom bitmaps when the UseBitmaps property is true. This is the default value.
Append	1	Use the default append bitmaps.
Bottom	2	Use the default bottom record bitmaps.
Flush Changes	3	Use the default flush bitmaps.
Next Page	4	Use the default next page bitmaps.
Next Record	5	Use the default next record bitmaps.
Pack	6	Use the default pack bitmaps.
Previous Page	7	Use the default previous page bitmaps.
Previous Record	8	Use the default previous record bitmaps.
Refresh Record	9	Use the default refresh record bitmaps.
Reindex	10	Use the default reindex bitmaps.
Search	11	Use the default search dialog bitmaps.
Toggle Deletion Status	12	Use the default deleted record bitmaps.
Top	13	Use the default top record bitmaps.
Undo	14	Use the default undo changes bitmaps.

See Also: **UseBitmaps, UseAltBitmaps, AltBitmapDisabled, AltBitmapDown, AltBitmapFocus, AltBitmapUp, BitmapDisabled, BitmapDown, BitmapFocus, BitmapUp** properties

DefaultDate Property

VB Usage: [*form*].*CbCtrl*.**DefaultDate**[= *setting As Date*]

VC++ Usage: DATE *m_CbCtrl*.**GetDefaultDate( )**
void *m_CbCtrl*.**SetDefaultDate(** DATE *setting***)**

Delphi Usage: [*form*].*CbCtrl*.**DefaultDate**[:= *setting:* TOleDate]

C++ Builder Usage: [*form*]->*CbCtrl*->**DefaultDate**[= double *setting*]

Applicable To: **CbEdit, CbCombo, CbList**

Description: Sets or returns the default date to be used with Date formatting. This property only has an effect when Date formatting is specified by setting the FormatType property to Date (1).

The DefaultDate property is an OLE DATE data type. The DATE type is a floating point value, measuring the days from midnight, December 30, 1899, which is represented by 0.0. For example, Midnight, December 31, 1899 is represented by the value 1.0 and Midnight, May 13, 1997 is represented by 35563.0.

Sometimes it is desirable to have a date field automatically filled with a default date or to fill in parts of the date automatically. The DefaultDate property allows you to specify the default date. If blank dates or partial dates are allowed in the data file then set the DefaultDate property to 0.0. In this case, the date field is not automatically filled with a default date. If you want the current system date to be automatically assigned to a blank date field, then set DefaultDate to 1.0. If you want a particular date to always be used to fill in blank or partial dates, then set DefaultDate to any other valid date.

If the DefaultDate is set to an invalid date, an exception is thrown.

The DefaultDate is set to 0.0 by default.

This property is read/write at design time and runtime.

See Also: **FormatType** property

Deleted Property

VB Usage: [*form*].*CbCtrl*.**Deleted**[= {**TRUE | FALSE**}]

VC++ Usage: BOOL *m_CbCtrl*.**GetDeleted( )**
void *m_CbCtrl*.**SetDeleted(** BOOL *setting***)**

Delphi Usage: [*form*].*CbCtrl*.**Deleted**[:= {**TRUE | FALSE**}]

C++ Builder Usage: [*form*]->*CbCtrl*->**Deleted**[= {**TRUE | FALSE**}]

Applicable To: **CbMaster**

Description: This property sets and returns the status of the deletion flag for the current record. When *setting* is false, the deletion mark is removed from the current record. When *setting* is true, then the current record is marked for deletion.

A record that is marked for deletion is not physically removed from the data file at this time. The records that are marked for deletion are physically removed when the data file is packed. The data file can be packed by calling the Pack method or by pressing the Pack button on the **CbMaster**.

Note that the deletion flag for the current record may also be altered by using the **CbMaster** Delete method.

This property is read/write at runtime.

See Also: **Delete**, **Pack** methods

Enabled Property

VB Usage: [*form*].*CbCtrl*.**Enabled**[= {**TRUE** | **FALSE**}]

VC++ Usage: BOOL *m_CbCtrl*.**GetEnabled( )**
void *m_CbCtrl*.**SetEnabled(** BOOL *setting***)**

Delphi Usage: [*form*].*CbCtrl*.**Enabled**[:= {**TRUE** | **FALSE**}]

C++ Builder Usage: [*form*]->*CbCtrl*->**Enabled**[= {**TRUE** | **FALSE**}]

Applicable To: **CbMaster, CbButton, CbCombo, CbEdit, CbList, CbSlider**

Description: This property determines whether the control is enabled. Enabled controls can receive mouse and keyboard input. Disabled controls are greyed out and can not received any mouse and keyboard input.

By default, all of the controls are enabled and the Enabled properties are set to true. If the Enabled property is set to false, then the control is disabled until the property is set back to true.

If an Error event is fired and the control does not recover from the error, the control is automatically disabled. The control can not be enabled until the error has been fixed. Once the control has recovered, the programmer must explicitly enable the control by setting Enabled to true.

This property are read/write at design time and runtime.

See the "Enabled Property" section of the "Programming CodeControls" chapter for more information.

See Also: ***buttonName*Enabled** properties

EOFAction Property

VB Usage: [*form*].*CbCtrl*.**EOFAction**[= *setting%*]

VC++ Usage: short *m_CbCtrl*.**GetEOFAction( )**
void *m_CbCtrl*.**SetEOFAction(** short *setting***)**

Delphi Usage: [*form*].*CbCtrl*.**EOFAction**[:= *setting:* short]

C++ Builder Usage: [*form*]->*CbCtrl*->**EOFAction**[= short *setting*]

Applicable To: **CbMaster**

Description: Sets or returns the action that takes place when the user attempts to skip past the last record in a data file or the selected tag.

Three different actions may take place when the user attempts to skip past the end of the data file or selected tag. The first action is to ignore the attempted skip and remain on the last record and thus remain positioned on a valid record. The second option is to set an end of file flag to be true. In this case there is no current record until the data file is repositioned to a valid record. The last option is to cause the **CbMaster** to append a record just as though the Append method had been called or the Append button had been pressed.

This property is read/write at design time and runtime.

EOFAction can have one of these values:

Settings	Value	Description
Last Record	0	When attempting to skip past the last record, the last record remains the current record. This is the default setting.
EOF	1	An end of file flag is set to true and the data file is not positioned on a valid record until the data file is repositioned.
Append New Record	2	This setting causes the CbMaster to go into append mode. The type of record appended depends on the AppendRecordType property.

See Also: **AppendRecordType, DataTag, BOFAction** properties**, Append** method

Font Property

VB Usage: [*form*].*CbCtrl*.**Font**

VC++ Usage: COleFont *m_CbCtrl*.**GetFont( )**
void *m_CbCtrl*.**SetFont(** LPDISPATCH *newFont* **)**

Delphi Usage: [*form*].*CbCtrl*.**Font**[:= *newFont:* Variant]

C++ Builder Usage: [*form*]->*CbCtrl*->**Font**[= Variant *newFont*)]

Applicable To: **CbMaster**, **CbButton, CbCombo, CbEdit, CbList**

Description: This property determines the font for the text that is displayed by the control.

Each control has a Font property page that can be accessed at design time. The Font property page provides a graphical interface with which to manipulate the font style and size.

By default, Font is set to the font of the Windows system text.

Under Visual Basic, use the Font property as you would for any Visual Basic control at runtime. See your Visual Basic online help for details.

This property is read/write at design time and runtime.

ForeColor Property

VB Usage: [*form*].*CbCtrl*.**ForeColor**[= *color&*]

VC++ Usage: OLE_COLOR *m_CbCtrl*.**GetForeColor( )**
void *m_CbCtrl*.**SetForeColor(** OLE_COLOR *color***)**

Delphi Usage: [*form*].*CbCtrl*.**ForeColor**[:= *setting:* TColor]

C++ Builder Usage: [*form*]->*CbCtrl*->**ForeColor**[= TColor *setting*]

Applicable To: **CbMaster, CbButton, CbCombo, CbEdit, CbList, CbSlider**

Description: Sets or returns the color used to paint the text on the control.

By default, ForeColor is set to the color of the Windows system text.

Any valid RGB value may be used for this property. The actual number of colors available is limited by the monitor's display adapter and display driver.

This property is read/write at design time and runtime.

See Also: **BackColor** property

FormatStripping Property

VB Usage: [*form*].*CbCtrl*.**FormatStripping**[= *setting%*]

VC++ Usage: short *m_CbCtrl*.**GetFormatStripping( )**
void *m_CbCtrl*.**SetFormatStripping(** short *setting***)**

Delphi Usage: [*form*].*CbCtrl*.**FormatStripping**[:= *setting:* short]

C++ Builder Usage: [*form*]->*CbCtrl*->**FormatStripping**[= short *setting*]

Applicable To: **CbEdit, CbList, CbCombo**

Description: This property determines whether formatting characters should be included when the contents of the control are either extracted through the Text property, copied to the clipboard, or are written to a data file field.

When format stripping is applied, all formatting characters are removed before the contents of the control are retrieved. When the FormatType property is Date, the contents of the control are converted into CCYYMMDD format.

This property is read/write at design time and runtime.

The following table lists the FormatStripping property settings:

Setting	Value	Description
None	0	No stripping. All data, including formatting characters are returned as part of the control's contents.
Data File	1	Datafile stripping. When the control is bound to a data field, formatting characters are stripped before the value is written to the field.
Clipboard & Text	2	Clipboard and Text stripping. When the contents of the control are copied to the Clipboard or retrieved through the Text property, all formatting characters are stripped.
Both.	3	Formatting characters are always stripped before the value of the control is returned. This is the default setting.

See Also: **Mask**, **FormatType, Text** properties

FormatType Property

VB Usage: [*form*].*CbCtrl*.**FormatType**[= *setting%*]

VC++ Usage: short *m_CbCtrl*.**GetFormatType( )**
void *m_CbCtrl*.**SetFormatType(** short *setting***)**

Delphi Usage: [*form*].*CbCtrl*.**FormatType**[:= *setting:* short]

C++ Builder Usage: [*form*]->*CbCtrl*->**FormatType**[= short *setting*]

Applicable To: **CbEdit, CbList, CbCombo**

Description: This property specifies the type formatting that is used by the control. At design time, when this property is set, CodeControls automatically creates a default setting for the Mask property.

The following table lists the FormatType property settings:

Settings	Value	Description
None	0	Any type of data can be assigned to the control. This is the default setting.
Date	1	Only date information can be assigned to the control. When the Date formatting is specified using the property pages, the Mask property is set to "CCYY/MM/DD", by default.
Generic Mask	2	Only data matching the input mask may be entered or displayed in the control. When the Generic Mask formatting is specified using the property pages, the Mask property is set to "(999) 999-9999", by default.
Numeric	3	Only numeric value can be assigned to the control.

NOTE

The FormatType property is read-only at runtime for the CbList and CbCombo when they are NOT linked to a data file. The reasons are explained in the "CodeControls Concepts" chapter. Otherwise, FormatType is read/write at design time and runtime.

See Also: **Mask** property

HideSelection Property

VB Usage: [*form*].*CbCtrl*.**HideSelection**[= {**TRUE** | **FALSE**}]

VC++ Usage: BOOL *m_CbCtrl*.**GetHideSelection( )**
void *m_CbCtrl*.**SetHideSelection(** BOOL *setting***)**

Delphi Usage: [*form*].*CbCtrl*.**HideSelection**[:= {**TRUE** | **FALSE**}]

C++ Builder Usage: [*form*]->*CbCtrl*->**HideSelection**[= {**TRUE** | **FALSE**}]

Applicable To: **CbEdit**

Description: This property determines whether the selected text will be hidden when the **CbEdit** loses the focus. By default, HideSelection is set to true, which causes the selection to be hidden when the **CbEdit** loses the focus and shows the selection when the controls receives the input focus. When HideSelection is set to false, the **CbEdit** always shows the selection regardless of the input focus.

This property is read/write at design time and read-only at runtime.

HighlightColor Property

VB Usage: [*form*].*CbCtrl*.**HighlightColor**[= *color&*]

VC++ Usage: OLE_COLOR *m_CbCtrl*.**GetHighlightColor( )**
void *m_CbCtrl*.**SetHighlightColor(** OLE_COLOR *color***)**

Delphi Usage: [*form*].*CbCtrl*.**HighlightColor**[:= *setting:* TColor]

C++ Builder Usage: [*form*]->*CbCtrl*->**HighlightColor**[= TColor *setting*]

Applicable To: **CbMaster, CbButton, CbSlider**

Description: Sets or returns the color used to draw the light shading lines that surround the control, which enhance its 3D appearance.

When the BevelType property of the **CbMaster** is set to Inset Bevel (2), the HighlightColor property affects the lower and right edges of the control. When set to Raised Bevel (3), the HighlightColor property affects the upper and left edges of the control.

By default, HighlightColor is set to the Windows system button highlight color.

Any valid RGB value may be used for this property. The actual number of colors available is limited by the monitor's display adapter and display driver.

This property is read/write at design time and runtime.

See Also: **BevelType**, **ShadowColor** properties

hWnd Property

VB Usage: [*form*].*CbCtrl*.**hWnd**[= *handle%*]

VC++ Usage: OLE_HANDLE *m_CbCtrl*.**GetHWnd( )**
void *m_CbCtrl*.**SetHWnd(** OLE_HANDLE *handle***)**

Delphi Usage: [*form*].*CbCtrl*.**hWnd**[:= *handle:* integer]

C++ Builder Usage: [*form*]->*CbCtrl*->**hWnd**[= int *handle*]

Applicable To: **CbMaster, CbButton, CbCombo, CbEdit, CbList, CbSlider**

Description: Sets or returns the window handle for the control. Windows automatically assigns the window handle for each instance of a control.

This property is read/write at design time and runtime. This property can not be accessed at design time under Visual Basic.

IndexFiles Property

VB Usage: [*form*].*CbCtrl*.**IndexFiles**[= *indexFileNames$*]

VC++ Usage: CString *m_CbCtrl*.**GetIndexFiles( )**
void *m_CbCtrl*.**SetIndexFiles(** CString *indexFileNames* **)**

Delphi Usage: [*form*].*CbCtrl*.**IndexFiles**[:= *indexFileNames:* string]

C++ Builder Usage: [*form*]->*CbCtrl*->**IndexFiles**[= AnsiString *indexFileNames*]

Applicable To: **CbMaster**

Description: Sets or returns the names of the index files that are used by the **CbMaster**.

The names of the index files to be used should be concatenated together to form the string *indexFileNames*. The paths of the index files should be included if necessary. The paths must be separated by semi-colons. For example *indexFileNames* could be set to "c:\data\employee.cdx;c:\data\customer.cdx" .

Adding an index file name to the IndexFiles property can be done automatically by using one of the **CbMaster** property pages at design time. Open the property pages for the **CbMaster** and choose the Files tab. Press the Add button and an Open dialog box appears. Select the appropriate index file and click Open. Now the name of the new index file, complete with its path, is added to the IndexFiles property.

NOTE

When you are using the client/server configuration of CodeBase with CodeControls, the Add button is not enabled on the Files property page. Under client/server, the index file names must be typed in manually on the property page. See the "Client/Server" section of the "Programming CodeControls" chapter for more information.

IndexFiles is initially set to an empty string ("").

This property is read/write at design time and runtime.

See Also: **DatabaseName**, **DataTag** properties

List Property

VB Usage: *str$* = [*form*].*CbCtrl*.**List(** *index&* **)**
[*form*].*CbCtrl*.**List(** *index&* **)** = *newStr$*

VC++ Usage: CString *m_CbCtrl*.**GetList(** long *index* **)**
void *m_CbCtrl*.**SetList(** long *index,* LPCTSTR *newStr* **)**

Delphi Usage: *str*: string := [*form*].*CbCtrl*.**List[** *index*: Integer **]**
[*form*].*CbCtrl*.**List[** *index*: Integer**]** := const *Value:* string

C++ Builder Usage: AnsiString *str* = [*form*]->*CbCtrl*->**List[** int *index* **]**
[*form*]->*CbCtrl*->**List[** int *index* **]** = const AnsiString *newStr*

Applicable To: **CbList, CbCombo**

Description: Sets or returns the contents of a specified item in the **CbList** or list portion of the **CbCombo**.

The *index* parameter is the zero-based index of the item that is to be returned or changed. When the **CbList** or list portion of the **CbCombo** is not linked to a data file then *index* has a maximum value of 32,767 under Windows 95. The *index* parameter also has the Windows 95 limitation of 32,727 when the **CbList** is linked to data file and the DataAction property is set to Field Value (2).

index represents a record number if the **CbList** is linked to a data file and the DataAction is set to Record Position (1) or if the list portion of the **CbCombo** is linked to a data file. In these cases, List is a read-only property and the list entry specified by *index* can not be changed, only the list entry may be returned.

This property is read/write at run time only.

ListCount Property

VB Usage: long [*form*].*CbCtrl*.**ListCount**

VC++ Usage: long *m_CbCtrl*.**GetListCount( )**

Delphi Usage: Integer [*form*].*CbCtrl*.**ListCount**

C++ Builder Usage: int [*form*]->*CbCtrl*->**ListCount**

Applicable To: **CbList, CbCombo**

Description: If the **CbList** or the list portion of the **CbCombo** is not linked to a data file, this property returns the number of items in the listbox.

If the **CbList** is bound to a data file and the DataAction is set to Field Value (2) then ListCount returns the number of items in the **CbList**.

If the **CbList** is bound to a data file and the DataAction is set to Record Position (1), ListCount returns -1.

If the list portion of the **CbCombo** is linked to a data file then ListCount returns -1.

This property is read-only runtime.

ListIndex Property

VB Usage: [*form*].*CbCtrl*.Object.**ListIndex**[= *index&*]

VC++ Usage: long *m_CbCtrl*.**GetListIndex( )**
void *m_CbCtrl*.**SetListIndex(** long *index***)**

Delphi Usage: [*form*].*CbCtrl*.**ListIndex**[:= *index:* Integer]

C++ Builder Usage: [*form*]->*CbCtrl*->**ListIndex**[= int *index*]

Applicable To: **CbList, CbCombo**

Description: If the **CbList** is not linked to a data file and is a single-selection list box then this property returns the zero-based index of the currently selected item. If the **CbList** is a multiple-selection list box then ListIndex returns the zero-based index of the item with the focus rectangle. In this case, the item with the focus may or may not be selected.

If the list portion of the **CbCombo** is not linked to a data file then ListIndex returns the zero-based index of the currently selected item.

The *index* parameter is the zero-based index of the new item to be selected when the **CbList** or the list portion of the **CbCombo** is not linked to a data file. The highlight of the previous item is removed and the new item is selected and scrolled into view if necessary. Under Windows 95 the value of *index* is limited to 32,767 since the number of items in a listbox is restricted.

Do not use ListIndex to select an item in a multiple-selection **CbList**. Use the SetSelected method to set multiple-selections, instead. Use the GetSelected method to check whether a particular item in the list box is selected or not.

If the **CbList** is linked to a data file and the Data Action is set to Record Position (1) or if the list portion of the **CbCombo** is linked to a data file, then *index* represents a record number. ListIndex will select the record in the data file corresponding to *index* and scroll it into view if necessary. The data file will automatically be repositioned to the newly selected record. If the record does not exist in the **CbMaster's** selected tag, the data file is not repositioned. If the edit portion of the **CbCombo** is also bound to a data file, then the selected item is also assigned to the field bound to the edit. The value of *index* is limited to the record numbers that occur in the selected tag. If there is no selected tag, *index* is limited to the number of records in the data file. If *index* is not supplied as a parameter then the record number of the selected item is returned.

If the **CbList** is linked to a data file and the Data Action is set to Field Value (2) then the *index* parameter represents the zero-based index of the items in the list box. ListIndex will select the list entry corresponding to *index* and scroll it into view if necessary. The field value corresponding to the specified *index* is automatically assigned to the current record. Under Windows 95 the value of *index* is limited to 32,767 since the number of items in a listbox is restricted.

If the list portion of the **CbCombo** is not linked to a data file and the edit portion is linked to a data file then the *index* parameter represents the zero-based index of the items in the list box. ListIndex will select the list entry corresponding to *index* and scroll it into view if necessary. The field value corresponding to the specified *index* is automatically assigned to the field bound to the edit. Under Windows 95 the value of *index* is limited to 32,767 since the number of items in a listbox is restricted.

This property is read/write at runtime only.

NOTE

Visual Basic supplies its own ListIndex property to the control, by default. Use the Visual Basic Object property to specify that you want to use ListIndex property supplied by the **CbList** or **CbCombo** control. See the Visual Basic online help for more information.

e.g. CbList1..Object.ListIndex = 5

See Also: **MultiSelect** property, **GetSelected, SetSelected** methods

ListItems Property

VB Usage: [*form*].*CbCtrl*.**ListItems**[= *itemsStr$*]

VC++ Usage: CString *m_CbCtrl*.**GetListItems( )**
void *m_CbCtrl*.**SetListItems(** LPCTSTR *itemsStr* **)**

Delphi Usage: [*form*].*CbCtrl*.**ListItems**[:= *itemsStr:* string]

C++ Builder Usage: [*form*]->*CbCtrl*->**ListItems**[= AnsiString *itemsStr*]

Applicable To: **CbList, CbCombo**

Description: The **CbList** and the **CbCombo** are initially drawn with an empty list when they are not linked to a data file. Use the ListItems property to specify items to be entered into the **CbList** or list portion of the **CbCombo** when they are first drawn.

If you want to add or remove items from the **CbList** or **CbCombo** at runtime, use the AddItem and RemoveItem methods.

The list entries are concatenated together to form one string which becomes the ListItems property. The list entries are separated by a carriage return character (i.e. "\r\n"). The ListItems property is initially set to an empty string ("").

This property is read/write at design time and read-only at runtime.

See Also: **AddItem, RemoveItem** methods

Mask Property

VB Usage: [*form*].*CbCtrl*.**Mask**[= *mask$*]

VC++ Usage: CString *m_CbCtrl*.**GetMask( )**
void *m_CbCtrl*.**SetMask(** LPCTSTR *mask***)**

Delphi Usage: [*form*].*CbCtrl*.**Mask**[:= *mask:* string]

C++ Builder Usage: [*form*]->*CbCtrl*->**Mask**[= AnsiString *mask*]

Applicable To: **CbEdit, CbList, CbCombo**

Description: The Mask property determines the format of data that can be entered/displayed in the control's edit portion, or listed in the control's list portion. The Mask property can only be set directly when the control's FormatType property has been set to Date (1) or Generic Mask (2).

When the FormatType property is set to Numeric (3), the Mask property is not set. The Numeric format is determined directly through the following properties: DecimalChar, Decimals, MaxLength, NegativeColor, NegativeLeader, NegativeTrailer, PositiveLeader, PositiveTrailer, ThouSeparatorChar and ThouSeparatorEnabled.

When the FormatType is set to Date (1) using the property pages, the default Mask property is automatically set to "CCYY/MM/DD". When the FormatType is set to Generic Mask (2) using the property pages, the default Mask property is automatically set to "(999) 999-9999".

