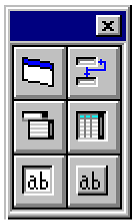
Vantage Control Set

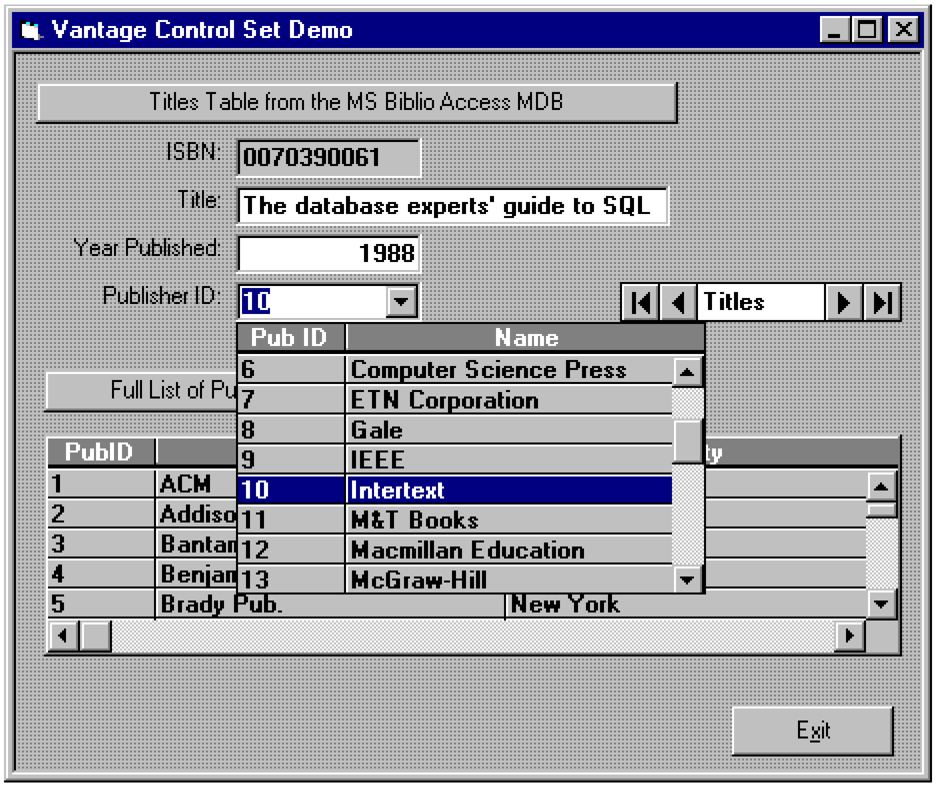
Version 2.0

Overview

Overview



Vantage Control Set is a library of custom controls for Visual Basic and other languages which supports ActiveX technology. This library of custom controls includes enhanced replacements for many of the standard controls as well as two new controls which help manage the user interface and operations of a program. The controls of this set are provided as ActiveX controls (32-bit OCXs) , and as VBX controls. The VBX version can be used in Visual Basic 3.0 and Visual Basic 4.0 16-bit projects. The ActiveX version will work in Visual Basic 4.0, Visual Basic 5.0, and other containers that support the ActiveX standard. The one exception is the VPFocus and VPForm controls which are designed to work with Visual Basic only.



Vantage Control Set Demo

Titles Table from the MS Biblio Access MDB

ISBN: 0070390061

Title: The database experts' guide to SQL

Year Published: 1988

Publisher ID: 10

Titles

Pub ID	Name
6	Computer Science Press
7	ETN Corporation
8	Gale
9	IEEE
10	Intertext
11	M&T Books
12	Macmillan Education
13	McGraw-Hill

Full List of Pu

PubID		ty
1	ACM	
2	Addiso	
3	Bantan	
4	Benjam	
5	Brady Pub.	New York

Exit

Vantage Control Set features a ...

VPTextBox TextBox Control - data-aware, 3D effects, data alignment options in either single or multiple line styles, and new Insert/Overtype modes of operation.

VPStatic Label Control - data-aware with custom 3D display effects.

VPComboBox ComboBox Control - data-aware, multiple columns, 3D effects, optional grid lines, column headings, output formatting, color settings, data alignment, custom sorting, matched entry property, locate data functions, standard unbound operations, independent sizing of edit and list portions, no 64K data limit.

VPListBox ListBox Control - data-aware, multiple columns, 3D effects, optional grid lines, column headings, output formatting, color settings, data alignment, custom sorting, locate data functions, standard unbound operations, no 64K data limit.

VPForm Control - used to control the appearance of a Form Object with 3D display effects and custom background properties. Also includes custom drag-and-drop operations.

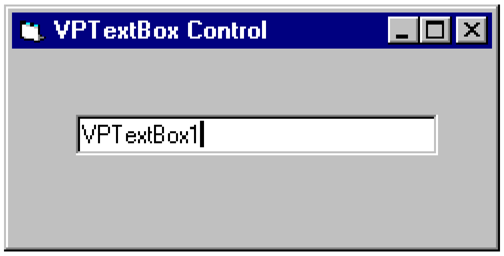
VPFocus Management Control - used to manage GotFocus and LostFocus events so you can do immediate data validation as **Focus** leaves a control. Allows Visual Basic's LostFocus event to work similar to the OnExit Control Property of Microsoft Access.

The replacement controls of this library go well beyond the capabilities of the standard Windows controls. While there are other control collections from 3rd-party vendors which provide enhanced versions of these same controls, none have the combination of features found within Vantage Control Set. With the Vantage Control Set we have tried to provide a set of controls that would typically be used in any programming project. The added functionality and utility associated with these controls are those that we simply could not get with the controls from either Microsoft or other 3rd-party vendors. This is not a large set of controls with the typical large price tag, but a small and useful set of workhorse controls which provide new and needed properties. They can be used to design a better user interface and provide increased functionality.

VPTextBox TextBox Control

Custom: Properties Standard: Properties Events Methods

A normal **TextBox** control, sometimes called an edit field or edit control, displays information entered at design time, entered by the user, or assigned to the control in code at run time. It is also data-aware. The **VPTextBox** custom control is similar to the standard **TextBox** control but has a number of added properties and features.



Syntax

VPTextBox

Remarks

To display multiple lines of text in a **VPTextBox** control, set the **MultiLine** property to **True**. If a multiple-line **VPTextBox** control doesn't have a horizontal scroll bar, text wraps automatically even when the **VPTextBox** control is resized. To customize the scroll bar combination on a **VPTextBox** control, set the **ScrollBars** property.

Unlike the normal **TextBox** control, you can use the **Alignment** property to set the alignment of text within the **VPTextBox** control, irrespective of the **MultiLine** property. The text is left-justified by default, but can be set to right or centered justification.

The **VPTextBox** control includes **Insert** and **Overtype** modes of operation. The **INS** key is used to toggle the modes of operation during run time. The default mode is the normal **Insert** behavior. Different carets are used as a visual cue to indicate the current mode of operation. There is a normal vertical line caret to indicate the insertion point when in the **Insert** mode, and an underscore caret to indicate the replacement position when in the **Overtype** mode.

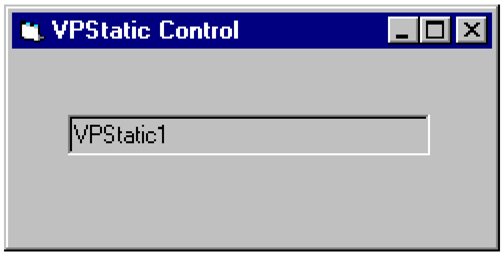
The **VPTextBox** control has a custom **Appearance** property that returns or sets the paint style of the control. This **Appearance** property includes 3D border effects with support for both **Inset** and **Raised** representations.

Another custom property of the **VPTextBox** control is the **Locked** property. This property returns or sets a value indicating whether a control can be edited. With this property a **VPTextBox** control can be placed into a display only mode of operation.

VPStatic Static or Label Control

Custom: Properties   Standard: Properties   Events   Methods

A **VPStatic** control is similar to the standard **Label** control you use to display text that a user can't change directly. As a data-aware non-edit control it can be bound to data sources as a "read-only" field. The VPStatic control is a regular window control, having a **hWnd** property unlike the standard **Label** control which is a graphical control. It also has a custom **Appearance** property for 3D display effects.



Syntax

VPStatic

Remarks

You can write code that changes the text displayed by a **VPStatic** control in response to events at run time. For example, if your application takes a few minutes to commit a change, you can display a processing-status message in a **VPStatic** control. You can also use the **VPStatic** control as a "read-only" or "display-only" database field when bound to a **Data** control. In addition you can use the **VPStatic** control to identify controls that don't have their own **Caption** properties.

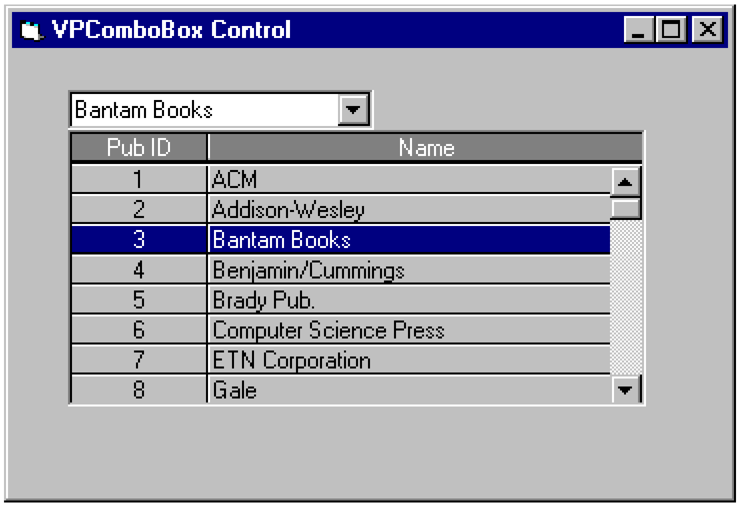
Set the **AutoSize** and **WordWrap** properties if you want the **VPStatic** control to properly display variable-length lines or varying numbers of lines.

The **VPStatic** control has a custom **Appearance** property that returns or sets the paint style of the control. This **Appearance** property includes 3D border effects with support for both **Inset** and **Raised** representations.

VPComboBox ComboBox Control

Custom: Properties Events Methods Functions Standard: Properties Events Methods

The **VPComboBox** control, like the standard **ComboBox** control, combines the features of a **Edit** or **Static** control and a **ListBox** control. Users can enter information in the Edit portion or select an item from the List portion of the control. The **VPComboBox** control has a number of added properties and features. These include being data-aware, having multi-columns, 3D display effects, optional grid lines, column headings, output formatting, color settings, data alignment, custom sorting, matched entry property, locate data functions, standard unbound operations, independent sizing of Edit and List portions, and no 64K data limit.



Pub ID	Name
1	ACM
2	Addison-Wesley
3	Bantam Books
4	Benjamin/Cummings
5	Brady Pub.
6	Computer Science Press
7	ETN Corporation
8	Gale

Syntax

VPComboBox

Remarks

Like the standard **ComboBox** control, the **VPComboBox** control has three styles or modes of operation controlled by the **Style** property. The default **Style** is a **Dropdown Combo** which includes a drop-down list and an Edit control. The user can select from the List or type in the Edit control portion. Another supported style is the **Simple Combo** where the List portion does not drop down, but is always displayed if the height of the **VPComboBox** control is sufficient. Like the **Dropdown Combo**, the user can select from the List or type in the Edit portion. The other style is the **Dropdown List**. This style only allows selection from the drop-down list and includes a Static control instead of an Edit control.

The Edit portion of the **VPComboBox** control has many of the custom properties and features of the **VPTextBox** custom control. This includes **Insert** and **Overtype** modes of operation as well as left, right or center justification of text.

The **VPComboBox** control has a number of custom properties for controlling the look of the control. The **Appearance** property can define either an **Inset** or **Raised** 3D display effect for the Edit or Static portion of the control. The **GridLines** and **GridAppearance** properties control the display for the List portion. There are also separate **ForeColor** and **BackColor** properties for the Edit or Static portion, the fixed Column Heading row of the List portion, and the other data rows of the List portion of the **VPComboBox** control.

A custom feature of the **VPComboBox** control is its support for defining and displaying multiple columns of information in the List portion of the control. These columns can be defined in code executed during run time, or before execution, during design time.

During run time, with code, you can set the number of columns and define their characteristics. The **MaxCols** property sets or returns the number of columns for the List portion. Setting this property to a number less than the current number of columns will delete the trailing columns from the List. Any columns added by setting this property to a higher number will create new columns with appropriate default values. The **Col** property allows a programmer to select a specific column. Once set, any column properties referenced or modified will be the array values for the specified column.

In design time, as part of the property sheet of the ActiveX version of the **VPComboBox** control, there is a tab for **Column layout** properties which is used to add, delete, or maintain column properties. In the VBX version there is a **Column Layout Properties dialog** which is used to maintain the same properties. This column layout dialog is called by clicking on the ellipse button that is part of the **(ColLayout)** property in the Visual Basic Properties Window or by double-clicking the **(ColLayout)** property, directly.

Shown below is a list of properties used to define columns in the List portion of the **VPComboBox** control. Array properties are italicized.

Property	Specifies
ColAlign	The alignment of text within a each column.
ColBound	Which column of a selected row in a List is bound directly to a data source.
ColFormat	An optional VB format string for each column. This property has the same effect and supports the same formats as the VB Format$ function.
ColHeading	The text for an optional heading or caption for each column.
ColHeadAlign	The alignment of the column heading or caption for each column.
ColLink	Which column of the List is linked to the Edit or Static portion of the control.
ColListField	Name of the **Field** object in the **Recordset** specified by the **RowSource** property bound to each column in the List.
ColSortBy	A Boolean flag which indicates if a given column is used for sorting the contents of the List.
ColSortOrder	Determines if a column of the List used in sorting is sorted in ascending or descending order.
ColWidth	The width in Twips of each column.

You will notice that there are properties for defining a column heading or caption row for the list portion of the **VPComboBox** control. Whether this control has a fixed heading row is determined by the Boolean **Heading** property.

The **VPComboBox** control has additional properties which define the colors used. The foreground and background colors for the fixed heading row are controlled by the **HeadingBackColor** and **HeadingForeColor** properties. The foreground and background colors for the data rows of the List are defined by the **ListBackColor** and **ListForeColor** properties. The foreground and background colors for the Edit or Static portion of the **VPComboBox** control are defined by the standard **BackColor** and **ForeColor** properties.

The **VPComboBox** control can be used as a bound control or used in unbound operations, equally well. Unbound, the control uses the same methods of the standard **ComboBox** control to populate the List portion. To add or delete items in the List portion, use the **AddItem** or **RemoveItem** method. Each string value to be added or updated to a specific row must use the **Tab** character, Chr$(09), as a delimiter between the columns of data. Set the **List**, **ListCount**, and **ListIndex** properties to enable a user to access items in the List portion of the control. With the **RowSource** and **ColListField** properties defined, the **VPComboBox** control automatically fills the List portion with fields from one **Data** control, and optionally passes data to the selected field of a second **Data** control, as defined in either the **ColDataSource** and **ColDataField** properties, or the standard **DataSource** and **DataField** properties. With the **ColLink** property a designated column of the selected row is copied to the Edit or Static portion of the control which may be used to edit the selected value. This Edit or Static value is then used to update a database field of a bound **Recordset** identified through the **DataSource** and **DataField** properties. Alternately, for the **VPComboBox** control you can have a different column of the selected row be bound for update through the use of the **ColBound**, **ColDataSource**, and **ColDataField** properties.

Shown below is a list of the properties used to fill and manage the List portion, and permit binding of the selected data to a **Data** control. Array properties are italicized.

Property	Specifies
ColBound	Index position of designated column bound to a **Field** object as defined by the **ColDataSource** and **ColDataField** properties.
ColDataSource	Name of **Data** control that is updated once a selection is made.
ColDataField	Name of **Field** object to be updated in the **Recordset** specified by **ColDataSource** based on a selection made and a designated column identified by the **ColBound** property.
DataSource	Name of **Data** control that is bound to the Edit or Static portion of the control.
DataField	Name of **Field** object to be updated in **Recordset** specified by **DataSource** based on data found in the Edit or Static portion of the control.
RowSource	Name of **Data** control used as a source of items for the List portion of the control.
ColListField	Array of Names of **Field** objects in **Recordset** specified by

	RowSource to be used to fill the List portion.
ColLink	Index position of designated column in List that is linked to the Edit or Static portion of control. If set to zero, no one column is linked and the whole row of data is linked to the Edit or Static portion of the control.

In either the **Dropdown** or **Simple Combo** styles, the **VPComboBox** control permits users to type data into the Edit portion of the control. Once entered, this data can be used to locate or match within the linked column (defined by the **ColLink** property) of the List, and the matched list item or row is selected. The **MatchEntry** property determines if matching occurs and what type of matching is employed. The **VPComboBox** control supports **Standard** matching modeled after the search dialog in the Windows help system, or **Extended** matching that is similar to the searches found in controls used in Microsoft Money and Intuits Quicken. For the **Dropdown List** style of combo box the **VPComboBox** control supports first character matching. This feature has the control search for the next matching item or row within the List portion for an alphabetic character entered using the first character of the linked column for matching purposes. Repeatedly typing the same letter cycles through all the items in the List beginning with the entered letter. Only one character can be entered at a time when the control has the **Dropdown List** style.

Other custom properties of the **VPComboBox** control include a **MaxDrop** and **MaxWidth** property. The **MaxDrop** property sets the maximum number of items or rows that are displayed when the drop-down list is dropped. Its default value is 8 items or rows. The **MaxWidth** property sets the maximum width for all columns within the List portion. This width value is expressed in Twips. One unique feature of the **VPComboBox** control is that the width of the List portion is not determined by the standard **Width** property used to set the width of the Edit or Static portion. If the custom **MaxWidth** property is set to the default value of zero (0), the width of the List portion is based on the cumulative widths of the individual columns. If the **MaxWidth** property is set to something other than zero, the width of the List portion is limited by this property and if the combined length of the columns exceeds this **MaxWidth** property, a horizontal scrollbar is added to the List portion.

The **VPComboBox** control List portion is always displayed in integral heights. This means the List resizes itself to display only complete items or rows.

One final aspect of the **VPComboBox** control is that it is not limited to 64K worth of data like the standard **ComboBox** control. This does not mean that it functions as a virtual List Box; it does have a limit on the amount of data, but its limits are not as restrictive as the standard **ComboBox** or **ListBox** control. The **VPComboBox** control has its own memory management routines which do not store a row of data within the List portion, but a pointer to the data. This allows the control to hold significantly more data. There still is a limit to the number of rows, but no limit to the amount of data within those rows or the number of columns of information for each row. In Windows 3.1 you are limited to a little over 6,500 rows. In Windows 95 or Windows NT you are limited to 32,767 rows. For most programming situations this should be sufficient.

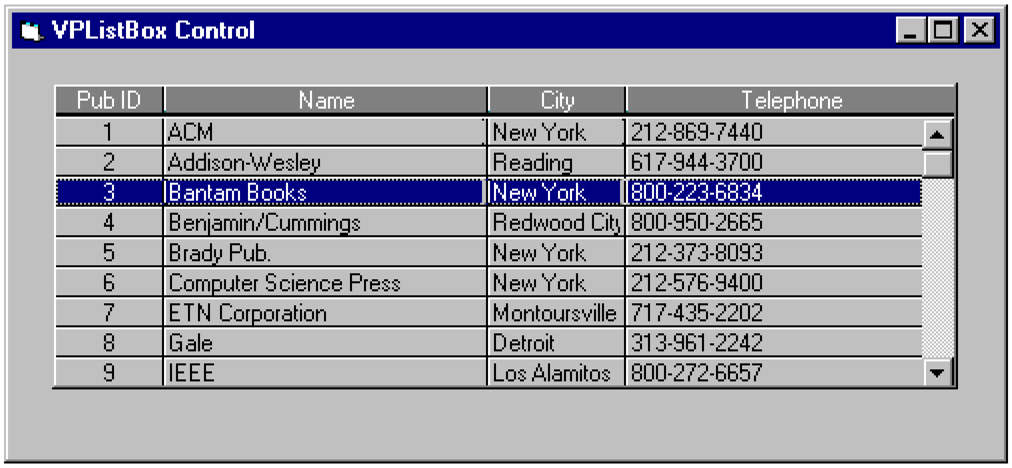
VPListBox ListBox Control

Custom: Properties Methods Functions Standard: Properties Events Methods

The **VPListBox** control, like the standard **ListBox** control, displays a list of items from which the user can select one or more. If the number of items exceeds the number that can be displayed, a scroll bar is automatically added to the **VPListBox** control.

If no item is selected, the **ListIndex** property value is -1. The first item in the list is **ListIndex** 0, and the value of the **ListCount** property is always one more than the largest **ListIndex** value.

The **VPListBox** control has a number of added properties and features. These include being data-aware, having multi-columns, 3D display effects, optional grid lines, column headings, output formatting, color settings, data alignment, custom sorting, locate data functions, standard unbound operations, and no 64K data limit.



Pub ID	Name	City	Telephone
1	ACM	New York	212-869-7440
2	Addison-Wesley	Reading	617-944-3700
3	Bantam Books	New York	800-223-6834
4	Benjamin/Cummings	Redwood City	800-950-2665
5	Brady Pub.	New York	212-373-8093
6	Computer Science Press	New York	212-576-9400
7	ETN Corporation	Montoursville	717-435-2202
8	Gale	Detroit	313-961-2242
9	IEEE	Los Alamitos	800-272-6657

Syntax

VPListBox

Remarks

The **VPListBox** control has a number of custom properties for controlling the look of the control. The **Appearance** property can define either an **Inset** or **Raised** 3D display effect for the List border. The **GridLines** and **GridAppearance** properties control the display for the List itself. There are also separate **ForeColor** and **BackColor** properties for the fixed Column Heading row or the other data rows of the List.

A custom feature of the **VPListBox** control is its support for defining and displaying multiple columns of information. These columns can be defined in code executed during run time, or before execution, during design time.

During run time, with code, you can set the number of columns and define their characteristics. The **MaxCols** property sets or returns the number of columns for the List. Setting this property to a number less than the current number of columns will delete the trailing columns from the List. Any columns added by setting this property to a higher number will create new columns with appropriate default values. The **Col** property allows a programmer to select a specific column. Once set, any column properties referenced or modified will be the array values for the specified column.

In design time, as part of the property sheet of the ActiveX version of the **VPListBox** control, there is a tab for **Column layout** properties which is used to add, delete, or maintain column properties. In the VBX version there is a **Column Layout Properties dialog** which is used to maintain the same column properties. This column layout dialog is called by clicking on the ellipse button that is part of the **(ColLayout)** property in the Visual Basic Properties Window or by double-clicking the **(ColLayout)** property, directly.

Shown below is a list of the properties used to define columns in the **VPListBox** control. Array properties are italicized.

Property	Specifies
ColAlign	The alignment of text within a each column.
ColBound	Which column of a selected row in a List is bound directly to a data source.
ColFormat	An optional VB format string for each column. This property has the same effect and supports the same formats as the VB Format$ function.
ColHeading	The text for an optional heading or caption for each column.
ColHeadAlign	The alignment of the column heading or caption for each column.
ColListField	Name of the **Field** object in the **Recordset** specified by the **RowSource** property bound to each column in the List.
ColSortBy	A Boolean flag which indicates if a given column is used for sorting the contents of the List.
ColSortOrder	Determines if a column of the List used in sorting is sorted in ascending or descending order.
ColWidth	The width in Twips of each column.

You will notice there are properties for defining a column heading or caption row for the **VPListBox** control. Whether this control has a fixed heading row is determined by the Boolean **Heading** property.

The **VPListBox** control has additional properties that define the colors used. The foreground and background colors for the fixed heading row are controlled by the **HeadingBackColor** and **HeadingForeColor** properties. The foreground and background colors for the data rows of the List are defined by the standard **BackColor** and **ForeColor** properties.

The **VPListBox** control can be used as a bound control or used in unbound operations, equally well. Unbound, the control uses the same methods of the standard **ListBox** control.

To add or delete items in a **VPListBox** control, use the **AddItem** or **RemoveItem** method. Each string value to be added or updated to a specific row must use the **Tab** character, Chr$(09), as a delimiter between the columns of data. Set the **List**, **ListCount**, and **ListIndex** properties to enable a user to access items in the **VPListBox** control. With the **RowSource** and **ColListField** properties defined, the **VPListBox** control automatically fills the List with fields from one **Data** control, and optionally passes data to the selected field of a second **Data** control, as defined in the **ColDataSource** and **ColDataField** properties. If the **ColDataSource** and **ColDataField** properties are defined, the control is restricted to a single select mode and the **MultiSelect** property is set to **False**.

Shown below is a list of the properties used to fill and manage the List, and bind the selected data to a **Data** control.

Property	Specifies
ColBound	Index position of designated column bound to a **Field** object as defined by the **ColDataSource** and **ColDataField** properties.
ColDataSource	Name of **Data** control that is updated once a selection is made.
ColDataField	Name of **Field** object to be updated in the **Recordset** specified by **ColDataSource** based on a selection made and a designated column identified by the **ColBound** property.
RowSource	Name of **Data** control used as a source of items for the List.
ColListField	Array of Names of **Field** objects in **Recordset** specified by **RowSource** to be used to fill the List.

The **VPListBox** control does not support LongBinary Fields, i.e. you cannot have the **ColListField** as a LongBinary field. If you do, nothing will display in the particular list column.

The **VPListBox** control is always displayed in integral heights. This means the List resizes itself to display only complete items or rows.

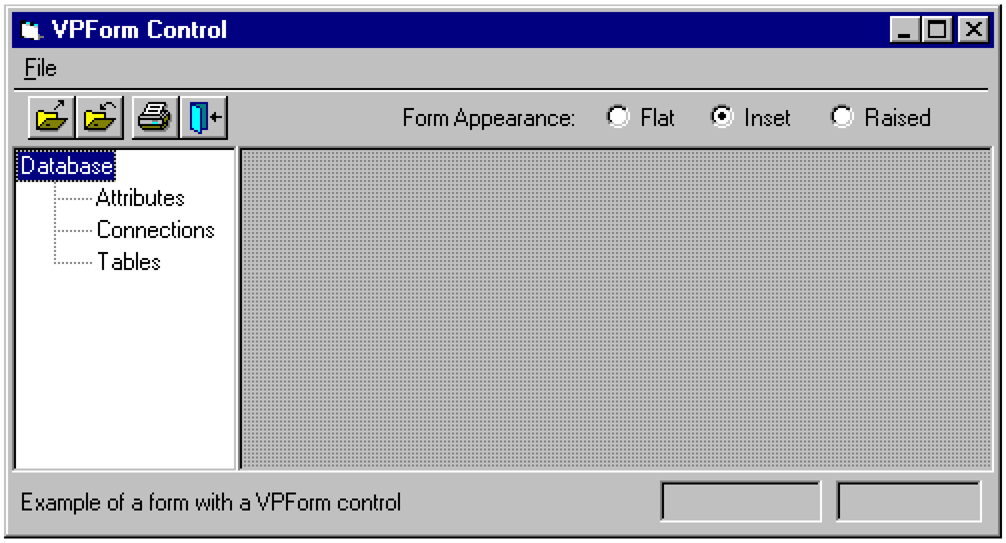
The **VPListBox** control has a custom **AutoSize** property that controls whether the width of the List is based on standard **Width** property or the List is automatically sized based on the cumulative widths of each column. If the **AutoSize** property is set to **False** and the sum of the column widths is greater than the standard **Width** property, a horizontal scroll bar is automatically added to the List at run time.

One final aspect of the **VPListBox** control is that it is not limited to 64K worth of data like the standard **ListBox** control. This does not mean that it functions as a virtual List Box; it does have a limit on the amount of data, but its limits are not as restrictive as the standard **ComboBox** or **ListBox** control. The **VPListBox** control has its own memory management routines which do not store a row of data within the List, but a pointer to the data. This allows the control to hold significantly more data. There still is a limit to the number of rows, but no limit to the amount of data within those rows or the number of columns of information for each row. In Windows 3.1 you are limited to a little over 6,500 rows. In Windows 95 or Windows NT you are limited to 32,767 rows. For most programming situations this should be sufficient.

VPForm Form Display Control

Custom: Properties Events Standard: Properties

The **VPForm** control is used to control the appearance of a **Form** object with 3D display effects and custom background properties. It also includes a custom **Cursor** property and Mouse properties and events to provide enhanced drag-and-drop operations.



Example of a form with a VPForm control

Syntax

VPForm

Remarks

The **VPForm** control is visible only during the design mode of Visual Basic and does things on a form level basis. It can be used to control the look of the form with a combination of a **BorderStyle** property and a custom **Appearance** property. The appearance options are **Flat**, **3D - Inset**, or **3D - Raised** with the 3D options drawing a 2-pixel border on the inside of the form's frame if the **BorderStyle** property is set to Single. This control also includes a form **Background** property that can apply a bitmap as a brush in displaying a form's background.

Both the **Appearance** and form **Background** properties take in account any controls on a form that have their align property set to top, bottom, left, or right. This means that any 3D or form display effects are within the boundaries of the form excluding aligned controls which provides for a more appealing visual effect.

The **VPForm** control also has a custom **Cursor** property and **MouseCapture** property that in conjunction with custom Mouse events, provides a robust and flexible drag-and-drop mechanism that goes beyond those functions provided by Visual Basic. Once the **MouseCapture** property is set to **True** the **VPForms** custom mouse events will be fired for any mouse operations until the **MouseCapture** property is reset to **False**. Cursor references can be assigned to the **VPForm's Cursor** property, allowing for different custom mouse pointers during drag-and-drop operations. The purpose of these custom properties and events is to provide centralized, easy to program, drag-and-drop operations.

VPFocus Focus Management Control

Custom: Properties   Methods   Standard: Properties

The **VPFocus** control is used to manage **GotFocus** and **LostFocus** events so a programmer can do immediate data validation as **Focus** leaves a control. This control allows Visual Basic's **LostFocus** event to work similar to the **OnExit Control Property** of Microsoft Access.

Syntax

VPFocus

Remarks

The **VPFocus** control has a set of properties and custom methods (ActiveX version only) that provide a mechanism for controlling **Focus** messages and **Focus** processing within a Visual Basic program. A single instance of this control is placed on the first Visual Basic form to be loaded. In the ActiveX version, a **VPFocus** control must be placed on each form. The control is visible only while in the design mode of Visual Basic. It works by serializing the **Focus** events generated within a Visual Basic program so that **data validation** processing in a **LostFocus** event can be properly completed before any **GotFocus** or **LostFocus** events are processed in any other controls. This control includes a **FocusAction** property that acts as a custom method. The ActiveX version also includes a set of custom methods. The **FocusAction** property has three property values or Actions which include:

0 (**vxTrapFocusOff**) - This action turns off the **Focus** control and sets the object pointer **ActiveControl** property to nothing. Same as the **TrapFocusOff** method in the ActiveX version.

