The Crescent Custom Data Control and Objects

The Custom Data control library (CCDATA32.OCX, CCDATA16.OCX) define the following control and supporting objects:

- Crescent Custom Data Control
- CFields Collection
- CField Object

This library allows you to access custom data sources that are not accessible with the standard Visual Basic Data control and Microsoft Jet Engine.

Crescent Custom Data Control

The Custom Data control provides access to data stored in custom data sources that are not accessible with the standard Visual Basic Data control and Microsoft Jet Engine. A custom data source can be anything ranging from a text file containing specialized field and record delimiters to data that originates from unconventional sources, like an HTML document or a serial port.

Like the Visual Basic Data control, the Custom Data control enables you to navigate from row to row and to display and manipulate data from the rows in bound controls. If fact, many of the properties, methods, and events of the Custom Data control are modeled after the those found on the Visual Basic Data control and the Recordset object.

Control Files

CCData16.OCX, CCData32.OCX

Object Type

CCData

CField Objects and the CFields Collection

The major difference between the Visual Basic Data control and the Custom Data control is the origin and management of data. The Visual Basic Data control manipulates data in a RecordSet object by making high-level requests to the Microsoft Jet Engine. The Microsoft Jet Engine performs the low-level functions that coordinate data with the data source and respond to high-level requests from the Visual Basic Data control. The data source must be one that the Microsoft Jet Engine understands.

In contrast, the Custom Data control manipulates data in the form of a collection of field objects. The CFields property of the Custom Data control contains a CFields collection object containing the values of the current row. Each item in this collection is a CField object representing a field from the current row.

Both the CFields collection and the CField objects have properties and methods that allow you to define the schema (i.e. the number of fields in row, field names, data types, etc.) of the custom data source. The custom property sheet of the Custom Data control allows you to define the CFields collection and CField objects at design time. You can also manipulate the contents of the CFields property at run-time.

The Custom Data control provides the *Data Source interface* which allows you to define the low-level routines that access data from custom data sources and populate the CFields collection with values for the current row.

Once you define the Data Source interface for a custom data control, you can perform most of the high-level data access operations (i.e. navigation, etc.) using the Custom Data control without writing any additional code. Data-aware controls bound to a Custom Data control automatically display data from one or more fields for the current row or, in some cases, for a set of rows.

If the Custom Data control is instructed to move to a different row, all bound controls automatically pass any changes to the Custom Data control to be saved in the database. The Custom Data control then moves to the requested row and passes back data from the current row to the bound controls where it's displayed.

Data Source Interface

The Data Source interface of the Custom Data control is a set of events for which you can code event procedures that do the following;

- Set up schema information for a custom data source.
- Read and write data to and from a custom data source.

These events occur when the Custom Data control requires information about or data from the custom data source. The events of the Data Source interface for the Custom Data control are listed below:

Use this Event ...	To Code an Event Procedure that ...
AddRow	Adds a new row to the data source.
DeleteRow	Deletes a row from the data source.
GetAttributes	Gets the attributes of the data source.
GetDefaultRow	Gets the default values for a new row in the data source.
GetFields	Gets the CField objects for the data source.
GetNumberOfRows	Gets the number of rows.
GetRow	Gets a specified row.
Open	Opens the data source.
Requery	Reopens the data source.
Resize	Changes the size of controls on the form when the form changes size.
UpdateRow	Modifies a row in the data source.

You must code the Data Source event procedures to handle a number of contingencies including empty data sets, adding new rows, editing and updating existing rows, and handling some types of errors.

Parameters of the Data Source event procedures provide that ability to indirectly access and manipulate the current CFields collection.

NOTE Accessing the CFields property directly from a Data Source event procedure will produce unexpected results. Always use the DataFields parameter of the current Data Source event procedure to manipulate the current CFields collection for the Custom Data control.

Bound Controls

The DBList, DBCombo, and DBGrid controls are all capable of managing sets of rows when bound to a Custom Data control.

All of these controls permit several rows to be displayed or manipulated at once.

The CheckBox, TextBox, Label, ListBox and ComboBox controls are also data-aware and can be bound to a single field of a row managed by the Custom Data control. Additional data-aware controls like the MaskedEdit and 3DCheckBox controls are available in the Professional Edition and from third-party vendors.

Validation

Use the Validate event to perform last minute checks on the rows being written to the data source.

BOF/EOF Handling

The Custom Data control can also manage what happens when you encounter a data set with no rows. By changing the EOFAction property, you can program the Custom Data control to enter AddNew mode automatically.

You can program the Custom Data control to automatically snap to the top or bottom of its parent form by using the Align property. In either case, the Custom Data control is resized horizontally to fill the width of its parent form whenever the parent form is resized. This property allows a Custom Data control to be placed on an MDI form without requiring an enclosing Picture control.

Properties

(About) *	Align	Appearance *
BackColor *	BOF *	BOFAction *
Caption	CFields *	Container
(Custom) *	DataSourceName *	DateCreated *
EditMode *	Enabled	EOF *
EOFAction *	Font	ForeColor *
Height	hWnd *	Index
LastModified *	LastUpdated *	Left
MouseIcon *	MousePointer *	Name
Object	Orientation *	Parent
PercentPosition *	RecordCount *	RowNumber
ShowWhatsThis	Tag	Top
Updatable *	Visible	Width

Methods

AddNew *	CancelUpdate *	Delete *
Drag	Drag Icon	DragMode
Edit *	Move	MoveFirst *
MoveLast *	MoveNext *	MovePrevious *
MoveRelative *	Refresh *	Update *
UpdateControls *	UpdateRecord *	ZOrder

Events

AddRow *	DeleteRow *	DragDrop
DragOver	Error *	GetAttributes *
GetDefaultRow *	GetFields *	GetNumberOfRows *
GetRow *	MouseDown *	MouseMove *
MouseUp *	Open *	Reposition *
Requery *	Resize	UpdateRow *
Validate		

* Indicates a custom, modified, or stock property for the CCData control.

(About) Property

Applies To

CCData

Purpose

Provides access to version and copyright information for the Custom Data control at design time.

Comments

Double-click this property in the Visual Basic property sheet for the Custom Data control to display the About dialog box for the control.

(Custom) Property

Applies To

CCData

Purpose

Provides access to custom property sheets for the Custom Data control at design time.

Comments

Double-click this property in the Visual Basic property sheet for the Custom Data control to display the CFields custom property sheet for the control. The CFields custom property sheet provides an interface to easily set field information for the Custom Data control.

AddNew Method

Applies To

CCData

Purpose

Creates a new row in the set of rows maintained by the control.

Syntax

```
CCData1.AddNew
```

Return Type

None

Comments

The AddNew method creates a new CFields collection and then populates the fields of the new row with default values by firing the GetDefaultRow event. This method sets the value of each CField objects in the new CFields collection to the empty value. You can modify the value of CField objects for the new row using the CFields property.

After you modify the new row, use the Update method to save the changes and add the row to the set of rows maintained by the control. No changes are made to the data source until you use the Update method. Execution of the Update method after the AddNew method triggers the AddRow event procedure. This procedure contains the code that writes the new row to the data source.

If you edit a row and then perform any operation that moves to another row without using Update, your changes are lost without warning. In addition, if you close the form which contains the control, the new row and the changes made to it are discarded without warning.