The following table lists the mask characters available for the control when the FormatType property is set to Date (1) and Masked (2).

FormatType set to Date (1)

Character	Meaning
C or c	Second digit of the century
CC or cc	First two digit of the century
YY or yy	First two digits of the year
DD or dd	Numeric representation of the day
MM or mm	Numeric representation of the month
MMM to MMMMMMMMM or mmm to mmmmmmmmm	Character representation of the month
\	When a backslash character '\' precedes any character, the character is treated as a literal, including any mask characters.

FormatType set to Masked (2)

Character	Meaning
!	Converts to upper case
9	Numeric data (0-9)
#	Extended numeric data (+, -, <space>, or 0-9)
A or a	Letters only (A-Z or a-z)
L or l	Logical data. (y, Y, n, N, t, T, f, or F) Converts to T or F.
N or n	Alphanumeric data. (A-Z, a-z, 0-9)
X or x	Any character
Y or y	Yes or No (y, Y, n, or N). Converts to Y and N
\	When a backslash character '\' precedes any character, the character is treated as a literal, including any mask characters.

NOTE

The Mask property is read-only at runtime for the CbList and CbCombo when they are NOT linked to a data file. The reasons are explained in the "CodeControls Concepts" chapter. Otherwise, the Mask property is read/write at design time and runtime.

See Also: **FormatType** property, "CodeControls Concepts" chapter.

MaxLength Property

VB Usage: [*form*].*CbCtrl*.**MaxLength**[= *setting&*]

VC++ Usage: long *m_CbCtrl*.**GetMaxLength( )**
void *m_CbCtrl*.**SetMaxLength(** long *setting***)**

Delphi Usage: [*form*].*CbCtrl*.**MaxLength**[:= *setting:* Integer]

C++ Builder Usage: [*form*]->*CbCtrl*->**MaxLength**[= int *setting*]

Applicable To: **CbEdit, CbList, CbCombo**

Description: This property sets or returns the maximum number of characters that can be entered or displayed in the control.

The maximum length allowed for an **CbEdit** control and the edit portion of a **CbCombo** control is the same as for a standard Windows edit control: approximately 32,000 characters. The default maximum number of characters allowed is 256 characters, including any masking characters.

If the control's Mask property is set, the length of the mask overrides this property setting.

If the control is bound to a data file field then the MaxLength will be equal to the length of the field. In the case of the **CbList** or **CbCombo** control, the length will be determined by the length of the expression specified by the DataExpr or DataExprList property.

When the FormatType property is set to Numeric (3), the MaxLength property must be large enough to accommodate the number of decimal places specified by the Decimals property, as well as a negative sign and the decimal point. If MaxLength is not large enough, an exception is thrown and the previous value is used.

This property is read/write at design time and runtime.

NOTE

The MaxLength property is read-only at runtime for the CbList when they are NOT linked to a data file. The reasons are explained in the "CodeControls Concepts" chapter. Otherwise, the MaxLength property is read/write at design time and runtime.

See Also: **Decimals** property

MaxValue Property

VB Usage: [*form*].*CbCtrl*.**MaxValue**[= *value#*]

VC++ Usage: double *m_CbCtrl*.**GetMaxValue( )**
void *m_CbCtrl***SetMaxValue(** double *value* **)**

Delphi Usage: [*form*].*CbCtrl*.**MaxValue**[:= *value:* Double]

C++ Builder Usage: [*form*]->*CbCtrl*->**MaxValue**[= double *value*]

Applicable To: **CbSlider**

Description: Sets or returns the maximum value of the range on the **CbSlider**. If the value assigned to MaxValue is less than the MinValue property, the MaxValue is automatically assigned the MinValue setting.

The default value is set to 1.0.

If the **CbSlider** is linked to a data file and the DataAction is set to Record Position (1) then the MaxValue is set to 1 automatically.

This property is read/write at design time and runtime.

See Also: **MinValue, ScrollUnit, Value** properties

MinValue Property

VB Usage: [*form*].*CbCtrl*.**MinValue**[= *value#*]

VC++ Usage: double *m_CbCtrl*.**GetMinValue( )**
void *m_CbCtrl***SetMinValue(** double *value* **)**

Delphi Usage: [*form*].*CbCtrl*.**MinValue**[:= *value:* Double]

C++ Builder Usage: [*form*]->*CbCtrl*->**MinValue**[= double *value*]

Applicable To: **CbSlider**

Description: Sets or returns the minimum value of the range on the **CbSlider**. If the value assigned to MinValue is greater than the MaxValue property, the MinValue is automatically assigned the MaxValue setting.

The default value is 0.

If the **CbSlider** is linked to a data file and the DataAction is set to Record Position (1) then the MinValue is set to 0 automatically.

This property is read/write at design time and runtime.

See Also: **MaxValue, ScrollUnit, Value** properties

MultiLine Property

VB Usage: [*form*].*CbCtrl*.**MulitLine**[= {**TRUE** | **FALSE**}]

VC++ Usage: BOOL *m_CbCtrl*.**GetMultiLine( )**
void *m_CbCtrl*.**SetMultiLine(** BOOL *setting***)**

Delphi Usage: [*form*].*CbCtrl*.**MultiLine**[:= {**TRUE** | **FALSE**}]

C++ Builder Usage: [*form*]->*CbCtrl*->**MultiLine**[= {**TRUE** | **FALSE**}]

Applicable To: **CbEdit**

Description: This property determines whether the **CbEdit** is a single line text box or a multiple line text box.

By default, the MultiLine property is set to false, which means that the **CbEdit** is a single line text box. Setting MultiLine to true, means the **CbEdit** can show multiple lines of text.

Horizontal and/or vertically scroll bars can be displayed by a multiline **CbEdit** control, by setting the ScrollBars property.

The MultiLine property is read/write at design time and read-only at runtime.

See Also: **ScrollBars** property

MultiSelect Property

VB Usage: [*form*].*CbCtrl*.**MultiSelect**[= *setting%*]

VC++ Usage: short *m_CbCtrl*.**GetMultiSelect( )**
void *m_CbCtrl*.**SetMultiSelect(** short *setting***)**

Delphi Usage: [*form*].*CbCtrl*.**MultiSelect**[:= *setting:* short]

C++ Builder Usage: [*form*]->*CbCtrl*->**MultiSelect**[= short *setting*]

Applicable To: **CbList**

Description: This property determines whether a **CbList** supports multiple selections.

The MultiSelect property only has an effect when the **CbList** is not bound to a **CbMaster** control. When the **CbList** is bound to a **CbMaster** the MultiSelect property is always set to the Default (0) setting and it can not be changed. Otherwise, the MultiSelect property is read/write at design time and read-only at runtime.

The possible values for *setting* are:

Settings	Value	Description
Default	0	This setting means that the CbList does not support multiple selections and that only one item may be selected at a time. Clicking or double clicking an item will select it and remove the previous selection. The arrow keys may also be used to select an item. This is the default setting.
Simple	1	This setting supports multiple selections. The selection is toggled by clicking or double clicking the item or pressing the spacebar.
Extended	2	This setting supports multiple selections. Pressing the SHIFT key and clicking the mouse or pressing the SHIFT key and an ARROW key will extend the selection from the previously selected item to the item currently with the focus. Pressing the CTRL key and clicking the mouse will select or deselect the item with the focus without removing the other selections. Clicking an item or pressing an ARROW key on its own will remove the previous selections and select only the current item.

See Also: **ListIndex** property, **GetSelected, SetSelected** methods

NameUnique Property

VB Usage: [*form*].*CbCtrl*.**NameUnique**[= *name$*]

VC++ Usage: CString *m_CbCtrl*.**GetNameUnique( )**
void *m_CbCtrl*.**SetNameUnique(** LPCTSTR *name* **)**

Delphi Usage: [*form*].*CbCtrl*.**NameUnique**[:= *name:* string]

C++ Builder Usage: [*form*]->*CbCtrl*->**NameUnique**[= AnsiString *name*]

Applicable To: **CbMaster**

Description: Each instance of a **CbMaster** on a particular container must have a unique name so that the other controls may have a list of available **CbMasters** at design time. Visual Basic, Delphi and C++ Borland automatically supply a unique name for each instance of a control, through the Name property. The dialog boxes under Visual C++ do not supply a unique name for each instance of a control. Consequently, the **CbMaster** generates its own unique name, which can be accessed through the **CbMaster's** NameUnique property. This NameUnique property may be changed at design time. An exception is thrown if the name is not unique.

This property is read/write at design time and read-only at runtime.

See Also: **Caption, Text** properties

NegativeColor Property

VB Usage: [*form*].*CbCtrl*.**NegativeColor**[= *color&*]

VC++ Usage: OLE_COLOR *m_CbCtrl*.**GetNegativeColor( )**
void *m_CbCtrl*.**SetNegativeColor(** OLE_COLOR *color***)**

Delphi Usage: [*form*].*CbCtrl*.**NegativeColor**[:= *setting:* TColor]

C++ Builder Usage: [*form*]->*CbCtrl*->**NegativeColor**[= TColor *setting*]

Applicable To: **CbEdit**

Description: Sets or returns the color used to draw a negative number. This property only has an effect when the FormatType property is set to Numeric (3). When a negative number is entered into a **CbEdit** control, the number is automatically drawn with the color specified by this property.

The NegativeColor property is set to red, by default.

Any valid RGB value may be used for this property. The actual number of colors available is limited by the monitor's display adapter and display driver.

This property is read/write at design time and runtime.

See Also: **ForeColor** property

NegativeLeader Property

VB Usage: [*form*].*CbCtrl*.**NegativeLeader**[= *newChar$*]

VC++ Usage: CSting *m_CbCtrl*.**GetNegativeLeader( )**
void *m_CbCtrl*.**SetNegativeLeader(** LPCTSTR *newChar* **)**

Delphi Usage: [*form*].*CbCtrl*.**NegativeLeader**[:= *newChar:* string]

C++ Builder Usage: [*form*]->*CbCtrl*->**NegativeLeader**[= AnsiString *newChar*]

Applicable To: **CbEdit, CbList, CbCombo**

Description: Sets or returns the character displayed to the immediate left of negative numeric entries. This property only applies when the FormatType property for the control is set to Numeric (3).

The default value is the negative symbol ("-").

NOTE

The NegativeLeader property is read-only at runtime for the CbList and CbCombo when they are NOT linked to a data file. The reasons are explained in the "CodeControls Concepts" chapter. Otherwise, the NegativeLeader property is read/write at design time and runtime.

See Also: **NegativeTrailer, FormatType** properties

NegativeTrailer Property

VB Usage: [*form*].*CbCtrl*.**NegativeTrailer**[= *newChar$*]

VC++ Usage: CString *m_CbCtrl*.**GetNegativeTrailer( )**
void *m_CbCtrl*.**SetNegativeTrailer(** LPCTSTR *newChar* **)**

Delphi Usage: [*form*].*CbCtrl*.**NegativeTrailer**[:= *newChar:* string]

C++ Builder Usage: [*form*]->*CbCtrl*->**NegativeTrailer**[= AnsiString *newChar*]

Applicable To: **CbEdit, CbList, CbCombo**

Description: Sets or returns the character to be displayed immediately to the right of a negative numeric entry. This property only applies when the FormatType property for the control is set to Numeric (3).

The default value is a null string ("").

This setting is typically used in conjunction with the NegativeLeader property. For example, specifying NegativeLeader as "(" and NegativeTrailer as ")" results in negative values being displayed in parenthesis (e.g. "(10000)").

NOTE

The NegativeTrailer property is read-only at runtime for the CbList and CbCombo when they are NOT linked to a data file. The reasons are explained in the "CodeControls Concepts" chapter. Otherwise, the NegativeTrailer property is read/write at design time and runtime.

See Also: **NegativeLeader, FormatType** properties

OptimisticLocking Property

VB Usage: [*form*].*CbCtrl*.**OptimisticLocking**[= *setting%*]

VC++ Usage: short *m_CbCtrl*.**GetOptimisticLocking( )**
void *m_CbCtrl*.**SetOptimisticLocking(** short *setting* **)**

Delphi Usage: [*form*].*CbCtrl*.**OptimisticLocking**[:= *setting:* short]

C++ Builder Usage: [*form*]->*CbCtrl*->**OptimisticLocking**[= short *setting*]

Applicable To: **CbMaster**

Description: This property specifies the type of locking used by the **CbMaster**. The **CbMaster** can use optimistic or pessimistic locking. If the OptimisticLocking property is set to 1 or other non-zero value, optimistic locking is performed. If the OptimisticLocking property is set to 0, pessimistic locking is performed.

If optimistic locking is specified, then the **CbMaster** attempts to lock the current record before any changes are flushed to disk. If the lock attempt fails, a NoUpdate event is fired. If the record can not be locked, the changes can not be flushed.

If pessimistic locking is specified, then the **CbMaster** attempts to lock the current record when the user attempts to modify the current record. If the lock attempt fails, a NoEdit event is fired. If the record can not be locked, the record can not be modified.

By default, the OptimisticLocking property is set to 0, which means that optimistic locking is performed.

This property is read/write at design time and runtime.

See Also: **NoUpdate, NoEdit** events

OptimizeRead Property

VB Usage: [*form*].*CbCtrl*.**OptimizeRead**[= *setting%*]

VC++ Usage: short *m_CbCtrl*.**GetOptimizeRead( )**
void *m_CbCtrl*.**SetOptimizeRead(** short *setting* **)**

Delphi Usage: [*form*].*CbCtrl*.**OptimizeRead**[:= *setting:* short]

C++ Builder Usage: [*form*]->*CbCtrl*->**OptimizeRead**[= short *setting*]

Applicable To: **CbMaster**

Description: This property specifies when the memory read optimization is used.

Call RefreshFiles after changing the OptimizeRead property at runtime. The RefreshFiles closes the data file and then reopens the data file according to the new OptimizeRead setting.

This property is read/write at design time and runtime.

Possible values for OptimizeRead:

Setting	Value	Description
Exclusive Mode Only	0	Read-optimize when files are opened exclusively, when the application is stand-alone or when the DOS read-only attribute is set for the file. Otherwise, do not read optimize. This is the default value.
Never Read Optimize	1	Do not read optimize.
Always Read Optimize	2	Read optimize files, including those opened in shared mode.

See Also: **OptimizeWrite** property, **RefreshFiles** method

OptimizeWrite Property

VB Usage: [*form*].*CbCtrl*.**OptimizeWrite**[= *setting%*]

VC++ Usage: short *m_CbCtrl*.**GetOptimizeWrite( )**
void *m_CbCtrl*.**SetOptimizeWrite(** short *setting* **)**

Delphi Usage: [*form*].*CbCtrl*.**OptimizeWrite**[:= *setting:* short]

C++ Builder Usage: [*form*]->*CbCtrl*->**OptimizeWrite**[= short *setting*]

Applicable To: **CbMaster**

Description: This property specifies when the memory write optimization is used. Read optimization must be enabled for write optimization to take effect. In addition, to write optimize shared data, index and memo files, it is important to lock the files.

Call RefreshFiles after changing the OptimizeWrite property at runtime. The RefreshFiles closes the data file and then reopens the data file according to the new OptimizeWrite setting.

This property is read/write at design time and runtime.

Possible values for OptimizeWrite:

Setting	Value	Description
Exclusive Mode Only	0	Write-optimize when files are opened exclusively or when the application is stand-alone. Otherwise, do not read optimize. This is the default value.
Never Write Optimize	1	Do not write optimize.
Always Write Optimize	2	Write optimize files, including those opened in shared mode. Shared files must be locked before write optimization takes effect.

See Also: **OptimizeRead** property, **RefreshFiles** method

Orientation Property

VB Usage: [*form*].*CbCtrl*.**Orientation**[= *setting%*]

VC++ Usage: short *m_CbCtrl*.**GetOrientaion( )**
void *m_CbCtrl*.**SetOrienation(** short *setting***)**

Delphi Usage: [*form*].*CbCtrl*.**Orientation**[:= *setting:* short]

C++ Builder Usage: [*form*]->*CbCtrl*->**Orientation**[= short *setting*]

Applicable To: **CbMaster**

Description: This property determines orientation of the **CbMaster** when it is drawn at design time and runtime. The buttons displayed by the **CbMaster** may be stacked vertically or lined up horizontally according to the Orientation property.

This property is read/write at design time and runtime.

The possible values for *setting* are:

Setting	Value	Description
Horizontal	0	The CbMaster is drawn horizontally. This is the default setting.
Vertical	non-zero	The CbMaster is drawn vertically.

PageSize Property

VB Usage: [*form*].*CbCtrl*.**PageSize**[= *setting&*]

VC++ Usage: long *m_CbCtrl*.**GetPageSize( )**
void *m_CbCtrl*.**SetPageSize(** long *setting***)**

Delphi Usage: [*form*].*CbCtrl*.**PageSize**[:= *setting:* Integer]

C++ Builder Usage: [*form*]->*CbCtrl*->**PageSize**[= int *setting*]

Applicable To: **CbMaster**

Description: The property determines how many records are skipped when the PreviousPage or NextPage button are clicked on the **CbMaster**.

This property also determines the number of records skipped when the **CbButton** is bound to a **CbMaster** and has its DataAction property set to PreviousPage (7) or NextPage (3).

The PageSize property is set to 10, by default.

This property is read/write at design time and runtime.

Password Property

VB Usage: [*form*].*CbCtrl*.**Password**[= *password$*]

VC++ Usage: CString *m_CbCtrl*.**GetPassword( )**
void *m_CbCtrl*.**SetPassword(** LPCTSTR *password***)**

Delphi Usage: [*form*].*CbCtrl*.**Password**[:= *password:* string]

C++ Builder Usage: [*form*]->*CbCtrl*->**Password**[= AnsiString *password*]

Applicable To: **CbMaster**

Description: This property is only used when the CodeBase client/server configuration is used with CodeControls 3.0. This property specifies the password to be used by the server for the user name and password verification.

Currently CodeBase does not support user name and password verification. This feature may be added in future versions of CodeBase at which time this property will be used.

The Password property is set to an empty string (""), by default.

The server should be running at design time as well as runtime because the **CbMaster** opens and closes data files to access field and tag information. For this reason, the ServerId, ProcessId, UserName and Password properties should be set before the DatabaseName and IndexFiles properties are specified.

Call the RefreshFiles method after changing this property at runtime. The RefreshFiles method will cause the **CbMaster** to reconnect to the server with the new password and then reopen the data file.

This property is read/write at design time and runtime.

Refer to the CodeBase *Getting Started* manual for more information the CodeBase client/server configuration.

See Also: **ServerId, ProcessId, UserName** properties, **RefreshFiles** method

PasswordChar Property

VB Usage: [*form*].*CbCtrl*.**PasswordChar**[= *chr$*]

VC++ Usage: CString *m_CbCtrl*.**GetPasswordChar( )**
void *m_CbCtrl*.**SetPasswordChar(** LPCTSTR *chr* **)**

Delphi Usage: [*form*].*CbCtrl*.**PasswordChar**[:= *chr:* string]

C++ Builder Usage: [*form*]->*CbCtrl*->**PasswordChar**[= AnsiString *chr*]

Applicable To: **CbEdit**

Description: When the Password property is set to a desired character, that character will be displayed as the user types.

If the PasswordChar is set to an empty string (""), the text is displayed normally, as typed by the user. PasswordChar is set to an empty string, by default.

This property will only have an effect when the **CbEdit** MultiLine property is set to false.

This property is read/write at design time and runtime.

See Also: **MultiLine** property

Position Property

VB Usage: [*form*].*CbCtrl*.**Position**[= *position#*]

VC++ Usage: double *m_CbCtrl*.**GetPosition( )**
void *m_CbCtrl***SetPosition(** double *position* **)**

Delphi Usage: [*form*].*CbCtrl*.**Position**[:= *position:* Double]

C++ Builder Usage: [*form*]->*CbCtrl*->**Position**[= double *position*]

Applicable To: **CbMaster**

Description: This property can be used to reposition the data file or return the current position of the data file. The Position property uses the currently selected tag to determine the position of the current record. If there is no selected tag, then natural ordering is used. The Position property returns a decimal value ranging between 0 and 1, which represents the position of the current record in the data file or selected tag. If the Position property is set to a value between 0 and 1, the data file is repositioned to the record located at that position occurring in the data file or tag.

If there is no record at the precise position specified, then the next record is chosen as the current record. For example, if a data file had three records and Position was set to 0.25, then the data file would be repositioned to the second record since there is no record at the 0.25 position. In this case, the Position property would now return 0.5 because the data file is not actually positioned at the 0.25 position.

When the Position property is set to value less than or equal to 0, the data file is repositioned to the top record. When the Position property is set to a value greater than or equal to 1, then data file is repositioned to the bottom record.

This property is read/write at runtime.

PositiveLeader Property

VB Usage: [*form*].*CbCtrl*.**PositiveLeader**[= *newChar$*]

VC++ Usage: CString *m_CbCtrl*.**GetPositiveLeader( )**
void *m_CbCtrl*.**SetPositiveLeader(** LPCTSTR *newChar* **)**

Delphi Usage: [*form*].*CbCtrl*.**PositiveLeader**[:= *newChar:* string]

C++ Builder Usage: [*form*]->*CbCtrl*->**PositiveLeader**[= AnsiString *newChar*]

Applicable To: **CbEdit, CbList, CbCombo**

Description: Sets or returns the character displayed to the immediate left of positive numeric entries. This property only applies when the FormatType property for the control is set to Numeric (3).

The default value is a space character (" "). A value of zero is considered a positive number for the purpose of displaying a leader character.

NOTE

The PositiveLeader property is read-only at runtime for the CbList and CbCombo when they are NOT linked to a data file. The reasons are explained in the "CodeControls Concepts" chapter. Otherwise, the PositiveLeader property is read/write at design time and runtime.

See Also: **PositiveTrailer, FormatType** properties

PositiveTrailer Property

VB Usage: [*form*].*CbCtrl*.**PositiveTrailer**[= *newChar$*]

VC++ Usage: CString *m_CbCtrl*.**GetPositiveTrailer( )**
void *m_CbCtrl*.**SetPositiveTrailer(** LPCTSTR *newChar* **)**

Delphi Usage: [*form*].*CbCtrl*.**PositiveTrailer**[:= *newChar:* string]

C++ Builder Usage: [*form*]->*CbCtrl*->**PositiveTrailer**[= AnsiString *newChar*]

Applicable To: **CbEdit, CbList, CbCombo**

Description: Sets or returns the character to be displayed immediately to the right of a positive or zero numeric entry. This property only applies when the FormatType property for the control is set to Numeric (3).

The default value is a null string ("").

This setting is typically used in conjunction with the PositiveLeader property. For example, specifying PositiveLeader as "[" and PositiveTrailer as "]" results in positive values being displayed in square brackets (e.g. "[10000]").

NOTE

The PositiveTrailer property is read-only at runtime for the CbList and CbCombo when they are NOT linked to a data file. The reasons are explained in the "CodeControls Concepts" chapter. Otherwise, the PositiveTrailer property is read/write at design time and runtime.