1 (**vxSendFocus**) - This action is usually set in a **LostFocus** event procedure of a control and sets the **ActiveControl** property to nothing and then sends **Focus** on to the control trapped when **Focus** changed. Same as the **SendFocus** method in the ActiveX version.

2 (**vxReturnFocus**) - This action is usually set in a **LostFocus** event procedure of a control and sends **Focus** back to the currently Active control. Same as the **ReturnFocus** method in the ActiveX version.

Most Visual Basic programmers have found that they would like to validate data entered into a control before **Focus** leaves that control. And they have all found out the problems associated with L**ostFocus/GotFocus** event pairs that make this programming task almost impossible. With the **VPFocus** custom control, this type of important programming can be accomplished easily and without any normal **Focus** transfer issues getting in the way. Most programmers will find that this **VPFocus** control, by itself, is worth the price of **Vantage Control Set**.

Close

Custom Property Summary

(About)
(ColLayout) **(VBX only)**
(Custom) **(OCX only)**

ActiveControl
Alignment
Appearance
AutoHeight
AutoSelect
AutoSize

Background
BorderStyle

CaseSensitive
CellText
Col
ColAlign
ColBound
ColDataField
ColDataSource
ColFormat
ColHeadAlign
ColHeading
ColLink
ColListField
ColSortBy
ColSortOrder
ColWidth
ControlIndex
Cursor

DataChanged
DataField
DataSource

FocusAction
ForceGotFocus **(OCX only)**
FormControlName
FormIndex

GridAppearance
GridLines

Heading
HeadingBackColor
HeadingForeColor

ListAppearance
ListBackColor
ListForeColor

Locked

MatchEntry
MaxCols
MaxDrop
MaxWidth
MouseCapture

OverType

RowSource

TargetControl

Close

Custom Event Summary

Change
CloseUp
MouseDown
MouseMove
MouseUp

Close

Custom Method Summary (OCX only)

LocateText
ReturnFocus
SendFocus
TrapFocusOff
TrapFocusOn

Close

Custom Function Summary

VLocateText

Using Custom Controls

Custom controls exist as separate files with either a .VBX or .OCX filename extension. They include specialized controls from Microsoft that are part of Visual Basic, such as the **CommonDialog** control, and third-party custom controls such as the library of controls found in **Vantage Control Set**.

Custom controls with the .OCX filename extension take advantage of ActiveX OLE technology and should work in any container that supports the ActiveX OLE interfaces. Custom controls with the .VBX filename extension use older 16-bit technology and are limited to working with Visual Basic.

Note You can use .VBX custom controls in Visual Basic 3.0 and in the 16-bit version of Visual Basic 4.0. You can't use these controls in the 32-bit version of Visual Basic 4.0, Visual Basic 5.0, or any other application. You can use ActiveX (.OCX) controls in the 32-bit version of Visual Basic 4.0 and in Visual Basic 5.0. You may be able to use the ActiveX controls in other containers that support ActiveX technology.

Visual Basic 3.0

To start using one of the .VBX controls of **Vantage Control Set** in a Visual Basic 3.0 project you use the Add File option of the File Menu to select the appropriate .VBX file (Alt-F-A). Adding a .VBX file will add the icons of the new control at the bottom of the Visual Basic ToolBox where you can start using them like you would any standard Visual Basic control.

The following controls are found in these .VBX files within **Vantage Control Set**.

Control	Object Name	.VBX File Name
VPTextBox	VPTextBox	VPTEXT.VBX
VPStatic	VPStatic	VPSTAT.VBX
VPComboBox	VPComboBox	VPCOMB.VBX
VPListBox	VPListBox	VPLIST.VBX
VPForm	VPForm	VPFORM.VBX
VPFocus	VPFocus	VPFOCUS.VBX

Visual Basic 4.0 and Visual Basic 5.0

When Visual Basic 4.0 or Visual Basic 5.0 opens a project containing a .VBX control, the default behavior is to replace the .VBX control with an .OCX control, but only if an .OCX version of the control is available.

To see a list of the custom controls included in your project

In VB4, from the Tools menu, choose Custom Controls. The checked boxes in the Available

Controls box indicate the .VBX controls, .OCX controls, and insertable objects that can be included in your project. In VB5, from the Project menu, choose Components. The check boxes in the Controls tab list indicate the ActiveX controls and insertable objects available in the Registry that can be included in your project.

To add a custom control

1. For VB4, From the Tools menu, choose Custom Controls. For VB5, from the Project menu, choose Components.

2. For VB4, under Show, select the Insertable Objects and Controls options. For VB5, select the Controls tab.

3. Select the check box next to the name of each .OCX control or .VBX control (VB416-bit only), and then choose OK. Once a custom control is placed in the Toolbox, you can add it to a **Form** just as you would with a standard control.

If the .OCX control or .VBX control that you want to use isn't listed in the Available Controls box, choose the Browse button to locate the file.

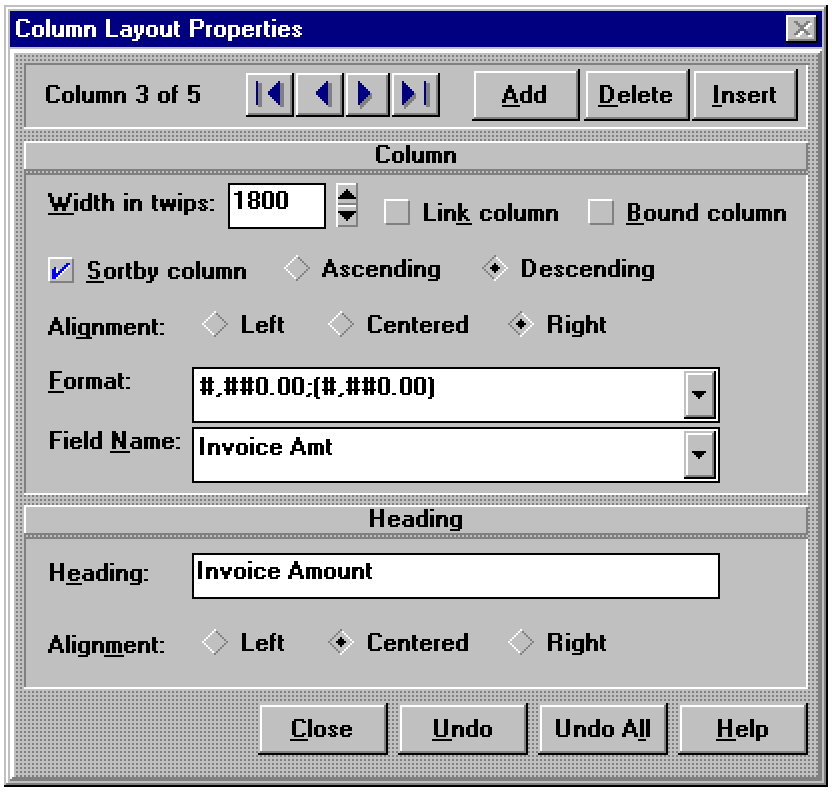
The following controls are found in these .OCX files within **Vantage Control Set**.

Control	Object Name	.OCX File Names
VPTextBox	VPTextBox	VPTEXT32.OCX
VPStatic	VPStatic	VPSTAT32.OCX
VPComboBox	VPComboBox	VPCOMB32.OCX
VPListBox	VPListBox	VPLIST32.OCX
VPForm	VPForm	VPFORM32.OCX
VPFocus	VPFocus	VPFOCS32.OCX

Column Layout Properties Dialog

See Also

The **Column Layout Properties Dialog**, associated with the VBX version, is used to add, delete, or maintain column properties for either the **VPComboBox** or **VPListBox** custom control (the ActiveX version uses a tabbed dialog associated with the property sheets). This layout dialog is called by clicking on the ellipse button that is part of the **(ColLayout)** property in the Visual Basic Properties Window or by double-clicking the **(ColLayout)** property, directly.



Column 3 of 5

Width in twips: 1800

Link column

Bound column

[✓ Sortby column]

Ascending [✓ Descending]

Alignment: Left Centered [✓ Right]

Format: #,##0.00;(#,##0.00)

Field Name: Invoice Amt

Heading: Invoice Amount

Alignment: Left [✓ Centered] Right

Remarks

The upper left corner of the Dialog displays the column currently pointed to and the number of columns defined for **VPComboBox** or **VPListBox** control being edited. The "VCR" buttons at the top of the Dialog are for navigating between the columns. You can quickly move to the first column, the previous column, the next column, or the last column by clicking on one of these "VCR" buttons. The other three command buttons in the top right of the Dialog are for **adding** or **inserting** new column definitions, or **deleting** existing column definitions.

The command buttons at the bottom of the Dialog are for saving column properties or for providing different levels of undoing pending edits or changes to the column properties. The **Close** button will save any currently pending changes for all columns edited and close the **Column Layout Properties Dialog**. The **Undo** button is used to undo any changes to the current column. Pending changes to other columns are not effected by clicking on this **Undo** button. The **Undo All** button will undo any and all changes to all columns and will close the **Column Layout Properties Dialog**. The **Help** button will display this help topic.

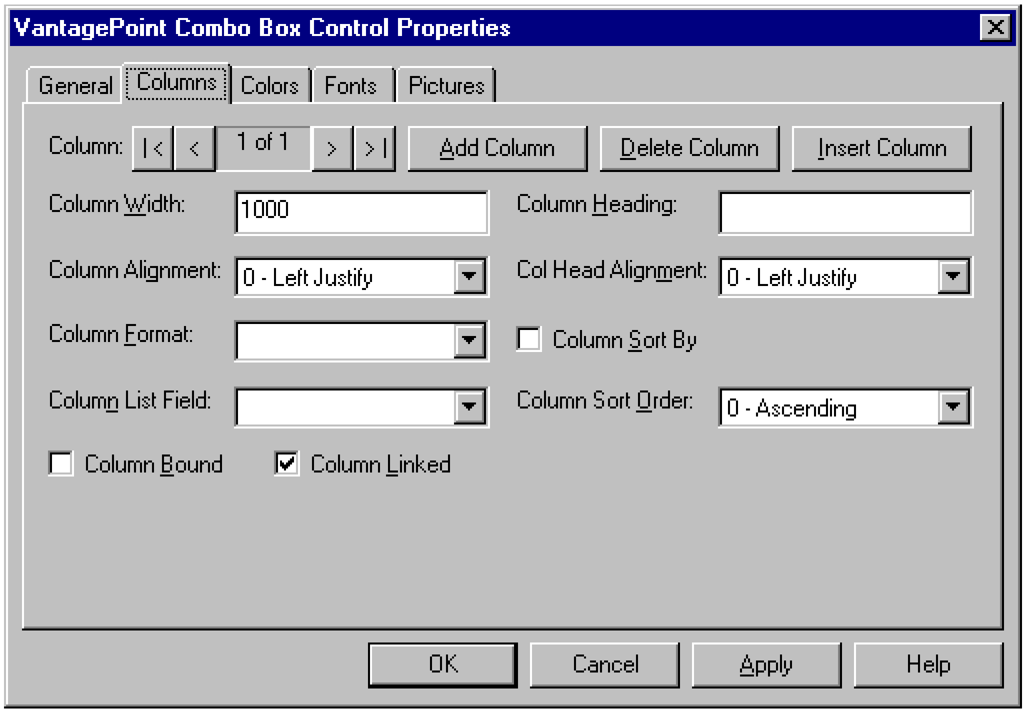
The column properties maintained in this Dialog are separated into two sections. The top section deals with general properties of each column. The lower section defines the heading properties for each column.

Shown below is a list of the array properties used to define columns that are maintained in the **Column Layout Properties Dialog**. Array properties are italicized.

Property	Specifies
ColAlign	The alignment of text within a each column.
ColBound	Which column of a selected row in a List is bound directly to a data source.
ColFormat	An optional VB format string for each column. This property has the same effect and supports the same formats as the VB Format$ function.
ColHeading	The text for an optional heading or caption for each column.
ColHeadAlign	The alignment of the column heading or caption for each column.
ColLink	Which column of the List is will pass back its data to the **DataSource** to update the **DataField**, once a selection is made.
ColListField	Name of the **Field** object in the **Recordset** specified by the **RowSource** property bound to each column in the List.
ColSortBy	A Boolean flag which indicates if a given column is used for sorting the contents of the List.
ColSortOrder	Determines if a column of the List used in sorting is sorted in ascending or descending order.
ColWidth	The width in Twips of each column.

Property Pages

Property Pages are a standard interface for displaying and modifying the design time properties for an ActiveX control. Sometimes referred to as a Property Sheet, Property Pages are tabbed dialogs that combine several dialog boxes into a single, compound dialog box. In a Property Sheet, an individual dialog is called a property page.



Column: 1 of 1

Column Width:	1000
Column Heading:	
Column Alignment:	0 - Left Justify
Col Head Alignment:	0 - Left Justify
Column Format:	
Column Sort By	☐
Column List Field:	
Column Sort Order:	0 - Ascending
Column Bound	☐
Column Linked	✓

How you access the Property Pages for a control depends on the container you are using the control in. To display the Property Pages in Visual Basic, either double-click on the **Custom** property in the properties list window, or click on the ellipsis button appearing on the properties list when the **Custom** property is selected. You can also right-button-click the control itself and select the properties menu option.

Property Pages allow the user to change the values for a property and then apply those changes to the control. The tabbed dialog interface of Property Pages include the following common command buttons for handling application of property changes:

Command	**Action**
OK	Applies all pending changes and closes the property sheet window.
Apply	Applies all pending changes but leaves the property sheet window open.
Cancel	Discards any pending changes and closes the property sheet window. Does not cancel or undo changes already applied.

If a user closes the Property Pages dialog and there are pending changes that have not been committed, a message box will open to allow the user to save the changes, discard the changes, or cancel the close, returning to the Property Pages dialog.

Technical Support

You can reach technical support at the following numbers:

Phone:	(801) 292-5344
Fax:	(801) 292-3142
Email:	vantage@vpsoft.com

Copyright Notice

VantagePoint Software, Inc.
1619 E. Lakeview Drive
Bountiful, UT 84010

Phone:	(801) 292-5344
Fax:	(801) 292-3142
BBS:	(801) 763-0860
Email:	vantage@vpsoft.com
Web:	http://www.vpsoft.com

(About) Property

Applies To

Displays the **About Dialog** for the Vantage Control Set. Available only in design mode.

Remarks

The dialog is called by clicking on the ellipse button that is part of the **(About)** property in the Visual Basic Properties Window or by double-clicking the **(About)** property, directly.

(ColLayout) Property

See Also   Applies To

Calls the **Column Layout Properties Dialog** in the VBX version. Available only in design mode.

Remarks

The layout dialog is called by clicking on the ellipse button that is part of the **(ColLayout)** property in the Visual Basic Properties Window or by double-clicking the **(ColLayout)** property, directly. This property is available only for the VBX version (the properties maintained through the column layout dialog are maintained through the property page in the ActiveX version).

(Custom) Property

See Also   Applies To

Displays the **Properties Pages dialog** in the ActiveX version. This tabbed dialog provides access to all the properties of the control. Available only in design mode.

Remarks

To display the property pages, either double-click on the **(Custom)** property in the properties bar/list or click on the ellipsis button that appears on the properties bar/list when the **(Custom)** property is selected. Property pages provide access to all the properties of the control through a tabbed dialog interface. This property is available only for the ActiveX version.

ActiveControl Property

See Also Example Applies To

Returns or sets the window handle of the currently active control. Setting this property will start the trapping of **Focus** events.

Syntax

object.**ActiveControl** [= *value*]

The **ActiveControl** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
value	The Window handle (Integer) of the current control.

Remarks

The **ActiveControl** property is an object pointer which records which control, if any, is currently being monitored for **Focus** events. This property should usually be set in a valid **GotFocus** event procedure. The **ActiveControl** property can be directly set or can be modified by using the **FocusAction** property. Setting the **FocusAction** property to an action state of 0 (vxTrapFocusOff) or 1 (vxSendFocus) will reset the **ActiveControl** property to zero (nothing). In the ActiveX version using the **TrapFocusOff** or the **SendFocus** methods will do the same.

Alignment Property

Applies To

Returns or sets a value that determines the alignment of text in a **VPTextBox** or **VPStatic** control, or the text of the Edit or Static portion of the **VPComboBox** control. Also returns or sets a value that determines the alignment of a **CheckBox** or **OptionButton** control, text in a standard **TextBox** control, or text in a standard **Label** control. Read-only at run time for **CheckBox**, **OptionButton**, and standard **TextBox** controls.

Syntax

object.**Alignment** [= *number*]

The **Alignment** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
number	An numeric expression which specifies the type of alignment, as described in Settings.

For **VPTextBox** and **VPStatic** controls, the edit portion of a **VPComboBox** control, the standard **Label** control, the standard **TextBox** control, the settings for *number* are:

Setting	Description
0	Left Justify. (Default)
1	Right Justify.
2	Center.

For **CheckBox** and **OptionButton** controls, the settings for *number* are:

Setting	Description
0	Control is left-aligned with text on the right. (Default)
1	Control is right-aligned with text on the left.

Remarks

If the **MultiLine** property setting of a standard **TextBox** control is **False**, the **Alignment** property is ignored. For the **VPTextBox** control this **Alignment** property applies even if the **MultiLine** property is set to **False**.

You can display text to the right or left of **OptionButton** and **CheckBox** controls. Text is always left-aligned.

Appearance Property

See Also Applies To

Returns or sets the paint style of **Form** objects and controls.

Syntax

object.**Appearance** [= *number*]

The **Appearance** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
number	An numeric expression which specifies the type of appearance, as described in Settings.

Settings

The settings for *number* are:

Setting	Description
0	Flat. Paints controls and forms without visual effects. (Default)
1	3D Inset. Paints controls with a lowered three-dimensional effect.
2	3D Raised. Paints controls with a raised three-dimensional effect.

Remarks

If set to 1 or 2 at either design time or run time, the **Appearance** property draws Forms or controls with three-dimensional effects. Display effects for **Form** objects are controlled through the **VPForm** control and its **Appearance** property. A **Form** object can have a normal flat appearance or acquire a three-dimensional look by drawing a 2-pixel border on the inside of the **Form** or window frame. The **VPTextBox**, **VPStatic**, **VPComboBox**, and **VPListBox**, controls each have their own **Appearance** property. This **Appearance** property will have no effect If a control's **BorderStyle** property is set to None (0).

AutoHeight Property

Applies To

Returns or sets a value that determines whether the **VPComboBox** control's **Height** property can be adjusted or is automatically set.

Syntax

object.**AutoHeight** [= *boolean*]

The **AutoHeight** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
boolean	A Boolean expression which specifies whether the control is resized, as described in Settings.

Settings

The settings for *boolean* are:

Setting	Description
True (Default)	Automatically sets the control's height based on the currently defined font.
False	Allows the **Height** property to be user defined.

Remarks

The **AutoHeight** property of the **VPComboBox** control determines if the **Height** property is automatically set or can be modified by the programmer. The normal **ComboBox** control's **Height** property is automatically set based on the font size defined for the control. You have no way of changing it. By setting the **AutoHeight** property of the **VPComboBox** control to **False** you can allow the control's height to be set to any value. This property and custom behavior can be useful if you want to link and match the **VPComboBox** control's size and position to that of a cell in a grid control, such as TrueGrid.

AutoSelect Property

Applies To

Returns or sets a value that determines whether the text of a control's Edit portion is automatically selected when the control receives **Focus**.

Syntax

object.**AutoSelect** [= *boolean*]

The **AutoSelect** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
boolean	A Boolean expression which specifies the behavior described in Settings.

Settings

The settings for *boolean* are:

Setting	Description
True	Text of control is automatically selected when control receives **Focus**.
False	Normal operation, text is not automatically selected. (Default)

Remarks

The **AutoSelect** property, when set to **True**, is useful for **VPTextBox** or **VPComboBox** controls that are used in "data entry" database applications. It makes it easy for the user to replace the current contents of a field that needs modification by automatically selecting all the text upon entering the field. Whatever the user types in will replace the selected text. If the **AutoSelect** property is set to **True** and the user does not want to replace the complete text, but only wants to edit the existing text, the user can use the keyboard cursor or arrow keys to reposition within the text and the text will become unselected.

Automatic selection of text works by setting the **SelStart** property of the control to zero (0) and the **SelLength** property of the control to the full length of the text. To have the **VPTextBox** or **VPComboBox** control operate in the same way as a standard **TextBox** or **ComboBox** control, set the **AutoSelect** property to **False.**

AutoSize Property

Applies To

Returns or sets a value that determines whether a control is automatically resized to display its entire contents.

Syntax

object.**AutoSize** [= *boolean*]

The **AutoSize** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
boolean	A Boolean expression which specifies whether the control is resized, as described in Settings.

Settings

The settings for *boolean* are:

Setting	Description
True	Automatically resizes the control to display its entire contents.
False	Keeps the size of the control constant. Contents are clipped when they exceed the area of the control. (Default)

Remarks

For the **VPListBox** control the **AutoSize** property determines whether the width of the control is based on the size set at design time (the **Width** property) or if the List is automatically sized based on the cumulative widths for each column (the **ColWidth** array property). If the **AutoSize** property is set to **True**, the control's width is resized to accommodate all the columns defined. Consequently a horizontal scroll bar will never be needed for the List. If the **AutoSize** property is set to **False**, the control retains its **Width** property. If the sum of the column widths is more then the **Width** property, a horizontal scroll bar will be added to the List at run time.

Background Property

Example   Applies To

Returns or sets a bitmap graphic used as a custom brush to be used when displaying a Form object's background.

Syntax

object.**Background** [= *picture*]

The **Background** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
picture	A string expression specifying a file containing a bitmap graphic, as described in Settings.

Settings

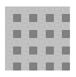
The settings for *picture* are:

Setting	Description
(None)	No picture. (Default)
(Bitmap)	Specifies a bitmap graphic. You can load the graphic from the Properties window at design time. At run time, you can also set this property using the LoadPicture function on a bitmap file.

Remarks

The **Background** property is a property of the **VPForm** control but effects the **Form** object that the **VPForm** control is placed on. This **Background** property contains a bitmap image used as a brush when painting the background of the target **Form**.

There are a number of pre-defined bitmaps which come with **Vantage Control Set** that can be used as a background brush. These bitmap files can be found in the Bitmaps folder (subdirectory) under the main **Vantage Control Set** folder (directory).

You can also create your own background brush bitmaps.  In order to be used as a brush, their dimensions must be 8 pixels by 8 pixels.

When setting the **Background** property at design time, the bitmap graphic is saved and loaded with the **Form**. If you create an executable file, the file contains the bitmap. When you load a graphic at run time, the graphic isn't saved with the application.

BorderStyle Property

See Also   Applies To

Returns or sets the border style for an object. Read-only at run time.

Syntax

object.**BorderStyle** [= *value*]

The **BorderStyle** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
value	A value or constant which determines the border style, as described in Settings.

Settings

The settings for *value* are:

Setting	Description
0	None. (Default)
1	Fixed Single. Meaning a single fixed line.

Remarks

If a control's **BorderStyle** property is set to None (0), the **Appearance** property of that control will have no effect.

The **BorderStyle** property is a custom property for the **VPTextBox**, **VPStatic**, **VPComboBox**, and **VPListBox** controls. It is also a property for the ActiveX version of the **VPForm** control. For all the visible controls of Vantage Control Set, setting the BorderStyle to None will eliminate the line border from the control if the Appearance property is set to a Flat style. For 3D effects of the **Appearance** property to be visible, the **BorderStyle** must be set to Single. This property can be useful if you want to link and match the **VPComboBox** control's size and position to that of a cell in a grid control, and not display its own border.

Caption Property

Applies To

The **Caption** property applies to several objects:

> Form - determines the text displayed in the **Form** or **MDIForm** object's title bar. When the form is minimized, this text is displayed below the form's icon.
>
> Control - determines the text displayed in or next to a control.
>
> MenuLine object - determines the text displayed for a **Menu** control or an object in the **MenuItems** collection.

Syntax

object.**Caption** [= *string*]

The **Caption** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list. If object is omitted, the form associated with the active form module is assumed to be object.
string	A string expression which evaluates to the text displayed as the caption.

Remarks

When you create a new object, its default caption is the default **Name** property setting. This default caption includes the object name and an integer, such as Command1 or Form1. For a more descriptive label, set the **Caption** property.

You can use the **Caption** property to assign an access key to a control. In the caption, include an ampersand (&) immediately preceding the character you want to designate as an access key. The character is underlined. Press the ALT key plus the underlined character to move the focus to that control. To include an ampersand in a caption without creating an access key, include two ampersands (&&). A single ampersand is displayed in the caption and no characters are underlined.

For a **Label** control, the caption is limited to 2048 characters. For forms and all other controls that have captions, the limit is 255 characters.

To display the caption for a form, set the **BorderStyle** property to either Fixed Single (1), Sizable (2), or Fixed Double (3). A caption too long for the form's title bar is clipped. When an MDI child form is maximized within an **MDIForm** object, the child form's caption is included in the parent form's caption.

Tip For the **Label** or **VPStatic** control, set the **AutoSize** property to **True** to automatically resize the control to fit its caption.

CaseSensitive Property

Applies To

Returns or sets a value that determines if the operation of the List portion of a control is case sensitive. This property effects how items in a List are selected and how they are sorted.

Syntax

object.**CaseSensitive** [= *boolean*]

The **CaseSensitive** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
boolean	A Boolean expression which specifies the behavior described in Settings.

Settings

The settings for *boolean* are:

Setting	Description
True	List operations are case sensitive.
False	List operations are not case sensitive. (Default)

Remarks

List operations of the standard ListBox or ComboBox control is case insensitive. This means the uniqueness of text placed within a List is not dependent on the case of the text. The string value of abc is considered the same as Abc or ABC. In this example, if all three strings are placed into a List, in the order referenced above, and a match against the string ABC was attempted, the first abc string would match in a case insensitive operation. The **CaseSensitive** property provides a means of setting the type of operations for the List portions of the **VPComboBox** control, or the **VPListBox** control. The default for this property is **False**, which indicates that all operations within a List are not case sensitive, like the standard ListBox or ComboBox control. If this property is set to **True**, the **VPComboBox** and **VPListBox** control operates in a case sensitive mode. Each instance of the above example strings would be considered different items within the List. This property effects how items are selected and how they are sorted.

CellText Property

See Also Example Applies To

Returns the contents of a cell within a List. Not available at design time.

Syntax

object.**CellText(***index***)**

The **CellText** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.

index A numeric expression which uniquely identifies the row within a List.

Remarks

The **CellText** property provides a mechanism to retrieve text data from a particular column and row position (or cell) within a List. The column position is first specified by the **Col** property, while the row position is determined by the supplied index value. This property, while it allows you to retrieve the text data of a cell, it does not allow you to assign or set the contents of a cell. To update the cell of a List you need to use the **List** property and update the contents of the complete row.

Col Property

See Also   Example   Applies To

Returns or sets the column number being addressed when setting column array properties for the **VPComboBox** and **VPListBox** controls. Not available at design time.

Syntax

object.**Col** [= *number*]

The **Col** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
number	A numeric expression which specifies a column (1 to 99).

Remarks

In design mode, column array properties for the **VPComboBox** and **VPListBox** controls are set and maintained through the **Column Layout Properties Dialog**. In run mode, you must first set the **Col** property to specific column number before reading or setting any properties for a given column.

ColAlign Property

See Also   Applies To

Returns or sets the alignment of data in a column.

Syntax

object.**ColAlign** [= *value*]

The **ColAlign** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
value	A value or constant which determines the column alignment, as described in Settings.

Settings

The settings for *value* are:

Setting	Description
0	Left Justify. (Default)
1	Right Justify.
2	Center.

Remarks

This is an array property of column alignments. Any column can have an alignment that is different from other columns. When setting this property in run mode, the affected column is specified using the **Col** property. Column alignments based on this property apply to all rows except the fixed heading row, if defined.

ColBound Property

See Also Applies To

Returns or sets which column is defined as the bound column. In the **VPComboBox** and **VPListBox** controls, this property determines if a column will pass back its data to the **ColDataSource** to update the **ColDataField**, once a selection is made.

Syntax

object.**ColBound** [= *column*]

The **ColBound** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
column	A numeric expression which represents the index position of the column assigned as the Bound column.

Remarks

The **ColBound** property determines if there is single bound column, or if the data from all columns are used for update for the currently selected row. The **ColBound** property is set to the index position of the column you want bound. If this property is set to zero (0) then no one column is bound and all columns of data within the selected row are used for update. Data associated with the selected item or row is updated to the database field defined by the **ColDataSource** and **ColDataField** properties.

In the **VPListBox** control, using this property is the normal method for binding selected data to a data source. In the **VPComboBox** control you have two ways to bind selected data for update to a data source. You can use the combination of the **ColBound** property associated with the **ColDataBound** and **ColDataField** properties, like the **VPListBox** control, or you can use the data found in the Edit or Static portion of the **VPComboBox** control and bind this data for update through the standard **DataSource** and **DataField** properties. Using this later approach you would also use the **ColLink** property to link a given column within the List portion of the **VPComboBox** control to the Edit or Static portion of the control. Using the **ColLink** property to assign the data from a given column of a selected row to the Edit or Static portion of the control and then having that data be bound to a data field through the **DataSource** and **DataField** properties would be the normal method of binding for a Combo box type control. But if you want an alternate column of data to be bound rather than the linked data displayed in the Edit or Static portion of the control, the use of this **ColBound** property becomes very handy.

Generally, you use two **Recordset** objects with the data-aware list controls of **Vantage Control Set**. One **Recordset** contains a read-only list of valid selections, while the other **Recordset** is updated with selections from the list. For example, the **VPComboBox** control's list could be generated from a query that returned a result set of valid part numbers and their descriptions. One column of the list would be bound to part numbers field of the **Recordset** through the **ColListField** property. The other column would be bound to the description field of the **Recordset** through its **ColListField** property. The **ColBound** property could be used to identify the first column of part numbers as bound to the part number field of the second **Recordset**, as defined through the **ColDataSource** and **ColDataField** properties. The second column could be linked to the Edit or Static portion of the control through the **ColLink** property, so the user would see the part description selected, but have the part number used for update.

ColDataField Property

See Also Applies To

Returns or sets a value that binds a control to a field in the current record.

Syntax

object.**ColDataField** [= *value*]

The **ColDataField** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
value	A string expression which evaluates to the name of one of the fields in the **Recordset** object specified by a **Data** control's **RecordSource** and **DatabaseName** properties.

Remarks

Bound controls provide access to specific data in your database. Bound controls that manage a single field typically display the value of a specific field in the current record. The **ColDataSource** property of a bound **VPComboBox** or **VPListBox** control specifies a valid **Data** control name, and the **ColDataField** property specifies a valid field name in the **Recordset** object created by the **Data** control. Together, these properties specify what data appears in the bound column of a control as defined in the **ColBound** property.

ColDataSource Property

See Also Applies To

Sets a value that specifies the **Data** control through which a column of the current control is bound to a database. Not available at run time.

Remarks

To bind a column of a control to a field in a database at run time, using the **ColBound** property, you must specify a **Data** control in the **ColDataSource** property at design time using the Properties window.