The position of the new row depends on the type of data source being managed by the control and the code in the AddRow event procedure. Typically, rows are inserted at the end of the set of rows, regardless of any sorting or ordering rules.

The row that was current before you used AddNew remains current. If you want to make the new row current, you can set the RowNumber property to the row number identified by the LastModified property setting.

AddRow Event

Applies To

CCData

Purpose

Occurs when there is a request to add new row to the data set managed by the Custom Data control. This is a Data Source Interface event.

Syntax

```
Private Sub CCData1_AddRow(ByVal DataFields As CFields, RowNumber As Long)
```

Comments

Use the AddRow event procedure to code the behavior that adds a row to the data maintained by the Custom Data control. This event occurs in response to an AddNew method followed by Update method, or when a bound control, such as the DBGrid, adds a new row.

DataFields	Provides access to the CFields collection containing the values for a new row as set using the AddNew method. Copy these values to your to the data source managed by the Custom Data control.
RowNumber	Represents the row number of the new row. You must assign this parameter the number of the new row. The number must be unique.

Appearance Property

Applies To

CCData

Purpose

Returns or sets the paint style of controls on an MDIForm or Form object. This property is applies only to the 32-bit version of the Custom Data control.

Syntax

```
CCData1.Appearance
```

Data Type

Integer

Usage

Read/Write at design time.
Read-only at run-time.

Comments

The Appearance property settings are;

Setting	Description
0	Flat the control is painted without 3D visual effects.
1	3D (Default) the control is painted 3D visual effects.

If you set this property to 1 (3D) at design time, VB draws the control with three-dimensional effects.

If you set this property to 0 (Flat, the control is drawn with a flat appearance.

BackColor, ForeColor Property

Applies To

CCData

Purpose

Returns or sets the background color of the caption area of the control. Returns or sets the foreground color of the caption text in the control.

Syntax

CCData1.BackColor [= *color*]

CCData1.ForeColor [= *color*]

Data Type

OLE_COLOR

Usage

Read/Write at design time and run-time.

Comments

Visual Basic uses the red-green-blue (RGB) color scheme of the Microsoft Windows operating environment. The color settings are;

Setting	Description
Normal RGB colors	Colors referenced from the Color palette or specified with the RGB or QBColor functions in code.
System default colors	Colors specified by system color constants jprovided in the Visual Basic (VB) object library. You can view these constants in the Object Browser. The Windows operating environment maps the user's selections as specified in the Control Panel settings.

The system default color specified by the constant vbWindowBackground is the default settings at design time for the BackColor property. The default setting for the ForeColor property is the system default color specified by the constant vbWindowText.

The displayed color changes immediately when you set the BackColor and ForeColor properties.

The valid range for a normal RGB color is 0 to 16,777,215 (&HFFFFFF). The high byte of a number in this range equals 0; the lower 3 bytes, from least to most significant byte, determine the amount of red, green, and blue, respectively. The red, green, and blue components are each represented by a number between 0 and 255 (&HFF). If the high byte isn't 0, Visual Basic uses the system colors, as defined in the user's Control Panel settings and by constants listed in the Visual Basic (VB) object library in the Object Browser.

To display text in the Windows, you specify solid colors for both the text and background. If the text or background colors do not appear on the display, one of the selected colors may be comprised of up to three different-colored pixels (dithered). Windows substitues the nearest solid color when you specify a dithered color for either the text or background.

BOF, EOF Property

Applies To

CCData

Purpose

Returns a value indicating whether the current row position is before the first row in the set of rows maintained by the control. Returns a value indicating whether the current row position is after the last row in the set of rows maintained by the control.

Syntax

```
CCData1.BOF

CCData1.EOF
```

Data Type

Boolean

Usage

Read-only at run-time.

Comments

The BOF property values are;

Setting	Description
True	The current row position is before the first row.
False	The current row position is on or after the first row.

The EOF property values are;

Setting	Description
True	The current row position is after the last row.
False	The current row position is on or before the last row.

Use the BOF and EOF properties to determine whether the set of rows maintained by the control is empty or whether you have navigated beyond the beginning or end of the set of rows. Typically, when you work with all the rows in a data set, your code will loop through the rows using MoveNext until the EOF property is set to True.

The location of the current pointer determines the values of the BOF and EOF properties. A True value for either BOF or EOF indicates that there is no current row.

If there are no rows in the controls data source, the BOF and EOF properties are True, and the RecordCount property setting is 0.

When there is at least one row in the data set, the first row is the current row and BOF and EOF are False; they remain False until you move beyond the beginning or end of the data set using the MovePrevious or MoveNext method, respectively. When you move beyond the beginning or end of the data set, there is no current row.

If you delete the last remaining row in the controls data set, BOF and EOF remain False until you change the current row.

If you use the MoveLast method and the controls data set contains rows, the last row becomes the current row; if you then use the MoveNext method, the current row becomes invalid and EOF is set to True. Conversely, if you use the MoveFirst method and the controls data set contains rows, the first row becomes the current row; if you then use the MovePrevious method, there is no current row and BOF is set to True.

An error occurs when you use MoveNext while EOF is True or MovePrevious while BOF is True.

The following table shows how the Move... methods work with different value combinations in BOF and EOF.

	MoveNext/ MoveRelative(n > 0)	**MovePrevious/ MoveRelative(n < 0)**	**MoveFirst/ MoveLast/ MoveRelative(n = 0)**
BOF/Not EOF	OK	Error	OK
Not BOF/EOF	Error	OK	OK
Not BOF/Not EOF	OK	OK	OK
BOF/EOF	Error	Error	Error

Allowing a Move method doesn't mean that the method will successfully locate a row. It merely indicates that an attempt to perform the specified Move method is allowed and won't generate an error. The state of the BOF and EOF flags may change as a result of the attempted Move.

The Requery method internally invokes a MoveFirst. Therefore, invoking the Requery method on an empty data set results in BOF and EOF being set to True. Invoking the Requery method on a data set with one or more rows results in BOF and EOF being set to False.

All Move methods that successfully locate a row clear (set to False) both BOF and EOF.

An AddNew method followed by an Update method that successfully inserts a new row will clear BOF, but only if EOF is already set to True. The state of EOF will always remain unchanged.

Any ~~Delete~~ method, even if it removes the only remaining row from the controls data set, won't change the setting of BOF or EOF.

The following table shows the effect of Move methods that don't locate a row on the BOF and EOF property settings.

	BOF	**EOF**
MoveFirst / MoveLast	True	True
MoveRelative(n = 0)	No change	No change
MovePrevious / MoveRelative(n < 0)	True	No change
MoveNext / MoveRelative(n > 0)	No change	True

BOFAction, EOFAction Property

Applies To

CCData

Purpose

Returns or sets a value indicating what action the control should take when the current row is the first row of the data set and a Move Previous operation is performed using the CCData control buttons. Returns or sets a value indicating what action the control should take when the current row is the last row of the data set and a Move Next operation is performed using the CCData control buttons.

Syntax

CCData1.BOFAction

CCData1.EOFAction

Data Type

Integer

Usage

Read/Write at design time and at run-time.