See Also: **PositiveLeader, FormatType** properties

ProcessId Property

VB Usage: [*form*].*CbCtrl*.**ProcessId**[= *processId$*]

VC++ Usage: CString *m_CbCtrl*.**GetProcessId( )**
void *m_CbCtrl*.**SetProcessId(** LPCTSTR *processId***)**

Delphi Usage: [*form*].*CbCtrl*.**ProcessId**[:= *processId:* string]

C++ Builder Usage: [*form*]->*CbCtrl*->**ProcessId**[= AnsiString *processId*]

Applicable To: **CbMaster**

Description: This property is only used when the client/server configuration of CodeBase is used with CodeControls 3.0. If the ProcessId field of the S4SERVER.DBF file for the server has been changed, then the **CbMaster** ProcessId property must be changed to the same value. If the ProcessId of the server has not be changed from the default, then the default ProcessId property for the **CbMaster** need not be changed.

The ProcessId property is set to "23165", by default.

The server should be running at design time as well as runtime because the **CbMaster** opens and closes data files to access field and tag information. For this reason, the ServerId, ProcessId, UserName and Password properties should be set before the DatabaseName and IndexFiles properties are specified.

Call the RefreshFiles method after changing this property at runtime. The RefreshFiles method will cause the **CbMaster** to reconnect to a server with the new process identification and then reopen the data file.

This property is read/write at design time and runtime.

Refer to the CodeBase *Getting Started* manual for more information on the server ProcessId.

See Also: **ServerId, Password, UserName** properties, **RefreshFiles** method

PromptChar Property

VB Usage: [*form*].*CbCtrl*.**PromptChar**[= *newChar$*]

VC++ Usage: CString *m_CbCtrl*.**GetPromptChar( )**
void *m_CbCtrl*.**SetPromptChar(** LPCTSTR *newChar***)**

Delphi Usage: [*form*].*CbCtrl*.**PromptChar**[:= *newChar:* string]

C++ Builder Usage: [*form*]->*CbCtrl*->**PromptChar**[= AnsiString *newChar*]

Applicable To: **CbEdit, CbCombo**

Description: Sets or returns the character used as the place holder when text formatting is used. This property is used when entering formatted data into a **CbEdit** or the edit portion of a **CbCombo**. The PromptChar is only used when the AllowPrompt property is set to true and when the FormatType property for the control is set to either Date (1) or Masked (2).

The default character is an underscore ("_").

This property is read/write at design time and runtime.

See Also: **AllowPrompt**, **FormatType** properties

PromptIncluded Property

VB Usage: [*form*].*CbCtrl*.**PromptIncluded**[= {**TRUE** | **FALSE**}]

VC++ Usage: BOOL *m_CbCtrl*.**GetPromptIncluded( )**
void *m_CbCtrl*.**SetPromptIncluded(** BOOL *setting***)**

Delphi Usage: [*form*].*CbCtrl*.**PromptIncluded**[:= {**TRUE** | **FALSE**}]

C++ Builder Usage: [*form*]->*CbCtrl*->**PromptIncluded**[= {**TRUE** | **FALSE**}]

Applicable To: **CbEdit, CbCombo**

Description: This property is used when entering formatted data into a **CbEdit** or the edit portion of a **CbCombo**. This property only applies when the property FormatType is set to Masked (2). When FormatStripping is set to None (0), then the formatted data is stored to disk when the field is flushed. If PromptIncluded is set to true, the place holder characters specified by PromptChar property are also saved to disk when the field is flushed. If PromptIncluded is set to false, then wherever there is a place holder character, a blank is stored on disk.

This property does not apply to the FormatType Date (1) because the date is stored in standard format CCYYMMDD without any formatting.

PromptIncluded is set to false, by default.

This property is read/write at design time and runtime.

See Also: **AllowPrompt**, **PromptChar**, **FormatType, FormatStripping** properties

ReadOnly Property

VB Usage: [*form*].*CbCtrl*.**ReadOnly**[= {**TRUE** | **FALSE**}]

VC++ Usage: BOOL *m_CbCtrl*.**GetReadOnly( )**
void *m_CbCtrl*.**SetReadOnly(** BOOL *setting***)**

Delphi Usage: [*form*].*CbCtrl*.**ReadOnly**[:= {**TRUE** | **FALSE**}]

C++ Builder Usage: [*form*]->*CbCtrl*->**ReadOnly**[= {**TRUE** | **FALSE**}]

Applicable To: **CbMaster**

Description: This property specifies whether files are to be opened in read-only mode.

The default setting is false, which means that a file is opened with read and write permissions. To open a file in read-only mode, set this property to true.

Call RefreshFiles after changing the ReadOnly property at runtime. The RefreshFiles closes the data file and then reopens the data file according to the new ReadOnly setting.

This property is read/write at design time and runtime.

See Also: **RefreshFiles** method

RecNo Property

VB Usage: [*form*].*CbCtrl*.**RecNo**[= *newRecno&*]

VC++ Usage: long *m_CbCtrl*.**GetRecNo( )**
void *m_CbCtrl*.**SetRecNo(** long *newRecno***)**

Delphi Usage: [*form*].*CbCtrl*.**RecNo**[:= *newRecno:* Integer]

C++ Builder Usage: [*form*]->*CbCtrl*->**RecNo**[= int *newRecno*]

Applicable To: **CbMaster**

Description: This property sets or returns the current record number for the **CbMaster** control. Setting this property to a new record number repositions the data file to that record.

If the record number specified by *newRecNo* does not exist in the data file, then the data file is not repositioned. Changing the RecNo property has the same effect as calling the Go method. Use the Go method instead of the RecNo property if you require return codes.

This property is read/write at run time only.

See Also: **Go** method

RecNoEnabled Property

VB Usage: [*form*].*CbCtrl*.**RecNoEnabled**[= {**TRUE** | **FALSE**}]

VC++ Usage: BOOL *m_CbCtrl*.**GetRecNoEnabled( )**
void *m_CbCtrl*.**SetRecNoEnabled(** BOOL *setting***)**

Delphi Usage: [*form*].*CbCtrl*.**RecNoEnabled**[:= {**TRUE** | **FALSE**}]

C++ Builder Usage: [*form*]->*CbCtrl*->**RecNoEnabled**[= {**TRUE** | **FALSE**}]

Applicable To: **CbMaster**

Description: This property determines whether current record number is displayed by the **CbMaster**.

By default, the record number is displayed and the RecNoEnabled property is set to true. If the RecNoEnabled property is set to false, then the current record number is not displayed.

The **CbMaster** record number can be accessed directly through the RecNo property.

This property are read/write at design time and runtime.

See the "Enabled Property" section of the "Programming CodeControls" chapter for more information.

See Also: ***buttonName*Enabled, *buttonName*Visible, TagListEnabled, SliderEnabled** properties

RecordChanged Property

VB Usage: bool [*form*].*CbCtrl*.**RecordChanged**

VC++ Usage: BOOL *m_CbCtrl*.**GetRecordChanged( )**

Delphi Usage: bool [*form*].*CbCtrl*.**RecordChanged**

C++ Builder Usage: bool [*form*]->*CbCtrl*->**RecordChanged**

Applicable To: **CbMaster**

Description: This property indicates whether the current record has been changed. This property returns true when the current record has been changed and false when the record has not been modified.

The RecordChanged property is read-only at runtime.

See Also: **FlushRecord, Undo, RefreshRecord** method

ScrollUnit Property

VB Usage: [*form*].*CbCtrl*.**ScrollUnit**[= *value#*]

VC++ Usage: double *m_CbCtrl*.**GetScrollUnit( )**
void *m_CbCtrl***SetScrollUnit(** double *value* **)**

Delphi Usage: [*form*].*CbCtrl*.**ScrollUnit**[:= *value:* Double]

C++ Builder Usage: [*form*]->*CbCtrl*->**ScrollUnit**[= double *value*]

Applicable To: **CbSlider**

Description: Sets or returns the scroll unit for the **CbSlider**. This property is used when the user clicks on the arrow buttons on the ends of the **CbSlider**. When an arrow button is clicked the thumb is moved one scroll unit in the arrow's direction.

Note that the thumb can be dragged to any position on the slider. The position of the thumb can be also be changed by clicking directly on the slider. This will cause the thumb to move directly to the mouse position on the slider.

If the slider is linked to a data file and the DataAction is set to Record Position (1), the ScrollUnit is ignored. The scroll unit is calculated internally as 1 divided by the number of records in the data file.

The default value is 0.1.

This property is read/write at design time and runtime.

See Also: **MinValue, MaxValue, Value, DataAction** properties

SelCount Property

VB Usage: long [*form*].*CbCtrl*.**SelCount**

VC++ Usage: long *m_CbCtrl*.**GetSelCount( )**

Delphi Usage: Integer [*form*].*CbCtrl*.**SelCount**

C++ Builder Usage: int [*form*]->*CbCtrl*->**SelCount**

Applicable To: **CbList**

Description: This property returns the number of selected items in a **CbList**. If there are no selections then 0 is returned.

The SelCount property is read-only at runtime.

See Also: **MultiSelect** properties**, GetSelected, SetSelected** methods

SelLength Property

VB Usage: [*form*].*CbCtrl*.**SelLength**[= *selectionLength%*]

VC++ Usage: short *m_CbCtrl*.**GetSelLength( )**
void *m_CbCtrl*.**SetSelLength(** short *selectionLength***)**

Delphi Usage: [*form*].*CbCtrl*.**SelLength**[:= *selectionLength:* short]

C++ Builder Usage: [*form*]->*CbCtrl*->**SelLength**[= short *selectionLength*]

Applicable To: **CbEdit, CbCombo**

Description: This property returns the length of the selected text displayed in a **CbEdit** or edit portion of a **CbCombo**.

This property may also be used to select text. In this case, the starting position of the original selection is used as the starting position for the new selection. If there is no selection, the cursor position is used as the starting position for the new selection. The selection is extended from the starting position to the specified length.

This property is read/write at runtime.

NOTE

> Visual Basic supplies its own SelLength property to the control, by default. Use the Visual Basic Object property to specify that you want to use SelLength property supplied by the **CbEdit** or **CbCombo** control. See the Visual Basic online help for more information.
>
> e.g. CbEdit1.Object.SelLength 5

See Also: **SelStart, SelText** properties

SelStart Property

VB Usage: [*form*].*CbCtrl*.**SelStart**[= *startPos%*]

VC++ Usage: short *m_CbCtrl*.**GetSelStart( )**
void *m_CbCtrl*.**SetSelStart(** short *startPos***)**

Delphi Usage: [*form*].*CbCtrl*.**SelStart**[:= *startPos:* short]

C++ Builder Usage: [*form*]->*CbCtrl*->**SelStart**[= short *startPos*]

Applicable To: **CbEdit, CbCombo**

Description: This property returns starting position of the selected text in a **CbEdit** or edit portion of a **CbCombo**.

This property may also be used to lengthen or shorten the selection. The end position of the current selection is used as the end position for the new selection. If there is no selection, the cursor position is used as the end position for the new selection. The selection is extended from the end position to the position specified by the *startPos* parameter.

This property is read/write at runtime.

NOTE

Visual Basic supplies its own SelStart property to the control by default. Use the Visual Basic Object property to specify that you want to use SelStart property supplied by the **CbEdit** control or **CbCombo**. See the Visual Basic online help for more information.

e.g. CbEdit1.Object.SelStart 2

See Also: **SelLength, SelText** properties

SelText Property

VB Usage: [*form*].*CbCtrl*.**SelText**[= *newText $*]

VC++ Usage: CString *m_CbCtrl*.**GetSelText( )**
void *m_CbCtrl*.**SetSelText(** LPCTSTR *newText***)**

Delphi Usage: [*form*].*CbCtrl*.**SelText**[:= *newText*: string]

C++ Builder Usage: [*form*]->*CbCtrl*->**SelText**[= AnsiString *newText*]

Applicable To: **CbEdit, CbCombo**

Description: This property returns or changes the selected text displayed in a **CbEdit** or edit portion of a **CbCombo**. The text specified by *newText* replaces the currently selected text. If there is no selected text, the *newText* is inserted in the text at the cursor position. The selection is removed when this property is used to insert text.

This property is read/write at runtime.

See Also: **SelLength, SelStart** properties

ScaleButton Property

VB Usage: [*form*].*CbCtrl*.**ScaleButton**[= {**TRUE** | **FALSE**}]

VC++ Usage: BOOL *m_CbCtrl*.**GetScaleButton( )**
void *m_CbCtrl*.**SetScaleButton(** BOOL *setting***)**

Delphi Usage: [*form*].*CbCtrl*.**ScaleButton**[:= {**TRUE** | **FALSE**}]

C++ Builder Usage: [*form*]->*CbCtrl*->**ScaleButton**[= {**TRUE** | **FALSE**}]

Applicable To: **CbButton**

Description: This property can be used to automatically resize the **CbButton**. When this property is set to true, the **CbButton** is automatically scaled to the size of the bitmaps displayed by the **CbButton** and the size can not be changed at design time. When ScaleButton is set to false the size of the **CbButton** may be changed and the bitmap is scaled to fit the size of the **CbButton**.

ScaleButton is set to false, by default. This property only has an effect when the UseBitmaps property is set to true.

This property is read/write at design time and runtime.

See Also: **UseBitmaps, UseAltBitmaps, DefaultBitmaps** properties

ScrollBars Property

VB Usage: [*form*].*CbCtrl*.**ScrollBars**[= *setting%*]

VC++ Usage: short *m_CbCtrl*.**GetScrollBars( )**
void *m_CbCtrl*.**SetScrollBars(** short *setting***)**

Delphi Usage: [*form*].*CbCtrl*.**ScrollBars**[:= *setting:* short]

C++ Builder Usage: [*form*]->*CbCtrl*->**ScrollBars**[= short *setting*]

Applicable To: **CbEdit**

Description: This property determines what kind of scroll bars will be displayed by a multiple line **CbEdit** control. This property only has an effect when the MultiLine property is set to true.

This property is read/write at design time and read-only at runtime.

The possible values for *setting* are:

Settings	Value	Description
No Scroll Bars	0	The multiple line CbEdit does no use any scroll bars. The text wraps around to the next line of the control when the user types past the end of the current line. This is the default setting.
Horizontal Scroll Bar	1	The multiple line CbEdit uses a horizontal scroll bar.
Vertical Scroll Bar	2	The multiple line CbEdit uses a vertical scroll bar. The text wraps around to the next line of the control when the user types past the end of the current line.
Both Horizontal and Vertical Scroll bars	3	The multiple line CbEdit uses both the horizontal and vertical scroll bars.

See Also: **MultiLine** property

ServerId Property

VB Usage: [*form*].*CbCtrl*.**ServerId**[= *serverId$*]

VC++ Usage: CString *m_CbCtrl*.**GetServerId( )**
void *m_CbCtrl*.**SetServerId(** LPCTSTR *serverId***)**

Delphi Usage: [*form*].*CbCtrl*.**ServerId**[:= *serverId*: string]

C++ Builder Usage: [*form*]->*CbCtrl*->**ServerId**[= AnsiString *serverId*]

Applicable To: **CbMaster**

Description: This property is only used when the CodeBase client/server configuration is used with CodeControls 3.0. The ServerId property sets or returns the network identification of the computer that is running the server. By default, the ServerId is set to "LOCALHOST". This setting is valid when the client application and the server are located on the same computer. Usually the client application and the server executable are located on different computers. In this case, the ServerId must be changed to the network identification of computer running the server.

The server should be running at design time as well as runtime because the **CbMaster** opens and closes data files to access field and tag information. For this reason, the ServerId, ProcessId, UserName and Password properties should be set before the DatabaseName and IndexFiles properties are specified. If the ServerId is incorrect or the server is not running at design time, a -1320 CodeBase error may be generated.

Call the RefreshFiles method after changing this property at runtime. The RefreshFiles method will cause the **CbMaster** to reconnect to a server with the new server identification and then reopen the data file.

This property is read/write at design time and runtime.

Refer to the CodeBase *Getting Started* manual for more information on the ServerId.

See Also: **Password, ProcessId, UserName** properties, **RefreshFiles** method

ShadowColor Property

VB Usage: [*form*].*CbCtrl*.**ShadowColor**[= *color&*]

VC++ Usage: OLE_COLOR *m_CbCtrl*.**GetShadowColor( )**
void *m_CbCtrl*.**SetShadowColor(** OLE_COLOR *color***)**

Delphi Usage: [*form*].*CbCtrl*.**ShadowColor**[:= *color:* TColor]

C++ Builder Usage: [*form*]->*CbCtrl*->**ShadowColor**[= TColor *color*]

Applicable To: **CbMaster, CbButton, CbSlider**

Description: Sets or returns the color used to draw the dark shading lines that surround the control, which enhance its 3D appearance.

When the BevelType property of the **CbMaster** is set to Raised Bevel (3), the ShadowColor property affects the lower and right edges of the control. When set to Inset Bevel (2), the ShadowColor property affects the upper and left edges of the control.

By default, ShadowColor is set to the Windows system button shadow color.

Any valid RGB value may be used for this property. The actual number of colors available is limited by the monitor's display adapter and display driver.

This property is read/write at design time and runtime.

See Also: **BevelType**, **HighlightColor** properties

ShowButtons Property

VB Usage: [*form*].*CbCtrl*.**ShowButtons**[= {**TRUE** | **FALSE**}]

VC++ Usage: BOOL *m_CbCtrl*.**GetShowButtons( )**
void *m_CbCtrl*.**SetShowButtons(** BOOL *setting***)**

Delphi Usage: [*form*].*CbCtrl*.**ShowButtons**[:= {**TRUE** | **FALSE**}]

C++ Builder Usage: [*form*]->*CbCtrl*->**ShowButtons**[= {**TRUE** | **FALSE**}]

Applicable To: **CbSlider**

Description: This property determines whether the arrow buttons are displayed by the **CbSlider**.

The ShowButtons property is set to true by default, which means that the arrow buttons are displayed. When the user clicks the an arrow button, the thumb is moved one scroll unit in the arrow's direction. The size of the scroll unit is determined by the ScrollUnit property.

When the ShowButtons property is false, the **CbSlider** is drawn without the arrow buttons.

This property is read/write at design time and runtime.

See Also: **ScrollUnit** property

SliderEnabled Property

VB Usage: [*form*].*CbCtrl*.**SliderEnabled**[= {**TRUE** | **FALSE**}]

VC++ Usage: BOOL *m_CbCtrl*.**GetSliderEnabled( )**
void *m_CbCtrl*.**SetSliderEnabled(** BOOL *setting***)**

Delphi Usage: [*form*].*CbCtrl*.**SliderEnabled**[:= {**TRUE** | **FALSE**}]

C++ Builder Usage: [*form*]->*CbCtrl*->**SliderEnabled**[= {**TRUE** | **FALSE**}]

Applicable To: **CbMaster**

Description: This property determines whether the slider is displayed by the **CbMaster**. The thumb on the slider indicates the relative position of the current record in the selected tag. If there is no selected tag, the thumb indicates the relative position of the current record in the data file. When the user clicks on the slider, the data file is repositioned.

By default, the slider is displayed and the SliderEnabled property is set to true. If the SliderEnabled property is set to false, then the slider is not displayed.

This property are read/write at design time and runtime.

See the "Enabled Property" section of the "Programming CodeControls" chapter for more information.

See Also: ***buttonName*Enabled, *buttonName*Visible, RecNoEnabled, TagListEnabled** properties

Sorted Property

VB Usage: [*form*].*CbCtrl*.**Sorted**[= {**TRUE** | **FALSE**}]

VC++ Usage: BOOL *m_CbCtrl*.**GetSorted( )**
void *m_CbCtrl*.**SetSorted(** BOOL *setting***)**

Delphi Usage: [*form*].*CbCtrl*.**Sorted**[:= {**TRUE** | **FALSE**}]

C++ Builder Usage: [*form*]->*CbCtrl*->**Sorted**[= {**TRUE** | **FALSE**}]

Applicable To: **CbCombo, CbList**

Description: This property determines whether items added to the **CbList** or to the list portion of the **CbCombo** are added in alphabetical order.

The Sorted property is set to false, by default, which means that items in the list are not sorted alphabetically.

When the property is set to true, the list is sorted alphabetically. See the AddItem method for more information on how to add items to a **CbList** or **CbCombo**.

This property will have an effect when the **CbList** or the list portion of the **CbCombo** is not linked to a data file. The Sorted property will also have an effect when the **CbList** is bound to a **CbMaster** and the DataAction property is set to Field Value (2).

When the **CbList** is linked to a data file and the DataAction is set to Record Position (1) then the records are sorted according to the tag specified by the DataTag property. Similarly for the **CbCombo**, when the list portion is linked to a data file, the records are sorted according to the tag specified by the DataTagList property. The Sorted property has no effect under these circumstances.

This property is read/write at design time and read-only at runtime.

See Also: **DataAction**, **DataSource**, **DataSourceList** properties, **AddItem** method

Style Property

VB Usage: [*form*].*CbCtrl*.**Style**[= *setting&*]

VC++ Usage: short *m_CbCtrl*.**GetStyle( )**
void *m_CbCtrl*.**SetStyle(** short *setting***)**

Delphi Usage: [*form*].*CbCtrl*.**Style**[:= *setting:* short]

C++ Builder Usage: [*form*]->*CbCtrl*->**Style**[= short *setting*]

Applicable To: **CbCombo**

Description: This property sets or returns the style of the **CbCombo**. The **CbCombo** is combination of an edit box and a list box. A **CbCombo** can have one of three styles, DropDown, Simple and DropDown List. These styles are discussed in detail in the "Control Description" chapter.

This property is read/write at design time and read-only at run time.

The possible values for *setting* are:

Settings	Value	Description
Drop Down Combo	0	The list portion of the CbCombo is hidden until the user clicks on the button located next to the edit portion. The user can also type into the edit portion. This is the default setting.
Simple Combo	1	The user can type into the edit portion and the list portion is always displayed.
Drop Down List Combo	2	The edit portion displays the selection from the list portion but the user can not type into the edit. The list portion is hidden until the user clicks the button located next to the edit portion.

Tag4 Property

VB Usage: [*form*].*CbCtrl*.**Tag4**

VC++ Usage: long *m_CbCtrl*.**GetTag4( )**

Delphi Usage: [*form*].*CbCtrl*.**Tag4**

C++ Builder Usage: [*form*]->*CbCtrl*->**Tag4**

Applicable To: **CbMaster**

Description: Specifies the index tag that is being used by the **CbMaster**. This property returns a pointer to a CodeBase **TAG4** structure, which specifies the selected tag. This pointer can be used as a parameter for other CodeBase functions that can be used to manipulate the **CbMaster**'s tag directly.

This property can not be used to change the tag. There are two ways to change the tag used by the **CbMaster**. The first way is to set the DataTag property to the name of the new tag. The second way is to choose a tag using the spinlist that is displayed by the **CbMaster** at run time. The spinlist is displayed when the TagListEnabled property is set to true.