To complete the connection with a field in the **Recordset** managed by the **Data** control, you must also provide the name of a **Field** object in the **ColDataField** property. Unlike the **ColDataField** property, the **ColDataSource** property setting isn't available at run time.

ColFormat Property

See Also   Applies To

Returns or sets an optional format string for data in a column.

Syntax

object.**ColFormat** [= *string*]

The **ColFormat** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
string	A VB format string to be applied to data of a specific column. Possible format strings include those listed in Settings.

Settings

Some of the possible settings for *string* include:

Settings			
General Number	#,##0;(#,##0)	hh:mm AM/PM	mmm
Currency	#,##0.00;(#,##0.00)	hh:mm am/pm	mmmm
Fixed	$#,##0	h:mm	q
Standard	$#,##0,00	h:mm:ss	y
Percent	$#,##0;($#,##0)	###-####	yy
Scientific	$#,##0.00;($#,##0.00)	(###) ###-####	yyyy
Yes/No	0%	#####	h
True/False	0.00%	#####-####	hh
On/Off	0.00E+00	###-##-####	n
General Date	0.00E-00	c	nn
Long Date	m/d/yy	d	s
Medium Date	mm/dd/yy	dd	ss
Short Date	d-mmm-yy	ddd	ttttt
Long Time	d-mmmm-yy	dddd	
Medium Time	d-mmm	ddddd	
Short Time	d-mmmm	dddddd	
0	mmmm-yyyy	w	
0.00	m/d/yy	ww	
#,##0	mmm-yy	m	
#,##0.00	h:mm	mm	

Remarks

This is an array property of optional column formats. Any column can have a format that is different from other columns. When setting this property in run mode, the affected column is specified using the **Col** property. Column format based on this property apply to all rows except the fixed heading row, if defined.

ColHeadAlign Property

See Also   Applies To

Returns or sets the alignment of the heading or caption in a column.

Syntax

object.**ColHeadAlign** [= *value*]

The **ColHeadAlign** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.

value A value or constant which determines the column alignment, as described in Settings.

Settings

The settings for *value* are:

Setting	Description
0	Left Justify. (Default)
1	Right Justify.
2	Center.

Remarks

This is an array property of column heading alignments. Any column can have an alignment that is different from other columns. When setting this property in run mode, the affected column is specified using the **Col** property. Column alignments based on this property apply to only the fixed heading row, if defined.

ColHeading Property

See Also   Applies To

Returns or sets the heading or caption in a column.

Syntax

object.**ColHeading** [= *string*]

The **ColHeading** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
string	A string expression which evaluates to the text displayed as the heading or caption.

Remarks

This is an array property of column headings or captions. Each column can have its own optional column heading. When setting this property in run mode, the affected column is specified using the **Col** property. Whether a column actually displays a column heading is determined by the **Heading** property.

ColLink Property

See Also Applies To

Returns or sets which column is defined as the linked column. In the **VPComboBox** control, this property determines which column is linked to the Edit or Static portion of the control.

Syntax

object.**ColLink** [= *column*]

The **ColLink** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
column	A numeric expression which represents the index position of the column linked to the Edit or Static portion of the control.

Remarks

This **ColLink** property determines if there is single linked column or if all the columns are linked to the Edit or Static portion of the control for the currently selected row. Any column can be designated as the linked column by setting the **ColLink** property to the Index position of the column. Only one column at a time can be defined as the linked column. If this property is set to zero (0) then no one column is linked and data for all columns for a selected row is treated as the linked data.

The data of the linked column is passed to Edit or Static portion of the control when a row is selected. The data in the Edit or Static portion is subsequently passed to the **DataSource** to update the **DataField**, if the these properties are defined.

ColListField Property

See Also   Applies To

Returns or sets the name of the field in the **Recordset** object used to fill a column in the list portion of the **VPComboBox** control or the list of the **VPListBox** control. Not available at run time.

Syntax

object.**ColListField** [= *fieldname*]

The **ColListField** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
fieldname	A string expression which specifies the name of a field in the **Recordset** created by the **Data** control specified by the **RowSource** property.

Remarks

This is an array property of optional field names to be bound to each column. The **ColListField** property enables you to select which field in the **Recordset** is used to fill a column in the list of a **VPComboBox** or **VPListBox** control. This property is used in conjunction with the **RowSource** property that specifies which **Data** control is used to create the **Recordset** used to fill the list. When setting this property in run mode, the affected column is specified using the **Col** property.

Generally, you use two **Recordset** objects with the data-aware list controls of **Vantage Control Set**. One **Recordset** contains a read-only list of valid selections, while the other **Recordset** is updated with selections from the list. For example, the **VPListBox** control's list could be generated from a query that returned a result set of valid part numbers and their descriptions. One column of the list would be bound to part numbers field of the **Recordset** through the **ColListField** property. The other column would be bound to the description field of the **Recordset** through its **ColListField** property. The **ColLink** property would identify the first column of part numbers as bound to the part number field of the second **Recordset**, as this is what needs to be updated.

If the field specified by the **ColListField** property can't be found in the **Recordset**, a trappable error occurs. This property can only be referenced within the **Column Layout Properties Dialog** at design time.

ColSortBy Property

See Also   Applies To

Returns or sets a value that determines if a given column of data will be used in sorting the contents of a List.

Syntax

object.**ColSortBy** [= *boolean*]

The **ColSortBy** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
boolean	A Boolean expression which specifies the behavior described in Settings.

Settings

The settings for *boolean* are:

Setting	Description
True	A column will be used when determining the order of items within a List.
False	Sorting within a List is not based on a given column. (Default)

Remarks

The **ColSortBy** property is an array of Boolean flags which determines if a given column is used as a basis for determining the order of items within a List. When setting this property in run mode, the affected column is specified using the **Col** property. If no columns have this property set to **True**, all the columns as a whole will be used as a basis for determining sort sequence and each full row of data is evaluated in a left-justified, ascending sort order to determine the sequence of items within a List. Once a column is designated as a SortBy column, its sorting sequence is determined by the **ColSortOrder**, and **ColAlign** properties, along with the general **CaseSensitive** property of the List.

ColSortOrder Property

See Also   Applies To

Returns or sets a value that determines the sorting order to be used within a column in a List.

Syntax

object.**ColSortOrder** [= *value*]

The **ColSortOrder** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
value	A value or constant which determines the sort order, as described in Settings.

Settings

The settings for *value* are:

Setting	Description
0	Ascending. (Default)
1	Descending.

Remarks

This is an array property of sort order definitions for each column. Any column can have a sorting order that is different from other columns. When setting this property in run mode, the affected column is specified using the **Col** property. This property works in conjunction with the **ColAlign** property, and the **CaseSensitive** property, to determine a proper sort sequence within a column. This sort sequence determines how items within a List are positioned. The **ColSortOrder** property determines if the sorting sequence for a column is in ascending or descending order. The **ColAlign** property sets how columnar text values are evaluated in a left-to-right or right-to-left ASCII sequence. Typically column text values that represent numeric data use a right justification, while alpha or alphanumeric data use left or center justification. Even with this property set, sorting items within a list will only be based on a given column if that column's **ColSortBy** property is set to **True**.

ColWidth Property

See Also Applies To

Returns or sets the width of the specified column in twips.

Syntax

object.**ColWidth** [= *number*]

The **ColWidth** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
number	A numeric expression which specifies the column width.

Remarks

This is an array property of column widths. You can use this property to set the width of any column in a **VPComboBox** or **VPListBox** control. Column widths are always defined in twips. When setting this property in run mode, the affected column is specified using the **Col** property.

ControlIndex Property

See Also   Example   Applies To

Returns a value that is used as an index into the **Control Collection** of a **Form** object.

Syntax

object.**ControlIndex**

The *object* placeholder represents an object expression which evaluates to an object in the Applies To list.

Return Values

The **ControlIndex** property return value is an *index* pointing within a **Form** object's **Control Collection**.

Remarks

The **ControlIndex** property's return value serves as an index into a **Form** object's **Control Collection**. Which **Form** object is determined by the **FormIndex** property. The **ControlIndex** and **FormIndex** properties are used in tandem to identify or test for a particular control object. The use of the **ControlIndex** property is dependent on if the property is being used in conjunction with a **VPForm** or **VPFocus** control.

In the **VPForm** control the **ControlIndex** property is used in conjunction with the **FormIndex** property to identify which control the mouse pointer is currently over. It is used in custom drag-and-drop operations associated with the **VPForm** control. If the mouse pointer is currently over a form or the desktop, and not over a control, the **ControlIndex** property will return a -1 value. You should check for this condition before using this property as an index within a **Form** object's **Control Collection.**

In the **VPFocus** control the **ControlIndex** property is used in conjunction with the **FormIndex** property to identify which control **Focus** is to be sent to next. This can be helpful in determining if normal processing in a **LostFocus** event should be continued or bypassed.   As an example, you may not want validation criteria to be evaluated for a control's contents if the Cancel or Exit command button has been clicked.

Cursor Property

See Also Example Applies To

Returns or sets the cursor image to be displayed as the mouse pointer in a drag-and-drop operation. Not available at design time.

Syntax

object.**Cursor** [= *cursor*]

The **Cursor** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
cursor	Specifies a graphic resource reference within a DLL or VBX.

Settings

The settings for *cursor* are:

Setting	Description
null string	Custom cursor is destroyed and the default mouse pointer is restored.
cursor	Any code reference which produces a custom mouse pointer. This code reference is made up of the name of the DLL or VBX module and the cursor resource name or id number. These two parameters are to be separated by one or more spaces.

Remarks

The **Cursor** property is a property of the **VPForm** control and can be set only during run time. This property provides a mechanism for assigning and displaying custom cursors (mouse pointers). It is used in conjunction with the **MouseCapture** property when using the custom drag-and-drop features of the **VPForm** control. Setting the **Cursor** property will change the mouse pointer until it is reset to another custom cursor reference or to a null string. If set to a null string the mouse pointer is returned to its previous Visual Basic supplied cursor. There are several custom cursor resources which you can use that are part of the VFORM.VBX file. When using these custom cursor resources, it is not necessary to include the module name along with the resource id. Actually, it is more efficient if you just assign the resource id so that the **VPForm** control does not do a LoadLibrary API call for each cursor assignment. The custom cursors of the **VPForm** control include the following:

Cursor	Resource	Resource ID
	IDCUR_NODROP	6001
	IDCUR_HANDBOX	6002

IDCUR_HANDOBJ	6003
IDCUR_HANDDOC	6004
IDCUR_HANDDOCS	6005
IDCUR_ARROWDOCS	6006
IDCUR_ARROWFLDR	6007
IDCUR_DROPDOC	6008
IDCUR_DROPFLDR	6009
IDCUR_HANDWAND	6010
IDCUR_HANDPT01	6011
IDCUR_HANDPT02	6012
IDCUR_HANDPT03	6013
IDCUR_ARROWPT01	6014
IDCUR_ARROWPT02	6015
IDCUR_ARROWPT03	6016
IDCUR_ARROWPT04	6017
IDCUR_DISK01	6018
IDCUR_DISK02	6019
IDCUR_FORM	6020

IDCUR_MAIL	6021
IDCUR_CLOCK	6022
IDCUR_IDEA	6023
IDCUR_WATCH	6024
IDCUR_PLUS01	6025
IDCUR_PLUS02	6026
IDCUR_CROSS	6027
IDCUR_KEY	6028
IDCUR_CLIP	6029
IDCUR_BOOK	6030
IDCUR_DOCS01	6031
IDCUR_DOCS02	6032
IDCUR_WRITE	6033
IDCUR_HANDITM01	6034
IDCUR_HANDITM02	6035

DataChanged Property

See Also   Applies To

Returns or sets a value indicating that the data in the bound control has been changed by some process other than that of retrieving data from the current record. Not available at design time.

Syntax

object.**DataChanged** [= *boolean*]

The **DataChanged** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
boolean	A Boolean expression hat indicates whether data has changed, as described in Settings.

Settings

The settings for *boolean* are:

Setting	Description
True	The data currently in the control isn't the same as in the current record.
False	The data currently in the control, if any, is the same as the data in the current record. (Default)

Remarks

When a **Data** control moves from record to record, it passes data from fields in the current record to controls bound to the specific field or the entire record. As data is displayed in the bound controls, the **DataChanged** property is set to **False**. If the user or any other operation changes the value in the bound control, the **DataChanged** property is set to **True**. Simply moving to another record doesn't affect the **DataChanged** property.

When the **Data** control starts to move to a different record, the **Validate** event occurs. If **DataChanged** is **True** for any bound control, the **Data** control automatically invokes the **Edit** and **Update** methods to post the changes to the database.

If you don't wish to save changes from a bound control to the database, you can set the **DataChanged** property to **False** in the **Validate** event.

Inspect the value of the **DataChanged** property in your code for a control's **Change** event to avoid a cascading event. This applies to both bound and unbound controls.

For the **VPComboBox** control this property is effected by either the values in the bound column or the Edit or Static portion of the control, depending on if you are using one or both binding methods.

DataField Property

See Also Applies To

Returns or sets a value that binds a control to a field in the current record.

Syntax

object.**DataField** [= *value*]

The **DataField** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
value	A string expression which evaluates to the name of one of the fields in the **Recordset** object specified by a **Data** control's **RecordSource** and **DatabaseName** properties.

Remarks

Bound controls provide access to specific data in your database. Bound controls that manage a single field typically display the value of a specific field in the current record. The **DataSource** property of a bound control specifies a valid **Data** control name, and the **DataField** property specifies a valid field name in the **Recordset** object created by the **Data** control. Together, these properties specify what data appears in the bound control.

DataSource Property

See Also   Applies To

Sets a value that specifies the **Data** control through which the current control is bound to a database. Not available at run time.

Remarks

To bind a control to a field in a database at run time, you must specify a **Data** control in the **DataSource** property at design time using the Properties window.

To complete the connection with a field in the **Recordset** managed by the **Data** control, you must also provide the name of a **Field** object in the **DataField** property. Unlike the **DataField** property, the **DataSource** property setting isn't available at run time.

FocusAction Property

See Also Example Applies To

Returns or sets the type of **Focus** management action to be executed. Not available at design time.

Syntax

object.**FocusAction** [= *value*]

The **FocusAction** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
value	A numeric expression specifying the type of **Focus** management action to be executed, as described in Settings.

Settings

The settings for *boolean* are:

Setting	Description
0	Turns off **Focus** control, **ActiveControl** property is set to zero (0). Can use the constant **vxTrapFocusOff** instead.
1	Sends **Focus** on to the trapped control and sets the **ActiveControl** property to zero (0). Can use the constant **vxSendFocus** instead.
2	Returns **Focus** to the current "active" control. Can use the constant **vxReturnFocus** instead.

Remarks

The **FocusAction** property is a custom property which acts as a custom method. Setting this property to one of the three action properties executes certain **Focus** management operations.

ForceGotFocus Property

See Also Applies To

Returns or sets a value that determines if special processing must be done to insure that a **GotFocus** event is fired in the target control that will next receive **Focus**. Only the ActiveX version has this property. Not available at design time.

Syntax

object.**ForceGotFocus** [= *boolean*]

The **ForceGotFocus** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
boolean	A Boolean expression which specifies whether special processing should be executed as defined in Settings.

Settings

The settings for *boolean* are:

Setting	Description
True	Special processing should be executed to insure a **GotFocus** event is generated for the target control.
False	No special processing should be executed. (Default)

Remarks

The **ForceGotFocus** property should only be set to **True** in the case where a programmer displays a message box in the **LostFocus** event code of the current "Active" control before sending **Focus** on. Setting this property to **True** before executing the **SendFocus** Method, or setting the **FocusAction** property to the **vxSendFocus** value, will insure the control recieving **Focus** will fire its **GotFocus** event. Setting the **ForceGotFocus** property to **True**, when you have not displayed a message box, is not necessary, and can interfere with other processing of the control that receives **Focus**, such as a target command button control that might miss its click event. This property is only needed and is available to the ActiveX version of the **VPFocus** control.

FormControlName Property

See Also Example Applies To

Returns a string that represents the form object name, control object name, and optional array index position.

Syntax

object.**FormControlName**

The *object* placeholder represents an object expression which evaluates to an object in the Applies To list.

Return Values

The **FormControlName** property returns a string value which provides an alternate way to identify or test for a particular form or control object. This property can be used instead of the **FormIndex** and **ControlIndex** properties. The returned string value of this property is made up of three string parameters delimited by the period character (.). These parameters or segments include a form name, a control name, and a control array index position. The index parameter will be null if the referenced control is not part of a control array.

The use of the **FormControlName** property is dependent on if the property is being used in conjunction with a **VPForm** or **VPFocus** control.

In the **VPForm** control, the **FormControlName** property is used to identify which form and which control the mouse pointer is currently over. It can be used in custom drag-and-drop operations associated with the **VPForm** control.

In the **VPFocus** control, the **FormControlName** property is used to identify which form and which control **Focus** is to be sent to next. This can be helpful in determining if normal processing in a **LostFocus** event should be continued or bypassed. As an example, you may not want validation criteria to be evaluated for a control's contents if the Cancel or Exit command button has been clicked.

FormIndex Property

See Also   Example   Applies To

Returns a value that is used as an index into the **Form Collection** of an application.

Syntax

object.**FormIndex**

The *object* placeholder represents an object expression which evaluates to an object in the Applies To list.

Return Values

The **FormIndex** property return value is an *index* pointing within an application's **Form Collection**.

Remarks

The **FormIndex** property's return value serves as an index into an application's **Form Collection**. The **FormIndex** and **ControlIndex** properties are used in tandem to identify or test for a particular control object. The use of the **FormIndex** property is dependent on if the property is being used in conjunction with a **VPForm** or **VPFocus** control.

In the **VPForm** control, the **FormIndex** property is used in conjunction with the **ControlIndex** property to identify which control the mouse pointer is currently over. It is used in custom drag-and-drop operations associated with the **VPForm** control.

In the **VPFocus** control, the **FormIndex** property is used in conjunction with the **ControlIndex** property to identify which control **Focus** is to be sent to next. This can be helpful in determining if normal processing in a **LostFocus** event should be continued or bypassed. As an example, you may not want validation criteria to be evaluated for a control's contents if the Cancel or Exit command button has been clicked.

GridAppearance Property

See Also Applies To

Returns or sets the paint style of any grid lines displayed within the list portion of a control.

Syntax

object.**GridAppearance** [= *number*]

The **GridAppearance** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
number	A numeric expression which specifies the type of appearance, as described in Settings.

Settings

The settings for *number* are:

Setting	Description
0	Flat. Paints grids lines without visual effects. (Default)
1	3D Inset. Paints grids with a lowered three-dimensional effect.
2	3D Raised. Paints grids with a raised three-dimensional effect.

Remarks

If set to 1 or 2 at either design time or run time, the **GridAppearance** property draws the grid lines within lists with three-dimensional effects. The grid can have a normal flat appearance or acquire a three-dimensional look by drawing a 2-pixel border on the inside of each cell frame. This **GridAppearance** property will have no effect If the **GridLines** property is set to None (0).

GridLines Property

See Also Applies To

Returns or sets a value that determines whether the list portion of controls display grid lines and the type or grid lines displayed.

Syntax

object.**GridLines** [= *number*]

The **GridLines** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
number	A numeric expression which specifies the type of grid lines, as described in Settings.

Settings

The settings for *number* are:

Setting	Description
0	None. (Default)
1	Horizontal grid lines are displayed.
2	Vertical grid lines are displayed.
3	Both horizontal and vertical grid lines are displayed.

Remarks

The **GridLines** property determines whether lines are display within the list portion of the **VPComboBox** or **VPListBox** controls. This property further determines whether the grid lines include horizontal lines between the rows of a list, vertical lines between the columns of a list, or both types of lines. The associated **GridAppearance** property determines if the representation of these grid lines are normal lines or take on a 3D display appearance.

Heading Property

See Also Applies To

Returns or sets a value that determines if a fixed row is displayed for column headings or captions within the List portion of a control.

Syntax

object.**Heading** [= *boolean*]

The **Heading** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
boolean	A Boolean expression which specifies whether the list has a heading, as described in Settings.

Settings

The settings for *boolean* are:

Setting	Description
True	List portion of control displays a fixed column heading row.
False	No heading row is displayed. (Default)

Remarks

The **VPComboBox** and **VPListBox** controls can display an optional fixed row which holds headings or captions for each column of the List. The columns of this heading row have their own alignment and captions and the fixed row has its own color properties. Whether this fixed heading row is displayed is determined by the **Heading** property which can be set to **True** or **False.** If set to **False**, any properties that define the headings, alignments, or colors for the heading row are ignored.

HeadingBackColor Property

See Also   Applies To

Returns or sets the background color of the fixed heading row associated with the List portion of an object.

Syntax

object.**HeadingBackColor** [= *color*]

The **HeadingBackColor** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
color	A value or constant which determines the background color of an object, as described in Settings.

Settings

The settings for *color* are:

Setting	Description
Normal RGB colors	Colors specified by using the Color palette or by using the RGB or QBColor functions in code.
System default colors	Colors specified by system color constants listed in the Visual Basic (VB) object library in the Object Browser (VB4). The Windows operating environment substitutes the user's choices as specified in the Control Panel settings. (VB4 only)

Remarks

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 (in VB4) by constants listed in the Visual Basic (VB) object library in the Object Browser.

To display text in the Windows operating environment, both the text and background colors must be solid. If the text or background colors you've selected aren't displayed, one of the selected colors may be dithered - that is, comprised of up to three different-colored pixels. If you choose a dithered color for either the text or background, the nearest solid color will be substituted.

If the **Heading** property is set to **False**, the **HeadingBackColor** property will be ignored.

HeadingForeColor Property

See Also   Applies To

Returns or sets the foreground color of the fixed heading row associated with the List portion of an object.

Syntax

object.**HeadingForeColor** [= *color*]

The **HeadingForeColor** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
color	A value or constant which determines the foreground color of an object, as described in Settings.

Settings

The settings for *color* are:

Setting	Description
Normal RGB colors	Colors specified by using the Color palette or by using the RGB or QBColor functions in code.
System default colors	Colors specified by system color constants listed in the Visual Basic (VB) object library in the Object Browser (VB4). The Windows operating environment substitutes the user's choices as specified in the Control Panel settings. (VB4 only)

Remarks

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 (in VB4) by constants listed in the Visual Basic (VB) object library in the Object Browser.

To display text in the Windows operating environment, both the text and background colors must be solid. If the text or background colors you've selected aren't displayed, one of the selected colors may be dithered - that is, comprised of up to three different-colored pixels. If you choose a dithered color for either the text or background, the nearest solid color will be substituted.

If the **Heading** property is set to **False**, the **HeadingForeColor** property will be ignored.

HideSelection Property

Applies To

Returns a value that determines whether selected text appears highlighted when a control loses the **Focus**.

Syntax

object.**HideSelection**

The object placeholder represents an object expression which evaluates to an object in the Applies To list.

Return Values

The **HideSelection** property return values are:

Value	Description
True (Default)	Selected text doesn't appear highlighted when the control loses the **Focus**.
False	Selected text appears highlighted when the control loses the **Focus**.

Remarks

You can use this property to indicate which text is highlighted while another form or a dialog box has the **Focus** - for example, in a spell-checking routine.

ListAppearance Property

Applies To

Returns or sets the paint style of the Dropdown List of a control.

Syntax

object.**ListAppearance** [= *number*]

The **ListAppearance** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
number	An numeric expression which specifies the type of appearance, as described in Settings.

Settings

The settings for *number* are:

Setting	Description
0	Flat. Paints List without visual effects. (Default)
1	3D Inset. Paints List with a lowered three-dimensional effect.
2	3D Raised. Paints List with a raised three-dimensional effect.

Remarks

If set to 1 or 2 at either design time or run time, the **Listppearance** property draws a Dropdown List with three-dimensional effects.

ListBackColor Property

See Also Applies To

Returns or sets the background color of the List portion of an object.

Syntax

object.**ListBackColor** [= *color*]

The **ListBackColor** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
color	A value or constant which determines the background color of an object, as described in Settings.

Settings

The settings for *color* are:

Setting	Description
Normal RGB colors	Colors specified by using the Color palette or by using the RGB or QBColor functions in code.
System default colors	Colors specified by system color constants listed in the Visual Basic (VB) object library in the Object Browser (VB4). The Windows operating environment substitutes the user's choices as specified in the Control Panel settings. (VB4 only)

Remarks

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 (in VB4) by constants listed in the Visual Basic (VB) object library in the Object Browser.

To display text in the Windows operating environment, both the text and background colors must be solid. If the text or background colors you've selected aren't displayed, one of the selected colors may be dithered - that is, comprised of up to three different-colored pixels. If you choose a dithered color for either the text or background, the nearest solid color will be substituted.

The **ListBackColor** property is provided for the **VPComboBox** control to allow for setting of colors for the List portion of the control, independent from the Edit or Static portion of the control, which is controlled by the standard **BackColor** property.

ListForeColor Property

See Also   Applies To

Returns or sets the foreground color of the List portion of an object.

Syntax

object.**ListForeColor** [= *color*]

The **ListForeColor** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
color	A value or constant which determines the foreground color of an object, as described in Settings.

Settings

The settings for *color* are:

Setting	Description
Normal RGB colors	Colors specified by using the Color palette or by using the RGB or QBColor functions in code.
System default colors	Colors specified by system color constants listed in the Visual Basic (VB) object library in the Object Browser (VB4). The Windows operating environment substitutes the user's choices as specified in the Control Panel settings. (VB4 only)

Remarks

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 (in VB4) by constants listed in the Visual Basic (VB) object library in the Object Browser.

To display text in the Windows operating environment, both the text and background colors must be solid. If the text or background colors you've selected aren't displayed, one of the selected colors may be dithered - that is, comprised of up to three different-colored pixels. If you choose a dithered color for either the text or background, the nearest solid color will be substituted.

The **ListForeColor** property is provided for the **VPComboBox** control to allow for setting of colors for the List portion of the control, independent from the Edit or Static portion of the control, which is controlled by the standard **ForeColor** property.

Locked Property

Applies To

Returns or sets a value indicating whether a control can be edited.

Syntax

object.**Locked** [= *boolean*]

The **Locked** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
boolean	A Boolean expression which specifies whether the control can be edited, as described in Settings.

Settings

The settings for *boolean* are:

Setting	Description
True	You can scroll and highlight the text in the control, but you can't edit it. The program can still modify the text by changing the **Text** property.
False	You can edit the text in the control. (Default)

Remarks

The **Locked** property when set to **True**, provides a display only mode for the **VPTextBox** and **VPComboBox** controls.

MatchEntry Property

See Also Applies To

Returns or sets a value that determines how the List portion of the **VPComboBox** control is searched, based on values entered into the Edit portion of the control. Applies only to a **VPComboBox** control with the **Style** property set to **Dropdown Combo**.

Syntax

object.**MatchEntry** [= *value*]

The **MatchEntry** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
value	A value or constant which specifies the type of search within the List portion, as described in Settings.

Settings

The settings for *value* are:

Setting	Description
0	No matching. As characters are type in to the edit portion, no searching occurs.
1	Standard matching. The control searches for an item with beginning characters matching all the characters entered. The search is done as characters are being typed, or backspaced, further refining the search. Any matching items row will be highlighted within the List portion, but not copied to the Edit or Static portion of the control until the user hits the Enter key, the control losses focus, or the user clicks on an item. This type of matching is modeled after the search dialog in the Windows help system. (Default)
2	Extended matching. The control waits until the user types in enough characters into the Edit portion of the control to uniquely match an item in the List portion. When a match is found the item is selected and the linked column is loaded and displayed in the Edit portion of the control. The portions not typed by the user are selected for easy over-typing. This type of matching is modeled after searches found in controls used in Microsoft Money and Intuits Quicken.

Remarks

The **MatchEntry** property allows for specific searching behavior that would otherwise require significant coding. A matching or auto searching property such as the **MatchEntry** property becomes very important when the List portions are either not sorted and have a large amount of items.

MaxCols Property

See Also   Example   Applies To

Returns or sets the number of columns for the List portion of a control. Not available during design time.

Syntax

object.**MaxCols** [= *number*]

The **MaxCols** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
number	A numeric expression representing the total number of the columns (1 to 99) for a List.

Remarks

The **MaxCols** property provides a mechanism to set the number of columns for a List a run time. If the number of columns set is less then the current number of columns, the trailing column definitions are deleted from the List. If the number of columns set is greater then the current number of columns, new column definitions will be added to the List. Any columns added will have the following default values:

Property	Default Value
ColAlign	0
ColFormat	"" (null string)
ColHeading	"" (null string)
ColHeadAlign	0
ColListField	"" (null string) or the next ordinal **Field Name** object if the List is bound
ColWidth	1000 (twips)

MaxDrop Property

Applies To

Returns or sets the number of items or rows that are displayed with the List portion of a control is dropped down.

Syntax

object.**MaxDrop** [= *number*]

The **MaxDrop** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
number	A numeric expression which specifies the number of rows to be displayed. Default number is 8.

Remarks

The **MaxDrop** property determines the size of the drop-down List associated with the **VPComboBox** control by setting the number of items or rows that are displayed when the List is dropped down.

MaxLength Property

Applies To

Returns or sets a value indicating whether there is a maximum number of characters that can be entered in the TextBox or **VPTextBox** control and, if so, specifies the maximum number of characters that can be entered.

Syntax

object.**MaxLength** [= value]

The **MaxLength** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
value	An integer specifying the maximum number of characters a user can enter in a **TextBox** or **VPTextBox** control. The default for the MaxLength property is 0, indicating no maximum other than that created by memory constraints on the user's system for single-line TextBox controls and a maximum of approximately 32K for multiple-line **TextBox** or **VPTextBox** controls. Any number greater than 0 indicates the maximum number of characters.

Remarks

Use the **MaxLength** property to limit the number of characters a user can enter in a **TextBox** or **VPTextBox** control.

If text that exceeds the **MaxLength** property setting is assigned to a **TextBox** or **VPTextBox** from code, no error occurs; however, only the maximum number of characters is assigned to the **Text** property, and extra characters are truncated. Changing this property doesn't affect the current contents of a **TextBox** or **VPTextBox** control but will affect any subsequent changes to the contents.

MaxWidth Property

See Also   Applies To

Returns or sets the maximum width for all columns with the List portion of the **VPComboBox** control.

Syntax

object.**MaxWidth** [= *number*]

The **MaxWidth** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
number	A numeric expression specifying the width dimensions of an object in twips.

Remarks

The **MaxWidth** property can be used to set the display width of the List portion of the **VPComboBox** control. The width is always expressed in twips. If the total of the individual column widths exceeds this property, a horizontal scrollbar is provided at the bottom of the List portion of the control. If the total of the individual column widths is less then this **MaxWidth** property, the **MaxWidth** property is ignored.

A **MaxWidth** property value of zero (0) indicates that there is no maximum width defined and the List portion of the control will be as wide as necessary to display all the columns.

MouseCapture Property

See Also Example Applies To

Returns or sets a value that determines if the **VPForm** control traps all mouse events and fires custom mouse events associated with the **VPForm** control. Not available at design time.