Comments

The BOFAction property settings are;

Setting	Value	Description
cdcMoveFirst	0	MoveFirst (Default) Keeps the first row as the current row.
cdcBOF	1	BOF Moving past the beginning of a set of rows triggers the Custom Data control's Validate event on the first row, followed by a Reposition event on the invalid (BOF) row. At this point, the Move Previous button on the Custom Data control is disabled.

The EOFAction property settings are;

Setting	Value	Description
cdcMoveLast	0	MoveLast (Default) Keeps the last row as the current row.
cdcEOF	1	EOF Moving past the end of a set of rows triggers the Custom Data control's Validation event on the last row, followed by a Reposition event on the invalid (EOF) row. At this point, the MoveNext button on the Custom Data control is disabled.
cdcAddNew	2	AddNew Moving past the last row triggers the Custom Data control's Validation event to occur on the current row, followed by an automatic AddNew, followed by a Reposition event on the new row.

The Custom Data control library contains definitions for these constants. You can view the structures defined in the Custom Data control library using the Object Browser.

When you use code to manipulate data set managed by the control, the EOFAction property has no effect. It only applies to navigation using the buttons of the Custom Data control.

If you set the EOFAction property to cdcAddNew, once the user moves the current row pointer to EOF using the Data control, the current row is positioned to a new row in the copy buffer. At this point you can edit the newly added row. If you make changes to the new row and the user subsequently moves the current row pointer using the Data control, the row is automatically appended to the data set. If you don't make changes to this new row, and reposition the current row to another row, the new row is discarded. If you use the control to position to the next row while positioned over this new row, another new row is created.

In situations where the controls data set is returned with no rows, or after the last row has been deleted, using the cdcAddNew option for the EOFAction property greatly simplifies your code because a new row is always editable as the current row.

CancelUpdate Method

Applies To

CCData

Purpose

Cancels any pending updates on the controls data set.

Syntax

CCData1.CancelUpdate

Return Type

None

Comments

The CancelUpdate method cancels any pending updates due to an Edit or AddNew operation. For example, if a user invokes the Edit or AddNew method and hasn't yet invoked the Update method, CancelUpdate cancels any changes made after Edit or AddNew was invoked.

NOTE Using the CancelUpdate method has the same effect as moving to another row without using the Update method except that the current row doesn't change, and various properties, such as BOF and EOF, aren't updated.

Use the EditMode property to determine if there is a pending operation that can be canceled.

Caption Property

Applies To

CCData

Purpose

Returns and sets the text displayed in the caption area of the control.

Syntax

CCData1.Caption [= *string*]

Data Type

String

Usage

Read/Write at design time and run time.

Comments

When you create a new control, its default caption is the default Name property setting. This default caption includes the control name and an integer (ie; Control1). For a more descriptive label, set the Caption property.

The caption is limited to 255 characters.

CFields Property

Applies To

CCData

Purpose

Contains the current collection of field (CField) objects for the data control.

Syntax

CCData1.CFields

Data Type

CFields

Usage

Read/Write at design time (only using Custom property sheets).
Read and write at run-time. (CField objects can only be defined and added to the CFields collection in the GetFields event procedure.)

Comments

A CFields collection contains a CField object for each field of the current row of the controls data set. You can use the CField objects in the CFields collection to read and set values of the fields in the current row of the data set. You normally set the value of a CField object during an update operation begun by the Edit or AddNew methods.

You can refer to the Value property of a CField object by its Name property setting using this syntax;

```
Control.CFields(name)
```

or

```
Control.CFields!name
```

CFields that have names with embedded spaces must be delimited with brackets as in [Part Number] or enclosed in single (' ') or double (" ") quotation marks, as in 'Part Number' or "Part Number". For example;

```
Control.CFields![long name]
```

or

```
Control.CFields("long name")
```

You can also refer to the Value property of a CField object by its position in the CFields collection using this syntax;

```
Control.CFields(0)
```

DataSourceName Property

Applies To

CCData

Purpose

Returns the name of the data source.

Syntax

CCData1.DataSourceName [= *string*]

Data Type

String

Usage

Read-only at run-time. (You can set the value of this property in the GetAttributes event procedure).

Comments

Since the Custom Data control requires you to manage data yourself and to populate the CFields property with a collection of fields representing the current row, the DataSourceName for the Custom Data control is optional. Some data sources may not have a name, in which case this property contains an empty string.

DateCreated, LastUpdated Property

Applies To

CCData

Purpose

Returns the date and time the controls data set was created.

Syntax

CCData1.DateCreated [= *date*]

CCData1.LastUpdated [= *date*]

Data Type

Date

Usage

Read-only at run-time. (You can set the value of these properties in the GetAttributes event procedure).

Comments

Since the Custom Data control requires you to manage data yourself and to populate the CFields property with a collection of fields representing the current row, the DateCreated and LastUpdate fields for the Custom Data control are optional.

Delete Method

Applies To

CCData

Purpose

Deletes the current row from the controls data set.

Syntax

CCData1.Delete

Return Type

None

Comments

To use the Delete method, there must be a current row in the controls data set; otherwise, a trappable error occurs.

The Delete method removes the current row and makes it inaccessible. Although you can't edit or use the deleted row, it remains current. Once you move to another row, however, you can't make the deleted row current again. Subsequent references to a deleted row in the controls data set are invalid and produce an error.

DeleteRow Event

Applies To

CCData

Purpose

Executes in response to Delete method. This is a Data Source Interface event.

Syntax

```
Private Sub CCData1_DeleteRow(ByVal RowNumber As Long)
```

Comments

Use the DeleteRow event procedure to code the behavior that deletes a row from the data maintained by the Custom Data control.

The RowNumber parameter specifies the row number to delete.

Edit Method

Applies To

CCData

Purpose

Copies the current row from the controls data set to the copy buffer for subsequent editing. You can set the Value property for each CField object in the CFields collection.

Syntax

CCData1.Edit

Return Type

None

Comments

Once you use the Edit method, changes made to the current row's fields are copied to the copy buffer. After you make the desired changes to the row, use the Update method to save your changes.

NOTE If you edit a row and then perform any operation that moves to another row without first using Update, your changes are lost without warning. In addition, if you close the form that the control is on, your edited row is discarded without warning.

The current row remains current after you use Edit.

To use Edit, there must be a current row. If there is no current row, an error occurs.

Using Edit produces an error under the following conditions;

- There is no current row.
- The data set is read-only.
- No fields in the row are Updatable.
- Another user is currently editing the current row.

EditMode Property

Applies To

CCData

Purpose

Returns a value that indicates the state of editing for the current row.

Syntax

CCData1.EditMode

Data Type

Integer

Usage

Read-only at run-time.

Comments

The EditMode property values are;

Setting	Value	Description
cdcEditNone	0	No editing operation is in progress.
cdcEditInProgress	1	The Edit method has been invoked, and the current row is in the copy buffer.
cdcEditAdd	2	The AddNew method has been invoked, and the current row in the copy buffer is a new row that hasn't been saved.

The Custom Data control library contains definitions for these constants. You can view the structures defined in the Custom Data control library using the Object Browser.

Enabled Property

Applies To

CCData

Purpose

Enables or disables the control for user input.

Syntax

CCData1.Enabled **[**= *boolean***]**

Data Type

Boolean

Usage

Read/Write at design time and run-time.