C/C++ compiler users must explicitly cast this pointer to a **TAG4** pointer before it may be used. CodeBase for C++ users should construct a **Tag4** object using the constructor that uses a **TAG4** pointer.

This property is read only at run time.

See Also: **DataTag**, **TagListEnabled** properties

TagListEnabled Property

VB Usage: [*form*].*CbCtrl*.**TagListEnabled**[= {**TRUE** | **FALSE**}]

VC++ Usage: BOOL *m_CbCtrl*.**GetTagListEnabled( )**
void *m_CbCtrl*.**SetTagListEnabled(** BOOL *setting***)**

Delphi Usage: [*form*].*CbCtrl*.**TagListEnabled**[:= {**TRUE** | **FALSE**}]

C++ Builder Usage: [*form*]->*CbCtrl*->**TagListEnabled**[= {**TRUE** | **FALSE**}]

Applicable To: **CbMaster**

Description: This property determines whether the spin list, which displays the available tags, is displayed by the **CbMaster**.

By default, the tag spin list is displayed and the TagListEnabled property is set to true. If the TagListEnabled property is set to false, then the tag spin list is not displayed.

The **CbMaster** tag can be accessed through the DataTag property regardless of the TagListEnabled setting.

This property are read/write at design time and runtime.

See the "Enabled Property" section of the "Programming CodeControls" chapter for more information.

See Also: ***buttonName*Enabled, *buttonName*Visible, RecNoEnabled, SliderEnabled** properties

Text Property

VB Usage: [*form*].*CbCtrl*.**Text**[= *text$*]

VC++ Usage: CString *m_CbCtrl*.**GetText( )**
void *m_CbCtrl*.**SetText(** LPCTSTR *text***)**

Delphi Usage: [*form*].*CbCtrl*.**Text**[:= *text*: string]

C++ Builder Usage: [*form*]->*CbCtrl*->**Text**[= AnsiString *text*]

Applicable To: **CbEdit, CbCombo, CbList**

Description: The Text property returns or sets the text displayed in a **CbEdit** or the edit portion of the **CbCombo**.

If the controls are using text formatting, the setting of the FormatStripping property will affect how the string is returned by the Text property. For example, if the FormatStripping is set to None (0) the Text returns exactly what is being displayed by the control. If FormatStripping is set to Data File Stripping (1), then Text returns the string as displayed, with the formatting intact. If the FormatStripping is set to Text Stripping (2) or Data File and Text Stripping (3), then the formatting characters are stripped from the string and the Text property returns the raw data.

The FormatStripping also determines how the text should be assigned to the Text property. If the FormatStripping is set to None (0) or Data File (1) then the programmer should assign formatted text to the Text property. If the Format Stripping is set to Text (2) or Data File and Text Stripping (3), then the programmer should assign raw data to the Text property and the data will be formatted internally.

The Text property for the **CbList** specifies the caption at design time.

Visual Basic, Delphi and C++ Builder provide a unique name for each instance of a control. The unique name can be accessed through the Name property. The Text property is initialized with this unique name, by default.

Visual C++ does not provide a unique name for each instance of a control. In this case, the Text property is initially set to the name of the control: **CbEdit**, **CbCombo** or **CbList**.

This property is read/write at design time and runtime.

See Also: **FormatStripping** property

ThouSeparatorChar Property

VB Usage: [*form*].*CbCtrl*.**ThouSeparatorChar**[= *newChar$*]

VC++ Usage: CString *m_CbCtrl*.**GetThouSeparatorChar( )**
void *m_CbCtrl*.**SetThouSeparatorChar(** LPCTSTR *newChar* **)**

Delphi Usage: [*form*].*CbCtrl*.**ThouSeparatorChar**[:= *newChar*: string]

C++ Builder Usage: [*form*]->*CbCtrl*->**ThouSeparatorChar**[= AnsiString *newChar*]

Applicable To: **CbEdit, CbList, CbCombo**

Description: Sets or returns the character used to separate hundreds from thousands, thousands from millions and so on. This property only applies when the FormatType for the control is set to Numeric (3) and the ThouSeparatorEnabled property is set to true.

The default value is a comma (",").

NOTE

> The ThouSeparatorChar property is read-only at runtime for the CbList and CbCombo when they are NOT linked to a data file. The reasons are explained in the "CodeControls Concepts" chapter. Otherwise, ThouSeparatorChar is read/write at design time and runtime.

See Also: **ThouSeparatorEnabled, FormatType** properties

ThouSeparatorEnabled Property

VB Usage: [*form*].*CbCtrl*.**ThouSeparatorEnabled**[= {**TRUE** | **FALSE**}]

VC++ Usage: BOOL *m_CbCtrl*.**GetThouSeparatorEnabled( )**
void *m_CbCtrl*.**SetThouSeparatorEnabled(** BOOL *setting* **)**

Delphi Usage: [*form*].*CbCtrl*.**ThouSeparatorEnabled**[:= {**TRUE** | **FALSE**}]

C++ Builder Usage: [*form*]->*CbCtrl*->**ThouSeparatorEnabled**[= {**TRUE** | **FALSE**}]

Applicable To: **CbEdit, CbList, CbCombo**

Description: Specifies whether a thousands separator character should be used when displaying numeric values. This property only applies when the FormatType property for the control is set to Numeric (3).

By default, the ThouSeparatorEnabled property is set to false, which means that the ThouSeparatorChar is not used during the numeric formatting. When ThouSeparatorEnabled is set to true, the ThouSeparatorChar is used during the numeric formatting.

NOTE

The ThouSeparatorEnabled property is read-only at runtime for the CbList and CbCombo when they are NOT linked to a data file. The reasons are explained in the "CodeControls Concepts" chapter. Otherwise, ThouSeparatorEnabled is read/write at design time and runtime.

See Also: **ThouSeparatorChar**, **FormatType** properties

ToolTipEnabled Property

VB Usage: [*form*].*CbCtrl*.**ToolTipEnabled**[= {**TRUE** | **FALSE**}]

VC++ Usage: BOOL *m_CbCtrl*.**GetToolTipEnabled( )**
void *m_CbCtrl*.**SetToolTipEnabled(** BOOL *setting***)**

Delphi Usage: [*form*].*CbCtrl*.**ToolTipEnabled**[:= {**TRUE** | **FALSE**}]

C++ Builder Usage: [*form*]->*CbCtrl*->**ToolTipEnabled**[= {**TRUE** | **FALSE**}]

Applicable To: **CbButton, CbEdit, CbCombo, CbList, CbSlider**

Description: This property determines whether the tool tips are displayed by the controls at runtime. A tool tip is small window that pops up when the mouse cursor is located over a control for a small amount of time. The tool tip window displays a string that gives a brief description of what the control does. The text that is displayed by the tool tip is specified by the ToolTipText property.

The ToolTipEnabled property is set to true, by default, which means that the tool tips are shown at runtime. Setting the ToolTipEnabled property to false, disables the tool tips and they are not shown at runtime.

This property is read/write at design time and runtime.

See Also: **ToolTipText** property

ToolTipsEnabled Property

VB Usage: [*form*].*CbCtrl*.**ToolTipsEnabled**[= {**TRUE** | **FALSE**}]

VC++ Usage: BOOL *m_CbCtrl*.**GetToolTipsEnabled( )**
void *m_CbCtrl*.**SetToolTipsEnabled(** BOOL *setting***)**

Delphi Usage: [*form*].*CbCtrl*.**ToolTipsEnabled**[:= {**TRUE** | **FALSE**}]

C++ Builder Usage: [*form*]->*CbCtrl*->**ToolTipsEnabled**[= {**TRUE** | **FALSE**}]

Applicable To: **CbMaster**

Description: This property determines whether the tool tips are displayed by the **CbMaster** at runtime.

The ToolTipsEnabled property is set to true, by default, which means that the tool tips are displayed at runtime. No tool tips are displayed if the ToolTipsEnabled property is set to false.

Each button on the **CbMaster** has its own individual tool tip that describes its function. The record number display, tag spin list and the slider parts of the **CbMaster** also have tool tips associated with them. The text in these tool tips can not be altered.

This property is read/write at design time and runtime.

ToolTipText Property

VB Usage: [*form*].*CbCtrl*.**ToolTipText**[= *text$*]

VC++ Usage: CString *m_CbCtrl*.**GetToolTipText( )**
void *m_CbCtrl*.**SetToolTipText(** LPCTSTR *text***)**

Delphi Usage: [*form*].*CbCtrl*.**ToolTipText**[:= *text*: string]

C++ Builder Usage: [*form*]->*CbCtrl*->**ToolTipText**[= AnsiString *text*]

Applicable To: **CbButton, CbEdit, CbCombo, CbList, CbSlider**

Description: The ToolTipText property determines what text is displayed when the tool tip window pops up.

Visual Basic, Delphi and C++ Borland automatically supply a Name property, which is a unique name given to each instance of a control. By default, the ToolTipText property is set to this unique name. Visual C++, does not provide a unique name for each instance of the control. In this case, the ToolTipText is set to the name of the control: **CbButton**, **CbEdit**, **CbCombo**, **CbList** or **CbSlider**.

This property is read/write at design time and runtime.

See Also: **ToolTipEnabled** property

TopIndex Property

VB Usage: [*form*].*CbCtrl*.**TopIndex**[= *index&*]

VC++ Usage: long *m_CbCtrl*.**GetTopIndex( )**
void *m_CbCtrl*.**SetTopIndex(** long *index***)**

Delphi Usage: [*form*].*CbCtrl*.**TopIndex**[:= *index:* Integer]

C++ Builder Usage: [*form*]->*CbCtrl*->**TopIndex**[= int *index*]

Applicable To: **CbList**

Description: The property returns the zero-based index of the item that is visible at the top of the **CbList**. The TopIndex property can also be used to scroll a particular item to the top of the **CbList**. The item at the position specified by *index* is scrolled to the top of the **CbList**. Under Windows 95 the value of *index* is limited to 32,767 since the number of items in a listbox is restricted.

If the **CbList** is linked to a data file and the DataAction is set to Record Position (1), then TopIndex returns the record number of the record visible at the top of the **CbList**. Similarly, the *index* parameter represents a record number and the specified record is scrolled to the top of the **CbList**. In this case, *index* is limited to the number of records in the data file.

This property is read/write at runtime.

Unique Property

VB Usage: [*form*].*CbCtrl*.**Unique**[= *setting%*]

VC++ Usage: short *m_CbCtrl*.**GetUnique( )**
void *m_CbCtrl*.**SetUnique(** short *setting***)**

Delphi Usage: [*form*].*CbCtrl*.**Unique**[:= *setting:* short]

C++ Builder Usage: [*form*]->*CbCtrl*->**Unique**[= short *setting*]

Applicable To: **CbMaster**

Description: This property determines how the **CbMaster** treats duplicate key entries for unique tags. This property only has an effect on unique tags.

If Unique is set to 0, then a duplicate key can be added to the data file. The tag only contains a reference to the first record in the data file that contains the key. This is the default behavior.

The DuplicateKey event is fired when an attempt is made to write a duplicate key to a unique tag and the Unique property is set to 1. The programmer may intercept this event or a default dialog box is displayed, indicating that a duplicate key was detected for a unique tag. Refer to the "Events" chapter for more information on the DuplicateKey event.

Call RefreshFiles after changing the Unique property at runtime. The RefreshFiles closes the data file and index files, and then reopens the data file and index files according to the new Unique setting.

This property is read/write at design time and runtime.

See Also: **RefreshFiles** method, **DuplicateKey** Event

UseAltBitmaps Property

VB Usage: [*form*].*CbCtrl*.**UseAltBitmaps**[= {**TRUE** | **FALSE**}]

VC++ Usage: BOOL *m_CbCtrl*.**GetUseAltBitmaps( )**
void *m_CbCtrl*.**SetUseAltBitmaps(** BOOL *setting***)**

Delphi Usage: [*form*].*CbCtrl*.**UseAltBitmaps**[:= {**TRUE** | **FALSE**}]

C++ Builder Usage: [*form*]->*CbCtrl*->**UseAltBitmaps**[= {**TRUE** | **FALSE**}]

Applicable To: **CbButton**

Description: When this property is set to true, the alternative set of custom or default bitmaps are displayed by the **CbButton**. Make sure that the UseBitmaps property is set to true, so that bitmaps are displayed by the **CbButton**. The DefaultBitmaps property determines which alternative bitmaps are displayed.

By default, this property is set to false, which means that the main set of custom or default bitmaps are used by the **CbButton**.

This property is read/write at design time and runtime.

See Also: **DefaultBitmaps**, **UseBitmaps** properties

UseBitmaps Property

VB Usage: [*form*].*CbCtrl*.**UseBitmaps**[= {**TRUE** | **FALSE**}]

VC++ Usage: BOOL *m_CbCtrl*.**GetUseBitmaps( )**
void *m_CbCtrl*.**SetUseBitmaps(** BOOL *setting***)**

Delphi Usage: [*form*].*CbCtrl*.**UseBitmaps**[:= {**TRUE** | **FALSE**}]

C++ Builder Usage: [*form*]->*CbCtrl*->**UseBitmaps**[= {**TRUE** | **FALSE**}]

Applicable To: **CbButton**

Description: This property determines whether the **CbButton** uses bitmaps for decoration as specified by the DefaultBitmaps property.

If UseBitmaps is set to false, then a caption may be displayed. The caption text is specified by the Caption property and the CaptionVisible property is used to determine whether the caption should be displayed.

When the UseBitmaps property is set to true, bitmaps are displayed by the **CbButton**. Be sure to set the CaptionVisible property to false, so that the Caption is not displayed at the same time as the bitmaps.

UseBitmaps is set to false, by default.

This property is read/write at design time and runtime.

See Also: **DefaultBitmaps, UseAltBitmaps, Caption, CaptionVisible** properties

UserName Property

VB Usage: [*form*].*CbCtrl*.**UserName**[= *name$*]

VC++ Usage: CString *m_CbCtrl*.**GetUserName( )**
void *m_CbCtrl*.**SetUserName(** LPCTSTR *name***)**

Delphi Usage: [*form*].*CbCtrl*.**UserName**[:= *name*: string]

C++ Builder Usage: [*form*]->*CbCtrl*->**UserName**[= AnsiString *name*]

Applicable To: **CbMaster**

Description: This property is only used when the CodeBase client/server configuration is used with CodeControls 3.0. The UserName property sets or returns the name of the registered user who is trying to connect to the server. The UserName may be set to an empty string ("") if the server does not use name authorizations.

By default, the UserName property is set to "PUBLIC", which will allow public access to the server.

Currently, CodeBase does not support user name and password verification. This feature may be added in future versions of CodeBase. The UserName property is currently used by CodeBase transactions to identify which client made changes to the data files.

The server should be running at design time as well as runtime because the **CbMaster** opens and closes data files to access field and tag information. For this reason, the ServerId, ProcessId, UserName and Password properties should be set before the DatabaseName and IndexFiles properties are specified.

Call the RefreshFiles method after changing this property at runtime. The RefreshFiles method will cause the **CbMaster** to reconnect to a server with the new user name and then reopen the data file.

This property is read/write at design time and runtime.

Refer to the CodeBase *Getting Started* manual for more information on the UserName.

See Also: **ServerId, Password, ProcessId** properties, **RefreshFiles** method

Value Property

VB Usage: [*form*].*CbCtrl*.**Value**[= *value#*]

VC++ Usage: double *m_CbCtrl*.**GetValue( )**
void *m_CbCtrl***SetValue(** double *value* **)**

Delphi Usage: [*form*].*CbCtrl*.**Value**[:= *position:* Double]

C++ Builder Usage: [*form*]->*CbCtrl*->**Value**[= double *position*]

Applicable To: **CbSlider**

Description: This property sets and returns the thumb position on the **CbSlider** control. Value lies between the MinValue and the MaxValue. For example, if MinValue was 0 and MaxValue was 20 and the thumb positioned in the middle of the slider, then Value would return 10.

If the **CbSlider** is linked to a data file, Value represents a percentage and the record closest to that percentage becomes the current record. Value lies between 0 and 1 regardless of MinValue and MaxValue properties. For example, if there were 100 records in a data file and Value was set to 0.5 then the record 50 would become the current record.

This property is read/write at run time only.

See Also: **MinValue, MaxValue, ScrollUnit** properties

Visibility Property

VB Usage: [*form*].*CbCtrl*.**Visibility**[= {**TRUE** | **FALSE**}]

VC++ Usage: BOOL *m_CbCtrl*.**GetVisibility( )**
void *m_CbCtrl*.**SetVisibility(** BOOL *setting***)**

Delphi Usage: [*form*].*CbCtrl*.**Visibility**[:= {**TRUE** | **FALSE**}]

C++ Builder Usage: [*form*]->*CbCtrl*->**Visibility**[= {**TRUE** | **FALSE**}]

Applicable To: **CbMaster, CbButton, CbCombo, CbEdit, CbList, CbSlider**

Description: This property determines whether the control is visible at runtime.

By default, the Visibility property is set to true, which means that the control is visible at runtime. When the property is set to false, the controls is invisible at runtime.

An invisible control can still be manipulated by its properties and methods. The **CbMaster** can open and manipulate a data file while invisible and other controls may be bound to an invisible **CbMaster**. Invisible controls may be bound to a **CbMaster**.

A control remains visible at design time for development purposes regardless of the setting of this property.

This property is read/write at design time and runtime.

NOTE

Visual Basic, Delphi and C++ Builder automatically provide a Visible property for each OLE control. Do NOT use this property if you want a control to be invisible at runtime. The default Visible property causes problems for the controls at runtime. Instead, use the Visibility property that is provided for each control in CodeControls.

6 Methods

Methods

The following reference lists all of the methods that are unique to CodeControls. The syntax given in the Usage statements illustrate how to call a method in a program written under Visual Basic, Visual C++, Delphi or Borland C++ Builder. This is not the syntax that would appear in a header file. The italicized part of the Usage statements represent the name of a particular instance of the control. Any part of the Usage statement in square brackets is optional.

CbMaster Methods			
Append	GetFieldNumber	RefreshRecord	SeekNextDouble
Bottom	GetFieldDecimals	Reindex	SeekNextN
Delete	GetFieldLength	Search	SetField
FlushRecord	GetNumberFields	Seek	Skip
GetField	Go	SeekDouble	Top
GetFieldName	Pack	SeekN	Undo
GetFieldType	RefreshFiles	SeekNext	Unlink

CbList Methods	
AddItem	Clear
Refresh	RemoveItem
GetSelected	SetSelected

CbCombo Methods	
AddItem	RemoveItem
Refresh	ValidDate
Clear	

CbButton Method
DoClick

CbEdit Method
ValidDate

Default Method provided by the compilers
SetFocus

AddItem Method

VB Usage: [*form*].*CbCtrl*.**AddItem(** *item$, index&***)**

VC++ Usage: *m_CbCtrl*.**AddItem(**LPCTSTR *item,* long *index***)**

Delphi Usage: [*form*].*CbCtrl*.**AddItem**(*item*: string, *index*: Integer)

C++ Builder Usage: [*form*]->*CbCtrl*->**AddItem**(AnsiString *item,* int *index*)

Applicable To: **CbList, CbCombo**

Description: This method adds an item to the **CbList** or the list portion of the **CbCombo** at the position specified by *index*.

AddItem may not add items to a list when the **CbList** or list portion of the **CbCombo** is linked to a data file. One exception to this rule, is when the **CbList** is linked to a data field and the DataAction property is set to Field Value (2). See the DataAction property for more information.

Parameters:

item *item* is the string to be added to the list box.

index *index* is the zero-based index of the position in the list box where the item is to be inserted. If *index* is set to -1 and the Sorted property is true, then the item is inserted in the list and the list is sorted. If index is set to -1 and the Sorted property is false, the item is inserted at the end of the list. If the *index* is 0 or greater, the item is added to specified position regardless of the setting of the Sorted property.

A list box under Windows 95 has a limit of 32,767 items, so *index* must be less than 32,767.

See Also: **DataAction** property

Append Method

VB Usage: *rc%* = [*form*].*CbCtrl*.**Append**

VC++ Usage: short *rc* = *m_CbCtrl*.**Append( )**

Delphi Usage: *rc:* Smallint **:=** [*form*].*CbCtrl*.**Append**

C++ Builder Usage: short *rc* = [*form*]->*CbCtrl*->**Append**( )

Applicable To: **CbMaster**

Description: When this method is called, the **CbMaster** appends a record to the data file. The type of record that is used for the append depends on the AppendRecordType property. The append may be aborted at this point by calling the Undo method or pressing the **CbMaster**'s Undo button. The record is not actually appended to the data file until after the new record is flushed explicitly or the data file is repositioned to a valid record.

The Append method flushes any changes to current record before appending the record.

This method fires a Validate event. If the event is intercepted and the response is set to true, the Append method is aborted. By default, the response is set to false and the Append method is allowed to continue.

Returns: This method returns 0 if it successfully appends a record to the data file.

The current record needs to be locked before it can be flushed. If another application has locked the record, then a NoUpdate event is fired. If the changes are discarded and the append continues, this method returns 0. If the append is aborted in order to keep the changes, this method will return -1.

A DuplicateKey event is fired when the Unique property is set to 1 and flushing the current record would result in a duplicate key for a unique tag. If the changes are discarded and the append continues, this method returns 0. If the append is aborted in order to keep the changes, this method will return -1.

This method also returns -1 if the **DATA4** structure is not valid or if the Validate event response is set to true.

See Also: **AppendRecordType** property**, NoUpdate, Validate DuplicateKey** events

Bottom Method

VB Usage: *rc%* = [*form*].*CbCtrl*.**Bottom**

VC++ Usage: *short rc = m_CbCtrl*.**Bottom( )**

Delphi Usage: *rc*: Smallint **:=** [*form*].*CbCtrl*.**Bottom**

C++ Builder Usage: short *rc* = [*form*]->*CbCtrl*->**Bottom**( )

Applicable To: **CbMaster**

Description: When this method is called, the **CbMaster** repositions the data file to the last record in the data file according to the selected tag. If there is no selected tag the data file is repositioned to the last physical record in the data file. The **CbMaster** also signals all of its bound controls to be repositioned accordingly.

This method fires a Validate event. If the event is intercepted and the response is set to true, the Bottom method is aborted. By default, the response is set to false and the Bottom method is allowed to continue.

Returns: This method returns 0 if it was successful in repositioning the data file to the last record according to the selected tag.

Before the data file can be repositioned, any changes made to the current record must be flushed. If the files could not be flushed because they were locked by another user then a NoUpdate event is fired. If the changes are discarded and the file reposition continues, this method returns 0. If the reposition is aborted in order to keep the changes, this method will return -1.

A DuplicateKey event is fired when the Unique property is set to 1 and flushing the current record would result in a duplicate key for a unique tag. If the changes are discarded and the file reposition continues, this method returns 0. If the reposition is aborted in order to keep the changes, this method will return -1.