Syntax

object.**MouseCapture** [= *boolean*]

The **MouseCapture** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
boolean	A Boolean expression which controls the capture of mouse events, as described in Settings.

Settings

The settings for *boolean* are:

Setting	Description
True	The **VPForm** control captures all mouse events and fires custom mouse events.
False	The capturing of mouse events is turned off.

Remarks

The **MouseCapture** property is a property of the **VPForm** control and can be set only during run time. This property turns on or off the capturing of mouse events for an application. Setting this property to **True** captures all mouse events and allows for the firing of custom mouse events associated with the **VPForm** control. These custom mouse events include the **MouseDown**, **MouseMove**, and **MouseUp** events. This **MouseCapture** property, the **Cursor** property, and custom mouse events are provided to enable a unique and centralized approach to drag-and-drop operations for an application.

Normal Visual Basic drag-and-drop operations involve writing code utilizing special methods in several events and setting several properties for a number of form and control objects. Typically, you start by setting the **DragIcon** property for the "source" control (usually in a Form's **Load** event) and executing the **Drag** method in its **MouseDown** event. Next you need to code for **DragOver** and **MouseUp** events in each object that the mouse can travel over. This includes forms and any controls. Finally, for any target object, you also need to code for the **DragDrop** event to implement any operation associated with the drag-and-drop process, such as moving a control and transferring data from a source object to a destination object.

Normal Visual Basic drag-and-drop operations also limit what you can use for a custom cursor or mouse pointer. You can only reference **Icon** files and not true **Cursor** resources.

Under this arrangement you have no control of where a custom mouse pointer's hot spot is located.

With the **VPForm** control of **Vantage Control Set** you can control all drag-and-drop operations from the one **MouseDown** event of a "source" control and the custom **MouseMove** and **MouseUp** events of the **VPForm** control. In the "source" control's **MouseDown** event you turn on mouse capturing by setting the **MouseCapture** property of the **VPForm** control to **True**. From this point, until the **MouseCapture** property is reset to **False**, all mouse events are redirected to the mouse events of the **VPForm** control. In these centralized mouse events you can code for all drag-and-drop operations. You can determine which objects the mouse-pointer is over through the **FormIndex** and **ControlIndex** properties and set any custom mouse pointers using the **Cursor** property of the **VPForm** control. For a full example on how drag-and-drop operations can be controlled see the example help on this **MouseCapture** property.

MultiLine Property

Applies To

Returns or sets a value indicating whether a **TextBox** or **VPTextBox** control can accept and display multiple lines of text. Read only at run time.

Syntax

object.**MultiLine**

The object placeholder represents an object expression which evaluates to an object in the Applies To list.

Settings

The **MultiLine** property settings are:

Setting	Description
True	Allows multiple lines of text.
False	Ignores carriage returns and restricts data to a single line. (Default)

Remarks

A multiple-line **TextBox** or **VPTextBox** control wraps text as the user types text extending beyond the edit control.

You can also add scrollbars to larger **TextBox** or **VPTextBox** controls using the **ScrollBars** property. If no **HScrollBar** control (horizontal scroll bar) is specified, the text in a multiple-line **TextBox** or **VPTextBox** control automatically wraps.

Note On a form with no default button, pressing ENTER in a multiple-line **TextBox** or **VPTextBox** control moves the **Focus** to the next line. If a default button exists, you must press CTRL+ENTER to move to the next line.

MultiSelect Property

Applies To

Returns or sets a value indicating whether a user can make multiple selections in a **ListBox** or **VPListBox** control and how the multiple selections can be made. Read only at run time.

Syntax

object.**MultiSelect**

The object placeholder represents an object expression which evaluates to an object in the Applies To list.

Settings

The **MultiSelect** property settings are:

Setting	Description
0	Multiple selection isn't allowed. (Default)
1	Simple multiple selection. A mouse click or pressing the SPACEBAR selects or deselects an item in the list. (Arrow keys move the **Focus**.)
2 pressing	Extended multiple selection. Pressing SHIFT and clicking the mouse or SHIFT and one of the arrow keys (UP ARROW, DOWN ARROW, LEFT ARROW, and RIGHT ARROW) extends the selection from the previously selected item to the current item. Pressing CTRL and clicking the mouse selects or deselects an item in the list.

OverType Property

Applies To

Determines the operational mode of the edit portion of the **VPTextBox** and **VPComboBox** controls. The edit portion can be in an **Insert** or **Overtype** mode.

Remarks

The **OverType** property is not available in design time nor in run time through code. It can only be toggled on or off by the user using the keyboard **INS** Key. The different modes of operation use different carets as a visual cue. There is a normal vertical bar caret to indicate the insert position when in the **Insert** mode, and an underscore caret to indicate the replacement position when in the **Overtype** mode. The default mode of operation is the **Insert** mode.

PasswordChar Property

Applies To

Returns or sets a value indicating whether the characters typed by a user or placeholder characters are displayed in a **TextBox** or **VPTextBox** control; returns or sets the character used as a placeholder.

Syntax

object.**PasswordChar** [= *value*]

The **PasswordChar** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
value	A string expression specifying the placeholder character.

Remarks

Use this property to create a password field in a dialog box. Although you can use any character, most Windows-based applications use the asterisk (*) (Chr$(42)).

This property doesn't affect the **Text** property; Text contains exactly what the user types or what was set from code. Set **PasswordChar** to a zero-length string (""), which is the default, to display the actual text.

You can assign any string to this property, but only the first character is significant; all others are ignored.

Note If the **MultiLine** Property is set to **True**, setting the **PasswordChar** property will have no effect.

RowSource Property

See Also   Applies To

Sets a value that specifies the **Data** control from which the list portion of a **VPComboBox** or **VPListBox** control is filled. Not available at run time.

Remarks

To fill the list in a **VPComboBox** or **VPListBox** control, you must specify a **Data** control in the **RowSource** property at design time using the Properties window.

To complete the connection with a field in the **Recordset** object managed by the **Data** control, you must also provide the name of a **Field** object in the **ColListField** property. Unlike the **ColListField** property, the **RowSource** property setting isn't available at run time.

TargetControl Property

See Also   Example   Applies To

Returns or sets the window handle of the control where **Focus** will next be sent.

Syntax

object.**TargetControl** [= *value*]

The **TargetControl** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
value	The Window handle (Integer) of a control.

Remarks

The **TargetControl** property is an object pointer which records the control where Focus will next be sent to when using the **FocusAction** property or the **SendFocus** method.

This property is used for **Focus** processing within MDI child forms as a further check to make sure that spurious events are not processed. Within Visual Basic, if you transfer focus from one MDI child form to another and then back, the control that last had focus within an MDI child form will temporarily get focus again when you return to that form, even if focus is being sent to a different target control. Under these circumstances the **GotFocus** event procedure must check if the current control is the same as the target control, in addition to checking if the active control is undefined, before processing the **GotFocus** procedure code.

The **TargetControl** property can also be used to explicitly change the control where **Focus** will be sent when setting the **FocusAction** property to the SendFocus value or using the **SendFocus** custom method. You assign the **TargetControl** property the control handle of the control where you want **Focus** to be sent.

ScrollBars Property

Applies To

Returns or sets a value indicating whether an object has horizontal or vertical scroll bars. Read only at run time.

Syntax

object.**ScrollBars** [= *number*]

The **ScrollBars** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
number scrollbars	A numeric expression which specifies whether a control has one or more and the type of scrollbars, as described in Settings.

Settings

The settings for *number* are:

Setting	Description
0	None (Default)
1	Horizontal scrollbar
2	Vertical scrollbar
3	Both types of scrollbars

Remarks

For a **TextBox** or **VPTextBox** controls with setting 1 (Horizontal), 2 (Vertical), or 3 (Both), you must set the **MultiLine** property to **True**.

At run time, the Microsoft Windows operating environment automatically implements a standard keyboard interface to allow navigation in **TextBox** and **VPTextBox** controls with the arrow keys (UP ARROW, DOWN ARROW, LEFT ARROW, and RIGHT ARROW), the HOME and END keys, and so on.

Scroll bars are displayed on an object only if its contents extend beyond the object's borders. For example, a vertical scroll bar appears on a **TextBox** or **VPTextBox** control when it can't display all of its lines of text. If the **ScrollBars** property is set to 0, the **TextBox** or **VPTextBox** won't have scroll bars, regardless of its contents.

Sorted Property

See Also Apples To

Returns a value indicating whether the elements of a control are automatically sorted. Can only be set at design time.

Syntax

object.**Sorted**

The object placeholder represents an object expression which evaluates to an object in the Applies To list.

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
boolean	A Boolean expression which specifies whether the control List contents are sorted, as described in Settings.

Return Values:

The **Sorted** property return values are:

Setting	Description
True	List items are sorted based on defined sort criteria.
False	List items are not sorted. (Default)

Remarks

When the **Sorted** property is **True**, the control handles almost all necessary string processing to maintain a proper sort order, including changing the index numbers for items as required by the addition or removal of items. The type of sorting and which columns are used as sort keys are determined by the **ColSortBy** and **ColSortOrder** properties.

Style Property

Applies To

Returns or sets a value indicating the type of combo box and the behavior of its List portion. Read only at run time.

Syntax

object.**Style**

The object placeholder represents an object expression which evaluates to an object in the Applies To list.

Settings

The **Style** property settings are:

Setting	Description
0 can	**Dropdown Combo**. Includes a drop-down List and an Edit control. The user select from the List or type in the Edit portion. (Default)
1	**Simple Combo**. Includes an Edit control and a List, which doesn't drop down. The user can select from the List or type in the Edit portion. The size of a Simple combo box includes both the Edit and List portions. By default, a Simple combo box is sized so none of the list is displayed. Increase the **Height** property to display more of the list.
2	**Dropdown List**. This style only allows selection from the drop-down List and includes a Static portion to display the selection.

Remarks

Follow these guidelines in deciding which setting to choose:

Use setting 0 (**Dropdown Combo**) or setting 1 (**Simple Combo**) to give the user a list of choices. Either style enables the user to enter a choice in the edit portion. Setting 0 saves space on the form because the List portion closes when the user selects an item. The Dropdown style allows use of the **MatchEntry** property and its custom data searching capabilities.

Use setting 2 (**Dropdown List**) to display a fixed list of choices from which the user can select one. The List portion closes when the user selects an item. With this style the control supports first character matching.

Text Property

Applies To

For **ComboBox** or **VPComboBox** controls (Style property set to 0 [Dropdown Combo] or to 1 [Simple Combo]) and **TextBox** or **VPTextBox** controls, this property returns or sets the text contained in the Edit area.

For **ComboBox** or **VPComboBox** controls (Style property set to 2 [Dropdown List]) and **ListBox** or **VPListBox** controls, this property returns the selected item in the List; the value returned is always equivalent to the value returned by the expression `List(ListIndex)`. Read-only at design time; read-only at run time.

For **Grid** control, this property returns or sets the text contained in a cell or range of cells. Not available at design time.

Syntax

object.**Text** [= *string*]

The **Text** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
string	A string expression specifying text.

Remarks

At design time only, the defaults for the **Text** property are:

ComboBox, **VPComboBox**, **TextBox**, and **VPTextBox** controls - the control's Name property.

ListBox and **VPListBox** controls - a zero-length string ("").

For a **ComboBox** or **VPComboBox** control with the **Style** property set to 0 (Dropdown Combo) or to 1 (Simple Combo) or for a **TextBox** or **VPTextBox** control, this property is useful for reading the actual string contained in the edit area of the control. For a **ComboBox**, **VPComboBox**, **ListBox**, or **VPListBox** control with the **Style** property set to 2 (Dropdown List), you can use the **Text** property to determine the currently selected item.

The **Text** setting for a **TextBox** or **VPTextBox** control is limited to 2048 characters unless the **MultiLine** property is **True**, in which case the limit is about 32K.

For a **Grid** control, you can add text to a single cell by setting the **Text** property. This property applies to the cell defined by the current values of the **Grid** control's **Row** and **Col** properties.

You can use the **Text** and **FillStyle** properties to add the same text to a highlighted range of cells. When **FillStyle** = 0, the text assigned to the **Text** property is added only to the cell defined by the current **Row** and **Col** property values. When **FillStyle** = 1, the text is added to all cells whose **CellSelected** property setting is **True**.

You can also use the **Clip** property to fill a range of cells. For example, you might want to paste a large block of information from the **Clipboard** into a **Grid** control.

WordWrap Property

Applies To

Returns or sets a value indicating whether a **Label** or **VPStatic** control with its **AutoSize** property set to **True** expands vertically or horizontally to fit the text specified in its **Caption** property.

Syntax

object.**WordWrap** [= *boolean*]

The **WordWrap** property syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
boolean	A Boolean expression specifying whether the control expands to fit the text, as described in Settings.

Settings

The settings for *boolean* are:

Setting	Description
True	The text wraps; the control expands or contracts vertically to fit the text and the size of the font. The horizontal size doesn't change.
False	The text doesn't wrap; the control expands or contracts horizontally to fit the length of the text and vertically to fit the size of the font and the number of lines. (Default)

Remarks

Use this property to determine how a **Label** or **VPStatic** control displays its contents. For example, a graph which changes dynamically might have a **Label** containing text that also changes. To maintain a constant horizontal size for the **Label** or **VPStatic** control and allow for increasing or decreasing text, set the **WordWrap** and **AutoSize** properties to **True**.

If you want a **Label** or **VPStatic** control to expand only horizontally, set **WordWrap** to **False**. If you don't want the **Label** or **VPStatic** control to change size, set **AutoSize** to False.

Note If **AutoSize** is set to **False**, the text always wraps, regardless of the size of the **Label** or **VPStatic** control or the setting of the **WordWrap** property. This may obscure some text because the **Label** or **VPStatic** control doesn't expand in any direction.

Change Event

Applies To

Indicates that the contents of a control have changed. How and when this event occurs varies with the control:

ComboBox or **VPComboBox** controls - changes in the text of the Edit portion of the control. For the standard **ComboBox** this event occurs only if the **Style** property is set to 0 (Dropdown Combo) or 1 (Simple Combo) and the user changes the text or you change the **Text** property setting through code. For the **VPComboBox** control this event also occurs when a selection is made from the List portion that changes the **Text** property.

Label or **VPStatic** controls - changes in the contents of the control. Occurs when a DDE link updates data or when you change the **Caption** property setting through code.

TextBox or **VPTextBox** controls - changes in the contents of the text box. Occurs when a DDE link updates data, when a user changes the text, or when you change the **Text** property setting through code.

Syntax

Sub *object*_**Change(**[*index* **As Integer**]**)**

The **Change** event syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
index	An integer which uniquely identifies a control if it's in a control array.

Remarks

The **Change** event procedure can synchronize or coordinate data display among controls. For example, you can use a **Change** event procedure to display data and formulas in a work area and results in another area.

Note A **Change** event procedure can sometimes cause a cascading event. This occurs when the control's **Change** event alters the control's contents, for example, by setting a property in code that determines the control's value, such as the **Text** property setting for a **TextBox** or **VPTextBox** control. To prevent a cascading event: 1) If possible, avoid writing a **Change** event procedure for a control that alters that control's contents. If you do write such a procedure, be sure to set a flag that prevents further changes while the current change is in progress, 2) Avoid creating two or more controls whose **Change** event procedures affect each other, for example, two **TextBox** or **VPTextBox** controls that update each other during their **Change** events.

Click Event

Applies To

Occurs when the user presses and then releases a mouse button over an object. It can also occur when the value of a control is changed.

For a **Form** object, this event occurs when the user clicks either a blank area or a disabled control. For a control, this event occurs when the user:

Clicks a control with the left or right mouse button. With a **CheckBox**, **CommandButton**, or **OptionButton** control, the **Click** event occurs only when the user clicks the left mouse button.

Selects an item in a **ComboBox**, **VPComboBox**, **VPListBox**, or **ListBox** control, either by pressing the arrow keys or by clicking the mouse button.

Presses the SPACEBAR when a **CommandButton**, **OptionButton**, or **CheckBox** control has the **Focus**.

Presses ENTER when a form has a **CommandButton** control with its **Default** property set to **True**.

Presses ESC when a form has a Cancel button - a **CommandButton** control with its **Cancel** property set to **True**.

Presses an access key for a control. For example, if the caption of a **CommandButton** control is "&Go", pressing ALT+G triggers the event.

You can also trigger the **Click** event in code by:

Setting a **CommandButton** control's **Value** property to **True**.

Setting an **OptionButton** control's **Value** property to **True**.

Changing a **CheckBox** control's **Value** property setting.

Syntax

Sub *object*_**Click (**[*index* **As Integer**]**)**

The **Click** event syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
index	An integer which uniquely identifies a control if it's in a control array.

Remarks

Typically, you attach a **Click** event procedure to a **CommandButton** control, **Menu** object, or **PictureBox** control to carry out commands and command-like actions. For the other applicable controls, use this event to trigger actions in response to a change in the control.

You can use a control's **Value** property to test the state of the control from code. Clicking a control generates **MouseDown** and **MouseUp** events in addition to the **Click** event. The order in which these three events occur varies from control to control. For example, for **ListBox**, **VPListBox**, and **CommandButton** controls, the events occur in this order: **MouseDown**, **Click**, **MouseUp**. But for **FileListBox**, **Label**, **Static**, or **PictureBox** controls, the events occur in this order: **MouseDown**, **MouseUp**, and **Click**. When you're attaching event procedures for these related events, be sure that their actions don't conflict. If the order of events is important in your application, test the control to determine the event order.

Note   To distinguish between the left, right, and middle mouse buttons, use the **MouseDown** and **MouseUp** events.

CloseUp Event

Applies To

Occurs when the List portion of the **VPComboBox** control is closed.

Syntax

Sub *object*_**CloseUp(**[*index* **As Integer**,]**)**

The **CloseUp** event syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
index	An integer which uniquely identifies a control if it's in a control array.

Remarks

The **CloseUp** event procedure can be used to trigger any processing that may be required after the user has made or even canceled a selection from the List portion of a **VPComboBox** control. This event will not fire if the **Style** property is set to 1 (Simple).

DblClick Event

Applies To

Occurs when the user presses and releases a mouse button and then presses and releases it again over an object.

For a form, the **DblClick** event occurs when the user double-clicks a disabled control or a blank area of a form. For a control, it occurs when the user:

Double-clicks a control with the left mouse button.

Double-clicks an item in a **ComboBox** or **VPComboBox** control whose **Style** property is set to 1 (Simple) or in a **FileListBox**, **ListBox**, **VPListBox**, **DBCombo** (VB4 only), or **DBList** (VB4 only) control.

Syntax

Sub *object*_**DblClick (**[*index* **As Integer**]**)**

The **DblClick** event syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
index	An integer which uniquely identifies a control if it's in a control array.

Remarks

The argument *Index* uniquely identifies a control if it's in a control array. You can use a **DblClick** event procedure for an implied action, such as double-clicking an icon to open a window or document. You can also use this type of procedure to carry out multiple steps with a single action, such as double-clicking to select an item in a List box and to close the dialog box.

To produce such shortcut effects in Visual Basic, you can use a **DblClick** event procedure for a List box or file List box in tandem with a default button - a **CommandButton** control with its **Default** property set to **True**. As part of the **DblClick** event procedure for the List box, you simply call the default button's **Click** event.

For those objects that receive **Mouse** events, the events occur in this order: **MouseDown**, **MouseUp**, **Click**, **DblClick**, and **MouseUp**.

If the **DblClick** event doesn't occur within the system's double-click time limit, the object recognizes another **Click** event. The double-click time limit may vary because the user can set the double-click speed in the Control Panel. When you're attaching procedures for these related events, be sure that their actions don't conflict. Controls that don't receive **DblClick** events may receive two clicks instead of a **DblClick**.

Note To distinguish between the left, right, and middle mouse buttons, use the **MouseDown** and **MouseUp** events.

DropDown Event

Applies To

Occurs when the List portion of a **ComboBox** or **VPComboBox** control is about to drop down; this event doesn't occur if a **ComboBox** or **VPComboBox** control's **Style** property is set to 1 (Simple Combo).

Syntax

Sub *object*_**DropDown (**[*index* **As Integer**]**)**

The **DropDown** event syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
index	An integer which uniquely identifies a control if it's in a control array.

Remarks

Use a **DropDown** event procedure to make final updates to a **ComboBox** or **VPComboBox** control's List before the user makes a selection. This enables you to add or remove items from the List using the **AddItem** or **RemoveItem** methods. This flexibility is useful when you want some interplay between controls - for example, if what you want to load into a **ComboBox** or **VPComboBox** List depends on what the user selects in an **OptionButton** group.

MouseDown Event

See Also Example Applies To

Occurs when the user presses a mouse button.

Syntax

Sub Form_MouseDown (*button* **As Integer**, *shift* **As Integer**, *x* **As Single**, *y* **As Single)**

Sub MDIForm_MouseDown (*button* **As Integer**, *shift* **As Integer**, *x* **As Single**, *y* **As Single)** (VB4 only)

Sub *object*_**MouseDown (**[*index* **As Integer**,] *button* **As Integer**, *shift* **As Integer**, *x* **As Single**, *y* **As Single)**

The **MouseDown** event syntax has these parts:

Part	Description
object	Returns an object expression which evaluates to an object in the Applies To list.
index	Returns an integer which uniquely identifies a control if it's in a control array.
button	Returns an integer which identifies the button that was pressed to cause the event. 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. Only one of the bits is set, indicating the button that caused the event.
shift	Returns an integer which corresponds to the state of the SHIFT, CTRL, and ALT keys when the button specified in the button argument is pressed. 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*	Returns a number which 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.

Remarks

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

When the **MouseCapture** property of the **VPForm** control is set to **True**, all mouse events are redirected to the mouse events of the **VPForm** control including the **MouseDown** event.

MouseMove Event

See Also Example Applies To

Occurs when the user moves the mouse.

Syntax

Sub Form_MouseMove (*button* **As Integer**, *shift* **As Integer**, *x* **As Single**, *y* **As Single)**

Sub MDIForm_MouseMove (*button* **As Integer**, *shift* **As Integer**, *x* **As Single**, *y* **As Single)** (VB4 only)

Sub *object*_**MouseMove (**[*index* **As Integer**,] *button* **As Integer**, *shift* **As Integer**, *x* **As Single**, *y* **As Single)**

The **MouseMove** event syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
index	An integer which uniquely identifies a control if it's in a control array.
button	An integer which 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 which 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 which 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.

Remarks

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.

When the **MouseCapture** property of the **VPForm** control is set to **True**, all mouse events are redirected to the mouse events of the **VPForm** control including the **MouseMove** event.

MouseUp Event

See Also Example Applies To

Occurs when the user releases a mouse button.

Syntax

Sub Form_MouseUp (*button* **As Integer**, *shift* **As Integer**, *x* **As Single**, *y* **As Single)**

Sub MDIForm_MouseUp (*button* **As Integer**, *shift* **As Integer**, *x* **As Single**, *y* **As Single)** (VB4 only)

Sub *object*_**MouseUp (**[*index* **As Integer**,] *button* **As Integer**, *shift* **As Integer**, *x* **As Single**, *y* **As Single)**

The **MouseUp** event syntax has these parts:

Part	Description
object	Returns an object expression which evaluates to an object in the Applies To list.
index	Returns an integer which uniquely identifies a control if it's in a control array.
button	Returns an integer which identifies the button that was released to cause the event. 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. Only one of the bits is set, indicating the button that caused the event.
shift	Returns an integer which corresponds to the state of the SHIFT, CTRL, and ALT keys when the button specified in the button argument is released. 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*	Returns a number which 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.

Remarks

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

When the **MouseCapture** property of the **VPForm** control is set to **True**, all mouse events are redirected to the mouse events of the **VPForm** control including the **MouseUp** event.

LocateText Method

See Also Example Applies To

Returns an index value to a row or item found based on the search text and locate parameters supplied. Available only in the ActiveX version.

Syntax

object.**LocateText** *text*, *col*, *start*, *stype*, *direction*, *scase*

The **LocateText** method syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
text	A string expression which represents the text value to search for. Required argument.
col	A numeric expression which specifies a column to search within the List. Required argument.
start	A numeric expression which specifies the starting row or item to begin the search. Required argument.
stype	A value or constant which specifies the type of search to be conducted, as described in Settings. Optional argument.
direction	A value or constant which specifies the direction to search within the List, as described in Settings. Optional argument.
scase	A value or constant which specifies the behavior of the locate operation, as described in Settings. Optional argument.

Settings

The settings for *stype* are:

Setting	Description
0	Exact match. Can use the constant **vxExactMatch** instead. (Default)
1	First characters match. Can use the constant **vxFirstCharMatch** instead.
2	Last characters match. Can use the constant **vxLastCharMatch** instead.
3	Sub-string match. Can use the constant **vxSubStringMatch** instead.

The settings for *direction* are:

Setting	Description
0	Down. Can use the constant **vxDown** instead. (Default)
1	Up. Can use the constant **vxUp** instead.

The settings for *scase* are:

Setting	Description
0	Locate operations within a List are not case sensitive. Can use the constant **vxCaseNotSensitive** instead.(Default)
1	Locate operations within a List are case sensitive. Can use the constant **vxCaseSensitive** instead.

Remarks

The **LocateText** method is used to search the contents of a List locating the first row or item which has column data that matches the supplied text. This method returns an integer index value identifying the location of the row or item that matches. If no match is made, executing this method, a -1 value is returned. How the search is conducted is determined by the arguments associated with this method.

The *col* argument identifies the column within the List to search within. Columns are numbered from 1 to however many columns are defined for a List. If a zero (0) is supplied, the search uses the whole row, including data from all columns to check for a match. In the case where the row is searched as a whole, the column delimiter character, Chr$(9), is ignored. This argument is required.

The *start* argument is used to designate the row or item of the List you want to start your search from. If a zero (0) or -1 value is supplied, the search starts with the first row or item of the List. To start a search with the last row or item you could pass the **ListCount** property -1 for this argument. If you passed the value of the **ListIndex** property, a search would start with the currently selected row or item of the List. This argument is required.

The *stype* argument is used to set what type of search should be conducted through the items of the List. The Locate Type options include *exact matching* (0), *first characters matching* (1), *last characters matching* (2), or *sub-string matching* (3).

Exact matching compares the locate text with the full text of the column being searched. Where both strings are equal a match is made and the row index position is returned.

First characters matching compares each character of the locate text with the first characters of the column being searched. Where both strings are equal, character for character, up to the length of the locate text, a match is made and the row index position returned. This type of matching is most appropriate for left, or centered justified, alphanumeric type data, that is part of the search column within a List.

Last characters matching compares each character of the locate text with the last characters of the column being searched. Where both strings are equal, character for character, up to the length of the locate text, a match is made and the row index position returned. This type of matching is most appropriate for right justified, numeric type data, that is part of the search column within a List.

Sub-string matching compares the locate text with any sub-set of characters within the column being searched. Where the sub-string of the locate text can be found within the search column text a match is made and the row index position is returned.

The *stype* argument is optional and if not supplied the default type is *Exact matching*.

The *direction* argument is used to set which direction a search should be conducted through the items of the List. If the *Down* option is supplied, the locate method operation will start at the supplied starting row, and search each subsequent row for the locate text. If the *Up* option is supplied, the locate method operation will start at the supplied starting row, and each previous row will be searched. The *direction* argument is optional and if not supplied the default direction is *Down*.

The *scase* argument determines if compare functions used by the search engine are case sensitive or case insensitive. If this argument is passed a zero (0), all locate operations within the List are not case sensitive. If this argument is passed a value of 1, all locate operations within the List are case sensitive. The *scase* argument is optional and if not supplied the default *case* is 0, case not sensitive.

This method is available only for the ActiveX version.

ReturnFocus Method

See Also Example Applies To

Returns **Focus** to the current active control. Available only in the ActiveX version.

Syntax

object.**ReturnFocus**

The *object* placeholder represents an object expression which evaluates to an object in the Applies To list.

Remarks

The **ReturnFocus** method is one of several custom methods used in **Focus** management by the **VPFocus** control. Using this method will send focus back to the current active control and is typically used when the data entered into a control fails the validation criteria set for the field. This method should usually be used in a valid **LostFocus** event procedure. For the ActiveX version of **Vantage Control Set**, this method should be used instead of setting the **FocusAction** property (although for backwards compatibility with the VBX version, you can still use the **FocusAction** property with the setting of 2, which will execute this method).

This method is available only for the ActiveX version.

SendFocus Method

See Also Example Applies To

Sends **Focus** on to the next control as trapped by the **VPFocus** control. Available only in the ActiveX version.

Syntax

object.**SendFocus**

The *object* placeholder represents an object expression which evaluates to an object in the Applies To list.

Remarks

The **SendFocus** method is one of several custom methods used in **Focus** management by the **VPFocus** control. Using this method will send focus on to the next control as trapped by the operation of the **VPFocus** control. This method also turns off **Focus** control, like the **TrapFocusOff** method and sets the **ActiveControl** property to zero (0).

This method is typically used when the data entered into a control passes the validation criteria set for the field. This method should usually be used in a valid **LostFocus** event procedure. For the ActiveX version of **Vantage Control Set**, this method should be used instead of setting the **FocusAction** property (although for backwards compatibility with the VBX version, you can still use the **FocusAction** property with the setting of 1, which will execute this method).

TrapFocusOff Method

See Also Applies To

Turns off **Focus** management operations and sets the **ActiveControl** property, of the **VPFocus** control, to the value zero (0). Available only in the ActiveX version.

Syntax

object.**TrapFocusOff**

The *object* placeholder represents an object expression which evaluates to an object in the Applies To list.

Remarks

The **TrapFocusOff** method is one of several custom methods used in **Focus** management by the **VPFocus** control. Using this method will set the **ActiveControl** property of the **VPFocus** control to a zero (0) value, turning off any **Focus** trapping. This method, if used, should be executed in a valid **LostFocus** event procedure. For the ActiveX version of **Vantage Control Set**, this method should be used instead of setting the **FocusAction** property (although for backwards compatibility with the VBX version, you can still use the **FocusAction** property with the setting of 0, which will execute this method). Under most circumstances this method is not needed. In a **LostFocus** event procedure the **SendFocus** or **ReturnFocus** methods are used instead.

This method is available only for the ActiveX version.

TrapFocusOn Method

See Also   Example   Applies To

Turns on **Focus** management operations and sets the **ActiveControl** property, of the **VPFocus** control, to the window handle of the current control. Available only in the ActiveX version.

Syntax

object.**TrapFocusOn (***hWnd* **As Integer)**

The **TrapFocusOn** method syntax has these parts:

Part	Description
object	An object expression which evaluates to an object in the Applies To list.
hWnd	The Window handle (Integer) of the current control.

Remarks

The **TrapFocusOn** method is one of several custom methods used in **Focus** management by the **VPFocus** control. Using this method will set the **ActiveControl** property of the **VPFocus** control to the Window handle of the current control. This method starts the trapping of **Focus** events and should usually be used in a valid **GotFocus** event procedure. For the ActiveX version of **Vantage Control Set**, this method should be used instead of setting the **ActiveControl** property directly (although for backwards compatibility with the VBX version, you can still set the **ActiveControl** property directly, which will execute this method).