Comments

The Enabled property settings are;

Setting	Description
True	(Default) Allows the control to respond to events.
False	Prevents the control from responding to events.

Use the Enabled property to enabled or disabled a control at run time. For example, you can disable a Custom Data control if there is a problem locating or reading the data source.

When you disable the Custom Data control, the navigation buttons of the control are greyed out and do not respond to user input.

Error Event

Applies To

CCData

Purpose

Occurs in response to a run-time error involving bound controls.

Syntax

```
Private Sub CCData1_Error(DataErr As Integer, Response As Integer)
```

Comments

Use the Error event procedure to handle data access errors. The Error event procedure has the following parameters:

DataErr	Specifies an integer representing the type of data error that occurred.
Response	Specifies an integer which determines the response to the data error.

Valid responses to an error are;

Constant	Value	Description
cdcErrContinue	0	Continue.
cdcErrDisplay	1	(Default) Display the error message.

The Custom Data control library contains definitions for these constants. You can view the structures defined in the Custom Data control library using the Object Browser.

You usually provide error-handling functionality for run-time errors in your code. However, run-time errors can occur in the following situations when none of your code is running:

- A user clicks a CCData control button.
- The CCData control automatically opens a database and loads a set of rows object after the Form_Load event.
- A bound control performs an operation such as the navigating, the adding, or the deleting rows.

If an error results from one of these actions, the Error event occurs. If you don't code an event procedure for the Error event, Visual Basic displays the message associated with the error.

You can search for the error constants in the Object Browser.

Font Property

Applies To

CCData

Purpose

Provides access to the Font object used by the control to draw caption. Use the properties of the Font object to return or set the font attributes of the caption text.

Syntax

CCData1.Font.*property* **[**= *value***]**

Data Type

IFontDisp

Usage

Read/Write at design time and run-time.

Comments

Use the Font property to access font attributes for the caption text of the Custom Data control. For example, the following code changes the Bold property setting of the controls Font object:

```
Control.Font.Bold = True
```

GetAttributes Event

Applies To

CCData

Purpose

Occurs after the Custom Data control opens or when the Refresh method executes. This is a Data Source Interface event.

Syntax

```
Private Sub CCData1_GetAttributes(DataSourceName As String, DateCreated As
Date, LastUpdated As Date, Updatable As Boolean)
```

Comments

Use the GetAttributes event procedure to set the values of the DataSourceName, DateCreated, LastUpdated, and Updatable properties of the Custom Data control. Values for the DataSourceName, DateCreated, and LastUpdated parameters are optional. The default value of the Updatable parameter is False. You must set the Updatable parameter to True to update data access by the Custom Data control.

GetDefaultRow Event

Applies To

CCData

Purpose

Occurs when a new row is added to the data set managed by the Custom Data control. This is a Data Source Interface event.

Syntax

```
Private Sub CCData1_GetDefaultRow(ByVal DataFields As CFields)
```

Comments

Use the GetDefaultRow event procedure to supply default values for a newly created row for the data set managed by the Custom Data control.

The DataFields parameter is a CFields collection which represents the field values of the new row. You must set the Value property of any CField objects in this collection which you wish to assign a default value.

This event occurs each time a new row is added to a data set so that you can handle calculated field values.

NOTE The GetDefaultRow event occurs at the start of an update operation that adds a new row. If that update operation is subsequently canceled, the supplied default values are discarded. This may effect the values of calculated/sequential fields. If the update operation is not canceled the AddRow event is fired.

GetFields Event

Applies To

CCData

Purpose

Occurs after the Custom Data control opens or after the Requery event occurs in response to the execution of the Refresh method. This is a Data Source Interface event.

Syntax

```
Private Sub CCData1_GetFields(ByVal DataFields As CFields)
```

Comments

Use the GetFields event procedure to define the schema of the data the Custom Data control maintains.

The DataFields parameter represents the CFields collection contained in the CFields property. You must create a new CField object for each field, set its properties, and then Append it to the CFields collection specified by the DataFields parameter. You must set the Name and Type properties of each CField object, you may also set the DataUpdatable property.

NOTE You can set the properties of the CField objects and the CFields collection at design time in the Custom property sheet for the Custom Data control. If you define the fields at design time using the CFields custom property sheet for the control, the GetFields event will not fire.

For more information about CField objects and the CFields collection, see the reference entries for those objects.

GetNumberOfRows Event

Applies To

CCData

Purpose

Occurs when the Custom Data control needs to determine the number of rows in the data set managed by the control. This is a Data Source Interface event.

The following actions may cause the GetNumberOfRows event to occur:

- Reading the PercentPosition property.
- Reading the RecordCount property.
- Executing the MoveLast method. The MustBeExact parameter is True when the GetNumberOfRows event occurs in response to a MoveLast method.
- Scrolling down and other actions in an advanced bound control (such as a grid) that is bound to the Custom Data control.

Syntax

```
Private Sub CCData1_GetNumberOfRows(ByVal MustBeExact As Boolean,
NumberOfRows As Long)
```

Comments

Use the GetNumberOfRows event procedure to set the number of rows managed by the Custom Data control. The GetNumberOfRows event procedure has the following parameters:

MustBeExact	Specifies whether or not the value in the NumberOfRows parameter must be exact. When MustBeExact is False, you must return the number of rows which have been fetched from the data set so far, but you do not need an exact count of rows in the data set.
NumberOfRows	Specifies the number of rows managed by the Custom Data control. DO NOT set the NumberOfRows to a number that is smaller than the value already contained in the parameter.

Once a GetNumberOfRows event occurs with the MustBeExact parameter set to True, the GetNumberOfRows event will not occur again.

GetRow Event

Applies To

CCData

Purpose

Occurs when the current row of the Custom Data control changes. The current row of the Custom Data control changes when a user clicks one of the direction buttons of the data control or when one of the Move... methods executes. This is a Data Source Interface event.

Syntax

```
Private Sub CCData1_GetRows(RowNumber As Long, DataFields As CFields,
NoSuchRow As Boolean)
```

Comments

Use the GetRows event procedure to return the field values of the requested row. The GetRows event procedure has the following parameters:

RowNumber	Specifies the row number of the row to fetch. The number for the first row in a data set is 0.
DataFields	Specifies a CFields collection. Set the Value property for each CField object in the collection to the field values of the row from the data set as specified by the RowNumber parameter.
NoSuchRow	Set this parameter to True if the row number specified in the RowNumber parameter does not exist in the data set. The parameter is False by default.

hWnd Property

Applies To

CCData

Purpose

Returns the Windows handle of the Custom Data control.

Syntax

```
CCData1.hWnd
```

Data Type

OLE_HANDLE

Usage

Read-only at run-time.

Comments

The Microsoft Windows operating environment identifies forms and controls in an application with a unique handle, or hWnd. Use the hWnd property to pass the handle of a control to Windows API calls.

NOTE The value of the hWnd property for a control can change while a program is running. Do

not store a hWnd value in a variable.

LastModified Property

Applies To

CCData

Purpose

Contains a row number of the most recently added or changed row.

Syntax

CCData1.LastModified

Data Type

Variant

Usage

Read-only at run-time.

Comments

Use this property to move to the most recently added or updated row. See the description of the RowNumber property for more information on using row numbers.