This method also returns -1 if the **DATA4** structure is not valid or if the Validate event response is set to true.

See Also: **NoUpdate, Validate DuplicateKey** events

Clear Method

VB Usage: [*form*].*CbCtrl*.**Clear**

VC++ Usage: *m_CbCtrl*.**Clear( )**

Delphi Usage: [*form*].*CbCtrl*.**Clear**

C++ Builder Usage: [*form*]->*CbCtrl*->**Clear( )**

Applicable To: **CbList, CbCombo**

Description: This method deletes all of the items from a **CbList** or the list portion of a **CbCombo**.

Clear is not supported when the **CbList** or list portion of the **CbCombo** is linked to a data file. One exception to this rule, is when the **CbList** is linked to a data field and the DataAction property is set to Field Value (2). In this case, the items may be deleted by Clear. See the DataAction property for more information.

See Also: **DataAction** Property

Delete Method

VB Usage: rc% = [*form*].*CbCtrl*.**Delete(**{**TRUE** | **FALSE**}**)**

VC++ Usage: short *rc* = *m_CbCtrl*.**Delete(**BOOL *setting***)**

Delphi Usage: *rc*: Smallint := [*form*].*CbCtrl*.**Delete**(*setting:* Boolean)

C++ Builder Usage: short *rc* = [*form*]->*CbCtrl*->**Delete**(bool *setting*)

Applicable To: **CbMaster**

Description: When this method is called, the **CbMaster** alters the deletion flag for the current record according to *setting*. When *setting* is false, the deletion mark is removed from the current record. When *setting* is true, the current record is marked for deletion. This method does not physically remove the record from the data file. The records that are marked for deletion are physically removed when the data file is packed. The data file can be packed by calling the Pack method or by pressing the Pack button on the **CbMaster**.

Note that the deletion flag for the current record may also be altered by using the **CbMaster** Deleted property.

Returns: This method returns -1 if the **DATA4** structure is not valid. Otherwise, Delete returns 0.

See Also: **Deleted** property**, Pack** method

DoClick Method

VB Usage: [*form*].*CbCtrl*.**DoClick**

VC++ Usage: *m_CbCtrl*.**DoClick( )**

Delphi Usage: [*form*].*CbCtrl*.**DoClick**

C++ Builder Usage: [*form*]->*CbCtrl*->**DoClick**( )

Applicable To: **CbButton**

Description: This method emulates a mouse click on a **CbButton**. When this method is called, the action that occurs is the same action that occurs when a mouse clicks the **CbButton**. In fact, a Click event is also fired when the DoClick method is called. The default action of the **CbButton** is determined by the DataAction property.

See Also: **DataAction** property, **Click** event

FlushRecord Method

VB Usage: *rc*% = [*form*].*CbCtrl*.**FlushRecord**

VC++ Usage: short *rc* = *m_CbCtrl*.**FlushRecord( )**

Delphi Usage: *rc*: Smallint **:=** [*form*].*CbCtrl*.**FlushRecord**

C++ Builder Usage: short *rc* **=** [*form*]->*CbCtrl*->**FlushRecord**( )

Applicable To: **CbMaster**

Description: Call this **CbMaster** method when you want to explicitly flush the changes to the current record. FlushRecord also updates the index files and memo files associated with the data file. If **CbMaster** is in append mode, FlushRecord causes the record to be appended to the end data file and this new record becomes the current record.

This method fires a Validate event. If the event is intercepted and the response is set to true, the FlushRecord method is aborted. By default, the response is set to false and the FlushRecord method is allowed to continue.

Returns: This method returns 0 if it was successful in flushing the changes.

If the files could not be flushed because they were locked by another user then a NoUpdate event is fired. If the changes are discarded, this method returns 0. If the flush is aborted in order to keep the changes, this method will return -1.

A DuplicateKey event is fired when the Unique property is set to 1 and flushing the current record would result in a duplicate key for a unique tag. If the changes are discarded, this method returns 0. If the flush is aborted in order to keep the changes, this method will return -1.

This method also returns -1 if the **DATA4** structure is not valid or if the Validate event response is set to true.

See Also: **NoUpdate, Validate, DuplicateKey** events

GetField Method

VB Usage: *rc%* = [*form*].*CbCtrl*.**GetField(** *fieldName$, fieldValue* As Variant**)**

VC++ Usage: short *rc* = *m_CbCtrl*.**GetField(**LPCTSTR *fieldName,*
VARIANT FAR* *fieldValue***)**

Delphi Usage: *rc:* Smallint **:=** [*form*].*CbCtrl*.**GetField(** *fieldName*: string,
fieldValue: Variant)

C++ Builder Usage: short *rc* = [*form*]->*CbCtrl*->**GetField(** AnsiString *fieldName,*
Variant *fieldValue*)

Applicable To: **CbMaster**

Description: This method returns the contents of a field from the current record in the *fieldValue* parameter. The type of Variant parameter passed in determines what form the field value is returned.

Parameters: *fieldName* is a string containing the name of a field. The *fieldValue* parameter stores the contents of field from the current record.

Returns: This method returns -1 if the field is empty or if there was an error. Otherwise it returns 0.

See Also: **SetField** method

GetFieldName Method

VB Usage: *fieldName$* = [*form*].*CbCtrl*.**GetFieldName(** *fieldNumber%* **)**

VC++ Usage: CString *fieldName* = *m_CbCtrl*.**GetFieldName(** short *fieldNumber* **)**

Delphi Usage: *fieldName:* string **:=** [*form*].*CbCtrl*.**GetFieldName**(*fieldNumber*: Smallint)

C++ Builder Usage: AnsiString *fieldName* **=** [*form*]->*CbCtrl*->**GetFieldName**(short *fieldNumber*)

Applicable To: **CbMaster**

Description: This method returns the name of the field in the *fieldNumber*th position in the record.

Parameters: *fieldNumber* is a number between 1 and the number of fields in the data file.

Returns: This method returns an empty string ("") if the *fieldNumber* is not valid, otherwise it returns a null-terminated string containing the field name.

GetFieldType Method

VB Usage: *fieldType%* = [*form*].*CbCtrl*.**GetFieldType(** *fieldName$***)**

VC++ Usage: short *fieldType* = *m_CbCtrl*.**GetFieldType(**LPCTSTR *fieldName***)**

Delphi Usage: *fieldType:* Smallint **:=** [*form*].*CbCtrl*.**GetFieldType**(*fieldName*: string)

C++ Builder Usage: short *fieldType* **=** [*form*]->*CbCtrl*->**GetFieldType**(AnsiString *fieldName*)

Applicable To: **CbMaster**

Description: This method returns the type of field specified by *fieldName*.

Parameters: *fieldName* is a string containing the name of a field.

Returns: Note that the integer returned under Visual Basic is the ASCII value of the characters listed below. Use the Visual Basic function Chr(*asciiVal*) to convert the *fieldType* into a character.

'B' Binary Field

'C' Character Field

'D' Date Field

'F' Floating Point Field

'G' General Field

'L' Logical Field

'M' Memo Field

'N' Numeric or Floating Point Field

If the field specified by *fieldName* does not exist in the data file then -1 is returned.

GetFieldNumber Method

VB Usage: *fieldNum%* = [*form*].*CbCtrl*.**GetFieldNumber(** *fieldName$***)**

VC++ Usage: short *fieldNum* = *m_CbCtrl*.**GetFieldNumber(** LPCTSTR *fieldName* **)**

Delphi Usage: *fieldNum:* Smallint **:=** [*form*].*CbCtrl*.**GetFieldNumber**(*fieldName*: string)

C++ Builder Usage: short *fieldNum* **=** [*form*]->*CbCtrl*->**GetFieldNumber**(AnsiString *fieldName*)

Applicable To: **CbMaster**

Description: A search is made for the specified field and its position in the data file is returned.

Parameters: *fieldName* is a string containing the name of a field.

Returns: The field number returned is a number between 1 and the number of fields in the data file. If the field specified by *fieldName* does not exist in the data file then -1 is returned.

GetFieldDecimals Method

VB Usage: *numDec%* = [*form*].*CbCtrl*.**GetFieldDecimals(** *fieldName$***)**

VC++ Usage: short *numDec* = *m_CbCtrl*.**GetFieldDecimals(** LPCTSTR *fieldName* **)**

Delphi Usage: *numDec:* Smallint **:=**[*form*].*CbCtrl*.**GetFieldDecimals**(*fieldName*: string)

C++ Builder Usage: short *numDec* **=** [*form*]->*CbCtrl*->**GetFieldDecimals**(AnsiString *fieldName*)

Applicable To: **CbMaster**

Description: This method returns the number of decimals in a field.

Parameters: *fieldName* is a string containing the name of a field.

Returns: This method always returns 0 for any field type other that Numeric or Floating Point. If the field specified by *fieldName* does not exist in the data file then -1 is returned.

GetFieldLength Method

VB Usage: *fieldLen&* = [*form*].*CbCtrl*.**GetFieldLength(** *fieldName$***)**

VC++ Usage: long *fieldLen* = *m_CbCtrl*.**GetFieldLength(** LPCTSTR *fieldName* **)**

Delphi Usage: *fieldLen:* Integer **:=** [*form*].*CbCtrl*.**GetFieldLength**(*fieldName*: string)

C++ Builder Usage: int *fieldLen* **=** [*form*]->*CbCtrl*->**GetFieldLength**(AnsiString *fieldName*)

Applicable To: **CbMaster**

Description: This method returns the length of a field.

Parameters: *fieldName* is a string containing the name of a field.

Returns: This method returns the length of the field as it was specified when the data file was originally created. If the field specified by *fieldName* does not exist in the data file then -1 is returned.

GetNumberFields Method

VB Usage: *rc%* = [*form*].*CbCtrl*.**GetNumberFields**

VC++ Usage: short *rc* = *m_CbCtrl*.**GetNumberFields( )**

Delphi Usage: *rc:* Smallint **:=** [*form*].*CbCtrl*.**GetNumberFields**

C++ Builder Usage: short *rc* = [*form*]->*CbCtrl*->**GetNumberFields( )**

Applicable To: **CbMaster**

Description: This method returns the number of fields in the data file linked to the **CbMaster** control.

Returns: This method returns -1 if an error occurred.

GetSelected Method

VB Usage: *selected* As Boolean = [*form*].*CbCtrl*..**GetSelected(** *index&***)**

VC++ Usage: BOOL *selected* = *m_CbCtrl*.**GetSelected(** long *index***)**

Delphi Usage: *selected:* Boolean := [*form*].*CbCtrl*.**GetSelected**(*index*: Integer)

C++ Builder Usage: bool *selected* **=** [*form*]->*CbCtrl*->**GetSelected**(int *index*)

Applicable To: **CbList**

Description: This method returns whether an item in the position specified by *index* is selected or not. If the **CbList** is linked to a data file and the DataAction property is set to Record Position (1), the GetSelected returns whether a particular record is selected.

This method is useful for a multiple-selection **CbList** where the MultiSelect property set to Simple Selection (1) or Extended Selection (2). The property ListIndex will return the index of the selected item in single-selection list boxes. In the case of multiple-selection list boxes, ListIndex returns the index of the item with the focus, which may or may not have a selection. GetSelected can be used with multiple-selection list boxes to determine quickly which items are selected.

Parameters:

index *index* is the zero-based index of the position of the item in the list box.

A list box under Windows 95 has a limit of 32,767 items, so *index* must be less than 32,767 when the **CbList** is not linked to a data file or when the **CbList** is linked to a data file and the DataAction property is set to Field Value (2).

If the **CbList** is linked to a data file and the DataAction property is set to Record Position (1), the *index* represents a record number. In this case, *index* can range from 0 to the number of records in the data file.

Returns: True is returned when the item in position *index* is selected, otherwise false is returned.

See Also: **SetSelected** method**, ListIndex, MultiSelect, DataAction** property

Go Method

VB Usage: *rc%* = [*form*].*CbCtrl*.**Go(** *recordNumber&***)**

VC++ Usage: short *rc* = *m_CbCtrl*.**Go(** long *recordNumber* **)**

Delphi Usage: *rc:* Smallint **:=** [*form*].*CbCtrl*.**Go**(*recordNumber*: Integer)

C++ Builder Usage: short *rc* **=** [*form*]->*CbCtrl*->**Go**(int *recordNumber*)

Applicable To: **CbMaster**

Description: When this method is called, the **CbMaster** repositions the data file to the record specified by *recordNumber*. The **CbMaster** also signals all of its bound controls to be repositioned accordingly.

This method fires a Validate event. If the event is intercepted and the response is set to true, the Go method is aborted. By default, the response is set to false and the Go method is allowed to continue.

Parameters: *recordNumber* is a long integer that contains a record number. This number must be between 1 and the number of records in the data file.

Returns: This method returns 0 if it was successful in repositioning the data file to the specified record.

Before the data file can be repositioned, any changes made to the current record must be flushed. If the files could not be flushed because they were locked by another user then a NoUpdate event is fired. If the changes are discarded and the file reposition continues, this method returns 0. If the reposition is aborted in order to keep the changes, this method will return -1.

A DuplicateKey event is fired when the Unique property is set to 1 and flushing the current record would result in a duplicate key for a unique tag. If the changes are discarded and the file reposition continues, this method returns 0. If the reposition is aborted in order to keep the changes, this method will return -1.

If no record exists that corresponds to the *recordNumber* parameter, this method returns 5, which is equivalent to the CodeBase return code of r4entry. In this case, the data file is not repositioned and it remains on the previously valid record.

This method also returns -1 if the **DATA4** structure is not valid or if the Validate event response is set to true.

See Also: **NoUpdate, Validate, DuplicateKey** events

Pack Method

VB Usage: *rc%* = [*form*].*CbCtrl*.**Pack**

VC++ Usage: short *rc* = *m_CbCtrl*.**Pack( )**

Delphi Usage: *rc*: Smallint **: =** [*form*].*CbCtrl*.**Pack**

C++ Builder Usage: short *rc* = [*form*]->*CbCtrl*->**Pack**( )

Applicable To: **CbMaster**

Description: This method packs the data file. Packing a data file physically removes all of the records marked for deletion. The Pack method also automatically reindexes any open index files associated with the data file. After the packing is finished, the data file is repositioned to the top.

This method can only be executed when the data file is opened exclusively. The data file will be opened exclusively if the AccessMode is set to Deny All (2) at design time. The data file can be opened exclusively at runtime by setting the AccessMode to Deny All (2) and then re-opening the data file by calling the RefreshFiles method.

Returns: This method returns 0 when it successfully packs the data file. If the data file is not opened exclusively then the pack does not occur and the method returns -1.

See Also: **Delete, RefreshFiles** method**, Deleted, AccessMode** property

Refresh Method

VB Usage: [*form*].*CbCtrl*.**Refresh**

VC++ Usage: *m_CbCtrl*.**Refresh( )**

Delphi Usage: [*form*].*CbCtrl*.**Refresh**

C++ Builder Usage: [*form*]->*CbCtrl*->**Refresh( )**

Applicable To: **CbList, CbCombo**

Description: This method forces the **CbList** or **CbCombo** controls to be repainted.

RefreshFiles Method

VB Usage: rc% = [*form*].*CbCtrl*.**RefreshFiles**

VC++ Usage: short *m_CbCtrl*.**RefreshFiles( )**

Delphi Usage: Smallint [*form*].*CbCtrl*.**RefreshFiles**

C++ Builder Usage: short [*form*]->*CbCtrl*->**RefreshFiles( )**

Applicable To: **CbMaster**

Description: This method refreshes the data file by closing it and reopening it. After the data file is re-opened, the data file is positioned according to the response from the DatafileOpened event. The **CbMaster** also updates its bound controls accordingly.

The DatafileClosed event is fired after the current data file is closed. The DatafileOpened event is fired after the data file is opened.

Call this method after the DatabaseName and IndexFiles properties for the **CbMaster** have been changed at runtime. RefreshFiles closes the current data file and then opens the new data file specified by the DatabaseName. An error will occur if the field names or expression used by the bound controls are no longer valid for the new data file. No error will occur if the DatabaseName has been set to a null string ("").

RefreshFiles should also be called after the following properties are changed at runtime: AccessMode, OptimizeRead, OptimizeWrite and Unique.

If the client/server configuration of CodeBase is being used with CodeControls, then call RefreshFiles after any of the following properties have been changed at runtime: ServerId, ProcessId, Password, UserName. RefreshFiles will disconnect from the current server and then reconnect to the server according to the new settings.

Returns: This method returns 0 when it successfully refreshes the data file. If the current data file could not be closed then RefreshFiles returns -1 and the current data file remains open and valid.

See Also: **DatafileOpened, DatafileClosed** events**, DatabaseName, IndexFiles, ServerId, ProcessId, Password, UserName** properties

RefreshRecord Method

VB Usage: *rc%* = [*form*].*CbCtrl*.**RefreshRecord**

VC++ Usage: short *rc* = *m_CbCtrl*.**RefreshRecord( )**

Delphi Usage: *rc*: Smallint **:=** [*form*].*CbCtrl*.**RefreshRecord**

C++ Builder Usage: short *rc* = [*form*]->*CbCtrl*->**RefreshRecord**( )

Applicable To: **CbMaster**

Description: This method refreshes the current record by reloading it from disk.

In multi-user situations, when the current record is not locked, it is possible for the record to be modified by another user. Calling this method causes the **CbMaster** to read the record directly from disk again, possibly returning a more up-to-date version of the record. When method is called, any changes made to the present copy of the current record are lost.

Bound controls are automatically updated.

Returns: This method returns 0, unless the **CbMaster** is in append mode, in which case it returns -1.

Reindex Method

VB Usage: *rc%* = [*form*].*CbCtrl*.**Reindex**

VC++ Usage: short *rc* = *m_CbCtrl*.**Reindex( )**

Delphi Usage: *rc*: Smallint **:=** [*form*].*CbCtrl*.**Reindex**

C++ Builder Usage: short *rc* **=** [*form*]->*CbCtrl*->**Reindex( )**

Applicable To: **CbMaster**

Description: This method causes all of the open index files that are associated with the data file to be recreated. After the reindexing is finished, the data file is repositioned to the top.

This method can only be executed when the data file is opened exclusively. The data file will be opened exclusively if the AccessMode is set to Deny All (2) at design time. The data file can be opened exclusively at runtime by setting the AccessMode to Deny All (2) and then re-opening the data file by calling the RefreshFiles method.

Returns: This method returns 0 when it successfully reindexes the data file. If the data file is not opened exclusively then the reindex does not occur and the method returns -1.

See Also: **AccessMode** property**, RefreshFiles** method

RemoveItem Method

VB Usage: [*form*].*CbCtrl*.**RemoveItem(** *index&***)**

VC++ Usage: *m_CbCtrl*.**RemoveItem(** long *index***)**

Delphi Usage: [*form*].*CbCtrl*.**RemoveItem**(*index*: Integer)

C++ Builder Usage: [*form*]->*CbCtrl*->**RemoveItem(** int *index*)

Applicable To: **CbList, CbCombo**

Description: This method removes the item at the position specified by *index* from a **CbList** or the list portion of the **CbCombo**.

RemoveItem may not remove items from a list when the **CbList** or list portion of the **CbCombo** is linked to a data file. One exception to this rule, is when the **CbList** is linked to a data field and the DataAction property is set to Field Value (2). See the DataAction property for more information.

Parameters:

index *index* is the zero-based index of the item in the list box to be removed.

A list box under Windows 95 has a limit of 32,767 items, so *index* must be less than 32,767.

See Also: **DataAction** property

Search Method

VB Usage: *rc%* = [*form*].*CbCtrl*.**Search**

VC++ Usage: short *rc* = *m_CbCtrl*.**Search( )**

Delphi Usage: *rc:* Smallint **=** [*form*].*CbCtrl*.**Search**( )

C++ Builder Usage: short *rc* **=** [*form*]->*CbCtrl*->**Search(**)

Applicable To: **CbMaster**

Description: This method opens the search dialog box. This is equivalent to pressing the search button on the **CbMaster**. See the "Control Description" chapter for more information on the how the search dialog box works.

This method fires a Validate event. If the event is intercepted and the response is set to true, the Search method is aborted. By default, the response is set to false and the Search method is allowed to continue.

Returns: This method also returns -1 if the **DATA4** structure is not valid. Otherwise, this method returns 0.

Seek Method

VB Usage: *rc%* = [*form*].*CbCtrl*.**Seek(** *seekStr$* **)**

VC++ Usage: short *rc* = *m_CbCtrl*.**Seek(** LPCTSTR *seekStr* **)**

Delphi Usage: *rc*: Smallint **:=** [*form*].*CbCtrl*.**Seek**(*seekStr*: string)

C++ Builder Usage: short *rc* **=** [*form*]->*CbCtrl*->**Seek(** AnsiString *seekStr*)

Applicable To: **CbMaster**

Description: This method searches for the first record in the selected tag that matches the string in *seekStr*. Once the search value is located in the tag, the corresponding data file record becomes the current record and the bound controls are updated accordingly. Searching always begins from the first logical record in the database as determined by the selected tag.

The Seek method may be used with any tag type, as long as the *seekStr* is formatted correctly. Seeking on memo fields is not allowed.

This method fires a Validate event. If the event is intercepted and the response is set to true, the Seek method is aborted. By default, the response is set to false and the Seek method is allowed to continue.

Parameters: *seekStr* is a string containing the value for which the search is conducted.

If the tag is of type Date, the search value should be in standard format: "CCYYMMDD" (Century, Year, Month, Day).

If the tag is of type Character, the *seekStr* may have fewer characters than the tag's key length, provided that the search value is null terminated. In this case, a search is done for the first key which matches the supplied characters. If *seekStr* contains data longer than the tag key length, the extra characters are ignored.

If the tag is of type Numeric, the character value should represent a valid number. e.g. *CbMaster*.Seek("33.7"). This number is internally converted to a double value before the seek is performed.

Returns: This method returns 0 if the Seek found an exact match to the search key and the data file is repositioned to that record.

If the search value was not found, then the data file is positioned to the record after the position where the search key would have been located had it existed. The Seek returns 2, which is equivalent to the CodeBase return code of r4after.

If the search value was not found and the search value was greater than the last key of the tag then the Seek creates an end of file condition. The action that takes place when an end of file condition is created depends on the value of the EOFAction property. In this case, the Seek returns 3, which is equal to the CodeBase return code of r4eof.

The seek can not be accomplished without a tag. If there is no tag associated with the data file, this method returns 80, which is equivalent to the CodeBase return code of r4noTag.

It may be possible that the Seek will attempt to reposition the data file to a non-existent record. In this case, the method returns 5, which is equivalent to the CodeBase return code of r4entry. Under these circumstances, the data file is not repositioned.

Before the data file can be repositioned, any changes made to the current record must be flushed. If the files could not be flushed because they were locked by another user then a NoUpdate event is fired. If the changes are discarded and the file reposition continues, this method returns 0. If the reposition is aborted in order to keep the changes, this method will return -1.