This method is available only for the ActiveX version.

VLocateText Function

See Also   Example   Applies To

Returns an index value to a row or item found within a supplied **VPComboBox** or **VPListBox** control, based on the search text, and locate parameters supplied. This function can be declared from either the VPLIST.VBX or the VPCOMB.VBX control file.

Declare Syntax

Declare Function VLocateText Lib VPLIST.VBX (*object* **As Control**, **ByVal** *text* **As String**, **ByVal** *col* **As Integer**, **ByVal** *start* **As Integer**, **ByVal** *stype* **As Integer**, **ByVal** *direction* **As Integer**, **ByVal** *scase* **As Integer) As Integer**

Syntax

found **= VLocateText (***object*, *text*, *col*, *start*, *stype*, *direction*, *scase***)**

The **VLocateText** function syntax has these parts:

Part	Description
found	An integer value returned from the function that identifies the found row or item.
object	An object expression which evaluates to an object in the Applies To list.
text	A string expression which represents the text value to search for.
col	A numeric expression which specifies a column to search within the List.
start	A numeric expression which specifies the starting row or item to begin the search.
stype	A value or constant which specifies the type of search to be conducted, as described in Settings.
direction	A value or constant which specifies the direction to search within the List, as described in Settings.
scase	A value or constant which specifies the behavior of the locate operation, as described in Settings.

Settings

The settings for *stype* are:

Setting	Description
0	Exact match. Can use the constant **vxExactMatch** instead.
1	First characters match. Can use the constant **vxFirstCharMatch** instead.
2	Last characters match. Can use the constant **vxLastCharMatch** instead.
3	Sub-string match. Can use the constant **vxSubStringMatch** instead.

The settings for *direction* are:

Setting	Description
0	Down. Can use the constant **vxDown** instead.
1	Up. Can use the constant **vxUp** instead.

The settings for *scase* are:

Setting	Description
0	Locate operations within a List are not case sensitive. Can use the constant **vxCaseNotSensitive** instead.
1	Locate operations within a List are case sensitive. Can use the constant **vxCaseSensitive** instead.

Remarks

The **VLocateText** function is used to search the contents of a List locating the first row or item which has column data that matches the supplied text. This function returns an integer index value identifying the location of the row or item that matches. If no match is made executing this function, a -1 value is returned. How the search is conducted is determined by the parameters associated with this function. Unlike the **LocateText** Method, all parameters are required for this function.

The *object* parameter identifies the **VPComboBox** or **VPListBox** control that will be searched. You pass an object variable as Control to the function.

The *col* parameter identifies the column within the List to search within. Columns are numbered from 1 to however many columns are defined for a List. If a zero (0) is supplied, the search uses the whole row, including data from all columns to check for a match. In the case where the row is searched as a whole, the column delimiter character, Chr$(9), is ignored.

The *start* parameter is used to designate the row or item of the List you want to start your search from. If a zero (0) or -1 value is supplied, the search starts with the first row or item of the List. To start a search with the last row or item you could pass the **ListCount** property -1 for this parameter. If you passed the value of the **ListIndex** property, a search would start with the currently selected row or item of the List.

The *stype* parameter is used to set what type of search should be conducted through the items of the List. The Locate Type options include *exact matching* (0), *first characters matching* (1), *last characters matching* (2), or *sub-string matching* (3).

Exact matching compares the locate text with the full text of the column being searched. Where both strings are equal a match is made and the row index position is returned.

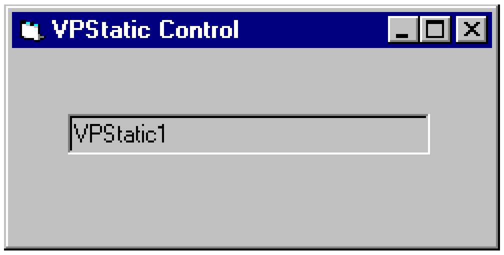
First characters matching compares each character of the locate text with the first characters of the column being searched. Where both strings are equal, character for character, up to the length of the locate text, a match is made and the row index position returned. This type of matching is most appropriate for left, or centered justified, alphanumeric type data, that is part of the search column within a List.

Last characters matching compares each character of the locate text with the last characters of the column being searched. Where both strings are equal, character for character, up to the length of the locate text, a match is made and the row index position returned. This type of matching is most appropriate for right justified, numeric type data, that is part of the search column within a List.

Sub-string matching compares the locate text with any sub-set of characters within the column being searched. Where the sub-string of the locate text can be found within the search column text a match is made and the row index position is returned.

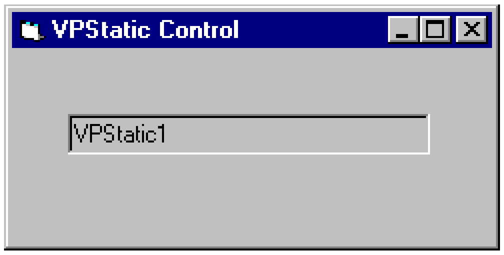
The *direction* parameter is used to set which direction a search should be conducted through the items of the List. If the *Down* option is supplied, the locate method operation will start at the supplied starting row, and search each subsequent row for the locate text. If the *Up* option is supplied, the locate method operation will start at the supplied starting row, and each previous row will be searched.

The *scase* parameter determines if compare functions used by the search engine are case sensitive or case insensitive. If this parameter is passed a zero (0), all locate operations within the List are not case sensitive. If this parameter is passed a value of 1, all locate operations within the List are case sensitive.



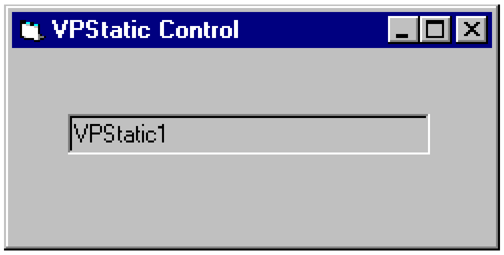
(About) Property Applies To

VPTextBox Control
VPStatic Control
VPComboBox Control
VPListBox Control
VPForm Control
VPFocus Control



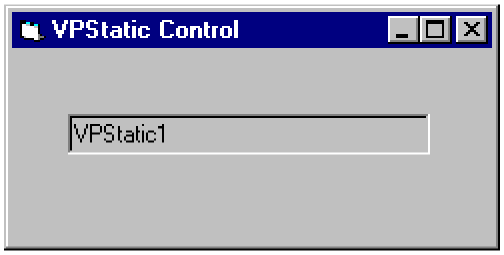
(ColLayout) Property Applies To

VPComboBox Control
VPListBox Control



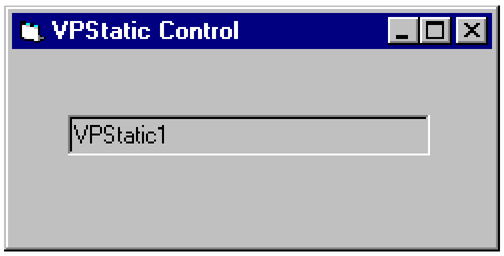
(Custom) Property Applies To

VPTextBox Control
VPStatic Control
VPComboBox Control
VPListBox Control
VPForm Control
VPFocus Control



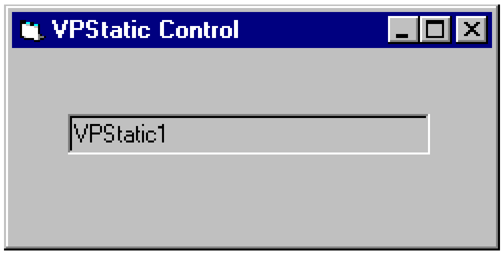
ActiveControl Property Applies To

VPFocus Control



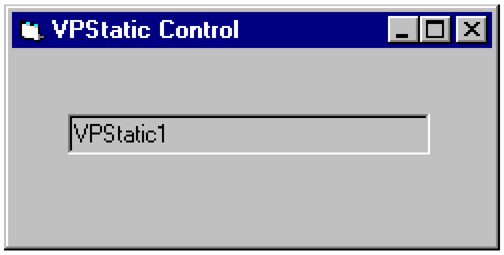
Alignment Property Applies To

VPTextBox Control
VPStatic Control
VPComboBox Control



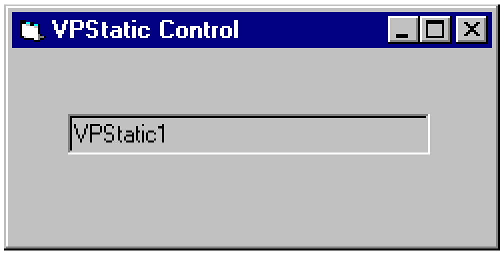
Appearance Property Applies To

VPTextBox Control
VPStatic Control
VPComboBox Control
VPListBox Control
VPForm Control



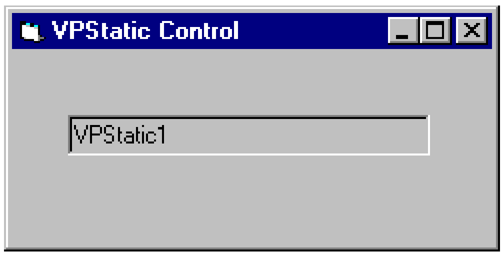
AutoHeight Property Applies To

VPComboBox Control



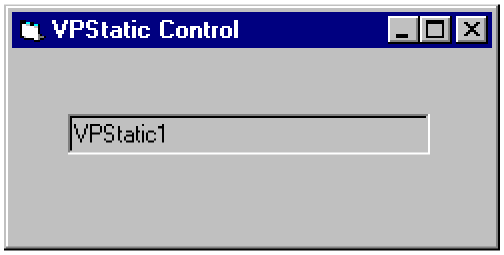
AutoSelect Property Applies To

VPTextBox Control
VPComboBox Control



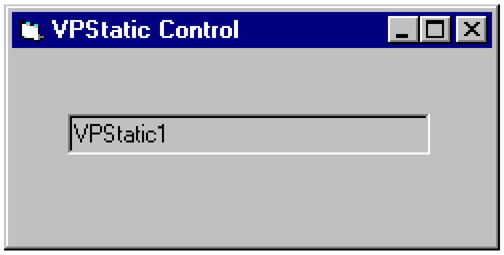
AutoSize Property Applies To

VPStatic Control
VPListBox Control



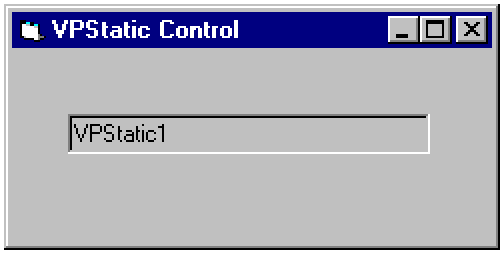
Caption Property Applies To

VPStatic Control



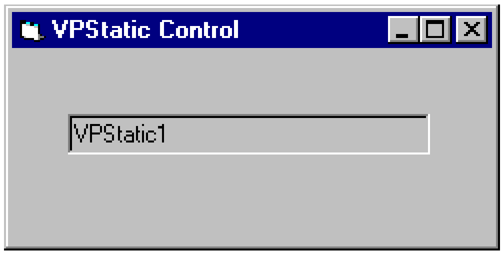
CaseSensitive Property Applies To

VPComboBox Control
VPListBox Control



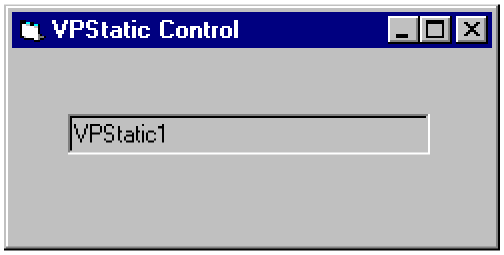
CellText Property Applies To

VPComboBox Control
VPListBox Control



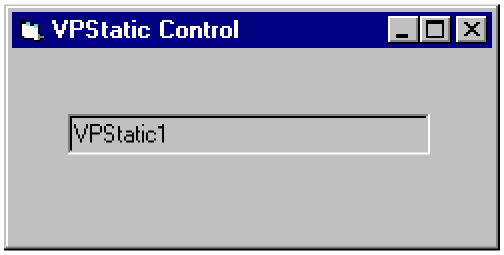
Background Property Applies To

VPForm Control



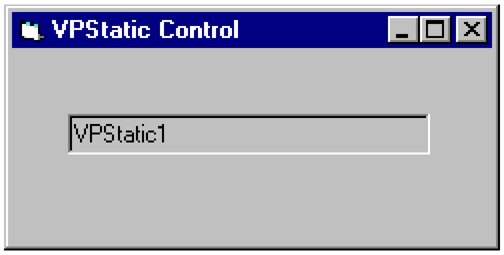
BorderStyle Property Applies To

VPTextBox Control
VPStatic Control
VPComboBox Control
VPListBox Control
VPForm Control