MouseDown, MouseUp Events

Applies To

CCData

Purpose

Occur when the user presses (MouseDown) or releases (MouseUp) a mouse button.

Syntax

```
Private Sub CCData1_MouseDown(button As Integer, shift As Integer, x As
Single, y As Single)

Private Sub CCData1_MouseUp(button As Integer, shift As Integer, x As
Single, y As Single)
```

Comments

The MouseDown and MouseUp event procedures have the following parameters:

button	Returns an integer identifying the button that was pressed (MouseDown) or released (MouseUp), causing the event. This argument is a bit field with bits corresponding to the left button (bit 0), right button (bit 1), and middle button (bit 2). These bits correspond to the values 1, 2, and 4, respectively. Only one of the bits is set at a time, indicating the button that caused the event.
shift	Returns an integer corresponding to the state of the SHIFT, CTRL, and ALT keys when the event occurred. A bit is set if the key is down. The shift argument is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. The shift argument indicates the state of these keys. None, some, or all of the bits can be set, indicating that none, some, or all of the keys are pressed. For example, if both CTRL and SHIFT were pressed, the shift value would be 3.
x, y	Returns a number specifying the current location of the mouse pointer. The x and y values are always expressed in terms of the coordinate system of the object.

Use a MouseDown or MouseUp event procedure to specify actions that will occur when a given mouse button is pressed or released. Unlike the Click and DblClick events, MouseDown and MouseUp events enable you to distinguish between the left, right, and middle mouse buttons. You can also use these events to write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keyboard modifiers.

Use the following constants to test for the button or shift arguments:

Constant	Value	Description
vbLeftButton	1	Left button is pressed.
vbRightButton	2	Right button is pressed.
vbMiddleButton	4	Middle button is pressed.

vbShiftMask	1	SHIFT key is pressed.
vbCtrlMask	2	CTRL key is pressed.
vbAltMask	4	ALT key is pressed.

The VB library contains definitions for these constants. You can view the structures defined in the this library using the Object Browser.

These constants then act as bit masks that you can use to test for any combination of buttons without having to figure out the unique bit field value for each combination.

NOTE You can use a MouseMove event procedure to respond to an event caused by moving the mouse. The button argument for MouseDown and MouseUp differs from the button argument used for MouseMove. For MouseDown and MouseUp, the button argument indicates exactly one button per event; for MouseMove, it indicates the current state of all buttons.

MouseIcon Property

Applies To

CCData

Purpose

Sets a custom mouse icon.

Syntax

```
CCData1.MouseIcon = LoadPicture(pathname)

CCData1.MouseIcon = picture
```

Data Type

IPictureDisp

Usage

Read/Write at design time and run-time.

Comments

The MouseIcon property provides a custom icon that is used when the MousePointer property is set to 99 (Custom).

Although Visual Basic 4.0 does not create cursor (.CUR) files, you can use the MouseIcon property to load either cursor or icon files. This provides your program with easy access to custom cursors of any size, with any desired hot spot location. The 32-bit version of Visual Basic does not load animated cursor (.ANI) files, even though 32-bit versions of Windows support these cursors.

MouseMove Event

Applies To

CCData

Purpose

Occurs when the user moves the mouse.

Syntax

```
Private Sub CCData1_MouseMove(button As Integer, shift As Integer, x As
Single, y As Single)
```

Comments

The MouseMove event procedure has the following parameters:

button	An integer that corresponds to the state of the mouse buttons in which a bit is set if the button is down. The button argument is a bit field with bits corresponding to the left button (bit 0), right button (bit 1), and middle button (bit 2). These bits correspond to the values 1, 2, and 4, respectively. It indicates the complete state of the mouse buttons; some, all, or none of these three bits can be set, indicating that some, all, or none of the buttons are pressed.
shift	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys. A bit is set if the key is down. The shift argument is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. The shift argument indicates the state of these keys. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT were pressed, the value of shift would be 6.
x, y	A number that specifies the current location of the mouse pointer. The x and y values are always expressed in terms of the coordinate system set by the ScaleHeight, ScaleWidth, ScaleLeft, and ScaleTop properties of the object.

The MouseMove event is generated continually as the mouse pointer moves across objects. Unless another object has captured the mouse, an object recognizes a MouseMove event whenever the mouse position is within its borders.

If you need to test for the button or shift arguments, use the following constants:

Constant	Value	Description
vbLeftButton	1	Left button is pressed.
vbRightButton	2	Right button is pressed.
vbMiddleButton	4	Middle button is pressed.
vbShiftMask	1	SHIFT key is pressed.
vbCtrlMask	2	CTRL key is pressed.
vbAltMask	4	ALT key is pressed.

The VB library contains definitions for these constants. You can view the structures defined in this library using the Object Browser.

The constants then act as bit masks you can use to test for any combination of buttons without having to figure out the unique bit field value for each combination.

You test for a condition by first assigning each result to a temporary integer variable and then comparing the button or shift arguments to a bit mask. Use the And operator with each argument to test if the condition is greater than zero, indicating the key or button is pressed, for example:

```
LeftDown = (Button And cdcLeftButton) > 0
CtrlDown = (Shift And cdcCtrlMask) > 0
```

Then, in a procedure, you can test for any combination of conditions, for example:

```
If LeftDown And CtrlDown Then
```

NOTE You can use MouseDown and MouseUp event procedures to respond to events caused by pressing and releasing mouse buttons.

The button argument for MouseMove differs from the button argument for MouseDown and MouseUp. For MouseMove, the button argument indicates the current state of all buttons; a single MouseMove event can indicate that some, all, or no buttons are pressed. For MouseDown and MouseUp, the button argument indicates exactly one button per event.

Any time you move a window inside a MouseMove event, it can cause a cascading event. MouseMove events are generated when the window moves underneath the pointer. A MouseMove event can be generated even if the mouse is perfectly stationary.

MousePointer Property

Applies To

CCData

Purpose

Returns or sets the type of mouse pointer displayed when the mouse cursor is over the Custom Data control at run time.

Syntax

CCData1.MousePointer [= *integer*]

Data Type

Integer

Usage

Read/Write at design time and run-time.

Comments

The MousePointer property settings are;

Setting	Description
0	(Default) Shape determined by the object.
1	Arrow.
2	Cross (cross-hair pointer).
3	I-Beam.
4	Icon (small square within a square).
5	Size (four-pointed arrow pointing north, south, east, and west).
6	Size NE SW (double arrow pointing northeast and southwest).
7	Size N S (double arrow pointing north and south).
8	Size NW SE (double arrow pointing northwest and southeast).
9	Size WE (double arrow pointing west and east).
10	Up Arrow.
11	Hourglass (wait).
12	No Drop.
13	Arrow and hourglass. (Only available in 32-bit Visual Basic.)
14	Arrow and question mark. (Only available in 32-bit Visual Basic.)
15	Size all. (Only available in 32-bit Visual Basic.)
99	Custom icon specified by the MouseIcon property.

Use this property when you want to indicate changes in functionality as the mouse pointer passes over the control. For example, the Hourglass setting (11) usually indicates that the user should wait for a process or operation to finish.

MoveFirst, MoveLast, MoveNext, MovePrevious Methods

Applies To

CCData

Purpose

Moves to the first, last, next, or previous row in the data set managed by the Custom Data control. and makes that row the current row.