A DuplicateKey event is fired when the Unique property is set to 1 and flushing the current record would result in a duplicate key for a unique tag. If the changes are discarded and the file reposition continues, this method returns 0. If the reposition is aborted in order to keep the changes, this method will return -1.

This method also returns -1 if the **DATA4** structure is not valid or if the Validate event response is set to true.

See Also: **Validate**, **NoUpdate and DuplicateKey** events

SeekDouble Method

VB Usage: *rc%* = [*form*].*CbCtrl*.**SeekDouble(** *seekVal#* **)**

VC++ Usage: short *rc* = *m_CbCtrl*.**SeekDouble(** double *seekVal* **)**

Delphi Usage: *rc*: Smallint **:=** [*form*].*CbCtrl*.**SeekDouble**(*seekVal*: Double)

C++ Builder Usage: short *rc* **=** [*form*]->*CbCtrl*->**SeekDouble(** double *seekVal*)

Applicable To: **CbMaster**

Description: This method searches for the first record in the selected tag that matches the value in *seekVal*. Once the search value is located in the tag, the corresponding record becomes the current record and the bound controls are updated accordingly. Searching always begins from the first logical record in the database as determined by the selected tag.

The SeekDouble method must use a tag key of type Numeric or Date.

This method fires a Validate event. If the event is intercepted and the response is set to true, the SeekDouble method is aborted. By default, the response is set to false and the SeekDouble method is allowed to continue.

Parameters: *seekVal* is a double value. If the key is of type Date, *seekVal* should represent a Julian day.

Returns: This method returns 0 if the SeekDouble found an exact match to the search key and the data file is repositioned to that record.

If the search value was not found, then the data file is positioned to the record after the position where the search key would have been located had it existed. The SeekDouble returns 2, which is equivalent to the CodeBase return code of r4after.

If the search value was not found and the search value was greater than the last key of the tag then the SeekDouble creates an end of file condition. The action that takes place when an end of file condition is created depends on the value of the EOFAction property. In this case, the SeekDouble returns 3, which is equal to the CodeBase return code of r4eof.

The seek can not be accomplished without a tag. If there is no tag associated with the data file, this method returns 80, which is equivalent to the CodeBase return code of r4noTag.

It may be possible that the SeekDouble will attempt to reposition the data file to a non-existent record. In this case, the method returns 5, which is equivalent to the CodeBase return code of r4entry. Under these circumstances, the data file is not repositioned.

Before the data file can be repositioned, any changes made to the current record must be flushed. If the files could not be flushed because they were locked by another user then a NoUpdate event is fired. If the changes are discarded and the file reposition continues, this method returns 0. If the reposition is aborted in order to keep the changes, this method will return -1.

A DuplicateKey event is fired when the Unique property is set to 1 and flushing the current record would result in a duplicate key for a unique tag. If the changes are discarded and the file reposition continues, this method returns 0. If the reposition is aborted in order to keep the changes, this method will return -1.

This method also returns -1 if the **DATA4** structure is not valid or if the Validate event response is set to true.

See Also: **Validate**, **NoUpdate and DuplicateKey** events

SeekN Method

VB Usage: *rc%* = [*form*].*CbCtrl*.**SeekN(** *seekStr$, length%* **)**

VC++ Usage: short *rc* = *m_CbCtrl*.**SeekN(** LPCTSTR *seekStr,* short *length* **)**

Delphi Usage: *rc*: Smallint **:=** [*form*].*CbCtrl*.**SeekN**(*seekStr*: string, *length*: Smallint)

C++ Builder Usage: short *rc* = [*form*]->*CbCtrl*->**SeekN**(AnsiString *seekStr,* short *length*)

Applicable To: **CbMaster**

Description: This method searches for the first record in the selected tag that matches the string in *seekStr*. Once the search value is located in the tag, the corresponding record becomes the current record and the bound controls are updated accordingly. Searching always begins from the first logical record in the database as determined by the selected tag.

If a character field is composed of binary data, SeekN may be used to seek without regard for nulls because the length of the key is specified. Seeking on memo fields is not allowed.

This method fires a Validate event. If the event is intercepted and the response is set to true, the SeekN method is aborted. By default, the response is set to false and the SeekN method is allowed to continue.

Parameters:

seekStr *seekStr* is a string that contains the value for which the search is conducted.

length This is the length of the data contained in *seekStr*. This should be less than or equal to the length of the key size for the selected tag. If *length* is greater than the tag key length, then the extra characters are ignored. If *length* is less than zero, then *length* is treated as though it is equal to zero. If *length* is greater than the key length, then *length* is treated as though it is equal to the key length.

seekStr can contain null characters and still remain a valid search key, since the *length* specifies the length of string.

Returns: This method returns 0 if the SeekN found an exact match to the search key and the data file is repositioned to that record.

If the search value was not found, then the data file is positioned to the record after the position where the search key would have been located had it existed.

The SeekN returns 2, which is equivalent to the CodeBase return code of r4after.

If the search value was not found and the search value was greater than the last key of the tag then the SeekN creates an end of file condition. The action that takes place when an end of file condition is created depends on the value of the EOFAction property. In this case, the SeekN returns 3, which is equal to the CodeBase return code of r4eof.

The seek can not be accomplished without a tag. If there is no tag associated with the data file, this method returns 80, which is equivalent to the CodeBase return code of r4noTag.

It may be possible that the SeekN will attempt to reposition the data file to a non-existent record. In this case, the method returns 5, which is equivalent to the CodeBase return code of r4entry. Under these circumstances, the data file is not repositioned.

Before the data file can be repositioned, any changes made to the current record must be flushed. If the files could not be flushed because they were locked by another user then a NoUpdate event is fired. If the changes are discarded and the file reposition continues, this method returns 0. If the reposition is aborted in order to keep the changes, this method will return -1.

A DuplicateKey event is fired when the Unique property is set to 1 and flushing the current record would result in a duplicate key for a unique tag. If the changes are discarded and the file reposition continues, this method returns 0. If the reposition is aborted in order to keep the changes, this method will return -1.

This method also returns -1 if the **DATA4** structure is not valid or if the Validate event response is set to true.

See Also: **Validate**, **NoUpdate and DuplicateKey** events

SeekNext Method

VB Usage: *rc%* = [*form*].*CbCtrl*.**SeekNext(** *seekStr$* **)**

VC++ Usage: short rc = *m_CbCtrl*.**SeekNext(** LPCTSTR *seekStr* **)**

Delphi Usage: *rc*: Smallint **:=** [*form*].*CbCtrl*.**SeekNext**(*seekStr*: string)

C++ Builder Usage: short *rc* = [*form*]->*CbCtrl*->**SeekNext**(AnsiString *seekStr*)

Applicable To: **CbMaster**

Description: The SeekNext method differs from the Seek method, in that the current position within the tag is used, instead of the beginning of the tag. This provides the capability of performing an incremental search through the database.

If the index key at the current position in the tag is not equal to that of the search key, then SeekNext looks for the first occurrence of the search key in the tag. Should the seek fail to find a match, **r4after** is returned.

If the index key at the current position in the tag is equal to the search key then SeekNext tries to find the next occurrence of the search key. If it fails to find an index key that matches the search key, **r4entry** is returned and the data file is positioned to the record after the position where the search key would have been located, had it existed.

SeekNext may be used with any tag type, as long as it is formatted correctly. (e.g. **CbMaster**.SeekNext(" 123.44") for seeking on a numeric tag). Seeking on memo fields is not allowed.

This method fires a Validate event. If the event is intercepted and the response is set to true, the SeekNext method is aborted. By default, the response is set to false and the SeekNext method is allowed to continue.

Parameters:

seekStr *seekStr* is a string containing the value for which the search is conducted. See the Seek method for more details.

Returns: This method returns 0 if the SeekNext found the next exact match to the search key and the data file was repositioned to that record.

If there is no index key that matches search value in the tag, then the data file is positioned to the record after the position where the search key would have been located had it existed. The SeekNext returns 2, which is equivalent to the CodeBase return code of r4after.

If SeekNext fails to find the next occurrence of the search key, then the data file is positioned to the record after the position where the search key would have been located had it existed. The SeekNext returns 5, which is equivalent to the CodeBase return code of r4entry.

If the search value was not found and the search value was greater than the last key of the tag then the SeekNext creates an end of file condition. The action that takes place when an end of file condition is created depends on the value of the EOFAction property. In this case, the SeekNext returns 3, which is equal to the CodeBase return code of r4eof.

The seek can not be accomplished without a tag. If there is no tag associated with the data file, this method returns 80, which is equivalent to the CodeBase return code of r4noTag.

Before the data file can be repositioned, any changes made to the current record must be flushed. If the files could not be flushed because they were locked by another user then a NoUpdate event is fired. If the changes are discarded and the file reposition continues, this method returns 0. If the reposition is aborted in order to keep the changes, this method will return -1.

A DuplicateKey event is fired when the Unique property is set to 1 and flushing the current record would result in a duplicate key for a unique tag. If the changes are discarded and the file reposition continues, this method returns 0. If the reposition is aborted in order to keep the changes, this method will return -1.

This method also returns -1 if the **DATA4** structure is not valid or if the Validate event response is set to true.

See Also: **Validate**, **NoUpdate and DuplicateKey** events

SeekNextDouble Method

VB Usage: *rc%* = [*form*].*CbCtrl*.**SeekNextDouble(** *seekVal#* **)**

VC++ Usage: short *rc* = *m_CbCtrl*.**SeekNextDouble(** double *seekVal* **)**

Delphi Usage: *rc*: Smallint **:=** [*form*].*CbCtrl*.**SeekNextDouble**(*seekVal*: Double)

C++ Builder Usage: short *rc* = [*form*]->*CbCtrl*->**SeekNextDouble(** double *seekVal*)

Applicable To: **CbMaster**

Description: The SeekNextDouble method differs from the SeekDouble, in that the current position within the tag is used, instead of the beginning of the tag. This provides the capability of performing an incremental search through the database.

If the index key at the current position in the tag is not equal to that of the search key, SeekNextDouble looks for the first occurrence of the search key in the tag. Should the seek fail to find a match, **r4after** is returned.

If the index key at the current position in the tag is equal to the search key then SeekNextDouble tries to find the next occurrence of the search key. If SeekNextDouble fails to find an index key that matches the search key, **r4entry** is returned and the data file is positioned to the record after the position where the search key would have been located, had it existed.

The SeekNextDouble method must use a tag key of type Numeric or Date.

This method fires a Validate event. If the event is intercepted and the response is set to true, the SeekNextDouble method is aborted. By default, the response is set to false and the SeekNextDouble method is allowed to continue.

Parameters: *seekVal* is a double value. If the key type is of type Date, *seekVal* should represent a Julian day.

Returns: This method returns 0 if the SeekNextDouble found the next exact match to the search key and the data file was repositioned to that record.

If there is no index key that matches search value in the tag, then the data file is positioned to the record after the position where the search key would have been located had it existed. The SeekNextDouble returns 2, which is equivalent to the CodeBase return code of r4after.

If SeekNextDouble fails to find the next occurrence of the search key, then the data file is positioned to the record after the position where the search key would have been located had it existed. The SeekNextDouble returns 5, which is equivalent to the CodeBase return code of r4entry.

If the search value was not found and the search value was greater than the last key of the tag then the SeekNextDouble creates an end of file condition.

The action that takes place when an end of file condition is created depends on the value of the EOFAction property. In this case, the SeekNextDouble returns 3, which is equal to the CodeBase return code of r4eof.

The seek can not be accomplished without a tag. If there is no tag associated with the data file, this method returns 80, which is equivalent to the CodeBase return code of r4noTag.

Before the data file can be repositioned, any changes made to the current record must be flushed. If the files could not be flushed because they were locked by another user then a NoUpdate event is fired. If the changes are discarded and the file reposition continues, this method returns 0. If the reposition is aborted in order to keep the changes, this method will return -1.

A DuplicateKey event is fired when the Unique property is set to 1 and flushing the current record would result in a duplicate key for a unique tag. If the changes are discarded and the file reposition continues, this method returns 0. If the reposition is aborted in order to keep the changes, this method will return -1.

This method also returns -1 if the **DATA4** structure is not valid or if the Validate event response is set to true.

See Also: **Validate**, **NoUpdate and DuplicateKey** events

SeekNextN Method

VB Usage: *rc%* = [*form*].*CbCtrl*.**SeekNextN(** *seekStr$, length%* **)**

VC++ Usage: short *rc* = *m_CbCtrl*.**SeekNextN(** LPCTSTR *seekStr,* short *length* **)**

Delphi Usage: *rc*: Smallint **:=** [*form*].*CbCtrl*.**SeekNextN**(*seekStr*: string, *length*: Smallint)

C++ Builder Usage: short *rc* = [*form*]->*CbCtrl*->**SeekNextN**(AnsiString *seekStr,* short *length*)

Applicable To: **CbMaster**

Description: The SeekNextN method differs from SeekN, in that the current position within the tag is used, instead of the beginning of the tag. This provides the capability of performing an incremental search through the database.

If the index key at the current position in the tag is not equal to that of the search key, SeekNextN tries to find the first occurrence of the search key in the tag. Should the seek fail to find a match, **r4after** is returned.

If the index key at the current position in the tag is equal to the search key then SeekNextN tries to find the next occurrence of the search key. If SeekNextN fails to find an index key that matches the search key, **r4entry** is returned and the data file is positioned to the record after the position where the search key would have been located, had it existed.

SeekNextN may be used to seek without regard to nulls. Seeking on memo fields is not allowed.

This method fires a Validate event. If the event is intercepted and the response is set to true, the SeekNextN method is aborted. By default, the response is set to false and the SeekNextN method is allowed to continue.

Parameters:

seekStr *seekStr* is a string that contains the value for which the search is conducted.

length This is the length of the data contained in *seekStr*. This should be less than or equal to the length of the key size for the selected tag. If *length* is greater than the tag key length, then the extra characters are ignored. If *length* less than zero, then *length* is treated as though it is equal to zero. If *length* is greater than the key length, then *length* is treated as though it is equal to the key length.

seekStr can contain null characters and still remain a valid search key, since the *length* specifies the length of string.

Returns: This method returns 0 if the SeekNextN found the next exact match to the search key and the data file was repositioned to that record.

If there is no index key that matches search value in the tag, then the data file is positioned to the record after the position where the search key would have been located had it existed. The SeekNextN returns 2, which is equivalent to the CodeBase return code of r4after.

If SeekNextN fails to find the next occurrence of the search key, then the data file is positioned to the record after the position where the search key would have been located had it existed. The SeekNextN returns 5, which is equivalent to the CodeBase return code of r4entry.

If the search value was not found and the search value was greater than the last key of the tag then the SeekNextN creates an end of file condition. The action that takes place when an end of file condition is created depends on the value of the EOFAction property. In this case, the SeekNextN returns 3, which is equal to the CodeBase return code of r4eof.

The seek can not be accomplished without a tag. If there is no tag associated with the data file, this method returns 80, which is equivalent to the CodeBase return code of r4noTag.

Before the data file can be repositioned, any changes made to the current record must be flushed. If the files could not be flushed because they were locked by another user then a NoUpdate event is fired. If the changes are discarded and the file reposition continues, this method returns 0. If the reposition is aborted in order to keep the changes, this method will return -1.

A DuplicateKey event is fired when the Unique property is set to 1 and flushing the current record would result in a duplicate key for a unique tag. If the changes are discarded and the file reposition continues, this method returns 0. If the reposition is aborted in order to keep the changes, this method will return -1.

This method also returns -1 if the **DATA4** structure is not valid or if the Validate event response is set to true.

See Also: **Validate**, **NoUpdate and DuplicateKey** events

SetField Method

VB Usage: *rc%* = [*form*].*CbCtrl*.**SetField(***fieldName$, fieldValue* As Variant **)**

VC++ Usage: short *rc* = *m_CbCtrl*.**SetField(**LPCTSTR *fieldName*, VARIANT *fieldValue***)**

Delphi Usage: *rc*: Smallint **:=** [*form*].*CbCtrl*.**SetField**(*fieldName*: string, *fieldValue*: Variant)

C++ Builder Usage: short *rc* = [*form*]->*CbCtrl*->**SetField**(AnsiString *fieldName*,
Variant *fieldValue*)

Applicable To: **CbMaster**

Description: This method sets the field value of the current record to the value supplied by *fieldValue*. This method also signals all the bound controls that the current record has changed. Any changes made to the current record by this method are not permanent until the data file has been repositioned or the record has been explicitly flushed. The changes may be discarded by calling the Undo method.

Parameters:

fieldName *fieldName* is a string containing the name of a field.

fieldValue *fieldValue* contains the new value for the field.

Returns: This method returns 0.

See Also: **GetField**, **Undo** method

SetSelected Method

VB Usage: [*form*].*CbCtrl*.**SetSelected(** *index&, selected* As Boolean **)**

VC++ Usage: *m_CbCtrl*.**SetSelected(** long *index*, BOOL *selected* **)**

Delphi Usage: [*form*].*CbCtrl*.**GetSelected**(*index*: Integer, *selected*: Boolean)

C++ Builder Usage: [*form*]->*CbCtrl*->**GetSelected**(int *index*, bool *selected*)

Applicable To: **CbList**

Description: SetSelected toggles the selection on the item at the position specified by *index*. This method works for single and multiple selection list boxes.

Parameters:

index *index* is the zero-based index of the position of the item in the list box.

A list box under Windows 95 has a limit of 32,767 items, so *index* must be less than 32,767 when the **CbList** is not linked to a data file or when the **CbList** is linked to a data file and the DataAction property is set to Field Value (2).

If the **CbList** is linked to a data file and the Data Action property is set to Field Value (2), then the field value selected is assigned to the current record.

If the **CbList** is linked to a data file and the DataAction property is set to Record Position (1), the *index* represents a record number. The value of *index* is limited to the record numbers that occur in the selected tag. If there is no selected tag, *index* is limited to the number of records in the data file.

selected — Setting *selected* to true causes the item at the *index* position is to be selected. The selection is removed from the item at the *index* position when *selected* is set to false.

See Also: **GetSelected** method, **ListIndex** property

Skip Method

VB Usage: *rc%* = [*form*].*CbCtrl*.**Skip(** *numRecords&***)**

VC++ Usage: short *rc* = *m_CbCtrl*.**Skip(** long *numRecords* **)**

Delphi Usage: *rc*: Smallint **:=** [*form*].*CbCtrl*.**Skip**(*numRecords*: Integer)

C++ Builder Usage: short *rc* = [*form*]->*CbCtrl*->**Skip**(int *numRecords*)

Applicable To: **CbMaster**

Description: This method skips *numRecords* from the current record number. The selected tag is used to determine the new record position. The bound controls are automatically updated.

If no tag is selected, natural order is used.

It is possible to skip one record past the last record in the data file and create an end of file condition or skip before the first record and create a beginning of file condition. Refer to the BOFAction and EOFAction properties on the effect of skipping past the end or beginning of file.

This method fires a Validate event. If the event is intercepted and the response is set to true, the Skip method is aborted. By default, the response is set to false and the Skip method is allowed to continue.

Parameters: *numRecords* is the number of logical records to skip forward. If *numRecords* is negative, the skip is made backwards.

Returns: This method returns 0 if it was successful in repositioning the data file. If there is no current record because the data file has no records then Skip has no effect and returns -1.

Before the data file can be repositioned, any changes made to the current record must be flushed. If the files could not be flushed because they were locked by another user then a NoUpdate event is fired. If the changes are discarded and the file reposition continues, this method returns 0. If the reposition is aborted in order to keep the changes, this method will return -1.

A DuplicateKey event is fired when the Unique property is set to 1 and flushing the current record would result in a duplicate key for a unique tag. If the changes are discarded and the file reposition continues, this method returns 0. If the reposition is aborted in order to keep the changes, this method will return -1.

This method also returns -1 if the **DATA4** structure is not valid or if the Validate event response is set to true.

See Also: **BOFAction** and **EOFAction** properties, **Validate, NoUpdate, DuplicateKey** events

Top Method

VB Usage: *rc%* = [*form*].*CbCtrl*.Object.**Top**

VC++ Usage: short *rc* = *m_CbCtrl*.**Top( )**

Delphi Usage: *rc*: Smallint **:=** [*form*].*CbCtrl*.**Top**

C++ Builder Usage: short *rc* = [*form*]->*CbCtrl*->**Top( )**

Applicable To: **CbMaster**

Description: When this method is called, the **CbMaster** repositions the data file to the first record in the data file according to the selected tag. If there is no selected tag the data file is repositioned to the first physical record in the data file. The bound controls are updated automatically.

NOTE

Visual Basic supplies its own Top property to the control, by default. Use the Visual Basic Object property to specify that you want to use Top method supplied by the **CbMaster** control. See the Visual Basic online help for more information.

e.g. CbMaster1.Object.Top

This method fires a Validate event. If the event is intercepted and the response is set to true, the Top method is aborted. By default, the response is set to false and the Top method is allowed to continue.

Returns: This method returns 0 if it was successful in repositioning the data file to the first record according to the selected tag.

Before the data file can be repositioned, any changes made to the current record must be flushed. If the files could not be flushed because they were locked by another user then a NoUpdate event is fired. If the changes are discarded and the file reposition continues, this method returns 0. If the reposition is aborted in order to keep the changes, this method will return -1.

A DuplicateKey event is fired when the Unique property is set to 1 and flushing the current record would result in a duplicate key for a unique tag. If the changes are discarded and the file reposition continues, this method returns 0. If the reposition is aborted in order to keep the changes, this method will return -1.

This method also returns -1 if the **DATA4** structure is not valid or if the Validate event response is set to true.

See Also: **NoUpdate, Validate, DuplicateKey** events

Undo Method

VB Usage: *rc%* = [*form*].*CbCtrl*.**Undo**

VC++ Usage: short *rc* = *m_CbCtrl*.**Undo( )**

Delphi Usage: *rc:* Smallint **:=** [*form*].*CbCtrl*.**Undo**

C++ Builder Usage: short *rc* = [*form*]->*CbCtrl*->**Undo( )**

Applicable To: **CbMaster**

Description: When this method is called, the **CbMaster** discards the changes made to the current record. The **CbMaster** keeps a copy of the original record when it is first loaded into memory. When Undo is called, the copy of the original record is copied into the record buffer.

If the **CbMaster** is in append mode and Undo is called, the append is aborted and file is repositioned to the record that was the current record before the append was invoked.

Returns: Undo returns 0 if it is successful in undoing the changes to the current record.

If the Undo aborts an append and successfully repositions the data file, 0 is returned.

If Undo aborts an append on an empty data file, an end of file condition is created according to the setting of the EOFAction property. In this case, 3 is returned, which is equivalent to the CodeBase return code of r4eof.