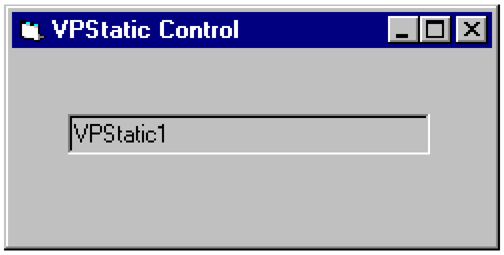
Col Property Applies To

VPComboBox Control
VPListBox Control



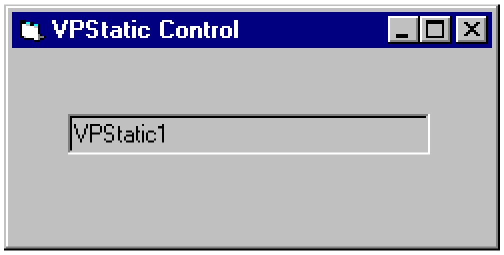
ColAlign Property Applies To

VPComboBox Control
VPListBox Control



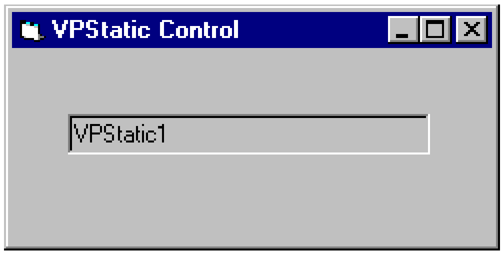
ColBound Property Applies To

VPComboBox Control
VPListBox Control



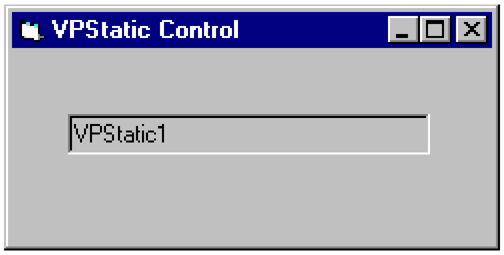
ColDataField Property Applies To

VPComboBox Control
VPListBox Control



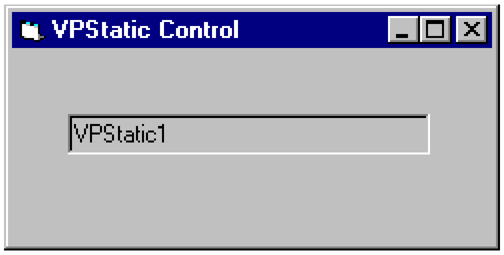
ColDataSource Property Applies To

VPComboBox Control
VPListBox Control



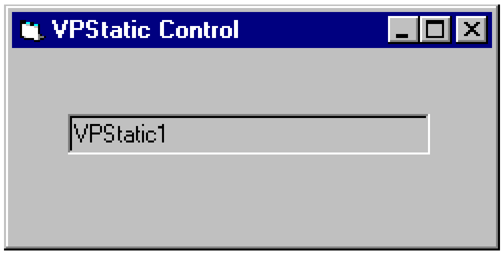
ColFormat Property Applies To

VPComboBox Control
VPListBox Control



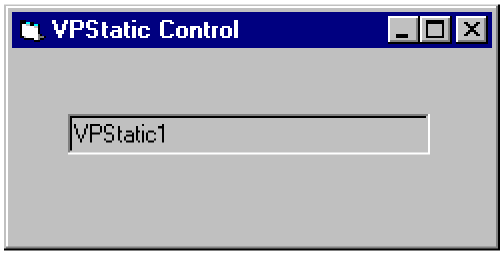
ColHeadAlign Property Applies To

VPComboBox Control
VPListBox Control



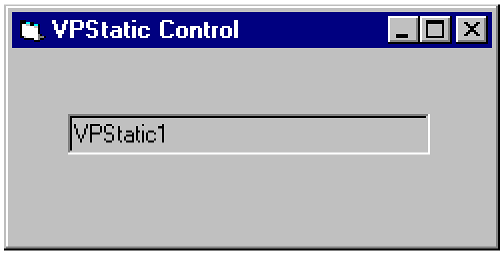
ColHeading Property Applies To

VPComboBox Control
VPListBox Control



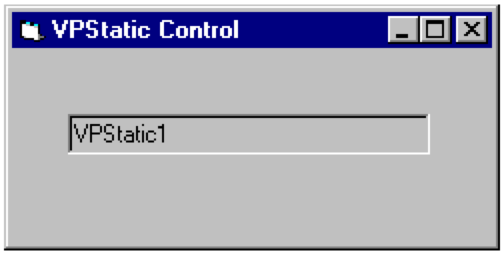
ColLink Property Applies To

VPComboBox Control



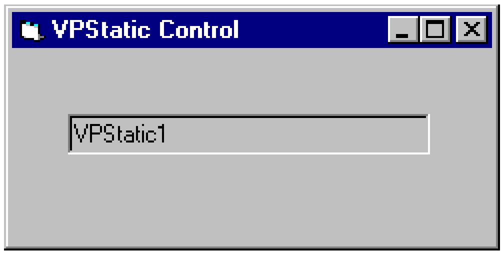
ColListField Property Applies To

VPComboBox Control
VPListBox Control



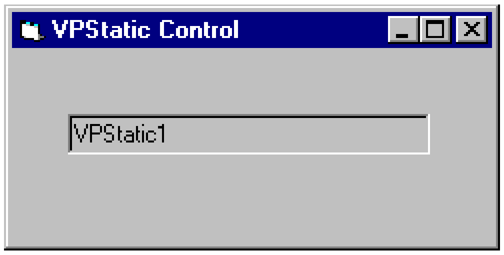
ColSortBy Property Applies To

VPComboBox Control
VPListBox Control



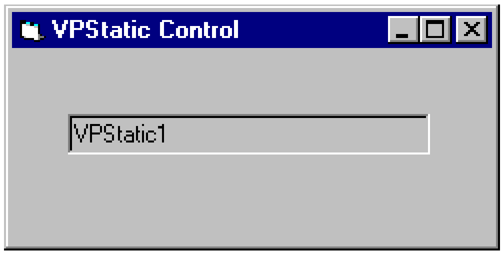
ColSortOrder Property Applies To

VPComboBox Control
VPListBox Control



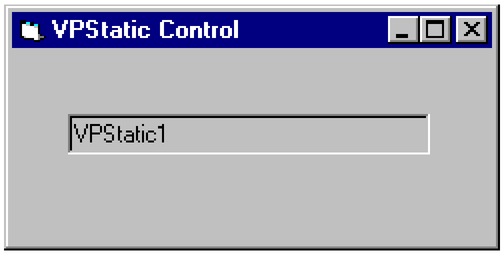
ColWidth Property Applies To

VPComboBox Control
VPListBox Control



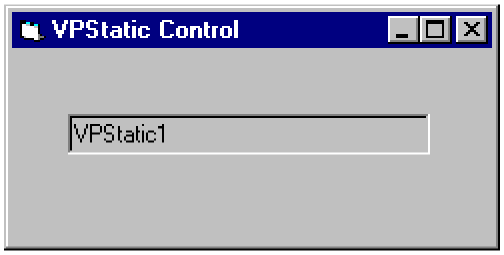
ControlIndex Property Applies To

VPForm Control
VPFocus Control



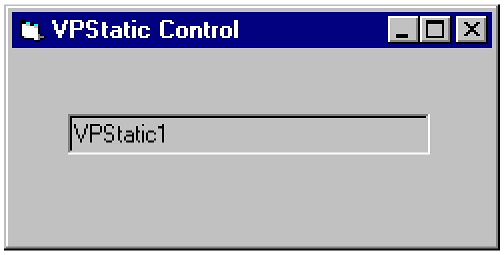
Cursor Property Applies To

VPForm Control



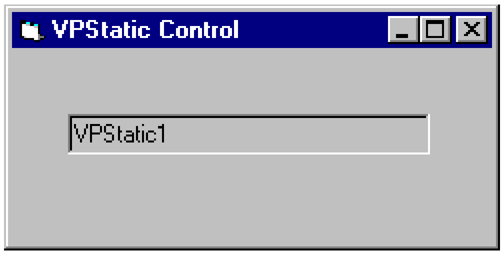
DataChanged Property Applies To

VPTextBox Control
VPStatic Control
VPComboBox Control
VPListBox Control



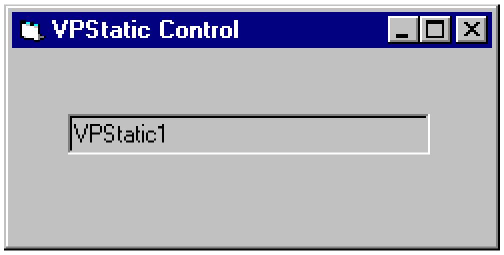
DataField Property Applies To

VPTextBox Control
VPStatic Control
VPComboBox Control



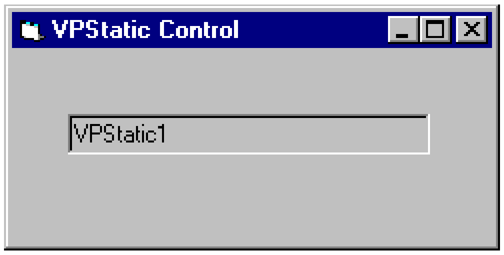
DataSource Property Applies To

VPTextBox Control
VPStatic Control
VPComboBox Control



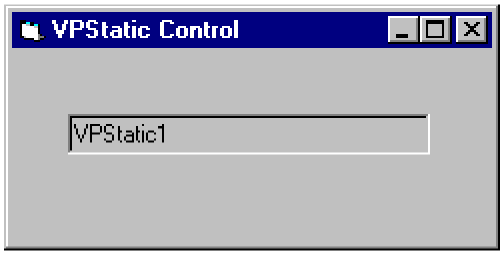
FocusAction Property Applies To

VPFocus Control



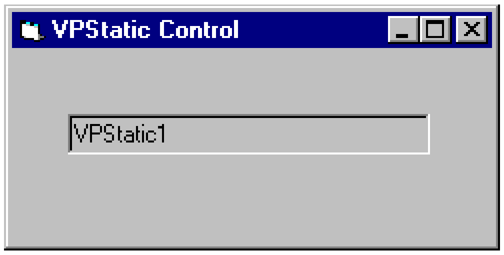
ForceGotFocus Property Applies To

VPFocus Control



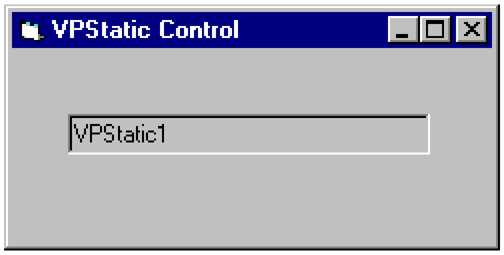
FormControlName Property Applies To

VPForm Control
VPFocus Control



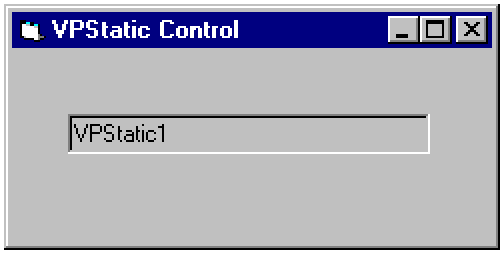
FormIndex Property Applies To

VPForm Control
VPFocus Control



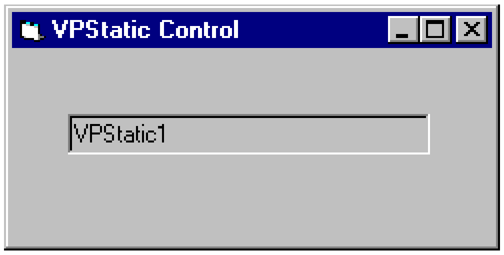
GridAppearance Property Applies To

VPComboBox Control
VPListBox Control



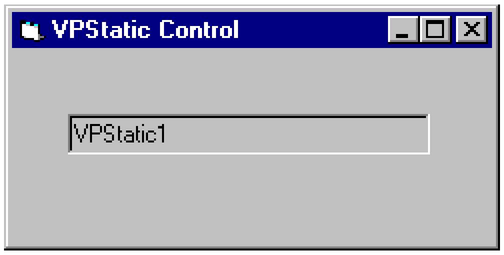
GridLines Property Applies To

VPComboBox Control
VPListBox Control



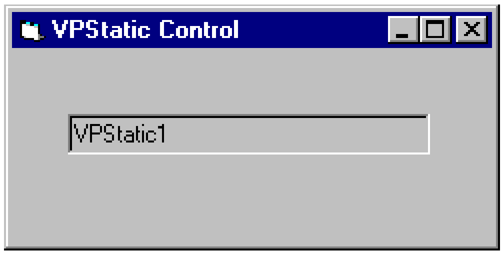
Heading Property Applies To

VPComboBox Control
VPListBox Control



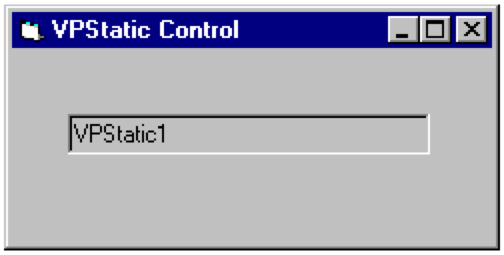
HeadingBackColor Property Applies To

VPComboBox Control
VPListBox Control



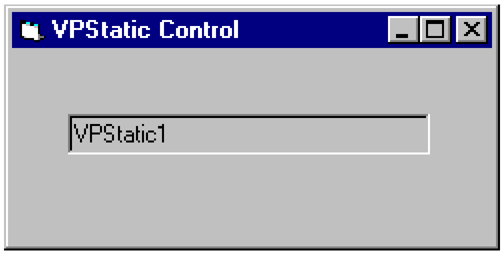
HeadingForeColor Property Applies To

VPComboBox Control
VPListBox Control



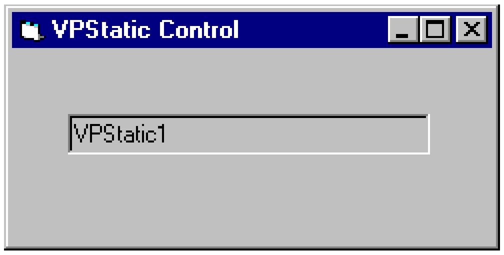
HideSelection Property Applies To

VPTextBox Control



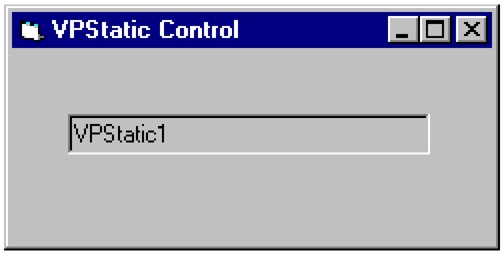
ListAppearance Property Applies To

VPComboBox Control



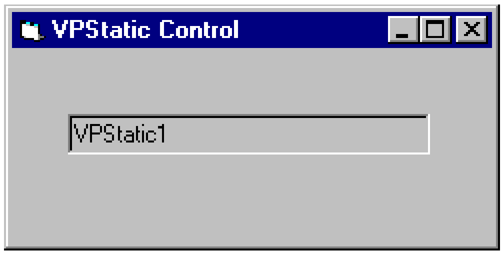
ListBackColor Property Applies To

VPComboBox Control



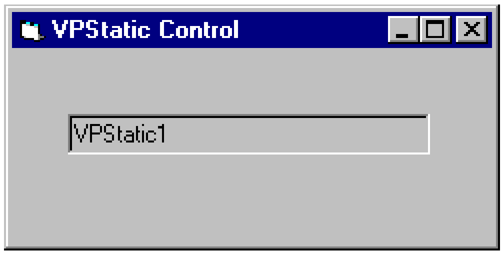
ListForeColor Property Applies To

VPComboBox Control



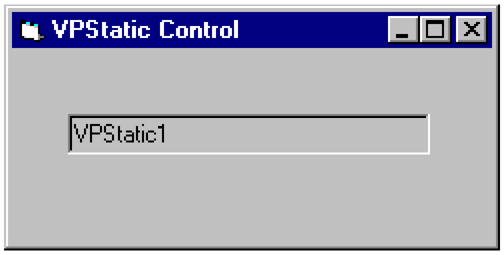
Locked Property Applies To

VPTextBox Control
VPComboBox Control



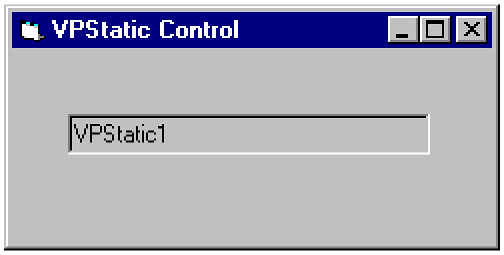
MatchEntry Property Applies To

VPComboBox Control



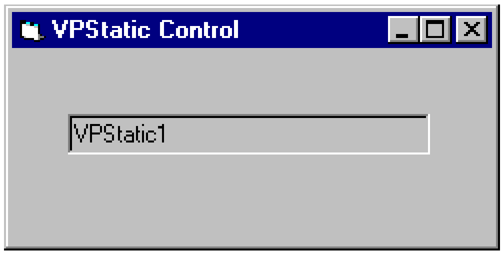
MaxCols Property Applies To

VPComboBox Control
VPListBox Control



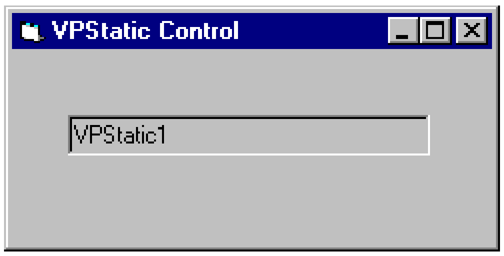
MaxDrop Property Applies To

VPComboBox Control



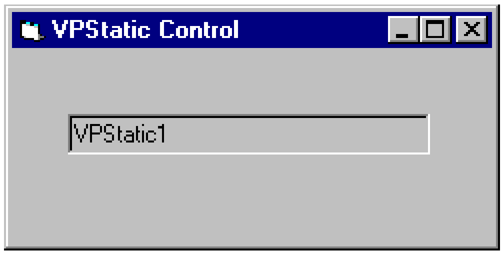
MaxLength Property Applies To

VPTextBox Control



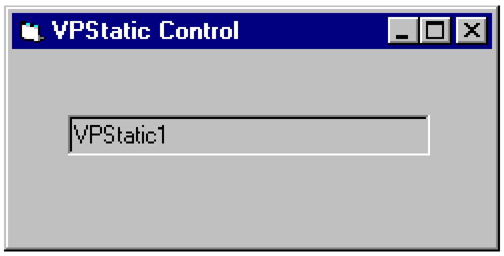
MaxWidth Property Applies To

VPComboBox Control



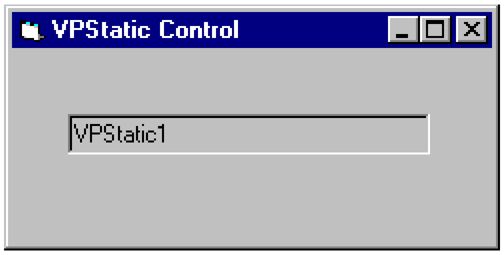
MouseCapture Property Applies To

VPForm Control



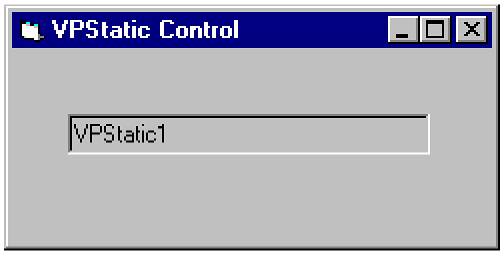
MultiLine Property Applies To

VPTextBox Control



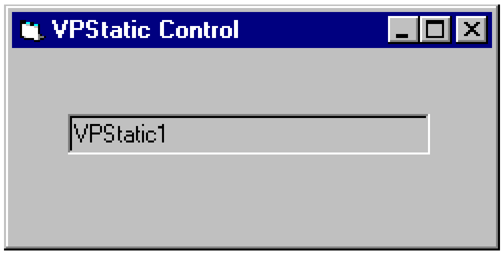
MultiSelect Property Applies To

VPListBox Control



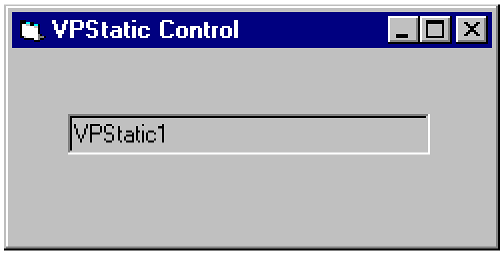
OverType Property Applies To

VPTextBox Control
VPComboBox Control



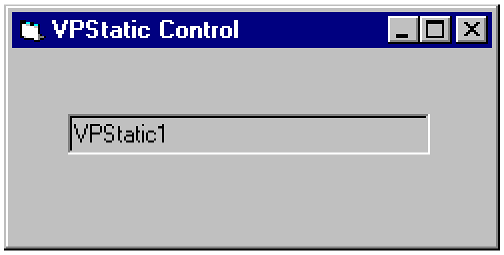
PasswordChar Property Applies To

VPTextBox Control



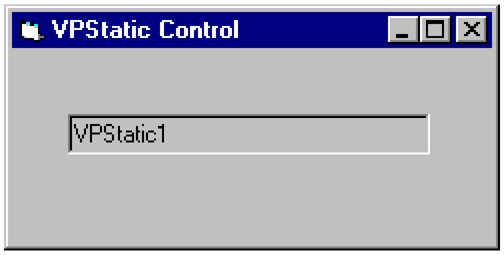
RowSource Property Applies To

VPComboBox Control
VPListBox Control



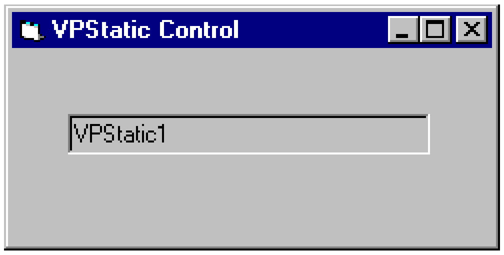
TargetControl Property Applies To

VPFocus Control



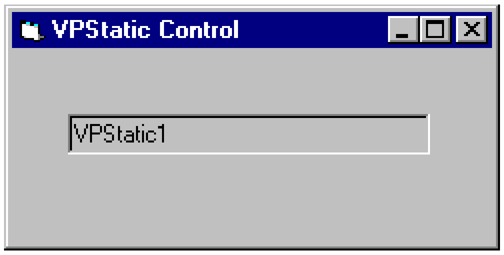
ScrollBars Property Applies To

VPTextBox Control



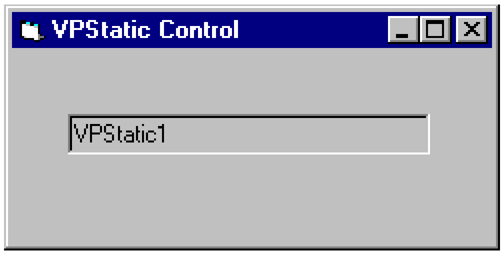
Sorted Property Applies To

VPComboBox Control
VPListBox Control



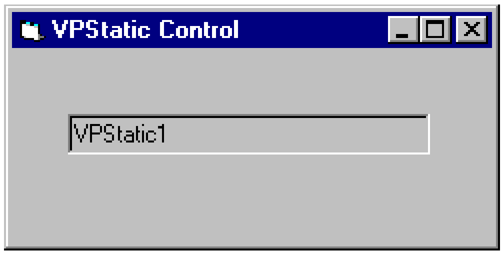
Style Property Applies To

VPComboBox Control



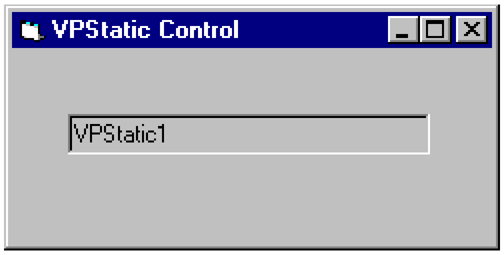
Text Property Applies To

VPTextBox Control
VPComboBox Control
VPListBox Control



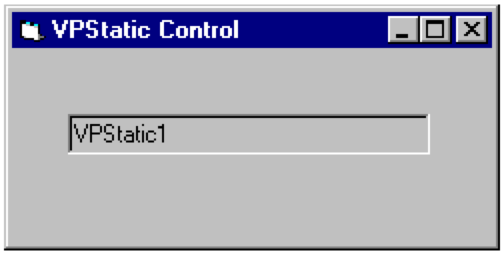
WordWrap Property Applies To

VPStatic Control



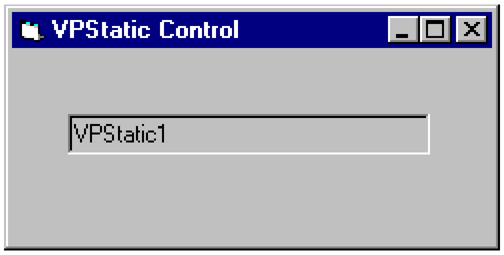
Change Event Applies To

VPTextBox Control
VPStatic Control
VPComboBox Control



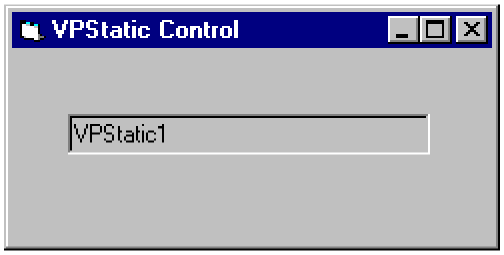
Click Event Applies To

VPTextBox Control
VPStatic Control
VPComboBox Control
VPListBox Control



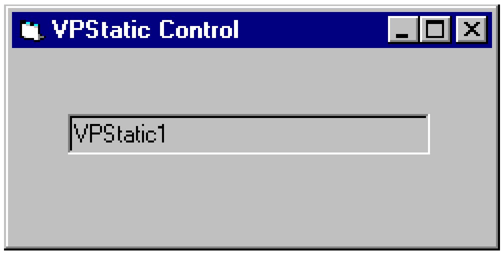
CloseUp Event Applies To

VPComboBox Control



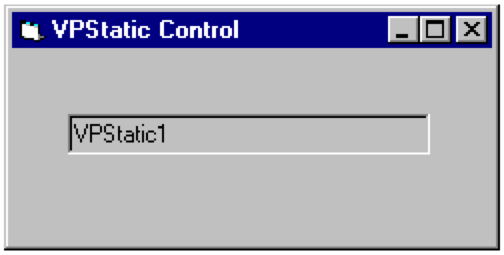
DblClick Event Applies To

VPTextBox Control
VPStatic Control
VPListBox Control



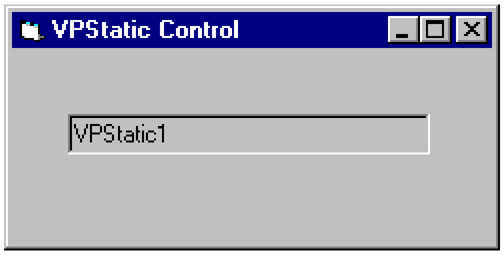
DropDown Event Applies To

VPComboBox Control



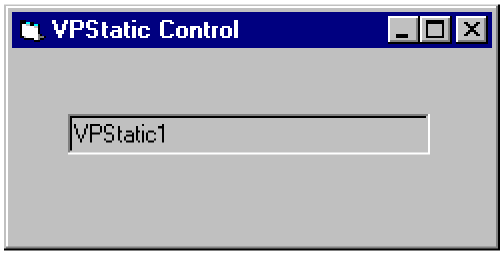
MouseDown Event Applies To

VPTextBox Control
VPStatic Control
VPComboBox Control
VPListBox Control
VPForm Control



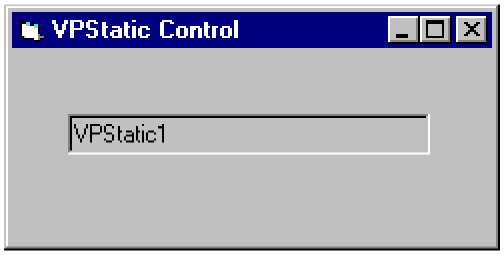
MouseMove Event Applies To

VPTextBox Control
VPStatic Control
VPComboBox Control
VPListBox Control
VPForm Control



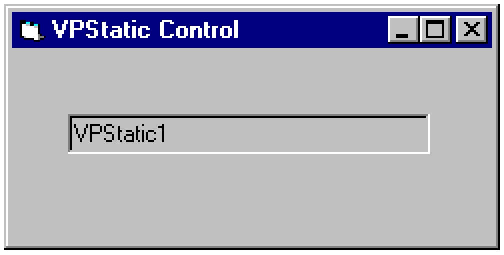
MouseUp Event Applies To

VPTextBox Control
VPStatic Control
VPComboBox Control
VPListBox Control
VPForm Control



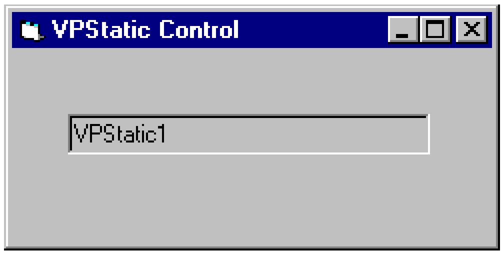
LocateText Method Applies To

VPComboBox Control
VPListBox Control



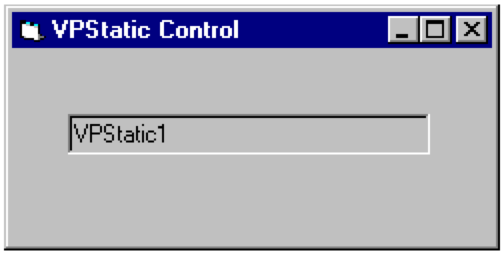
ReturnFocus Method Applies To

VPFocus Control



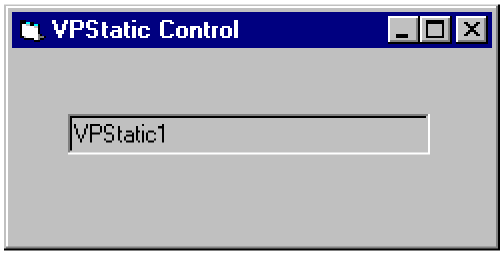
SendFocus Method Applies To

VPFocus Control



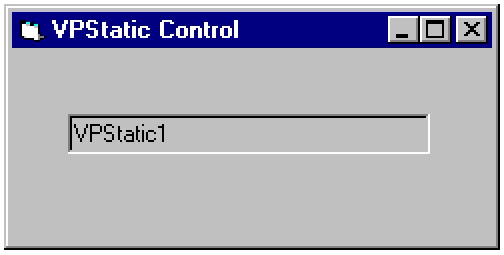
TrapFocusOff Method Applies To

VPFocus Control



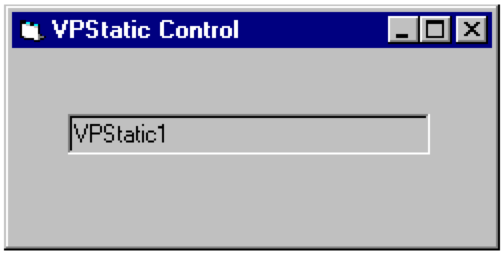
TrapFocusOn Method Applies To

VPFocus Control



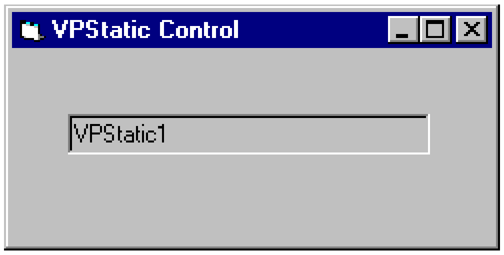
VLocateText Function Applies To

VPComboBox Control
VPListBox Control



VPTextBox Control Custom Properties

(About)
(Custom) (OCX only)
Alignment
Appearance
AutoSelect
Locked
OverType



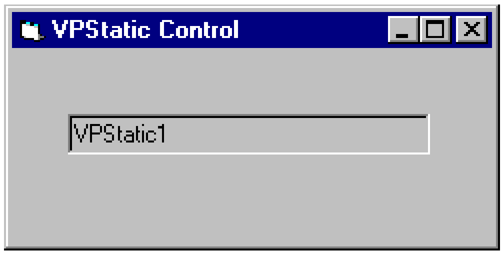
VPTextBox Control Standard Properties

For more information on these properties

BackColor
BorderStyle
Container (OCX only)
DataChanged
DataField
DataSource
DragIcon
DragMode
Enabled
Font (OCX only)
FontBold (VBX only)
FontItalic (VBX only)
FontName (VBX only)
FontSize (VBX only)
FontStrikethru (VBX only)
FontUnderline (VBX only)
ForeColor
Height
HelpContextID
HideSelection
hWnd
Index
Left
LinkItem (VBX only)
LinkMode (VBX only)
LinkTimeout (VBX only)
LinkTopic (VBX only)
MaxLength
MouseIcon (OCX only)
MousePointer
MultiLine
Name
Parent
PasswordChar
ScrollBars
SelLength
SelStart
SelText
TabIndex
TabStop
Tag

Text
Top
Visible
WhatsThisHelpID (OCX only)
Width

Return to Vantage Control Set Contents

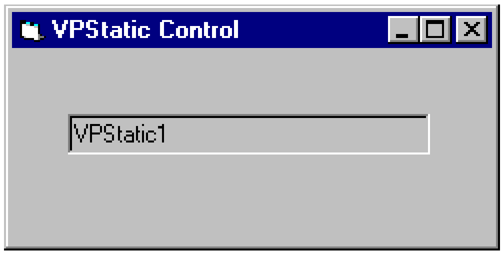


VPTextBox Control Standard Events

For more information on these events

Change
Click
DblClick
DragDrop
DragOver
GotFocus
KeyDown
KeyPress
KeyUp
LinkClose (VBX only)
LinkError (VBX only)
LinkNotify (VBX only)
LinkOpen (VBX only)
LostFocus
MouseDown
MouseMove
MouseUp

Return to Vantage Control Set Contents

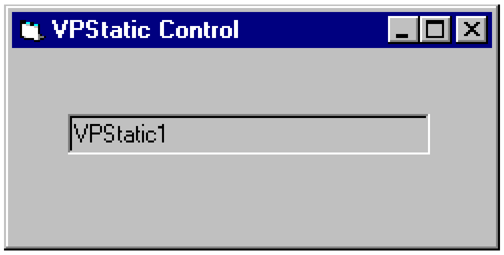


VPTextBox Control Standard Methods

For more information on these methods

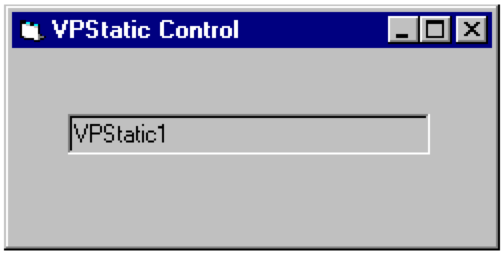
Drag
LinkExecute (VBX only)
LinkPoke (VBX only)
LinkRequest (VBX only)
Move
Refresh
SetFocus
ShowWhatsThis (OCX only)
Zorder

Return to Vantage Control Set Contents



VPStatic Control Custom Properties

(About)
(Custom) (OCX only)
Alignment
Appearance

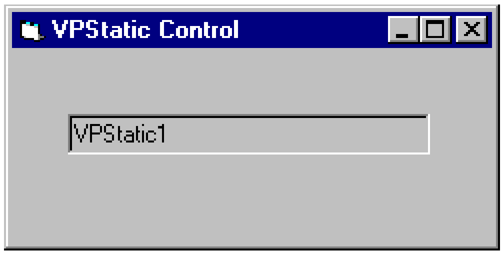


VPStatic Control Standard Properties

For more information on these properties

AutoSize
BackColor
BorderStyle
Caption
Container (OCX only)
DataChanged
DataField
DataSource
DragIcon
DragMode
Enabled
Font (OCX only)
FontBold (VBX only)
FontItalic (VBX only)
FontName (VBX only)
FontSize (VBX only)
FontStrikethru (VBX only)
FontUnderline (VBX only)
ForeColor
Height
hWnd
Index
Left
LinkItem (VBX only)
LinkMode (VBX only)
LinkTimeout (VBX only)
LinkTopic (VBX only)
MouseIcon (OCX only)
MousePointer
Name
Parent
TabIndex
Tag
Top
Visible
WhatsThisHelpID (OCX only)
Width
WordWrap

Return to Vantage Control Set Contents

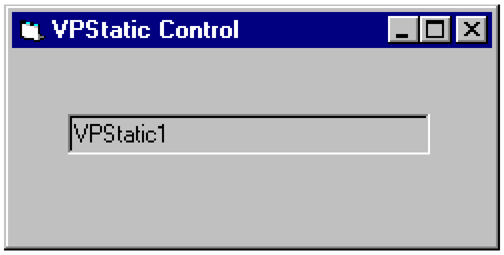


VPStatic Control Standard Events

For more information on these events

Change
Click
DblClick
DragDrop
DragOver
LinkClose (VBX only)
LinkError (VBX only)
LinkNotify (VBX only)
LinkOpen (VBX only)
MouseDown
MouseMove
MouseUp

Return to Vantage Control Set Contents

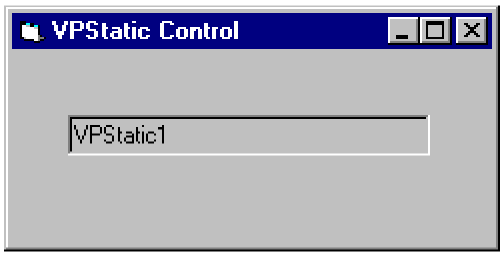


VPStatic Control Standard Methods

For more information on these methods

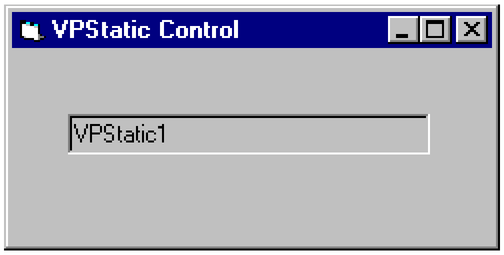
Drag
LinkExecute (VBX only)
LinkPoke (VBX only)
LinkRequest (VBX only)
Move
Refresh
ShowWhatsThis (OCX only)
Zorder

Return to Vantage Control Set Contents



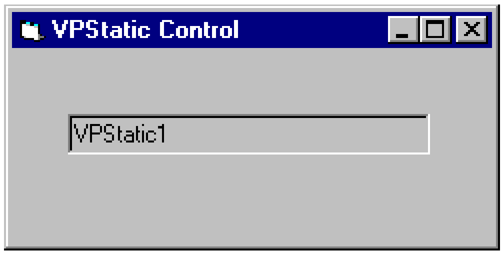
VPComboBox Control Custom Properties

(About)
(ColLayout) (VBX only)
(Custom) (OCX only)
Alignment
Appearance
AutoHeight
AutoSelect
BorderStyle
CaseSensitive
CellText
Col
ColAlign
ColBound
ColDataField
ColDataSource
ColFormat
ColHeadAlign
ColHeading
ColLink
ColListField
ColSortBy
ColSortOrder
ColWidth
DataChanged
DataField
DataSource
GridAppearance
GridLines
Heading
HeadingBackColor
HeadingForeColor
ListAppearance
ListBackColor
ListForeColor
Locked
MatchEntry
MaxCols
MaxDrop
MaxWidth
RowSource



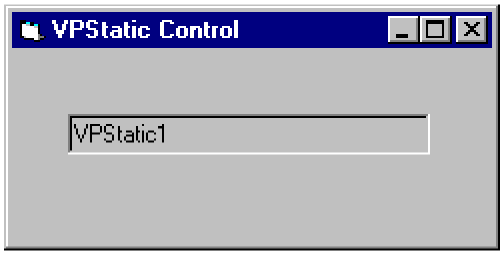
VPComboBox Control Custom Events

Change
CloseUp
MouseDown
MouseMove
MouseUp



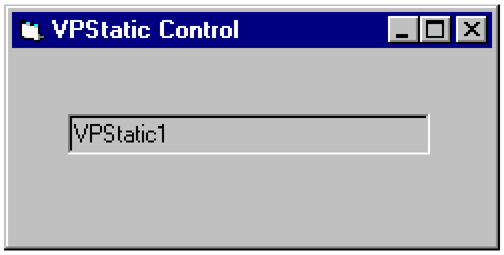
VPComboBox Control Custom Methods (OCX only)

LocateText



VPComboBox Control Custom Functions

VLocateText

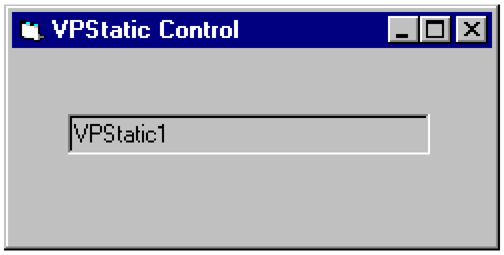


VPComboBox Control Standard Properties

For more information on these properties

BackColor
Container (OCX only)
DragIcon
DragMode
Enabled
Font (OCX only)
FontBold (VBX only)
FontItalic (VBX only)
FontName (VBX only)
FontSize (VBX only)
FontStrikethru (VBX only)
FontUnderline (VBX only)
ForeColor
Height
HelpContextID
hWnd
Index
ItemData
Left
List
ListCount
ListIndex
MouseIcon (OCX only)
MousePointer
Name
NewIndex
Parent
SelLength
SelStart
SelText
Sorted
Style
TabIndex
TabStop
Tag
Text
Top
TopIndex
Visible
WhatsThisHelpID (OCX only)
Width

Return to Vantage Control Set Contents

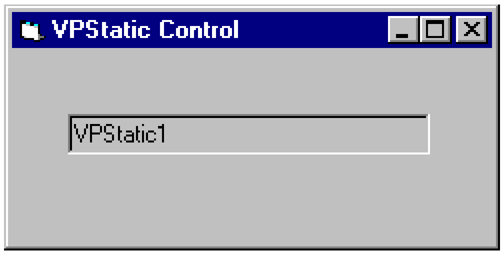


VPComboBox Control Standard Events

For more information on these events

Click
DblClick
DragDrop
DragOver
DropDown
GotFocus
KeyDown
KeyPress
KeyUp
LostFocus

Return to Vantage Control Set Contents

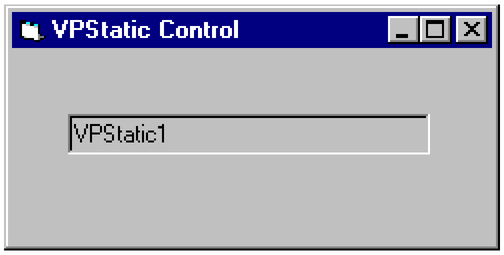


VPComboBox Control Standard Methods

For more information on these methods

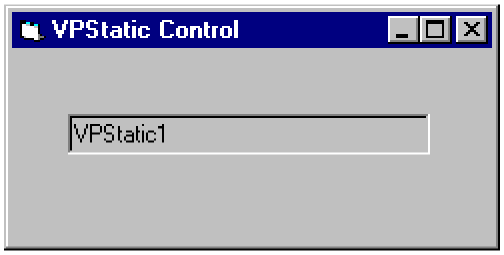
AddItem
Clear
Drag
Move
Refresh
RemoveItem
SetFocus
ShowWhatsThis (OCX only)
Zorder

Return to Vantage Control Set Contents



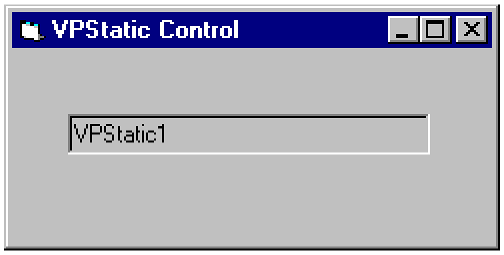
VPListBox Control Custom Properties

(About)
(ColLayout)
(Custom) (OCX only)
Appearance
AutoSize
BorderStyle
CellText
Col
ColAlign
ColBound
ColDataField
ColDataSource
ColFormat
ColHeadAlign
ColHeading
ColListField
ColSortBy
ColSortOrder
ColWidth
GridAppearance
GridLines
Heading
HeadingBackColor
HeadingForeColor
ListAppearance
MaxCols
RowSource



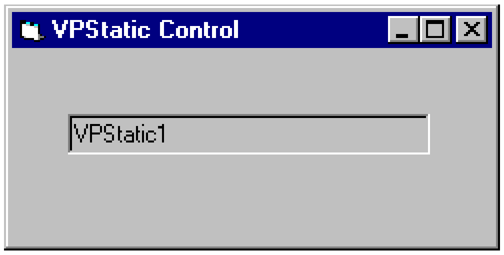
VPListBox Control Custom Methods (OCX only)

LocateText



VPListBox Control Custom Functions

VLocateText

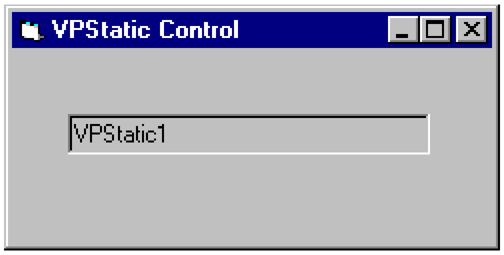


VPListBox Control Standard Properties

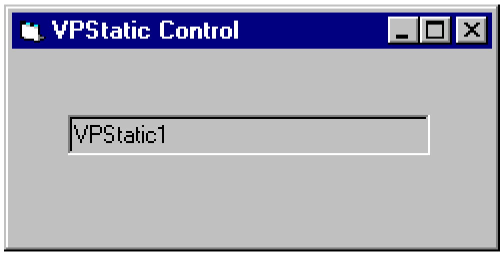
For more information on these properties

BackColor
Container (OCX only)
DragIcon
DragMode
Enabled
Font (OCX only)
FontBold (VBX only)
FontItalic (VBX only)
FontName (VBX only)
FontSize (VBX only)
FontStrikethru (VBX only)
FontUnderline (VBX only)
ForeColor
Height
HelpContextID
hWnd
Index
Left
List
ListCount
ListIndex
MouseIcon (OCX only)
MousePointer
MultiSelect
Name
NewIndex
Parent
Selected
Sorted
TabIndex
TabStop
Tag
Text
Top
TopIndex
Visible
WhatsThisHelpID (OCX only)
Width

Return to Vantage Control Set Contents



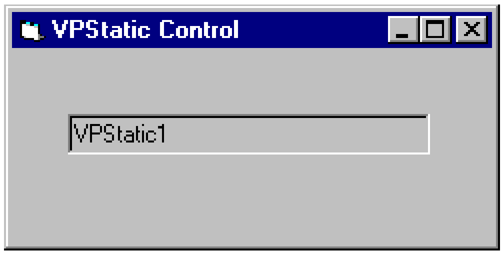
VPListBox Control Standard Events



For more information on these events

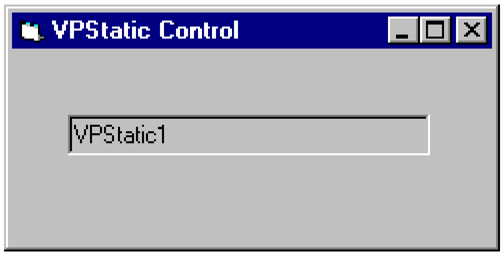
Click
DblClick
DragDrop
DragOver
GotFocus
KeyDown
KeyPress
KeyUp
MouseDown
MouseMove
MouseUp

Return to Vantage Control Set Contents


VPStatic Control

VPStatic1

VPListBox Control Standard Methods

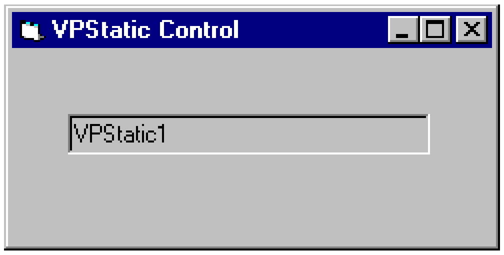

VPStatic Control

VPStatic1

For more information on these methods

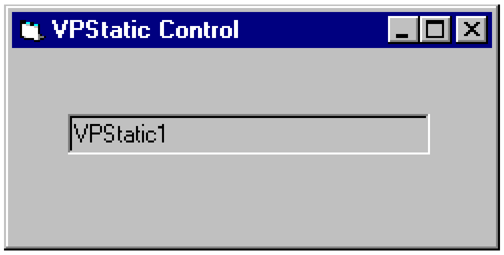
AddItem
Clear
Drag
Move
Refresh
RemoveItem
SetFocus
ShowWhatsThis (OCX only)
Zorder