Syntax

CCData1.{MoveFirst|MoveLast|MoveNext|MovePrevious}

Return Type

None

Comments

Use the Move methods to move from row to row based on the row sequence in the data set.

NOTE If the user is editing the current row, use the Update method before moving to save the changes. If you allow the user to move to another row without executing an Update method, all changes will be lost without warning..

Initially, the first row is current and the BOF property is set to False. If the controls data set contains no rows, the BOF property is set to True, and there is no current row.

If the first row is current when you use MoveFirst, the current row does not change. If the last row is current when you use MoveLast, the current row does not change.

If the BOF property is True and you use MovePrevious when the first row is current, there is no current row. If you use MovePrevious again, an error occurs and BOF remains True.

If the EOF property is True and you use MoveLast when the last row is current, there is no current row. If you use MoveLast again, an error occurs and EOF remains True.

The behavior set by the BOFAction and EOFAction property values do not apply to row navigation with the Move... methods. The BOFAction and EOFAction properties only affect row navigation behavior using the buttons of the Custom Data control.

Use the MoveRelative method to move to a row that is a specific number of rows forward or backward from the current row in the data set.

Using the MoveLast method may cause the GetNumberOfRows event to occur with the MustBeExact parameter set to True. In this case, the event procedure must return the exact number of rows in the data set managed by the Custom Data control. All of the Move... methods cause the GetRow event to occur. The code of the GetRow event procedure must locate the appropriate row and returns the CFields collection containing the row values.

MoveRelative Method

Applies To

CCData

Purpose

Moves a specified number of rows forward or backward from the current row in the data set managed by the Custom Data control.

Syntax

```
ReturnValue = Object.MoveRelative(NumberOfRows)
```

Return Type

None

Comments

This method causes a GetRow event to occur. The code of the GetRow event procedure must locate the appropriate row and return the CFields collection containing the row values.

Use a positive value in the NumberOfRows parameter to move forward in the data set or a negative value to move backward in the data set.

If the EOF property is True and you attempt to move past the last row in the data set, there is no current row. If the BOF property is True and you attempt to move before the first row in the data set, there is no current row.

Open Event

Applies To

CCData

Purpose

Occurs when the form containing the Custom Data control opens. This is a Data Source Interface event.

Syntax

```
Private Sub CCData1_Open()
```

Comments

Use the Open event procedure to prepare the data source of the Custom Data control for access within the DataSource event procedures. This event procedure is usually used to cache the rows of the data set in a data structure.

Orientation Property

Applies To

CCData

Purpose

Returns or sets the orientation, horizontal or vertical, of the Custom Data control.

Syntax

CCData1.Orientation [= *integer*]

Data Type

Integer

Usage

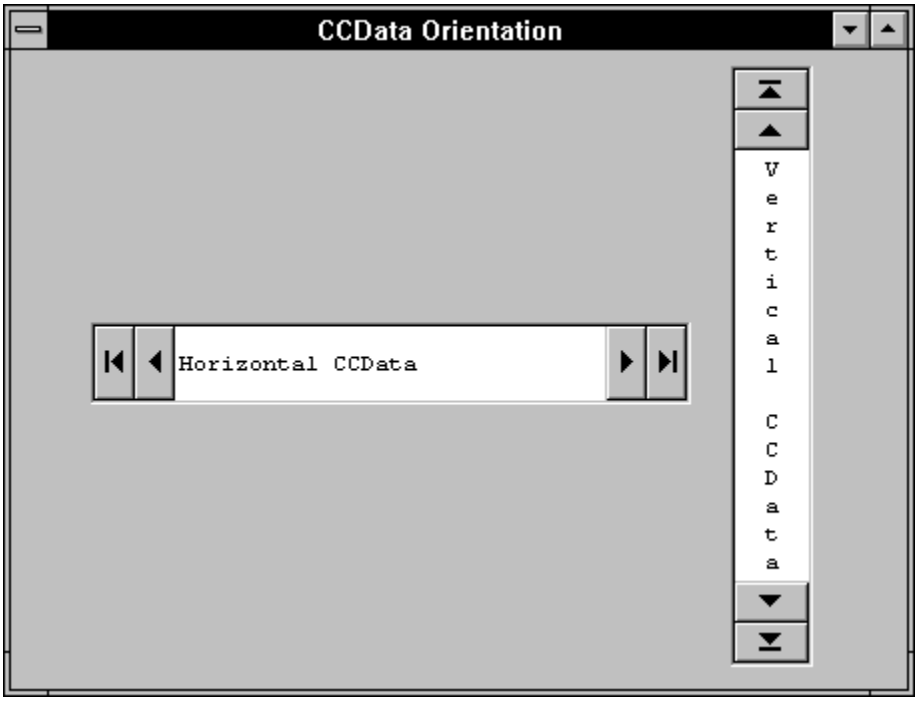
Read/Write at design time and run-time.

Comments

The Orientation property settings are;

Constant	Value	Description
cdcOrientationHorizontal	0	Horizontal - the control is drawn horizontally.
cdcOrientationVertical	1	Vertical - the control is drawn vertically.
cdcOrientationAutomatic	2	(Default) Automatic - the control is drawn horizontally if its Width >= Height, otherwise it is drawn vertically.

The Orientation property determines how the control will be drawn. The following figure illustrates the two orientations:



PercentPosition Property

Applies To

CCData

Purpose

Returns value indicating the approximate location of the current row in the number of known rows in the data set managed by the Custom Data control. Sets the current row to a new row in the data set managed by the Custom Data control based upon a specified percentage value relative to the number of known rows.

Syntax

CCData1.PercentPosition **[**= *value***]**

Data Type

Single

Usage

Read/Write at run-time.

Comments

The number of known rows in a data set is determined by the highest value assigned to the NumberOfRows parameter in the GetNumberOfRows event procedure.

If you use the PercentPosition property before navigating to the last row of the data set, the amount of movement is relative to the number of rows accessed by the Custom Data control. This number is indicated by the RecordCount property. For the percentage value to be relative to the total number of rows in the data set, you must first move to the last row in the data set before specifying a value for the PercentPosition property. Use the MoveLast method to move to the last row of the data set.

When you set the PercentPosition property to a value, the row closest to the specified percentage becomes current, and the PercentPosition property is reset to a value that reflects the approximate position of the current row. For example, if your data set contains only 10 rows, and you set its PercentPosition value to 57, the value returned from the PercentPosition property is 60, not 57.

NOTE Use the RowNumber property, instead of the PercentPosition property, to move to a specific row in the data set.

RecordCount Property

Applies To

CCData

Purpose

Returns the number of known rows in the data source managed by the Custom Data control.

Syntax

CCData1.RecordCount

Data Type

Integer

Usage

Read-only at run-time.

Comments

The number of known rows in a data set is determined by the highest value assigned to the NumberOfRows parameter in the GetNumberOfRows event procedure.

Reading the RecordCount property may trigger the GetNumberOfRows event. The RecordCount property returns the highest value assigned to the NumberOfRows parameter of the GetNumberOfRows event.

Refresh Method

Applies To

CCData

Purpose

Refreshes the data from the data source managed by the Custom Data control and makes the first row the current row.