See Also: **RefreshRecord** method, **EOFAction** property

Unlink Method

VB Usage: *rc%* = [*form*].*CbCtrl*.**Unlink**

VC++ Usage: short *rc* = *m_CbCtrl*.**Unlink( )**

Delphi Usage: *rc:* Smallint **:=** [*form*].*CbCtrl*.**Unlink**

C++ Builder Usage: short *rc* = [*form*]->*CbCtrl*->**Unlink( )**

Applicable To: **CbMaster**

Description: The Unlink method closes the **CbMaster**'s data file and signals its bound controls that the data file has been closed and the **CbMaster** becomes disabled. Calling the RefreshFiles method will cause the **CbMaster** to re-open the data file, automatically enable the **CbMaster** and update the bound controls.

After the Unlink method is called, the bound controls remain enabled and bound to the **CbMaster**. You can unbind a control from the it's **CbMaster** by changing the DataSource (DataSourceEdit or DataSourceList) property to the name of a different **CbMaster** or setting it to an empty string ("").

Returns: This method returns 0 when it successfully unlinks the data file. If the current data file could not be closed then Unlink returns -1 and the current data file remains open and valid.

See Also: **RefreshFiles** method

ValidDate Method

VB Usage: *valid* As Boolean = [*form*].*CbCtrl*.**ValidDate**

VC++ Usage: BOOL *valid* = *m_CbCtrl*.**ValidDate( )**

Delphi Usage: *rc*: Boolean **:=** [*form*].*CbCtrl*.**ValidDate**

C++ Builder Usage: bool *rc* = [*form*]->*CbCtrl*->**ValidDate( )**

Applicable To: **CbEdit, CbCombo**

Description: This method returns whether the current date displayed by the **CbEdit** or edit portion of the **CbCombo** forms a valid date. This method only works on date fields.

Consider the following examples. Assume that there is no date formatting and the date is in standard format ("CCYYMMDD"). If "19901301" was typed into the **CbEdit**, ValidDate would return false. In this case, there is no 13th month.

Assume that the date mask is "CCYY MMM, DD" and "1996 Sep, 31" was typed into the **CbEdit**, ValidDate would return false since September only has 30 days.

Returns: True is returned when the date is valid, otherwise false is returned. ValidDate returns false if the date is blank or incomplete.

7 Events

Events

The following reference lists all of the events that are unique to CodeControls. The syntax for the event procedures are given for Visual Basic, Visual C++, Delphi and Borland C++ Builder.

CbMaster Events			
ButtonClick	Error	MouseDown	NoUpdate
DatafileClosed	KeyDown	MouseMove	Reposition
DatafileOpened	KeyPress	MouseUp	TagChanged
DuplicateKey	KeyUp	NoEdit	Validate

CbButton Events			
Click	KeyDown	KeyUp	MouseMove
Error	KeyPress	MouseDown	MouseUp

CbEdit Events			
Change	Error	KeyUp	MouseUp
Click	KeyDown	MouseDown	
DblClick	KeyPress	MouseMove	

CbList Events			
Click	KeyDown	MouseDown	Reposition
DblClick	KeyPress	MouseMove	SelChange
Error	KeyUp	MouseUp	

CbCombo Events			
Change	KeyDown	MouseDown	SelChange
Click	KeyPress	MouseMove	
DblClick	KeyUp	MouseUp	
Error	LookUp	Reposition	

CbSlider Events			
Changed	KeyDown	MouseDown	Reposition
Click	KeyPress	MouseMove	
Error	KeyUp	MouseUp	

Visual Basic Events			
DragDrop	DragOver	GotFocus	LostFocus

Delphi and C++ Builder Events			
DragDrop	EndDrag	Exit	
DragOver	Enter	StartDrag	

ButtonClick Event

VB Usage: Sub *CbCtrl*_**ButtonClick**(*Action* As Integer)

VC++ Usage: void *CUserDialog*::**OnButtonClick***CbCtrl*(short FAR* *Action*)

Delphi Usage: procedure *form*.*CbCtrl***ButtonClick**(*Sender*: TObject;
var *Action*: Smallint)

C++ Builder Usage: void __fastcall *form*::*CbCtrl***ButtonClick**(TObject *Sender, short &*Action*)

Applicable To: **CbMaster**

Description: This event occurs when the user clicks one of the buttons on a **CbMaster** control. The ButtonClick event also occurs when a button on the **CbMaster** has the focus and the user presses the spacebar key. *Action* specifies which button was clicked. The following table lists the possible values for *Action*.

Setting	Value	Action
Top	0	The Top button was clicked.
Previous Page	1	The Previous Page button was clicked.
Previous Record	2	The Previous record button was clicked.
Search	3	The Search dialog button was clicked.
Next Record	4	The Next record button was clicked.
Next Page	5	The Next Page button was clicked.
Bottom	6	The Bottom button was clicked
Append	7	The Append button was clicked.
Deleted	8	The Deleted button was clicked.
Undo	9	The Undo button was clicked.
Reindex	10	The Reindex button was clicked.
Pack	11	The Pack method was clicked.
Flush	12	The Flush record button was clicked.
Refresh	13	The Refresh record button was clicked.

Setting the *Action* parameter to a value other than that originally passed into *Action* during the ButtonClick event prevents the default **CbMaster** control action from being performed.

Any program code in the ButtonClick event procedure is executed before the default action of the control. This gives you the ability to enhance or substitute the control's default action.

You cannot directly change *Action* to a different value in order to change the database operation initially specified. However, this can be accomplished indirectly by calling the appropriate **CbMaster** method (which executes the new operation immediately), and then setting *Action* to a different value to cancel the initial operation.

NOTE

Calling a method in the ButtonClick event does not trigger another ButtonClick event.

Change Event

VB Usage: Sub *CbCtrl*_**Change**( )

VC++ Usage: void *CUserDialog*::**OnChange***CbCtrl*( )

Delphi Usage: procedure *form*.*CbCtrl***Change**(*Sender*: TObject)

C++ Builder Usage: void __fastcall *form*::*CbCtrl***Change**(TObject *Sender)

Applicable To: **CbEdit, CbCombo**

Description: When the text is changed in a **CbEdit** or the edit portion of a **CbCombo**, the Change event is fired. This event is fired when the control is linked to data file and when the controls are not bound to a data file.

NOTE

This event is triggered each time the edit portion changes. For example, if the user starts by typing "S", a Change event is fired. If the user then presses the Backspace key, the event is triggered again.

Changed Event

VB Usage: Sub *CbCtrl*_**Changed**(ByVal *Value* As Double)

VC++ Usage: void *CUserDialog*::**OnChanged***CbCtrl*(double *Value*)

Delphi Usage: procedure *form*.*CbCtrl***Changed**(*Sender*: TObject; *Value*: Double)

C++ Builder Usage: void __fastcall *form*::*CbCtrl***Changed**(TObject *Sender, double *Value*)

Applicable To: **CbSlider**

Description: This event is fired when the thumb on the **CbSlider** is moved to a new position. The new value of the position is passed to the event through the parameter *Value*.

Click Event

VB Usage: Sub *CbCtrl*_**Click**( )

VC++ Usage: void *CUserDialog*::**OnClick***CbCtrl*( )

Delphi Usage: procedure *form*.*CbCtrl***Click**(*Sender*: TObject)

C++ Builder Usage: void __fastcall *form*::*CbCtrl***Click**(TObject *Sender)

Applicable To: **CbButton, CbEdit, CbList, CbCombo, CbSlider**

Description: The Click event is fired when these controls have received one of the following Windows messages: WM_LBUTTONUP, WM_MBUTTONUP or WM_RBUTTONUP. Refer to your compiler's online help for more information about this event.

DblClick Event

VB Usage: Sub *CbCtrl*_**DblClick**( )

VC++ Usage: void *CUserDialog*::**OnDblClick***CbCtrl*( )

Delphi Usage: procedure *form*.*CbCtrl***DblClick**(*Sender*: TObject)

C++ Builder Usage: void __fastcall *form*::*CbCtrl***DblClick**(TObject *Sender)

Applicable To: **CbEdit, CbList, CbCombo**

Description: The DblClick event is fired when these controls have received one of the following Windows messages: WM_LBUTTONDBLCLICK, WM_MBUTTONDBLCLICK or WM_RBUTTONDBLCLICK. Refer to your compiler's online help for more information about this event.

DatafileClosed Event

VB Usage: Sub *CbCtrl*_**DatafileClosed**( )

VC++ Usage: void *CUserDialog*::**OnDatafileClosed***CbCtrl*( )

Delphi Usage: procedure *form*.*CbCtrl***DatafileClosed**(*Sender*: TObject)

C++ Builder Usage: void __fastcall *form*::*CbCtrl***DatafileClosed**(TObject *Sender)

Applicable To: **CbMaster**

Description: This event is fired after the **CbMaster** closes the data file successfully. This event lets programmers know exactly when the data file has been closed and thus allows them to program any necessary clean up actions.

See Also: **DatafileOpened** event

DatafileOpened Event

VB Usage: Sub *CbCtrl*_**DatafileOpened**(ByVal *DatafileName* As String,
Response As Integer)

VC++ Usage: void *CUserDialog*::**OnDatafileOpened***CbCtrl*(LPCTSTR *DatafileName,*
short FAR* *Response*)

Delphi Usage: procedure *form*.*CbCtrl***DatafileOpened**(*Sender*: TObject;
DatafileName: string;
var *Response:* Smallint)

C++ Builder Usage: void __fastcall *form*::*CbCtrl*__DatafileOpened__(TObject *Sender,
const AnsiString D*atafileName*,
short &*Response*)

Applicable To: **CbMaster**

Description: This event is fired after the **CbMaster** opens the data file successfully. The *DatafileName* parameter contains the name of the data file that was opened. This event signals to programmers when the data file has been opened and thus allows them to program any necessary actions.

After the data file is opened, it needs to be positioned to a valid record. The *Response* parameter is initially set to 1 and if it is not altered, the data file is automatically positioned to the top record according to the selected tag. If there is no selected tag, the data file is positioned to the first record in the data file.

If the programmer intercepts the DatafileOpened event and changes the *Response* parameter to 0, the data file is not automatically positioned valid record. This gives the programmer the opportunity to position the data file to any desired record from the DatafileOpened procedure.

See Also: **DatafileClosed** event

DuplicateKey Event

VB Usage: Sub *CbCtrl*__DuplicateKey(__ ByVal *Action* As Integer,
Response As Integer**)**

VC++ Usage: void *CUserDialog*::__OnDuplicateKey__*CbCtrl*(short *Action*,
short FAR* *Response*)

Delphi Usage: procedure *form*.*CbCtrl*__DuplicateKey__(*Sender*: TObject;
Action: Smallint;
var *Response:* Smallint)

C++ Builder Usage: void __fastcall *form*::*CbCtrl*__DuplicateKey__(TObject *Sender,
short *Action*,
short &*Response*)

Applicable To: **CbMaster**

Description: This event is triggered when CodeControls attempts to write a duplicate key to a unique tag and the Unique property is set to 1. This event occurs when the **CbMaster** attempts to flush a record that has been modified in such a way that it would become a duplicate key in a unique tag.

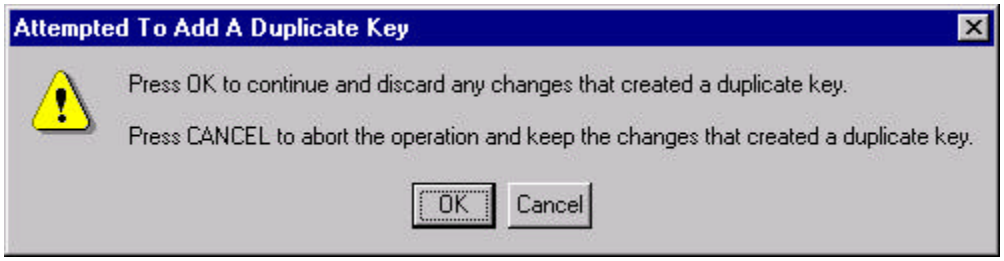
The **CbMaster** attempts to a flush record changes when the user tries to move to a different record, append a new record, or the data file is about to be closed. The data file is closed when either the form containing the **CbMaster** control is unloaded, or when the RefreshFiles or Unlink methods are called. In addition, the user may explicitly flush any changes by calling the FlushRecord method or by clicking the Flush button on the **CbMaster** or **CbButton**.

You can signal which action you want the **CbMaster** control to perform by setting *Response* to one of the following values listed in this table.

Settings	Value	Response Action
OK	1	Continue with the operation, do not write record changes to the data or index file. The changes to the record are discarded.
Cancel	2	Abort the operation that triggered the event. Current record position does not change and any changes to the current record remain intact.

For example, if the user modifies the current record so that a given field (upon which a unique index tag is based) matches that of an existing record and then attempts to go to another record by clicking the Top button on the **CbMaster**, a DuplicateKey event is generated.

If you do not modify the value of *Response* to one of the constants listed above, the **CbMaster** will display the following dialog:



Duplicate Key Dialog

The choices available in the above dialog are equivalent to setting the *Response* parameter to one of the two values listed in the previous table. For example, clicking the "Cancel" button would be equivalent to intercepting the event and setting *Response* to Cancel (2).

The advantage of the DuplicateKey event is that if you don't want the standard dialog response supplied by CodeControls, you can supply your own, or have none at all. In addition you can augment the event with your own code if you wish to perform other operations as well.

Error Event

VB Usage: Sub *CbCtrl*_**Error(** *ErrorCode* As Long,
Description As String,
Response As Integer)

VC++ Usage: void *CUserDialog*::**OnError***CbCtrl*(long *ErrorCode*, LPCTSTR *Description*, short FAR* *Response*)

Delphi Usage: procedure *form*.*CbCtrl***Error**(*Sender*: TObject;
ErrorCode: Integer;
const *Description*: string;
var *Response:* Smallint)

C++ Builder Usage: void __fastcall *form*::*CbCtrl***Error**(TObject *Sender,
Integer *ErrorCode*,
const AnsiString *Description*,
short &*Response*)

Applicable To: **CbMaster**

Description: The Error event is generated whenever an internal CodeControls error occurs. This occurs when the following properties are set to illegal values: DatabaseName, DataField, DataExpr, DataSource, DataSourceEdit, DataSourceList, DataExprList, DataTag and DataTagList.

Generally, when an Error event is fired, the control is automatically disabled. After the application has recovered from the error, the control must be explicitly enabled by setting the Enabled property to true.

If the programmer intercepts the Error event and fixes the error in the Error procedure code, the control will not be disabled. If the program intercepts the Error event and inadvertently causes another Error event to be fired, a infinite recursive loop may occur. This can cause the application to run out of stack space.

The control is not disabled if Error is caused by an invalid DataTag or DataTagList property. In these cases, the Error event is fired, but the control remains enabled and DataTag or DataTagList property is reset to the previously valid tag.

The *ErrorCode* may have one of the following values:

ErrorCode	Description
-10	Invalid DatabaseName property for the CbMaster.
-20	Invalid IndexFiles property for the CbMaster.
-30	Invalid DataTag property for the CbMaster.
-40	Invalid DataSource property for the CbButton.
-50	Invalid DataSourceEdit property for the CbCombo.
-60	Invalid DataSourceList property for the CbCombo.
-70	Invalid DataFieldEdit property for the CbCombo.

-80	Invalid DataExprList property for the CbCombo.
-90	Invalid DataTagList property for the CbCombo.
-100	Invalid DataSource property for the CbEdit.
-110	Invalid DataField property for the CbEdit.
-120	Invalid DataSource property for the CbList.
-130	Invalid DataExpr property for the CbList.
-140	Invalid DataTag property for the CbList.
-150	Invalid DataField property for the CbSlider.
-160	Invalid DataSource property for the CbSlider.

The *Description* parameter will contain a short general description about the function call that failed.

The default action for the Error event is to display a message box containing a description of the error. This message may be suppressed by setting the *Response* parameter to 0.

See Also: **Enabled** property

KeyDown Event

VB Usage: Sub *CbCtrl*_**KeyDown**(*KeyCode* As Integer, *Shift* As Integer)

VC++ Usage: void *CUserDialog*::**OnKeyDown***CbCtrl*(short FAR* *KeyCode*, short *Shift*)

Delphi Usage: procedure *form*.*CbCtrl***KeyDown**(*Sender*: TObject;
var *Key*: Word;
Shift: TShiftState)

C++ Builder Usage: void __fastcall *form*::*CbCtrl***KeyDown**(TObject *Sender,
WORD &*Key*,
TShiftState *Shift*)

Applicable To: **CbMaster, CbButton, CbEdit, CbList, CbCombo, CbSlider**

Description: This event is fired when the Windows message WM_SYSKEYDOWN or WM_KEYDOWN is received by the control. Refer to your compiler's online help for more information about this event.

KeyPress Event

VB Usage: Sub *CbCtrl*_**KeyPress**(*KeyAscii* As Integer)

VC++ Usage: void *CUserDialog*::**OnKeyPress***CbCtrl*(short FAR* *KeyAscii*)

Delphi Usage: procedure *form*.*CbCtrl***KeyPress**(*Sender*: TObject; var *Key*: Char)

C++ Builder Usage: void __fastcall *form*::*CbCtrl***KeyPress**(TObject *Sender, char *&Key*)

Applicable To: **CbMaster, CbButton, CbEdit, CbList, CbCombo, CbSlider**

Description: This event is fired when the Windows message WM_CHAR is received by the control. Refer to your compiler's online help for more information about this event.

KeyUp Event

VB Usage: Sub *CbCtrl*_**KeyUp**(*Keycode* As Integer, *Shift* As Integer)

VC++ Usage: void CUserDialog::**OnKeyUp***CbCtrl*(short FAR* *Keycode*, short *Shift*)

Delphi Usage: procedure *form*.*CbCtrl***KeyUp**(*Sender*: TObject;
var *Key*: Word;
Shift: TShiftState)

C++ Builder Usage: void __fastcall *form*::*CbCtrl***KeyUp**(TObject *Sender,
WORD &*Key*,
TShiftState *Shift*)

Applicable To: **CbMaster, CbButton, CbEdit, CbList, CbCombo, CbSlider**

Description: This event is fired when the Windows message WM_SYSKEYUP or WM_KEYUP is received by the control. Refer to your compiler's online help for more information about this event.

LookUp Event

VB Usage: Sub *CbCtrl*_**LookUp**(ByVal *str* As String, *newStr* As String)

VC++ Usage: void *CUserDialog*::**OnLookUp***CbCtrl*(LPCTSTR *str*, BSTR FAR* *newStr*)

Delphi Usage: procedure *form*.*CbCtrl***LookUp**(*Sender*: TObject;
const *str:* string;
newStr: string)

C++ Builder Usage: void __fastcall *form*::*CbCtrl***LookUp**(TObject *Sender,
const AnsiString *str*,
AnsiString *newStr*)

Applicable To: **CbCombo**

Description: As explained in the, "CodeControls Concepts" chapter, the **CbCombo** supports a search feature. As the user types into the edit portion of the **CbCombo**, an incremental search of the list contents is executed, based on the current contents of the edit.

Before the search actually occurs, the LookUp event is called, with the contents of the edit portion being passed into the *str* parameter. You can modify this string and assign it to the *newStr* parameter.

NOTE

> The LookUp event is triggered each time the edit contents change. For example, if the user starts by typing "S", a LookUp event is generated with *str* being equal to "S". If the user then presses the Backspace key, the event is triggered again, this time with *str* being a null string, and so on.

If the list is not bound to **CbMaster**, the search is accomplished by doing a string comparison of the edit text and the list items. When the edit portion is bound to field, the contents of the edit may be padded with spaces. If the list portion is filled with strings, the searches may fail to find a match merely because the list strings have not be padded with spaces. If this is the case, intercept the LookUp event and trim the trailing spaces from the *str* parameter and assign the resulting string to the *newStr* parameter. The **CbCombo** will execute the search using the *newStr* value and thus the trailing spaces will no longer affect the search.

If the list is bound to a **CbMaster,** the search is accomplished by performing a query operation based on a dBASE expression created by using the DataExprList expression and edit text. The query can be made more efficient if an appropriate tag is specified by the DataTagList property.

The LookUp event will need to be intercepted if the text typed into the edit does not match the format of the dBASE expression specified by the DataExprList. For example, suppose you have a data file with two fields, L_NAME and F_NAME, each of size 10 characters, and the DataExprList property was based on the expression "L_NAME + F_NAME". The resulting key value for this expression will be 20 characters long, and each field within the key will be padded with blanks to match the field size.

Figure 7.1 shows the key value for the record of John Smith, based on this expression:

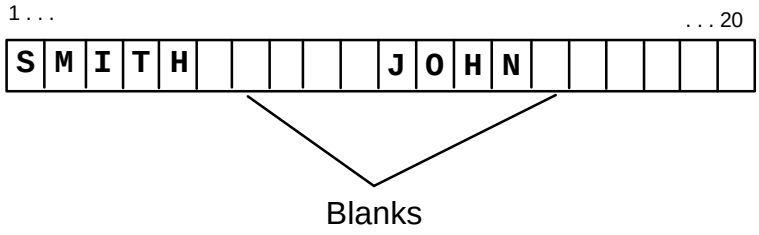


Figure 7.1 Key value for John Smith. DataExprList is L_NAME + F_NAME

Suppose that user types in the search key as "Smith,John". As you can see, the search key typed into the edit does not match the format of the DataExprList expression. Consequently the search will fail to find a match.

The way to solve this problem, involves modifying the contents of the edit to match the format of the DataExprList expression before the search is performed. Specifically, when the entire name has been entered in the edit portion, the last name must be extracted and padded with blanks before the first name is appended. Developers can modify the search key by intercepting the LookUp event.

NOTE

> Be sure to assign an appropriate expression to the DataExprList property to minimize the programming needed in the LookUp event. The above example was exaggerated for illustration purposes. Obviously, if the DataExprList had been set to "TRIM(L_NAME) + ',' + F_NAME", the contents of the edit would not have to be modified.

See Also: **DataExprList** and **DataTagList** properties, and "CodeControls Concepts" chapter.

MouseDown Event

VB Usage:
```
Sub CbCtrl_MouseDown( Button As Integer,
                      Shift As Integer,
                      x As Single, y As Single )
```

VC++ Usage:
```
void CUserDialog::OnMouseDownCbCtrl( short Button,
                                     short Shift,
                                     long x, long y )
```

Delphi Usage:
```
procedure form.CbCtrlMouseDown( Sender: TObject;
                                Button: TMouseButton;
                                Shift: TShiftState;
                                x: Integer; y: Integer )
```

C++ Builder Usage:
```
void __fastcall form::CbCtrlMouseDown( TObject *Sender,
                                       TMouseButton Button,
                                       TShiftState Shift,
                                       int x, int y )
```

Applicable To: **CbMaster, CbButton, CbEdit, CbList, CbCombo, CbSlider**

Description: This event is fired when the Windows message WM_LBUTTONDOWN, WM_MBUTTONDOWN or WM_RBUTTONDOWN is received by the control. Refer to your compiler's online help for more information about this event.