Return to Vantage Control Set Contents



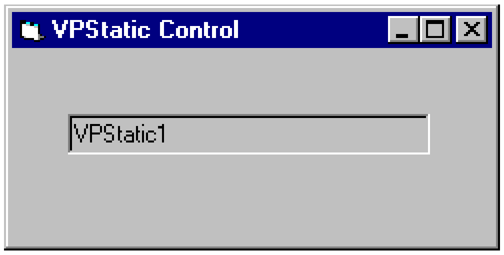
VPForm Control Custom Properties

(About)
(Custom) (OCX only)
Appearance
Background
BorderStyle (OCX only)
ControlIndex
Cursor
FormControlName
FormIndex
MouseCapture

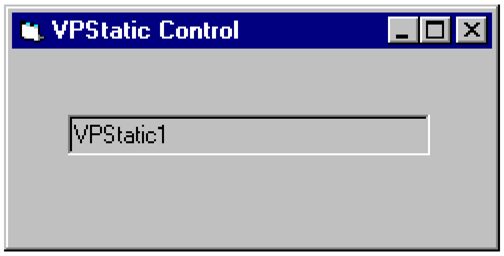


VPForm Control Custom Events

MouseDown
MouseMove
MouseUp



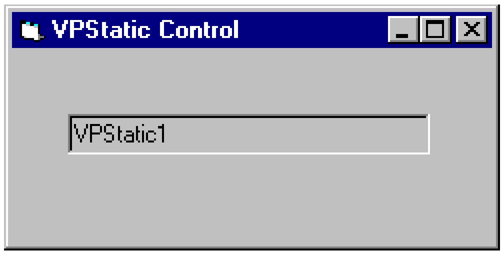
VPForm Control Standard Properties



For more information on these properties

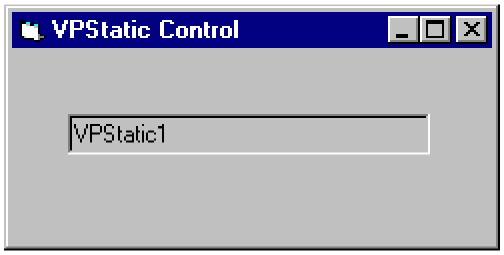
Align
DragIcon (OCX only)
DragMode (OCX only)
Height
HelpContextID (OCX only)
Index (OCX only)
Left
Name
Negotiate (OCX only)
TabIndex (OCX only)
TabStop (OCX only)
Tag (OCX only)
Top
Visible (OCX only)
WhatsThisHelpID (OCX only)
Width

Return to Vantage Control Set Contents



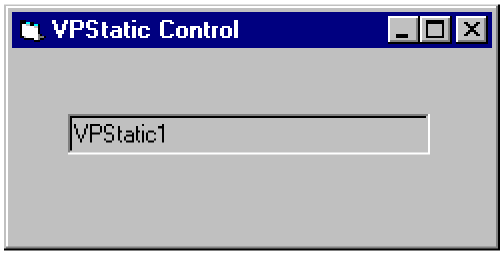
VPFocus Control Custom Properties

(About)
(Custom) (OCX only)
ActiveControl
ControlIndex
FocusAction
ForceGotFocus **(OCX only)**
FormControlName
FormIndex
TargetControl

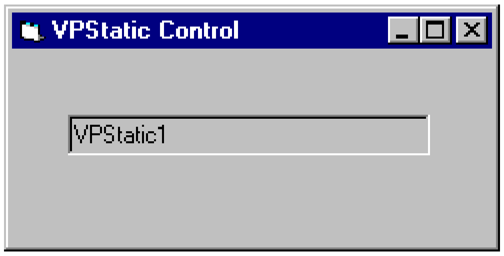


VPFocus Control Custom Methods (OCX only)

ReturnFocus
SendFocus
TrapFocusOff
TrapFocusOn



VPFocus Control Standard Properties



For more information on these properties

Align (OCX only)
DragIcon (OCX only)
DragMode (OCX only)
Height
HelpContextID (OCX only)
Index (OCX only)
Left
Name
Negotiate (OCX only)
TabIndex (OCX only)
TabStop (OCX only)
Tag (OCX only)
Top
Visible (OCX only)
WhatsThisHelpID (OCX only)
Width

Return to Vantage Control Set Contents

See Also

Column Layout Properties Dialog
ColAlign Property
ColBound Property
ColFormat Property
ColHeadAlign Property
ColHeading Property
ColLink Property
ColListField Property
ColSortBy Property
ColSortOrder Property
ColWidth Property
VPComboBox Control
VPListBox Control

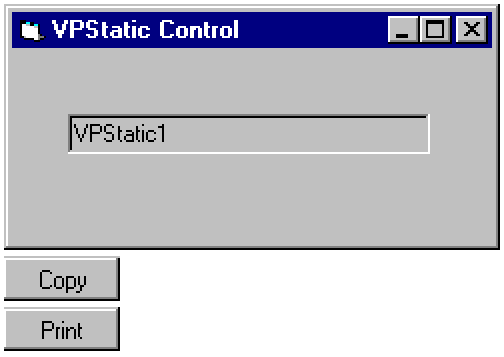
See Also

Property Pages

See Also

FocusAction Property
TargetControl Property
ReturnFocus Method
SendFocus Method
TrapFocusOff Method
TrapFocusOn Method
VPFocus Control

ActiveControl Property Example



The **ActiveControl** property is part of the **VPFocus** control that is used to manage **GotFocus**/**LostFocus** event processing. This property is usually used in the **GotFocus** event of a control.

Rather than placing code in each **GotFocus** event procedure, one generalized procedure can be created and called from each event procedure. In this example, we will name the generalized procedure *GotFocusDefProc*. The **ActiveControl** property is used in two situations;   One, to check if processing for a **GotFocus** event is valid, and two, to assign the "current" control as the "active" control when you have a valid event.   This is done by setting this **ActiveControl** property to the window handle of the current control.

In the *GotFocusDefProc* procedure we first check if the **GotFocus** event is valid. A **GotFocus** event will be valid if no control is currently "active". We can test this by seeing if the **ActiveControl** property is set to zero (0).

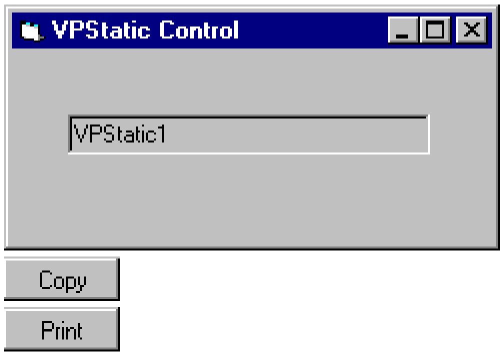
If we have a valid **GotFocus** event we record the current control as the "active" control by setting the **ActiveControl** property to the control's Window handle. This will start the **VPFocus** control trapping for **Focus** events.

```
Sub GotFocusDefProc (Frm As Form, ctl As Control)
    'Check if GotFocus event is valid
    If Frm.VPFocus1.ActiveControl = 0 Then
        'Set Active control and start trapping for focus events.
        Frm.VPFocus1.ActiveControl = ctl.hWnd
        'If Text Box control then change Colors to help show where focus is
        If TypeOf ctl Is VPTextBox Then
            glBackColor = ctl.BackColor
            glForeColor = ctl.ForeColor
            ctl.BackColor = glHLBackColor
            ctl.ForeColor = glHLForeColor
        End If
    End If
End Sub
```

See Also

BorderStyle Property

Background Property Example



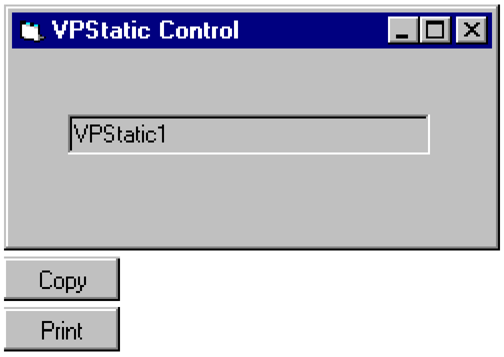
This example loads a bitmap from the Vantage Control Set bitmap library into the **Background** property of a **VPForm** control.

```
Private Sub Form_Load ()
        'Load the bitmap
        VPForm1.Background = LoadPicture("C:\VCS\BITMAPS\VFBGRD01.BMP")
End Sub
```

See Also

Appearance Property

CellText Property Example



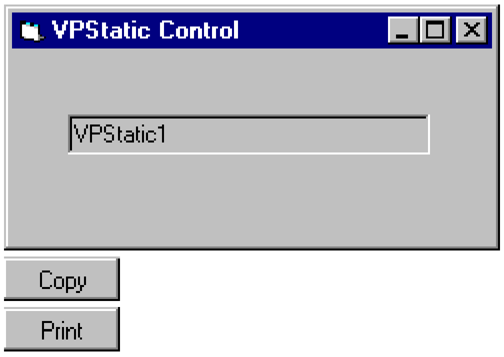
This example uses the **CellText** property to retrieve the contents of the currently selected row and a given column and assign the text value to a local string variable.

```
Sub GetData ()
    Dim sColData As String
    VPListBox1.Col = 3
    sColData = VPListBox1.CellText(VPListBox1.ListIndex)
End Sub
```

See Also

Col Property
VPComboBox Control
VPListBox Control

Col Property Example



In this example we will change the alignment, format, and width of the second and third columns in a **VPComboBox** ComboBox control. In this example the second column will be used to display a name or description field and the third column will be used to display a dollar amount field.

```
VPComboBox1.Col = 2
VPComboBox1.ColAlign = 0 'Left justification
VPComboBox1.ColFormat = "" 'No formatting
VPComboBox1.ColWidth = 3200 'Width in twips
VPComboBox1.Col = 3
VPComboBox1.ColAlign = 1 'Right justification
VPComboBox1.ColFormat = "$#,##0.00;($#,##0.00)" 'Currency format
VPComboBox1.ColWidth = 1600 'Width in twips
```

See Also

CellText Property
ColAlign Property
ColBound Property
ColDataField Property
ColDataSource Property
ColFormat Property
ColHeadAlign Property
ColHeading Property
ColLink Property
ColListField Property
ColSortBy Property
ColSortOrder Property
ColWidth Property
MaxCols Property
VPComboBox Control
VPListBox Control

See Also

Col Property
ColFormat Property
ColHeadAlign Property
ColHeading Property
MaxCols Property
VPComboBox Control
VPListBox Control

See Also

Col Property
ColDataField Property
ColDataSource Property
ColLink Property
ColListField Property
DataField Property
DataSource Property
RowSource Property
Data Control
VPComboBox Control
VPListBox Control

See Also

Col Property
ColBound Property
ColDataSource Property
ColLink Property
ColListField Property
DataField Property
DataSource Property
RowSource Property
Data Control
VPComboBox Control
VPListBox Control

See Also

Col Property
ColBound Property
ColDataField Property
ColLink Property
ColListField Property
DataField Property
DataSource Property
RowSource Property
Data Control
VPComboBox Control
VPListBox Control

See Also

Col Property
ColAlign Property
ColWidth Property
MaxCols Property
VPComboBox Control
VPListBox Control

See Also

Col Property
ColAlign Property
ColHeading Property
ColWidth Property
Heading Property
MaxCols Property
VPComboBox Control
VPListBox Control

See Also

Col Property
ColAlign Property
ColHeadAlign Property
ColWidth Property
Heading Property
MaxCols PropertyVCS_MaxCols_Property>main
VPComboBox Control
VPListBox Control

See Also

Col Property
ColBound Property
ColDataField Property
ColDataSource Property
ColListField Property
DataField Property
DataSource Property
MaxCols Property
RowSource Property
Data Control
VPComboBox Control
VPListBox Control

See Also

Col Property
ColBound Property
ColDataField Property
ColDataSource Property
ColLink Property
DataField Property
DataSource Property
MaxCols Property
RowSource Property
Data Control
VPComboBox Control
VPListBox Control

See Also

CaseSensitive Property
Col Property
ColSortOrder Property
MaxCols Property
Sorted Property
VPComboBox Control
VPListBox Control

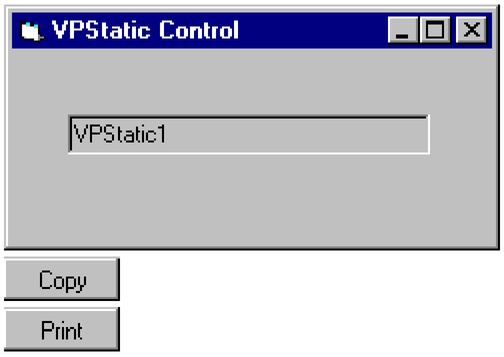
See Also

CaseSensitive Property
Col Property
ColSortBy Property
MaxCols Property
Sorted Property
VPComboBox Control
VPListBox Control

See Also

Col Property
ColAlign Property
ColFormat Property
MaxCols Property
MaxWidth Property
VPComboBox Control
VPListBox Control

ControlIndex Property Example



When used with the **VPForm** control the **ControlIndex** property is used to identify the control where the mouse pointer is currently over. This property is useful in both the **MouseMove** and **MouseUp** events of the **VPForm** control.

In the **MouseMove** event the **ControlIndex** is tested to see if it equals a -1 value which would indicate the mouse pointer is not over a control but is either over a **Form** or outside the application. In this situation the **Cursor** property of the **VPForm** control can be set to the "No Drop" cursor, which is resource 6001 in the VFORM.VBX file. If the value of the **ControlIndex** property is not -1 then this property can be used together with the **FormIndex** property to identify the control the mouse pointer is currently over. The local object variable **Cntrl** can be set to the identified control in the **Controls Collection**. This object variable can further be tested to see if the control, the mouse pointer is over, is the drop target, VPTextBox2 in this example, or some other control. If the mouse pointer is over some other control then set the **Cursor** property to the same "No Drop" cursor. If the mouse pointer is over the target VPTextBox2 control then set the **Cursor** property to a valid drag cursor, such as the cursor resource 6003 in the VFORM.VBX file.

```
Sub VPForm1_MouseMove (Button As Integer, Shift As Integer, X As Single, Y As Single)
    Dim Cntrl As Control
    If VPForm1.ControlIndex = -1 Then
        VPForm1.Cursor = "6001"
    Else
        Set Cntrl = Forms(VPForm1.FormIndex).Controls(VPForm1.ControlIndex)
        If Cntrl Is VPTextBox2 Then
            VPForm1.Cursor = "6003"
        Else
            VPForm1.Cursor = "6001"
        End If
    End If
End Sub
```

In the **MouseUp** event the **ControlIndex** can again be used to see if the drop occurred on a control or some other object. If **ControlIndex** does not have a -1 value, the local object variable **Cntrl** can be set to the currently pointed to control where the mouse button was released. If this object variable is the same as our target VPTextBox2 control, the results of the drag operation can be executed. In this example, the drag-and-drop operation transfers the contents of the source **VPTextBox1** TextBox control to the targeted **VPTextBox2** TextBox control. Note: at the time of the original mouse down event the module level object variable, **Source**, was set to the **VPTextBox1** control.

```
Sub VPForm1_MouseUp (Button As Integer, Shift As Integer, X As Single, Y As Single)
    Dim Cntrl As Control
    VPForm1.Cursor = ""
    VPForm1.MouseCapture = False
    If VPForm1.ControlIndex > -1 Then
```

```
        Set Cntrl = Forms(VPForm1.FormIndex).Controls(VPForm1.ControlIndex)
        If Cntrl Is VPTextBox2 Then
            VPTextBox2.Text = Source.Text
        End If
    End If
End Sub
```

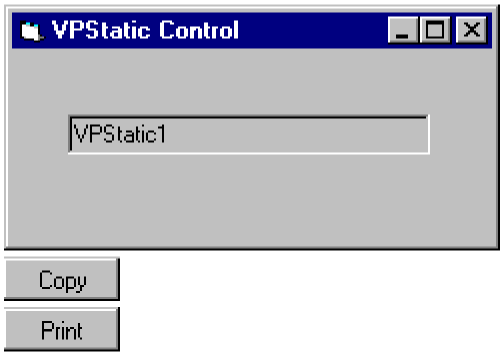
When used with the **VPFocus** control, the **ControlIndex** property is used to identify the control where **Focus** should go to next. If **Focus** is to go to an object that executes an event code that cancels the current process or exits the application, it may not make sense to continue the normal processing of a **LostFocus** event. In this example we use the **ControlIndex** property along with the **FormIndex** property to identify the "next" control and to test if **Focus** is next being passed to the "Cancel" command button.

```
Sub LostFocusDefProc (Frm As Form, ctl As Control, rsDataVal)
    'Check if LostFocus event is valid
    If Frm.VPFocus1.ActiveControl = ctl.hWnd Then
        'Check if next target control is cancel button
        If Forms(Frm.VPFocus1.FormIndex).controls(Frm.VPFocus1.ControlIndex) Is Frm.cmdCancel
Then
            'Send focus on to Trapped control
            Frm.VPFocus1.FocusAction = vxSendFocus '1
        Else
            Select Case UCase$(rsDataVal)
                Case "BAD"
                    Beep
                    MsgBox "Invalid Data - Please enter again."
                    'Reset data
                    ctl.Text = ""
                    'Invalid data so send focus back to itself
                    Frm.VPFocus1.FocusAction = vxReturnFocus '2
                    Exit Sub
                Case "MSG"
                    'Having a MsgBox normally eats up any GotFocus and causes real problems
                    Beep
                    MsgBox "This is a valid entry - Data accepted."
                    'Send focus on to Trapped control
                    Frm.VPFocus1.ForceGotFocus = True
                    Frm.VPFocus1.FocusAction = vxSendFocus '1
                Case Else 'GOOD or any other input
                    'Send focus on to Trapped control
                    Frm.VPFocus1.FocusAction = vxSendFocus '1
            End Select
        End If
        'Turn off focus highlight color
        If TypeOf ctl Is VPTextBox Then
            ctl.BackColor = glBackColor
            ctl.ForeColor = glForeColor
        End If
    End If
End Sub
```

See Also

FocusAction Property
FormIndex Property
FormControlName Property
ReturnFocus Method
SendFocus Method
TrapFocusOff Method
TrapFocusOn Method
VPForm Control
VPFocus Control

Cursor Property Example



The **Cursor** property is used change the mouse pointer to a custom cursor during drag-and-drop operations. The appropriate place to make these changes is in the **MouseMove** and **MouseUp** events of the **VPForm** control, once the **MouseCapture** property has been set to **True**.

In the **MouseMove** event we set the **Cursor** property to either the "No Drop" cursor resource (id 6001), to a valid drag cursor (id 6003), or a valid drop cursor (id 6008). If the mouse pointer is over some control other than the target control, **VPTextBox2**, we supply the "No Drop" cursor resource from the VFORM.VBX file. If the mouse pointer is over our source control or the form, an appropriate drag cursor is used. If the mouse pointer is over our target control, an appropriate drop cursor can be used. In this example we assign the resource id of 6003 from the VFORM.VBX file as our drag cursor and the resource id of 6008 as our drop cursor.

```
Sub VPForm1_MouseMove (Button As Integer, Shift As Integer, X As Single, Y As Single)
    Dim Cntrl As Control
    If VPForm1.FormIndex = -1 Then
        'Mouse outside any application form - use No Drop cursor
        VPForm1.Cursor = "6001"
    Else
        'Mouse within an application form
        If VPForm1.ControlIndex = -1 Then
            'Mouse not on a control - on form itself - use valid drag cursor
            VPForm1.Cursor = "6003"
        Else
            'Identify control mouse is currently over
            Set Cntrl = Forms(VPForm1.FormIndex).Controls(VPForm1.ControlIndex)
            If Cntrl Is VPTextBox1 Then
                'Mouse is over source control - use valid drag cursor
                VPForm1.Cursor = "6003"
            Else
                If Cntrl Is VPTextBox2 Then
                    'Mouse is over target control - use valid drop cursor
                    VPForm1.Cursor = "6008"
                Else
                    'Mouse is on some other control - use No Drop cursor
                    VPForm1.Cursor = "6001"
                End If
            End If
        End If
    End If
End Sub
```

In the **MouseUp** event we turn off any custom cursor being used and return the mouse pointer to the previous cursor assigned by Visual Basic. This is done by assigning the **Cursor** property to a null string.

```
Sub VPForm1_MouseUp (Button As Integer, Shift As Integer, X As Single, Y As Single)
    Dim Cntrl As Control
    VPForm1.MouseCapture = False
    VPForm1.Cursor = ""
    If VPForm1.ControlIndex > -1 Then
        Set Cntrl = Forms(VPForm1.FormIndex).Controls(VPForm1.ControlIndex)
        If Cntrl Is VPTextBox2 Then
            VPTextBox2.Text = Source.Text
        End If
    End If
End Sub
```

See Also

MouseCapture Property
MouseDown Event
MouseMove Event
MouseUp Event
VPForm Control

See Also

ColBound Property
ColDataField Property
ColDataSource Property
ColLink Property
DataField Property
DataSource Property
Data Control
VPTextBox Control
VPStatic Control
VPComboBox Control
VPListBox Control

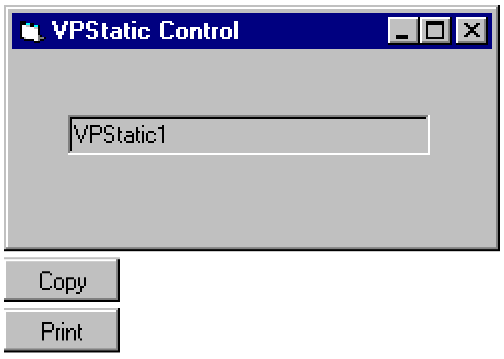
See Also

ColBound Property
ColDataField Property
ColDataSource Property
ColLink Property
DataChanged Property
DataSource Property
Data Control
VPTextBox Control
VPStatic Control
VPComboBox Control
VPListBox Control

See Also

ColBound Property
ColDataField Property
ColDataSource Property
ColLink Property
DataChanged Property
DataField Property
Data Control
VPTextBox Control
VPStatic Control
VPComboBox Control
VPListBox Control

FocusAction Property Example



The **FocusAction** property is part of the **VPFocus** control that is used to manage **GotFocus**/**LostFocus** event processing. This property is used in conjunction with the **ActiveControl** property and is usually used in the **LostFocus** event of a control.

Rather than placing code in each **GotFocus** or **LostFocus** event procedure, one generalized procedure can be created and called from each event procedure type. In this example, we will define two generalized procedures, one named *GotFocusDefProc* and the other *LostFocusDefProc*. In the first generalized procedure the **ActiveControl** property is used in two situations;   One, to check if processing for a **GotFocus** event is valid, and two, to assign the "current" control as the "active" control when you have a valid event.   This is done by setting this **ActiveControl** property to the window handle of the current control.

In the *GotFocusDefProc* procedure we first check if the **GotFocus** event is valid. A **GotFocus** event will be valid if no control is currently "active". We can test this by seeing if the **ActiveControl** property is set to zero (0).

If we have a valid **GotFocus** event we record the current control as the "active" control by setting the **ActiveControl** property to the control's Window handle. This will start the **VPFocus** control trapping for **Focus** events.

```
Sub GotFocusDefProc (Frm As Form, ctl As Control)
    'Check if GotFocus event is valid
    If Frm.VPFocus1.ActiveControl = 0 Then
        'Set Active control and start trapping for focus events.
        Frm.VPFocus1.ActiveControl = ctl.hWnd
        'If Text Box control then change Colors to help show where focus is
        If TypeOf ctl Is VPTextBox Then
            glBackColor = ctl.BackColor
            glForeColor = ctl.ForeColor
            ctl.BackColor = glHLBackColor
            ctl.ForeColor = glHLForeColor
        End If
    End If
End Sub
```

In the *LostFocusDefProc* procedure we first check if the **LostFocus** event is valid. A **LostFocus** event will be valid if the object reference in the **ActiveControl** property is equal to the current control. We can test this by seeing if the **ActiveControl** property is equal to window handle of the current control.

If we have a valid **LostFocus** event we next check if the control where **Focus** will next be sent is a special "Cancel" command button control. In this case, any processing of our **LostFocus** event should be by-passed. If **Focus** is going to any other control we check the validity of the data entered into the current control. If it fails any validation checks the

FocusAction property is set to 2, returning focus back to the current control. If the data is acceptable then the **FocusAction** property is set to 1, sending **Focus** on to the next control.

```
Sub LostFocusDefProc (Frm As Form, ctl As Control, rsDataVal)
    'Check if LostFocus event is valid
    If Frm.VPFocus1.ActiveControl = ctl.hWnd Then
        'Check if next target control is cancel button
        If Forms(Frm.VPFocus1.FormIndex).controls(Frm.VPFocus1.ControlIndex) Is Frm.cmdCancel
Then
            'Send focus on to Trapped control
            Frm.VPFocus1.FocusAction = vxSendFocus '1
        Else
            Select Case UCase$(rsDataVal)
                Case "BAD"
                    Beep
                    MsgBox "Invalid Data - Please enter again."
                    'Reset data
                    ctl.Text = ""
                    'Invalid data so send focus back to itself
                    Frm.VPFocus1.FocusAction = vxReturnFocus '2
                    Exit Sub
                Case "MSG"
                    'Having a MsgBox normally eats up any GotFocus and causes real problems
                    Beep
                    MsgBox "This is a valid entry - Data accepted."
                    'Send focus on to Trapped control
                    Frm.VPFocus1.ForceGotFocus = True
                    Frm.VPFocus1.FocusAction = vxSendFocus '1
                Case Else 'GOOD or any other input
                    'Send focus on to Trapped control
                    Frm.VPFocus1.FocusAction = vxSendFocus '1
            End Select
        End If
        'Turn off focus highlight color
        If TypeOf ctl Is VPTextBox Then
            ctl.BackColor = glBackColor
            ctl.ForeColor = glForeColor
        End If
    End If
End Sub
```

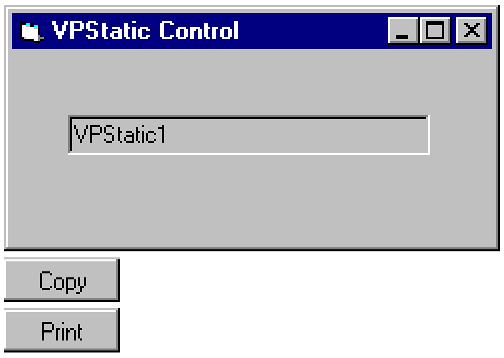
See Also

ActiveControl Property
ForceGotFocus Property
ReturnFocus Method
SendFocus Method
TrapFocusOff Method
TrapFocusOn Method
VPFocus Control

See Also

ActiveControl Property
FocusAction Property
SendFocus Method
VPFocus Control

FormControlName Property Example



The **FormControlName** property can be used to identify a form or control as used within the **VPForm** or **VPFocus** controls. In this example we show how the **FormIndex** and **ControlIndex** properties are used to test against a control. We next show how this same test might be done using the **FormControlName** property.

This example tests if the next control to receive focus is the "cancel" command button.

```
If Forms(Frm.VPFocus1.FormIndex).controls(Frm.VPFocus1.ControlIndex) Is Frm.cmdCancel Then
    'Send focus on to Trapped control
    Frm.VPFocus1.FocusAction = vxSendFocus '1
End If
```

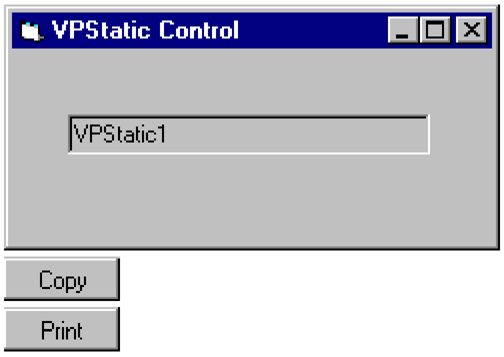
This is an alternate way to test the same thing using the **FormControlName** property.

```
If Frm.VPFocus1.FormControlName = "Form1.cmdCancel." Then
    'Send focus on to Trapped control
    Frm.VPFocus1.FocusAction = vxSendFocus '1
End If
```

See Also

ControlIndex Property
FormIndex Property
VPForm Control
VPFocus Control

FormIndex Property Example



When used with the **VPForm** control the **FormIndex** property along with the **ControlIndex** property is used to identify the control where the mouse pointer is currently over. This property is useful in both the **MouseMove** and **MouseUp** events of the **VPForm** control.