Syntax

CCData1.Refresh

Return Type

None

Comments

The Refresh method causes the Requery, GetAttributes, GetFields, GetRow, and sometimes the GetNumberOfRows events to occur. The event procedure associated with the Requery event is responsible for refreshing the data from the data source.

Reposition Event

Applies To

CCData

Purpose

Occurs after the current row of the Custom Data control changes to a new row.

Syntax

```
Private Sub CCData1_Reposition()
```

Comments

The Reposition event occurs after the current row of the Custom Data control changes. This event occurs after a users clicks a Custom Data control button to navigate to a new row, after the execution of a Move... method, or after setting the value of a property or the execution of a method that changes the current row. When Visual Basic loads a Custom Data control, the GetRow event finds the first row from the data source triggering a Reposition event.

In contrast to the Reposition event, the Validate event occurs before moving to a new row.

Use this event to perform calculations based on data in the new current row or to change the form in response to data in the current row.

Requery Event

Applies To

CCData

Purpose

Occurs in response to the execution of the Refresh method. This is a Data Source Interface event.

Syntax

```
Private Sub CCData1_Requery()
```

Comments

Use the Requery event procedure to refresh the data from the data source managed by the Custom Data control. Under most circumstances, the processing that occurs in the Requery event procedure should be similar to the code of the Open event procedures. You can also use this event to update the working data set from a dynamic data source.

RowNumber Property

Applies To

CCData

Purpose

Returns the row number of the current row in the controls data set. When set, the RowNumber property causes the control to be repositioned to the row identified by the row number being assigned.

Syntax

CCData1.RowNumber **[**= *value***]**

Data Type

Long

Usage

Read/write at run time.

Comments

You can save the row number for the current row by assigning the value of the RowNumber property to a variable. To quickly return to that row at any time after moving to a different row, set the RowNumber property to the value of that variable.

If you set the RowNumber property to a value that represents a row which does not exist, a trappable error occurs.

To refresh the contents of a row, set the RowNumber property to itself. (This technique, however, also cancels any pending operations invoked by the Edit or AddNew methods.) For example:

```
CCData1.RowNumber = CCData1.RowNumber
```

Updatable Property

Applies To

CCData

Purpose

Returns a value indicating whether or not changes can be made to rows managed by the Custom Data control.

Syntax

CCData1.Updatable

Data Type

Boolean

Usage

Read-only at run-time. (You can set the value of this property in the GetAttributes event procedure).

Comments

The Updatable property values are;

Value	Description
True	Rows from the controls data source can be modified.
False	Rows from the controls data source can NOT be modified.

Update Method

Applies To

CCData

Purpose

Saves the contents of the edit buffer to the data source managed by the Custom Data control.

Syntax

CCData1.Update

Return Type

None

Comments

Use Update to save any modified values for the current row to the data source. The Update method triggers a Validate event and then an UpdateRow event.

The following actions cause the loss of modified values in the current row of the Custom Data control:

- Execute the Edit or AddNew method, and then move to another row without first executing the Update method.
- Execute Edit or AddNew, and then use Edit or AddNew again without first executing the Update method.
- Set the RowNumber property to another row.
- Close the form the control is on without first executing the Update method.

To edit a row, use the Edit method to copy the current row to the edit buffer. If you do not use Edit first, an error occurs when you use Update or change a field's value.

To add a new row to the controls data source, use the AddNew method.

UpdateControls Method

Applies To

CCData

Purpose

Gets the current row from the CFields collection of the Custom Data control and displays the data in controls bound to the data control.

Syntax

CCData1.UpdateControls

Return Type

None

Comments

Use this method to restore the contents of bound controls to their original values. This is useful when a user makes changes to data and then decides to cancel the changes.

This method creates the same effect as making the current row current again, except that no events occur.

UpdateRecord Method

Applies To

CCData

Purpose

Saves the current values of bound controls to the data source managed by the Custom Data control.

Syntax

```
CCData1.UpdateRecord
```

Return Type

None

Comments

Use this method to trigger an UpdateRow event. Use this method to save the contents of bound controls to the data source maintained by the Custom Data control during the Validate event, without triggering the Validate event again.

The UpdateRecord method is similar to executing the Edit method, changing a field, and then executing the Update method, however no events occur.

Use this method to avoid retriggering the Validate event.

UpdateRow Event

Applies To

CCData

Purpose

Executes in response to an Update or an UpdateRecord method. This is a Data Source Interface event.

Syntax

```
Private Sub CCData1_UpdateRow(ByVal RowNumber As Long, ByVal DataFields As CFields)
```

Comments

Use this event procedure to write changes made to the current row to the data source managed by the Custom Data control. The UpdateRow event procedure has the following parameters.

RowNumber	Represents the number of the row to be updated.
DataFields	Passes the CFields collection associated with the current row into the event procedure. Write these values to the data source associated with the Custom Data control.

In response to an Update method, this event procedure executes after the Validate event procedure.

Validate Event

Applies To

CCData

Purpose

Occurs before a navigation, update (except with the UpdateRecord method), delete, data source close, or form unload operation. The Validate event procedure executes prior to the operation to allow you to validate or cleanup the data in the current row of the Custom Data control.

Syntax

```
Private Sub CCData1_Validate(Action As Integer)
```

Comments

The Action parameter is an integer that indicates the operation that caused the Validate event. Valid values for the Action parameter are;

Constant	Value	Description
cdcActionCancel	0	Cancel the operation when the Sub exits.
cdcActionMoveFirst	1	MoveFirst method.
cdcActionMovePrevious	2	MovePrevious method.
cdcActionMoveNext	3	MoveNext method.
cdcActionMoveLast	4	MoveLast method.
cdcActionAddNew	5	AddNew method.
cdcActionUpdate	6	Update operation (not UpdateRecord).
cdcActionDelete	7	Delete method.
cdcActionBookmark	9	The Bookmark property has been set.
cdcActionClose	10	The Close method.
cdcActionUnload	11	The form is being unloaded.

The Custom Data control library contains definitions for these constants. You can view the structures defined in the Custom Data control library using the Object Browser.

This event occurs even if the field values of the current row are unchanged and no bound controls exist. Use the Validate event to change values and update the data of the current row. You can also save the current row or stop whatever action is causing the event and execute a different action.

You can set the Action parameter value in the Validate event procedure to convert one action into another. For example, you can change the behavior of the different Move... actions and AddNew action. When using AddNew, you can use MoveNext and then execute another AddNew to examine the EditMode property to determine if an Edit or AddNew operation is in progress.

You cannot change an AddNew or one of the Move actions into any of the other actions. This is either ignored or produces a trappable error. You can stop any action by setting action to 0.

Use the DataChanged property for each bound control to check for new or modified field data. If you set the DataChanged property of a bound control to False, any new or modified data in that control will not be saved to the underlying data set.

You cannot use any Custom Data control methods (such as MoveNext) during this event.

CFields Collection

Control Files

CCData16.OCX, CCData32.OCX

Object Type

CFields

Comments

A CFields collection contains all of the CField objects that represent the current row from a custom data source. The methods and properties of the CFields collection allow you to define the order and number of fields per row for a custom data source.

The CFields property of the Custom Data control contains a CField collection for a custom data source. This property also provides access to the field values of the current row from the custom data source through the Value property of each CField object in the collection.