MouseMove Event

VB Usage: Sub *CbCtrl*_**MouseMove**(*Button* As Integer,
Shift As Integer,
x As Single**,** *y* As Single)

VC++ Usage: void CUserDialog::**OnMouseMove***CbCtrl*(short *Button*,
short *Shift*,
long *x*, long *y*)

Delphi Usage: procedure *form*.*CbCtrl***MouseMove**(*Sender*: TObject;
Shift: TShiftState;
x: Integer; *y:* Integer)

C++ Builder Usage: void __fastcall *form*::*CbCtrl***MouseMove**(TObject *Sender,
TShiftState *Shift*,
int *x*, int *y*)

Applicable To: **CbMaster, CbButton, CbEdit, CbList, CbCombo, CbSlider**

Description: This event is fired when the Windows message WM_MOUSEMOVE is received by the control. Refer to your compiler's online help for more information about this event.

MouseUp Event

VB Usage: Sub *CbCtrl*_**MouseUp**(*Button* As Integer, *Shift* As Integer,
x As Single**,** *y* As Single)

VC++ Usage: void CUserDialog::**OnMouseUp***CbCtrl*(short *Button*, short *Shift*,
long *x*, long *y*)

Delphi Usage: procedure *form*.*CbCtrl***MouseUp**(*Sender*: TObject;
Button: TMouseButton;
Shift: TShiftState;
x: Integer; *y:* Integer)

C++ Builder Usage: void __fastcall *form*::*CbCtrl***MouseUp**(TObject *Sender,
TMouseButton *Button*,
TShiftState *Shift*,
int *x*, int *y*)

Applicable To: **CbMaster, CbButton, CbEdit, CbList, CbCombo, CbSlider**

Description: This event is fired when the Windows message WM_LBUTTONUP, WM_MBUTTONUP or WM_RBUTTONUP is received by the control. Refer to your compiler's online help for more information about this event.

NoEdit Event

VB Usage: Sub *CbCtrl*_**NoEdit**(ByVal *Action* As Integer, *Response* As Integer)

VC++ Usage: void *CUserDialog*::**OnNoEdit***CbCtrl*(short *Action*, short FAR* *Response*)

Delphi Usage: procedure *form*.*CbCtrl***NoEdit**(*Sender*: TObject; *Action*: Smallint;
var *Response*: Smallint)

C++ Builder Usage: void __fastcall *form*::*CbCtrl***NoEdit**(TObject *Sender, short *Action*,
short &*Response*)

Description: This event occurs when the **CbMaster** is performing pessimistic locking and an attempt is made to edit a field. Under this locking method, the record is locked when the user attempts to edit the record. If the record can not be locked because another application has the record locked, the NoEdit event is fired.

You can signal which action you want the master control to perform during a NoEdit event by setting *Response* to one of the following values listed in this table.

Settings	Value	Response Action
Cancel	2	Abort the attempt to modify the record. The current record position does not change and contents of the record are not modified.
Retry	4	Attempt the lock again.

For example, if the user tries to modify the current record and the current record is locked by another application, this event will occur. If you wish to try the lock again, set the parameter *Response* to Retry (4) within the procedure code of this event.

If you do not modify the value of *Response* to one of the constants listed above, the **CbMaster** will display the following dialog:

NoEdit Dialog

The choices available in the above dialog are equivalent to setting the *Response* parameter to one of the two values listed in the previous table. For example, the user clicking the "Cancel" button would be equivalent to intercepting the event and setting *Response* to Cancel (2).

The advantage of the NoEdit event is that if you don't want the standard dialog response supplied by CodeControls, you can supply your own, or have none at all. In addition you can augment the event with your own code if you wish to perform other operations as well.

See Also: **NoUpdate** event

NoUpdate Event

VB Usage: Sub *CbCtrl*_**NoUpdate**(ByVal *Action* As Integer, *Response* As Integer)

VC++ Usage: void *CUserDialog*::**OnNoUpdate***CbCtrl*(short *Action*,
short FAR* *Response*)

Delphi Usage: procedure *form*.*CbCtrl***NoUpdate**(*Sender*: TObject;
Action: Smallint;
var *Response*: Smallint)

C++ Builder Usage: void __fastcall *form*::*CbCtrl***NoUpdate**(TObject *Sender,
short *Action*,
short &*Response*)

Description: This event occurs when the **CbMaster** is performing optimistic locking and an attempt is made to flush record changes. Under this locking method, the **CbMaster** attempts to lock the record before it tries to flush any changes to disk. If the record can not be locked because another application has the record locked, the NoUpdate event is fired.

The **CbMaster** will automatically try to flush a modified record under different circumstances. For instance, when the user attempts to move to a different record or append a new record, the **CbMaster** will attempt to flush any changes to the current record. If the data file is about to be closed, then the **CbMaster** will try to flush any changes before the data file is closed.

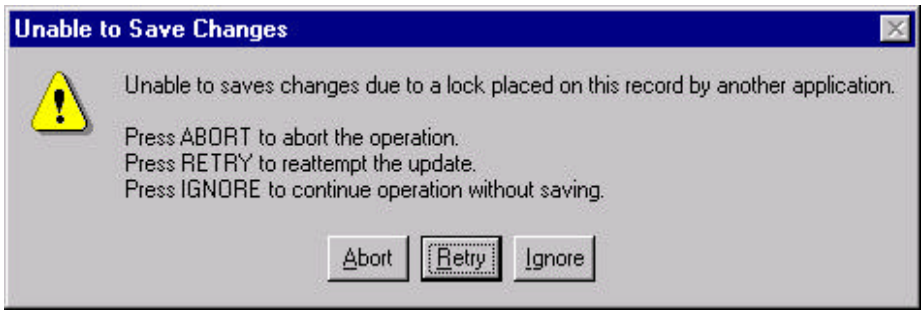
The user may also flush any changes explicitly by calling Flush method or by clicking Flush button of the **CbMaster** or **CbButton**.

You can signal which action you want the master control to perform during a NoUpdate event by setting *Response* to one of the following values listed in this table.

Settings	Value	Response Action
Abort	3	Abort the operation that triggered the event. Current record position does not change and any changes to the current record remain intact.
Retry	4	Attempt the lock again.
Ignore	5	Perform the next operation without flushing the current record. Any changes made to the current record are discarded.

For example, if the user modifies the current record, clicks on the Top button, and the current record is locked by another application, this event will occur. If you wish to try the lock again, set the parameter *Response* to Retry (4) within the procedure code of this event.

If you do not modify the value of *Response* to one of the constants listed above, **CbMaster** will display the following dialog:



NoUpdate Dialog

The choices available in the above dialog are equivalent to setting the *Response* parameter to one of the three values listed in the previous table. For example, the user clicking the "Abort" button would be equivalent to code in the event procedure setting *Response* to a value of Abort (3).

The advantage of the NoUpdate event is that if you don't want the standard dialog response supplied by CodeControls, you can supply your own, or have none at all. In addition you can augment the event with your own code if you wish to perform other operations as well.

NOTE

If the application is closing and there are record changes that need to be flushed and the data file is locked by another application, the NoUpdate event will be fired. In this case, the Abort option will have no effect because the application will continue to close at this point and it can not be stopped from the control level. The data file will be closed and the changes will be lost if the Abort or Ignore option is chosen. It is up to the programmer to take into account the possibility of this event and attempt to flush the changes before the application begins shutting down.

See Also: **NoEdit** event

Reposition Event

VB Usage: Sub *CbCtrl*_**Reposition**(ByVal *Action* As Integer)

VC++ Usage: void *CUserDialog*::**OnReposition***CbCtrl*(short *Action*)

Delphi Usage: procedure *form*.*CbCtrl***Reposition**(*Sender*: TObject; *Action*: Smallint)

C++ Builder Usage: void __fastcall *form*::*CbCtrl***Repostion**(TObject *Sender, short *Action*)

Applicable To: **CbMaster**

Description: This event is triggered after the **CbMaster's** data file has been repositioned.

This event occurs when a **CbMaster** control is first loaded, when the user positions to a new record using one of the **CbMaster** control buttons, or when method causes the **CbMaster** control to position to a new record.

Use this event whenever you need to perform additional operations or calculations based on the contents of the current record.

The *Action* parameter indicates how the data file was repositioned. *Action* will be set to one of the following values:

Settings	Value	Description
Top	0	The Top button was clicked or the Top method was called.
Previous Page	1	The PreviousPage button was clicked or the Skip method was called with its parameter set to (-1*PageSize). The PageSize property determines how many records to skip when the PreviousPage and NextPage buttons are clicked.
Previous Record	2	The PreviousRecord button was clicked or the Skip method was called with its parameter set to -1.
Search	3	The search dialog was used to reposition the data file. The search dialog is invoked by clicking the Search button or by calling the Search method
Next Record	4	The NextRecord button was clicked or the Skip method was called with its parameter set to 1.
Next Page	5	The NextPage button was clicked or the Skip method was called with its parameter set to the PageSize value.
Bottom	6	The Bottom button was clicked or the Bottom method was called.
Append	7	The Append Button was clicked or the Append method was called.
Undo	9	The Undo Button was clicked or the Undo method was called.
Flush Record	12	The FlushRecord Button was clicked or the FlushRecord method was called.
Refresh Record	13	The RefreshRecord Button was clicked or the RefreshRecord method was called.
Skip	14	The Skip method was called.
Go	15	The Go method was called.
Slider	16	The user clicked the CbMaster's slider, or the Position property was changed.
Seek	18	The Seek method was called.
SeekDouble	19	The SeekDouble method was called.
SeekN	20	The SeekN method was called.
SeekNext	21	The SeekNext method was called.
SeekNextDouble	22	The SeekNextDouble method was called.
SeekNextN	23	The SeekNextN method was called.

Reposition Event

VB Usage: Sub *CbCtrl*_**Reposition**(*NewPosition* As Double,
ByVal *OldPosition* As Double)

VC++ Usage: void *CUserDialog*::**OnReposition***CbCtrl*(double FAR* *NewPosition*,
double *OldPosition*)

Delphi Usage: procedure *form*.*CbCtrl***Reposition**(*Sender*: TObject;
var *NewPosition*: Double;
OldPosition: Double)

C++ Builder Usage: void __fastcall *form*::*CbCtrl***Reposition**(TObject *Sender,
double &*NewPostion*,
double *OldPosition*)

Applicable To: **CbSlider**

Description: This event is triggered when the **CbSlider** DataAction property is set to RecordPosition (1) and the slider thumb is moved to a new position on the slider, which corresponds to a new data file position. The **CbSlider** signals the **CbMaster** control to reposition the data file to the new record.

The *NewPosition* parameter indicates the position of the new current record. The Reposition event is fired just before the reposition takes place. The programmer may intercept the Reposition event and change the *NewPosition* parameter and thereby change how the data file is repositioned. The *OldPostion* parameter indicates the position of the record that was the current record previously.

Use this event whenever you need to perform additional operations or calculations based on the contents of the current record.

Reposition Event

VB Usage: Sub *CbCtrl*_**Reposition**(ByVal *NewRecord* As Long,
ByVal *OldRecord* As Long)

VC++ Usage: void *CUserDialog*::**OnReposition***CbCtrl*(long *NewRecord*,
long *OldRecord*)

Delphi Usage: procedure *form*.*CbCtrl***Reposition**(*Sender*: TObject;
NewRecord: Integer;
OldRecord: Integer)

C++ Builder Usage: void __fastcall *form*::*CbCtrl***Reposition**(TObject *Sender,
int *NewRecord*,
int *OldRecord*)

Applicable To: **CbCombo, CbList**

Description: This event is triggered when the **CbList** or **CbCombo** is used to reposition the data file. The **CbList** DataAction property must be set to RecordPosition (1) before the **CbList** can be used to reposition a data file. **CbList** and **CbCombo** reposition the data file when an entry in the list box is selected. The **CbList** and **CbCombo** signal the **CbMaster** control to reposition the data file to the new record.

This event is fired after the data file has been repositioned.

The *NewRecord* parameter represents the record number of the new current record. The *OldRecord* parameter represents the record number of the previous current record.

Use this event whenever you need to perform additional operations or calculations based on the contents of the current record.

SelChange Event

VB Usage: Sub *CbCtrl*_**SelChange**( )

VC++ Usage: void CUserDialog::**OnSelChange***CbCtrl*( )

Delphi Usage: procedure *form*.*CbCtrl***SelChange**(*Sender*: TObject)

C++ Builder Usage: void __fastcall *form*::*CbCtrl***SelChange**(TObject *Sender)

Applicable To: **CbCombo, CbList**

Description: This event is triggered when the selection has changed in a **CbList** or the list portion of the **CbCombo**. This event is only fired when the controls are linked to a data file.

TagChanged Event

VB Usage: Sub *CbCtrl*_**TagChanged**(*NewTag* As String, *Response* As Integer)

VC++ Usage: void *CUserDialog*::**OnTagChanged***CbCtrl*(BSTR FAR* *NewTag*,
short FAR* *Response*)

Delphi Usage: procedure *form*.*CbCtrl***TagChanged**(*Sender*: TObject; var *NewTag*: string;
var *Response*: Smallint)

C++ Builder Usage: void __fastcall *form*::*CbCtrl***TagChanged**(TObject *Sender,
AnsiString &*NewTag*,
short &*Response*)

Applicable To: **CbMaster**

Description: The TagChanged event is fired before the tag is changed for the **CbMaster**. This gives the programmer the chance to respond before the tag is actually changed. The parameter *NewTag* is a string that contains the name of new tag. Set the *Response* parameter to 0 if you do not want the tag to be changed to *NewTag*, otherwise the tag will be changed.

Validate Event

VB Usage: Sub *CbCtrl*_**Validate**(ByVal *Action* As Integer, Response As Integer)

VC++ Usage: void *CUserDialog*::**OnValidate***CbCtrl*(short *Action*, short FAR* Response)

Delphi Usage: procedure *form*.*CbCtrl***Validate**(*Sender*: TObject; *Action*: Smallint; var *Response*: Smallint)

C++ Builder Usage: void __fastcall *form*::*CbCtrl***Validate**(TObject *Sender, short *Action*, short &*Response*)

Applicable To: **CbMaster**

Description: This event is fired when a different record is about to become the current record of the control; for example, when the user clicks on the Next button to move to the next record in the file.

This event only occurs if the current record is changed by the user by pressing a **CbMaster** button or by calling a **CbMaster** repositioning method.

The second Validate event argument, *Response*, is used to specify what to do in response to the action specified. If you set this argument to true, the **CbMaster** control action that triggered the event is canceled, the user is left on the current record.

By default, *Response* is set to false, which means that the **CbMaster** attempts to update the current record and then complete the action that triggered the event.

The programmer can intercept this event and discard any changes to the current record before the **CbMaster** attempts to flush the record and carry out the action. The programmer can use the RecordChanged property to check if current record was modified and then discard any changes by calling the Undo or RefreshRecord method.

The *Action* parameter determines what caused the validation event to occur. See the list below for possible *Action* values.

Setting	Value	Action
Top	0	The Top button was clicked or the Top method was called.
Previous Page	1	The PreviousPage button was clicked or the Skip method was called with its parameter set to (-1*PageSize). The PageSize property determines how many records to skip when the PreviousPage and NextPage buttons are clicked.
Previous Record	2	The PreviousRecord button was clicked or the Skip method was called with its parameter set to -1.
Search	3	The data file is about to be repositioned using the search dialog. The search dialog is invoked by clicking the Search button or by calling the Search method.
Next Record	4	The NextRecord button was clicked or the Skip method was called with its parameter set to 1.
Next Page	5	The NextPage button was clicked or the Skip method was called with its parameter set to the PageSize value.

Bottom	6	The Bottom button was clicked or the Bottom method was called.
Append	7	The Append button was clicked or the Append method was called.
Flush Record	12	The Flush Record button was clicked or the FlushRecord method was called.
Skip	14	The Skip method was called.
Go	15	The Go method was called.
Seek	18	The Seek method was called.
SeekDouble	19	The SeekDouble method was called.
SeekN	20	The SeekN method was called.
SeekNext	21	The SeekNext method was called.
SeekNextDouble	22	The SeekNextDouble method was called.
SeekNextN	23	The SeekNextN method was called.
Close	24	The data file is about to be closed. The data file is closed when the RefreshFiles or Unlink methods are called or when the CbMaster control is being unloaded.

Appendix

Registering CodeControls

The following discussion describes how to register CodeControls on to your computer. The CodeControls ActiveX controls are automatically registered when they are installed with CodeBase. This chapter provides supplementary information on how to register the controls when they are not installed using the CodeBase install program.

CodeControls 3.0 consists of 6 custom OLE controls :

Control Name	Name of the OCX file
CbButton	BTN4.OCX
CbCombo	CMB4.OCX
CbEdit	EDT4.OCX
CbList	LST4.OCX
CbMaster	MTR4.OCX
CbSlider	SLD4.OCX

The following instructions assume that the controls are being registered from the command line. It will also be assumed that the important files are located in the C:\CODEBASE\CODECTRL directory. There are two sets of instructions given below, one set for stand-alone and one set for client/server.

Stand-Alone

There are two versions of CodeControls, stand-alone and client/server. Assuming that the CD-ROM drive is designated as the D: drive, the stand-alone files are located in the D:\CB63\CODECTRL directory.

Step 1

Copy the stand-alone files into the C:\CODEBASE\CODECTRL directory.

C:\CODEBASE\CODECTRL> COPY D:\CB63\CODECTRL*.*

Now the C:\CODEBASE\CODECTRL directory should contain the following files:

BTN4.OCX, BTN4.LIC

CMB4.OCX, CMB4.LIC

EDT4.OCX, EDT4.LIC

LST4.OCX, LST4.LIC

MTR4.OCX, MTR4.LIC

SLD4.OCX, SLD4.LIC

MFC42.DLL,

MSVCRT.DLL,

OLEAUT32.DLL

REGSVR32.EXE

Step 2

Now you must copy a CodeBase DLL into the C:\CODEBASE\CODECTRL directory. CodeBase supports four different computer languages and three different index file formats. As a result there are 12 different CodeBase DLLs to choose from. First determine which language you want to use and then decide which index file format you want to support. You only need to copy one CodeBase DLL into the C:\CODEBASE\CODECTRL. The following are examples that show where the DLLs are located for each language and for illustration purposes each example assumes that the user wants FoxPro support.

Visual Basic

If you are using Visual Basic, choose a DLL from D:\CB63\BASIC\DLL32 and rename it C4DLL.DLL.

```
C:\CODEBASE\CODECTRL>COPY D:\CB63\BASIC\DLL32\C4FOX.DLL C4DLL.DLL
```

C

If you are using C, choose a DLL from D:\CB63\C\DLL32 and rename it C4DLL.DLL.

```
C:\CODEBASE\CODECTRL>COPY D:\CB63\C\DLL32\C4FOX.DLL C4DLL.DLL
```

C++

If you are using C++, choose a DLL from D:\CB63\CPP\DLL32 and rename it C4DLL.DLL.

```
C:\CODEBASE\CODECTRL>COPY D:\CB63\CPP\DLL32\C4FOX.DLL C4DLL.DLL
```

Delphi

If you are using Delphi, choose a DLL from C:\CB63\PASCAL\DLL32 and rename it C4DLL.DLL

```
C:\CODEBASE\CODECTRL>COPY D:\CB63\PASCAL\DLL32\C4FOX.DLL C4DLL.DLL
```

Step 3

Now you can register each CodeControls OCX, so that compilers can have access to the custom controls. The **CbMaster** control must be registered first. Type in the following on the command line:

```
C:\CODEBASE\CODECTRL>REGSVR32 MTR4.OCX
```

A window will pop up saying that MTR4.OCX was successfully registered. Register the each OCX in the same manner. If the registration did not succeed, it could be possible that the above DLLs are out of date. Make sure that the Microsoft DLLs that you use are the ones supplied on the CodeBase CD-ROM. Make sure that the C4DLL.DLL is a CodeBase 6.3 (or later) DLL. Try registering the OCXs again. Always register the **CbMaster** control first.

Once the OCXs have been successfully registered then you can access the OCXs through your compiler.

Client/Server

There are two versions of CodeControls, stand-alone and client/server. Assuming that the CD-ROM drive is designated as the D: drive, the client/server files are located in the D:\CB63CS\CODECTRL directory.

Step 1

Copy the client/server files into the C:\CODEBASE\CODECTRL directory.

```
C:\CODEBASE\CODECTRL> COPY D:\CB63CS\CODECTRL\*.*
```

Now the C:\CODEBASE\CODECTRL directory should contain the following files:

BTN4.OCX, BTN4.LIC
CMB4.OCX, CMB4.LIC
EDT4.OCX, EDT4.LIC
LST4.OCX, LST4.LIC
MTR4.OCX, MTR4.LIC
SLD4.OCX, SLD4.LIC
MFC42.DLL,
MSVCRT.DLL,
OLEAUT32.DLL
REGSVR32.EXE

Step 2

Now you must copy a CodeBase DLL into the C:\CODEBASE\CODECTRL directory. CodeBase supports four different languages. Copy the CodeBase DLL that supports the language of your choice into the C:\CODEBASE\CODECTRL directory. The following examples show where the various CodeBase DLLs are located. Only copy one C4DLL.DLL into the C:\CODEBASE\CODECTRL directory.

Visual Basic If you are using Visual Basic, copy the C4DLL.DLL from D:\CB63CS\BASIC\DLL32 into C:\CODEBASE\CODECTRL.

```
C:\CODEBASE\CODECTRL>COPY D:\CB63CS\BASIC\DLL32\C4DLL.DLL
```

C If you are using C, copy the C4DLL.DLL from D:\CB63CS\C\DLL32 into C:\CODEBASE\CODECTRL.

```
C:\CODEBASE\CODECTRL>COPY D:\CB63CS\C\DLL32\C4DLL.DLL
```

C++ If you are using C++, copy the C4DLL.DLL from D:\CB63CS\CPP\DLL32 into C:\CODEBASE\CODECTRL.

```
C:\CODEBASE\CODECTRL>COPY D:\CB63CS\CPP\DLL32\C4DLL.DLL
```

Delphi If you are using Delphi, copy the C4DLL.DLL from D:\CB63CS\PASCAL\DLL32 into C:\CODEBASE\CODECTRL.

```
C:\CODEBASE\CODECTRL>COPY D:\CB63CS\PASCAL\DLL32\C4DLL.DLL
```

Step 3

Now you can register each CodeControls OCX, so that compilers can have access to the custom controls. The **CbMaster** control must be registered first. Type in the following on the command line:

```
C:\CODEBASE\CODECTRL>REGSVR32 MTR4.OCX
```

A window will pop up saying that MTR4.OCX was successfully registered. Register the each OCX in the same manner. If the registration did not succeed, it could be possible that the above DLLs are out of date. Make sure that the Microsoft DLLs that you use are the ones supplied on the CodeBase CD-ROM. Make sure that the C4DLL.DLL is a CodeBase 6.3 (or later) DLL. Try registering the OCXs again. Always register the **CbMaster** control first.

Once the OCXs have been successfully registered then you can access the OCXs through your compiler.