Properties

Count

Methods

Append	Delete	Item

Append Method

Applies To

CFields

Purpose

Adds a new CField object to a CFields collection.

Syntax

CField.Append *CField*

Return Type

None

Comments

Use the Append method to add CField objects to the CFields collection in the GetFields event procedure for the Custom Data control.

Count Property

Applies To

CFields

Purpose

Returns the number of CField objects in a CFields collection.

Syntax

CFields.Count

Data Type

Long

Usage

Read-only at run-time.

Comments

Use this property to determine the number of fields in a row from a custom data source.

Members of a collection are numbered starting with 0, so loops are always coded starting with the 0 member; for example;

```
For I = 0 to CCData1.CFields.Count - 1
   ' Work with CCData1.CFields(I) here...
Next I
```

You can also use the For Each *object* In *collection* syntax to enumerate all members of a collection. For example;

```
Dim Fld As CField

For Each Fld In CCData1.CFields
   ' Work with Fld object here...
Next
```

If the Count property is 0, there are no CField objects in the collection.

Delete Method

Applies To

CFields

Purpose

Deletes a CField object from a CFields collection.

Syntax

CField.Delete *CFieldName*

Return Type

None

Comments

The *CFieldName* parameter specifies the name of the CField object to delete. The Name property of a CField object contains a string that uniquely identifies the object in the CFields collection.

Item Method

Applies To

CFields

Purpose

Returns the CField object at the specified index from a CFields collection.

Syntax

```
CCData.CFields.Item(n).Value
CCData.CFields.Item(Pn).Value
CCData.CFields(n).Value
CCData.CFields(Pn).Value
```

Data Type

CField object

Comments

- You can index an item either by offset , that is, 0 ... (n-1), or by name.
- If you specify an invalid index, you will receive the following error message:

```
Run-time error 9:Subscript out of range
```

- Specify an index that is in the valid range of indexes. The valid range is from 0 to the integer specified in the Count property.

 CField Object

Control Files

CCData16.OCX, CCData32.OCX

Object Type

CField

Comments

A CField object contains the definition for a single field in CFields collection that represents a row from a custom data source. The methods and properties of the CField object allow you to define the attributes and value of a field.

Properties

DataUpdatable	Name	Size
Type	Value	

Methods

FieldSize

DataUpdatable Property

Applies To

CField

Purpose

Determines whether or not the value for a CField object can be set outside of the GetDefaultRow event procedure or in an AddNew editting block.

Syntax

CField.DataUpdatable **[**= *boolean***]**

Data Type

Boolean

Usage

Read and write at run-time. Read-only for CField objects in the CFields collection.

Comments

The DataUpdatable property values are;

Value	Description
True	The value of the CField object can be modified.
False	The value of the CField object can NOT be set outside of the GetDefaultRow or in an AddNew editing block.

FieldSize Method

Applies To

CField

Purpose

Returns the number of bytes of a value in a CField object.

Syntax

```
longVar = CField.FieldSize()
```

Return Type

Long

Comments

Use the FieldSize property to determine the length of Text fields.

Name Property

Applies To

CField

Purpose

Returns or sets the name of CField object.

Syntax

CField.Name **[**= *string***]**

Data Type

String

Usage

Read and write at run-time. Read-only for CField objects in the CFields collection.

Comments

The Name string must uniquely identify the CField object. You cannot add two fields with the same Name string to a CFields collection.

Size Property

Applies To

CField

Purpose

Returns or sets the maximum size, in bytes, of a CField object.

Syntax

CField.Size [= *value*]

Data Type

Integer

Usage

Read and write at run-time. Read-only for CField objects in the CFields collection.

Comments

The Size value depends on the Type property setting of the CField object and can be one of the following:

Type Setting	Size Value	Description
cdcBoolean	1	True/False
cdcByte	1	Byte
cdcCurrency	8	Currency
cdcDate	8	Date/Time
cdcDouble	8	Double
cdcInteger	2	Integer
cdcLong	4	Long
cdcSingle	4	Single
cdcText	1-255	Text

The Custom Data control library contains definitions for these constants. You can view the structures defined in the Custom Data control library using the Object Browser.

When you create a CField object with a data type other than Text, the Type property setting automatically determines the Size property setting, and you don't need to set the Size property.

For a CField object with the Text data type, Size can be set to any integer up to 255 where the Size property sets the maximum size for the CField.

Type Property

Applies To

CField

Purpose

Returns or sets the data type for a value in a CField object.

Syntax

CField.Type [= *value*]

Data Type

Integer

Usage

Read and write at run-time. Read-only for CField objects in the CFields collection.

Comments

The settings for the Type property are;

Value	Setting	Description
1	cdcBoolean	Yes/No
2	cdcByte	Byte
3	cdcInteger	Integer
4	cdcLong	Long
5	cdcCurrency	Currency
6	cdcSingle	Single
7	cdcDouble	Double
8	cdcDate	Date/Time
10	cdcText	Text

NOTE A Long Binary data type is the same as a String. However, it's recommended that you use the Variant data type instead of Long Binary (OLE Object) data types because doing so enables you to store Nulls.

The Custom Data control library contains definitions for these constants. You can view the structures defined in the Custom Data control library using the Object Browser.

Value Property

Applies To

CField

Purpose

Returns or sets the value of a CField object.

Syntax

CField.Value [= *value*]

Data Type

Variant

Usage

Read and write at run-time.

Comments

Use the Value property to read and manipulate the data of a CField object in the current CFields collection maintained by the Custom Data control.

The Value property is the default property of the CField objects. Therefore, the following lines of code are equivalent (assuming CField1 is at the first ordinal position in the CFields collection);

```
X = CCData1.CFields!Field1
X = CCData1.CFields!Field1.Value
X = CCData1.CFields(0)
X = CCData1.CFields(0).Value
X = CCData1.CFields(Field1).Value
X = CCData1.CFields(Field1)
F$ = CField1
X = CCData1.CFields(F$).Value
X = CCData1.CFields(F$)
```

CFields Tab - Crescent Custom Data Control

Use this tab to define the schema information for the fields and rows managed by the Custom Data control. A row for the Custom Data control is a CFields collection. Each field in the row is a CField object. The Fields tab allows you to define these objects at design time, instead of at run-time in the GetFields event procedure.

Add Field

The text boxes in this frame allow you to define and add a CField object to the CFields collection associated with the current Custom Data control. You can specify the following property values for a CField object.

Name A unique string used to reference a CField object in the CFields collection. See the Name property of the CField object for more information.

Type The data type of the value contained in the CField object. See the Type property of the CField object for more information.

Size The maximum size, in bytes, of the CField object. See the Size property of the CField object for more information.

Add Add the current CField object definition in the **Add Fields** frame to the CFields collection listed in the **Fields in Fields Collection** list box.

Fields in Fields Collection

Lists the CFields object that are currently defined in the CFields collection associated with the current Custom Data control.

Delete Selected Fields

Deletes the currently highlighted CField objects listed in the **Fields in Fields Collection** list box.

OK

Apply changes to the CFields collection for the Custom Data control and exit the property sheet.

Cancel

Exit the property sheet without saving unapplied changes to the CFields collection of the Custom Data control.

Apply

Apply changes to the CFields collection.