In the **MouseMove** event the **FormIndex** can be used together with the **ControlIndex** property to identify the control the mouse pointer is currently over. The local object variable **Cntrl** can be set to the identified control in the **Controls Collection**. This object variable can further be tested to see if the control, the mouse pointer is over, is the drop target, VPTextBox2 in this example, or some other control. If the mouse pointer is over some other control then set the **Cursor** property to the same "No Drop" cursor. It the mouse pointer is over the target VPTextBox2 control then set the **Cursor** property to a valid drag cursor, such as the cursor resource 6003 in the VFORM.VBX file.

```
Sub VPForm1_MouseMove (Button As Integer, Shift As Integer, X As Single, Y As Single)
    Dim Cntrl As Control
    If VPForm1.ControlIndex = -1 Then
        VPForm1.Cursor = "6001"
    Else
        Set Cntrl = Forms(VPForm1.FormIndex).Controls(VPForm1.ControlIndex)
        If Cntrl Is VPTextBox2 Then
            VPForm1.Cursor = "6003"
        Else
            VPForm1.Cursor = "6001"
        End If
    End If
End Sub
```

In the **MouseUp** event the **FormIndex** property along with the **ControlIndex** property can again be used to set the local object variable, **Cntrl**, to the currently pointed to control, where the mouse button was released. If this object variable is the same as our target VPTextBox2 control, the results of the drag operation can be executed. In this example, the drag-and-drop operation transfers the contents of the source **VPTextBox1** TextBox control to the targeted **VPTextBox2** TextBox control. Note: at the time of the original mouse down event the module level object variable, **Source**, was set to the **VPTextBox1** control.

```
Sub VPForm1_MouseUp (Button As Integer, Shift As Integer, X As Single, Y As Single)
    Dim Cntrl As Control
    VPForm1.Cursor = ""
    VPForm1.MouseCapture = False
    If VPForm1.ControlIndex > -1 Then
        Set Cntrl = Forms(VPForm1.FormIndex).Controls(VPForm1.ControlIndex)
        If Cntrl Is VPTextBox2 Then
            VPTextBox2.Text = Source.Text
        End If
    End If
End Sub
```

When used with the **VPFocus** control, the **FormIndex** property along with the **ControlIndex** property is used to identify the control where **Focus** should go to next. If **Focus** is to go to an object which executes an event code that cancels the current process or exits the application, it may not make sense to continue the normal processing of a **LostFocus** event. In this example we use the **FormIndex** property along with the **ControlIndex** property to identify the "next" control and to test if **Focus** is next being passed to the "Cancel" command button.

```
Sub LostFocusDefProc (Frm As Form, ctl As Control, rsDataVal)
    'Check if LostFocus event is valid
    If Frm.VPFocus1.ActiveControl = ctl.hWnd Then
        'Check if next target control is cancel button
        If Forms(Frm.VPFocus1.FormIndex).controls(Frm.VPFocus1.ControlIndex) Is Frm.cmdCancel
Then
            'Send focus on to Trapped control
            Frm.VPFocus1.FocusAction = vxSendFocus '1
        Else
            Select Case UCase$(rsDataVal)
                Case "BAD"
                    Beep
                    MsgBox "Invalid Data - Please enter again."
                    'Reset data
                    ctl.Text = ""
                    'Invalid data so send focus back to itself
                    Frm.VPFocus1.FocusAction = vxReturnFocus '2
                    Exit Sub
                Case "MSG"
                    'Having a MsgBox normally eats up any GotFocus and causes real problems
                    Beep
                    MsgBox "This is a valid entry - Data accepted."
                    'Send focus on to Trapped control
                    Frm.VPFocus1.ForceGotFocus = True
                    Frm.VPFocus1.FocusAction = vxSendFocus '1
                Case Else 'GOOD or any other input
                    'Send focus on to Trapped control
                    Frm.VPFocus1.FocusAction = vxSendFocus '1
            End Select
        End If
        'Turn off focus highlight color
        If TypeOf ctl Is VPTextBox Then
            ctl.BackColor = glBackColor
            ctl.ForeColor = glForeColor
        End If
    End If
End Sub
```

See Also

ControlIndex Property
FocusAction Property
FormControlName Property
ReturnFocus Method
SendFocus Method
TrapFocusOff Method
TrapFocusOn Method
VPForm Control
VPFocus Control

See Also

GridLines Property
VPComboBox Control
VPListBox Control

See Also

GridAppearance Property
VPComboBox Control
VPListBox Control

See Also

ColHeadAlign Property
ColHeading Property
HeadingBackColor Property
HeadingForeColor Property
VPComboBox Control
VPListBox Control

See Also

Heading Property
HeadingForeColor Property
ListBackColor Property
ListForeColorProperty
VPComboBox Control
VPListBox Control

See Also

Heading Property
HeadingBackColor Property
ListBackColor Property
ListForeColor Property
VPComboBox Control
VPListBox Control

See Also

HeadingBackColor Property
HeadingForeColor Property
ListForeColor Property
VPComboBox Control

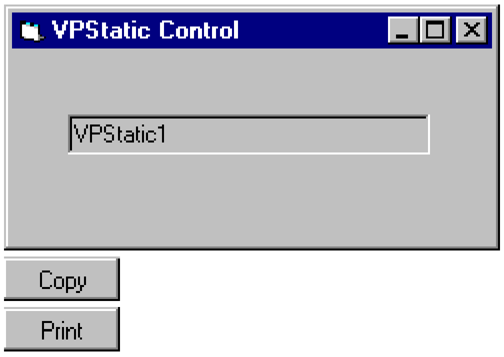
See Also

HeadingBackColor Property
HeadingForeColor Property
ListBackColor Property
VPComboBox Control

See Also

Style Property
VPComboBox Control

MaxCols Property Example



In this example we will add two new columns by setting the **MaxCols** property to a value two higher then the current number. The List currently had 6 columns and by setting it to 8 we add two new columns. We then set the alignment, format, and width of these new columns in a **VPComboBox** control. We also set the heading or caption and the alignment for a column heading. In this example the first new column will be used to display a name or description field and the next new column will be used to display a dollar amount field.

```
VPComboBox1.MaxCols = 8

VPComboBox1.Col = 7
VPComboBox1.ColAlign = 0 'Left justification
VPComboBox1.ColFormat = "" 'No formatting
VPComboBox1.ColWidth = 3200 'Width in twips
VPComboBox1.ColHeading = "Description"
VPComboBox1.ColHeadAlign = 2 'Center justification

VPComboBox1.Col = 8
VPComboBox1.ColAlign = 1 'Right justification
VPComboBox1.ColFormat = "$#,##0.00;($#,##0.00)" 'Currency format
VPComboBox1.ColWidth = 1600 'Width in twips
VPComboBox1.ColHeading = "Amount"
VPComboBox1.ColHeadAlign = 2 'Center justification
```

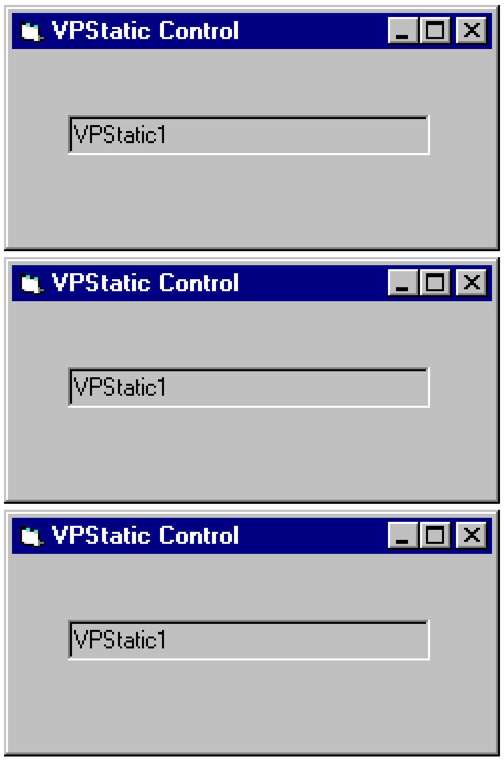
See Also

CellText Property
Col Property
ColAlign Property
ColBound Property
ColDataField Property
ColDataSource Property
ColFormat Property
ColHeadAlign Property
ColHeading Property
ColLink Property
ColListField Property
ColSortBy Property
ColSortOrder Property
ColWidth Property
VPComboBox Control
VPListBox Control

See Also

ColWidth Property
VPComboBox Control

MouseCapture Property Example



In this example we will identify all the code that is necessary to implement drag-and-drop operations using the **VPForm** control. This will include specific code that involves the **MouseCapture** property.

This example involves two **TextBox** controls (actually two **VPTextBox** controls) where we will provide the ability to drag the contents of one **TextBox** control (**VPTextBox1)** to another (**VPTextBox2)**. In order to avoid confusing normal mouse operations, such as setting an insertion point or selecting text within a **TextBox,** and the mouse operations associated with a drag-and-drop operation, we will employ a **Timer** control. The **Timer** control will be set to fire its **Timer** event in 500 milliseconds. In the first or "source" **TextBox** control (**VPTextBox1**) we will add code to the **MouseDown** event to enable the **Timer** control.

```
Sub VPTextBox1_MouseDown (Button As Integer, Shift As Integer, X As Single, Y As Single)
    Timer1.Enabled = True
End Sub
```

In the **Timer1** control **Timer** event we will turn on the mouse capturing feature of the **VPForm** control, initialize our drag mouse cursor, and record which control is our "source" control by setting a form level object variable named **Source** to the **VPTextBox1** control. In this event we also disable the **Timer** control. By setting the **MouseCapture** property we start the drag-and-drop process.

```
Sub Timer1_Timer ()
    VPForm1.MouseCapture = True
    VPForm1.Cursor = "6003"
    Set Source = VPTextBox1
    Timer1.Enabled = False
End Sub
```

Although not part of the drag-and-drop process, we add code to the **MouseUp** event of the first **TextBox** control to turn off the **Timer** control if the mouse button is released while in the "source" **TextBox** control. This is done so that the drag-and-drop process is not started if it hasn't started already.

```
Sub VPTextBox1_MouseUp (Button As Integer, Shift As Integer, X As Single, Y As Single)
    Timer1.Enabled = False
End Sub
```

Once the mouse capture process is initiated, all mouse events for the system are redirected to the custom mouse events of the **VPForm** control. In the **MouseMove** event we check to see where the mouse pointer is currently pointing to and set the appropriate custom mouse pointer. If over the form (without being on a control) we set the **Cursor** property of the **VPForm** control to an appropriate drag cursor resource (id 6003) located in the VFORM.VBX file. If on any control other than our "source" or "target" control or if outside any form of our application, we set the mouse pointer to a "No-Drop" cursor resource (id 6001). If the mouse pointer is over our "source" control we set the mouse pointer to our valid drag cursor resource (id 6003). And finally, if the mouse is over our "target" control we can set it to a valid drop cursor resource (id 6008).

```
Sub VPForm1_MouseMove (Button As Integer, Shift As Integer, X As Single, Y As Single)
    Dim Cntrl As Control
    If VPForm1.FormIndex = -1 Then
        'Mouse outside any application form - use No Drop cursor
        VPForm1.Cursor = "6001"
    Else
        'Mouse within an application form
        If VPForm1.ControlIndex = -1 Then
            'Mouse not on a control - on form itself - use valid drag cursor
            VPForm1.Cursor = "6003"
        Else
            'Identify control mouse is currently over
            Set Cntrl = Forms(VPForm1.FormIndex).Controls(VPForm1.ControlIndex)
            If Cntrl Is VPTextBox1 Then
                'Mouse is over source control - use valid drag cursor
                VPForm1.Cursor = "6003"
            Else
                If Cntrl Is VPTextBox2 Then
                    'Mouse is over target control - use valid drop cursor
                    VPForm1.Cursor = "6008"
                Else
                    'Mouse is on some other control - use No Drop cursor
                    VPForm1.Cursor = "6001"
                End If
            End If
        End If
    End If
End Sub
```

The last code we need is for the custom **MouseUp** event of the **VPForm** control. This event is fired when the mouse button is released. In this event we reset the mouse pointer to the cursor defined by Visual Basic previous to our assigning a custom cursor. We also turn off the drag-and-drop operation by setting the **MouseCapture** property to **False**. In this event we also check to see if the mouse button was release over our target control or somewhere else. If over the target **VPTextBox2** control we transfer the text from our first **TextBox** control (**VPTextBox1)** to the **Text** property of the target **VPTextBox2** control and set focus to our target control. This completes the drag-and-drop process.

```
Sub VPForm1_MouseUp (Button As Integer, Shift As Integer, X As Single, Y As Single)
    Dim Cntrl As Control
    VPForm1.Cursor = ""
    VPForm1.MouseCapture = False
    If VPForm1.ControlIndex > -1 Then
```

```
        Set Cntrl = Forms(VPForm1.FormIndex).Controls(VPForm1.ControlIndex)
        If Cntrl Is VPTextBox2 Then
            VPTextBox2.Text = Source.Text
            VPTextBox2.SetFocus
        End If
    End If
End Sub
```

As you can see, only a handful of events need to be programmed for the custom drag-and-drop operations of the **VPForm** control as compared to the many **MouseUp**, **MouseDown**, **DragOver**, and **DragDrop** events of the normal Visual Basic operations.

See Also

Cursor Property
MouseDown Event
MouseMove Event
MouseUp Event
VPForm Control

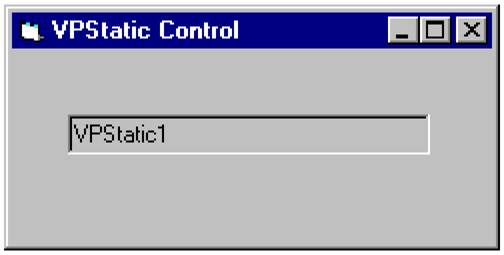
See Also

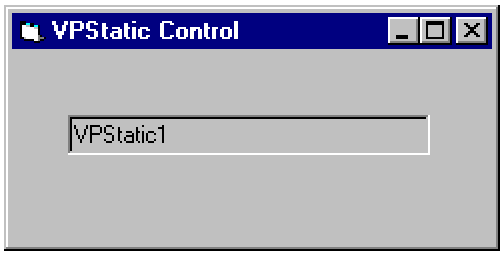
ColBound Property
ColDataField Property
ColDataSource Property
ColLink Property
ColListField Property
DataField Property
DataSource Property
Data Control
VPComboBox Control
VPListBox Control

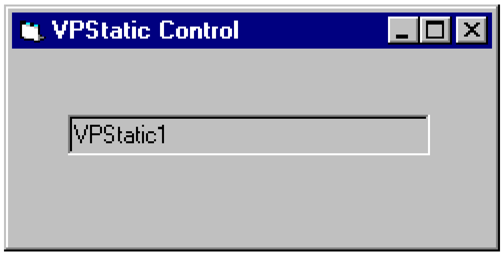
See Also

CaseSensitive Property
ColSortBy Property
ColSortOrder Property
VPComboBox Control
VPListBox Control

TargetControl Property Example







The **TargetControl** property is part of the **VPFocus** control that is used to manage **GotFocus**/**LostFocus** event processing when dealing with MDI Child forms. This property is usually used in the **GotFocus** event of a control.

Rather than placing code in each **GotFocus** event procedure, one generalized procedure can be created and called from each event procedure. In this example, we will name the generalized procedure *GotFocusDefProc*. The **TargetControl** property is to check if processing for a **GotFocus** event is valid in addition to the normal **ActiveControl** property.

In the *GotFocusDefProc* procedure we first check if the **GotFocus** event is valid. A **GotFocus** event will be valid if no control is currently active. We can test this by seeing if the **ActiveControl** property is set to zero (0). If the ActiveControl property is zero we next check to see if the TargetControl handle, or the control where the **VPFocus** control is sending **Focus**, is the same as the current controls handle. If the handles are the same, we have a valid **GotFocus** event. If not, it is usually because Visual Basic has fired an erroneous **GotFocus** event for the last control that had **Focus** when **Focus** left the MDI child form. In this case we want all **Focus** processing to fail for current control.

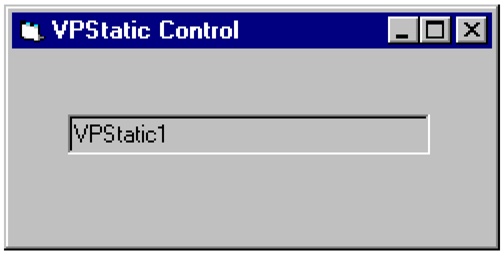
If we have a valid **GotFocus** event we record the current control as the active control by setting the **ActiveControl** property to the controls Window handle as we normally would. This will start the **VPFocus** control trapping for **Focus** events.

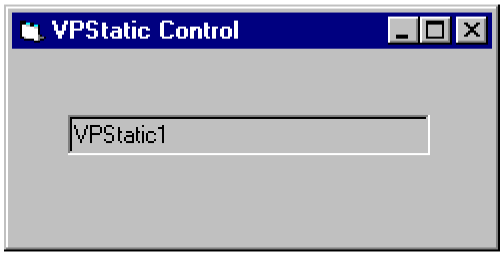
```
Sub GotFocusDefProc (Frm As Form, ctl As Control)

    'Check if GotFocus event is valid

    If Form1.VPFocus1.ActiveControl = 0 Then

        'Necessary check for MDI Child forms
        If Form1.VPFocus1.TargetControl = ctl.hWnd Then
            'Set Active control and start trapping for focus events
            Form1.VPFocus1.ActiveControl = ctl.hWnd
            'If Text Box control change Colors to show where focus is
            If TypeOf ctl Is VPTextBox Then
                glBackColor = ctl.BackColor
                glForeColor = ctl.ForeColor
                ctl.BackColor = glHLBackColor
                ctl.ForeColor = glHLForeColor
            End If
        End If
    End If
End Sub
```

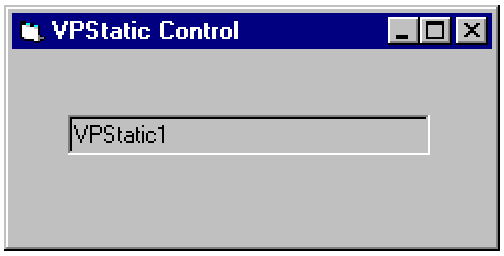
See Also

FocusAction Property
SendFocus Method
VPFocus Control

MouseDown Event Example







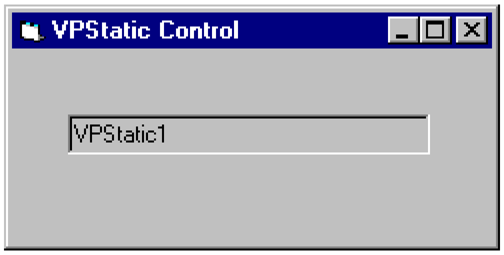
This example demonstrates a simple paint application. The MouseDown event procedure works with a related MouseMove event procedure to enable painting when any mouse button is pressed. The MouseUp event procedure disables painting.

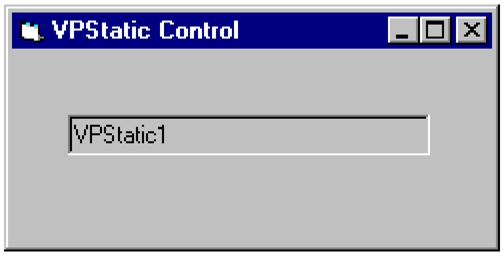
```
Dim PaintNow As Integer

Sub Form_MouseDown (Button As Integer, Shift As Integer, X As Single, Y As Single)
        PaintNow = True 'Brush on
End Sub

Sub Form_MouseUp (Button As Integer, X As Single, Y As Single)
        PaintNow = False 'Turn off painting
End Sub

Sub Form_MouseMove (Button As Integer, Shift As Integer, X As Single, Y As Single)
        If PaintNow Then
                PSet (X, Y) 'Draw a point
        End If
End Sub

Sub Form_Load ()
        DrawWidth = 10 'Use wider brush
        ForeColor = RGB(0, 0, 255) 'Set drawing color
End Sub
```

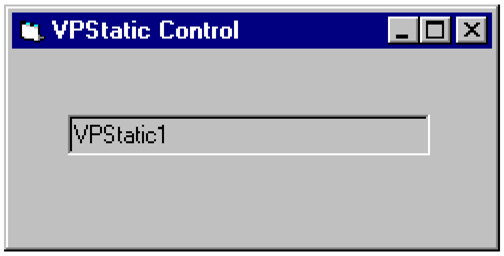
See Also

Cursor Property
MouseCapture Property
MouseMove Event
MouseUp Event
VPForm Control
VPTextBox Control
VPStatic Control
VPComboBox Control
VPListBox Control

MouseMove Event Example







This example demonstrates a simple paint application. The MouseDown event procedure works with a related MouseMove event procedure to enable painting when any mouse button is pressed. The MouseUp event procedure disables painting.

```
Dim PaintNow As Integer

Sub Form_MouseDown (Button As Integer, Shift As Integer, X As Single, Y As Single)
        PaintNow = True 'Brush on
End Sub

Sub Form_MouseUp (Button As Integer, X As Single, Y As Single)
        PaintNow = False 'Turn off painting
End Sub

Sub Form_MouseMove (Button As Integer, Shift As Integer, X As Single, Y As Single)
        If PaintNow Then
                PSet (X, Y) 'Draw a point
        End If
End Sub

Sub Form_Load ()
        DrawWidth = 10 'Use wider brush
        ForeColor = RGB(0, 0, 255) 'Set drawing color
End Sub
```

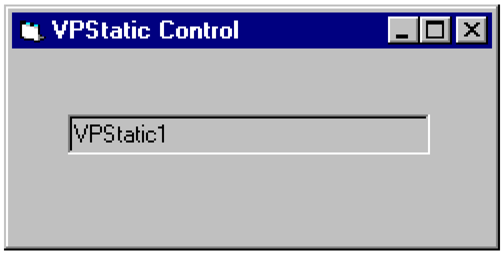
Another example is the **MouseMove** event of the **VPForm** control. This example involves programming drag-and-drop operations through the custom mouse events of the **VPForm** control. Once the mouse capture process is initiated with the **MouseCapture** property, all mouse events for the system are redirected to the custom mouse events of the **VPForm** control. In the **MouseMove** event we check to see where the mouse pointer is currently pointing to and set the appropriate custom mouse pointer. If over the form (without being on a control) we set the **Cursor** property of the **VPForm** control to an appropriate drag cursor resource (id 6003)   located in the VFORM.VBX file. If on any control other than our "source" or "target" control or if outside any form of our application, we set the mouse pointer to a "No-Drop" cursor resource (id 6001). If the mouse pointer is over our "source" control we set the mouse pointer to our valid drag cursor resource (id 6003). And finally, if the mouse is over our "target" control we can set it to a valid drop cursor resource (id 6008).

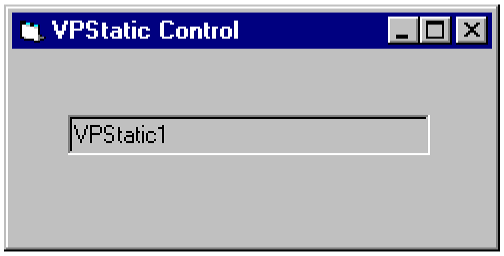
```
Sub VPForm1_MouseMove (Button As Integer, Shift As Integer, X As Single, Y As Single)
    Dim Cntrl As Control
    If VPForm1.FormIndex = -1 Then
        'Mouse outside any application form - use No Drop cursor
        VPForm1.Cursor = "6001"
    Else
        'Mouse within an application form
        If VPForm1.ControlIndex = -1 Then
            'Mouse not on a control - on form itself - use valid drag cursor
            VPForm1.Cursor = "6003"
        Else
            'Identify control mouse is currently over
            Set Cntrl = Forms(VPForm1.FormIndex).Controls(VPForm1.ControlIndex)
            If Cntrl Is VPTextBox1 Then
                'Mouse is over source control - use valid drag cursor
                VPForm1.Cursor = "6003"
            Else
                If Cntrl Is VPTextBox2 Then
                    'Mouse is over target control - use valid drop cursor
                    VPForm1.Cursor = "6008"
                Else
                    'Mouse is on some other control - use No Drop cursor
                    VPForm1.Cursor = "6001"
                End If
            End If
        End If
    End If
End Sub
```

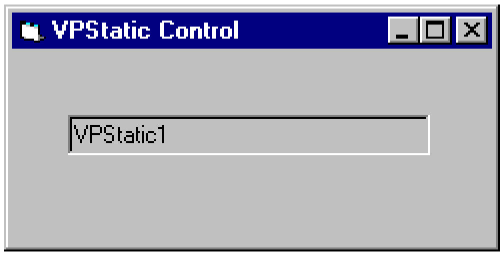
See Also

Cursor Property
MouseCapture Property
MouseDown Event
MouseUp Event
VPForm Control
VPTextBox Control
VPStatic Control
VPComboBox Control
VPListBox Control

MouseUp Event Example







This example demonstrates a simple paint application. The MouseDown event procedure works with a related MouseMove event procedure to enable painting when any mouse button is pressed. The MouseUp event procedure disables painting.

```
Dim PaintNow As Integer

Sub Form_MouseDown (Button As Integer, Shift As Integer, X As Single, Y As Single)
        PaintNow = True 'Brush on
End Sub

Sub Form_MouseUp (Button As Integer, X As Single, Y As Single)
        PaintNow = False 'Turn off painting
End Sub

Sub Form_MouseMove (Button As Integer, Shift As Integer, X As Single, Y As Single)
        If PaintNow Then
                PSet (X, Y) 'Draw a point
        End If
End Sub

Sub Form_Load ()
        DrawWidth = 10 'Use wider brush
        ForeColor = RGB(0, 0, 255) 'Set drawing color
End Sub
```

Another example is the **MouseUp** event of the **VPForm** control. This example involves programming drag-and-drop operations through the custom mouse events of the **VPForm** control. Once the mouse capture process is initiated with the **MouseCapture** property, all mouse events for the system are redirected to the custom mouse events of the **VPForm** control. In this **MouseUp** event we reset the mouse pointer to the cursor defined by Visual Basic previous to our assigning a custom cursor. We also turn off the drag-and-drop operation by setting the **MouseCapture** property to **False**. In this event we also check to see if the mouse button was release over our target control or somewhere else. If over the target **VPTextBox2** control we transfer the text from our first **TextBox** control (**VPTextBox1)** to the **Text** property of the target **VPTextBox2** control and set focus to our target control. This completes the drag-and-drop process.

```
Sub VPForm1_MouseUp (Button As Integer, Shift As Integer, X As Single, Y As Single)
    Dim Cntrl As Control
    VPForm1.Cursor = ""
    VPForm1.MouseCapture = False
    If VPForm1.ControlIndex > -1 Then
        Set Cntrl = Forms(VPForm1.FormIndex).Controls(VPForm1.ControlIndex)
        If Cntrl Is VPTextBox2 Then
            VPTextBox2.Text = Source.Text
            VPTextBox2.SetFocus
        End If
    End If
End Sub
```

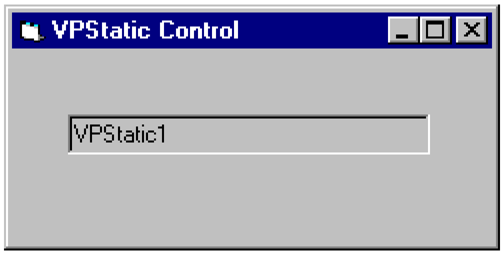
See Also

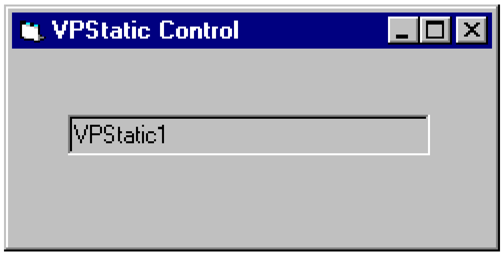
Cursor Property
MouseCapture Property
MouseDown Event
MouseMove Event
VPForm Control
VPTextBox Control
VPStatic Control
VPComboBox Control
VPListBox Control

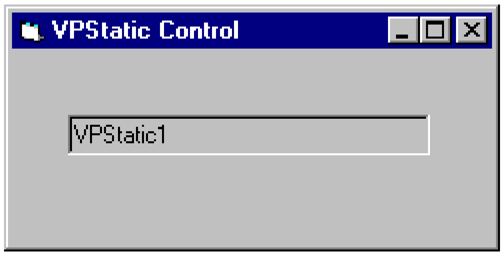
See Also

VLocateText Function
VPComboBox Control
VPListBox Control

LocateText Method Example







This first example searches the items of a **VPListBox** control for the sub-string elect, locating the first row which may have any phrase or word that has the characters elect (such as the word electronics) within the third column, which, in this example, is the company name column. The row index returned is set as the selected row or item.

```
Sub Command1_Click ()
    VPListBox1.ListIndex = VPListBox1.LocateText "elec", 3, 0, vxSubStringMatch, _
        vxDown, vxCaseSensitive
End Sub
```

This second example recursively searches the items of a **VPListBox** control for the sub-string elect, locating any rows which may have any phrase or word that has the characters elect (such as the word electronics) within the third column, which, in this example, is the company name column. In this example, the search is executed in a loop, with the starting position modified each time, continuing the search after each successful locate. When a row is matched, the row is added to a second **VPListBox** control. The loop is terminated upon an unsuccessful match.

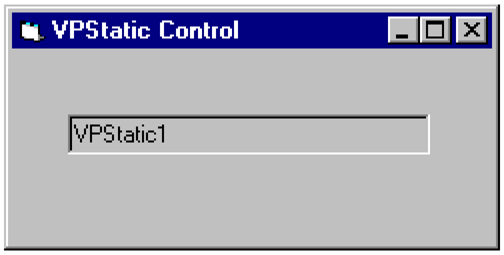
```
Sub Command1_Click ()
    Dim iFound As Integer
    iFound = -1
    Do
```

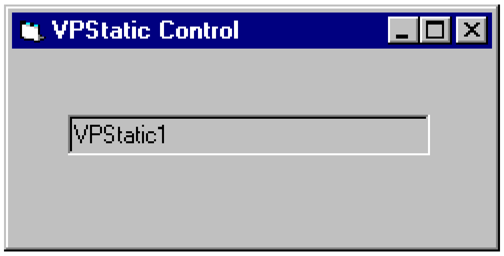
```
        iFound = VPListBox1.LocateText "elec", 3, iFound, vxSubStringMatch, _
            vxDown, vxCaseSensitive
        If iFound <> -1 Then
            VPListBox2.AddItem VPListBox1.List(iFound)
        End If
    Loop Until iFound = -1
End Sub
```

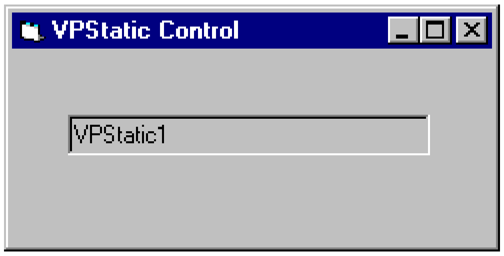
See Also

FocusAction Property
SendFocus Method
TrapFocusOff Method
TrapFocusOn Method
VPFocus Control

ReturnFocus Method Example







Rather than placing code in each **GotFocus** or **LostFocus** event procedure, one generalized procedure can be created and called from each event procedure type. In this example, we will define two generalized procedures, one named *GotFocusDefProc* and the other *LostFocusDefProc*. In the first generalized procedure the **ActiveControl** property is first used to check if processing for a **GotFocus** event is valid, and if valid, the **ActiveControl** property is assign the "current" control as the "active" control through the **TrapFocusOn** method. This is done by passing the window handle of the current control as an argument to the **TrapFocusOn** method.

In the *GotFocusDefProc* procedure we first check if the **GotFocus** event is valid. A **GotFocus** event will be valid if no control is currently "active". We can test this by seeing if the **ActiveControl** property is set to zero (0).

If we have a valid **GotFocus** event we record the current control as the "active" control by passing the control's Window handle when we execute the **TrapFocusOn** method. This will also start the **VPFocus** control trapping for **Focus** events.

```
Sub GotFocusDefProc (Frm As Form, ctl As Control)
    'Check if GotFocus event is valid
    If Frm.VPFocus1.ActiveControl = 0 Then
        'Set Active control and start trapping for focus events.
        Frm.VPFocus1.TrapFocusOn ctl.hWnd
        'If Text Box control then change Colors to help show where focus is
        If TypeOf ctl Is VPTextBox Then
            glBackColor = ctl.BackColor
            glForeColor = ctl.ForeColor
            ctl.BackColor = glHLBackColor
            ctl.ForeColor = glHLForeColor
        End If
    End If
End Sub
```

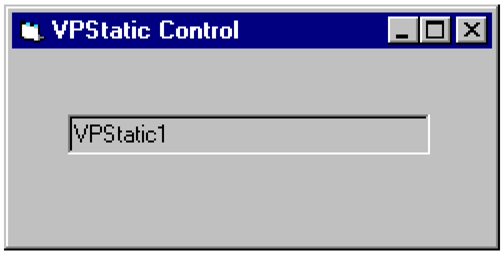
In the *LostFocusDefProc* procedure we first check if the **LostFocus** event is valid. A **LostFocus** event will be valid if the object reference in the **ActiveControl** property is equal to the current control. We can test this by seeing if the **ActiveControl** property is equal to window handle of the current control.
If we have a valid **LostFocus** event we next check if the control where **Focus** will next be sent is a special "Cancel" command button control. In this case, any processing of our **LostFocus** event should be by-passed. If **Focus** is going to any other control we check the validity of the data entered into the current control. If it fails any validation checks the **ReturnFocus** method is executed, returning focus back to the current control. If the data is acceptable then the **SendFocus** method is executed, sending **Focus** on to the next control.

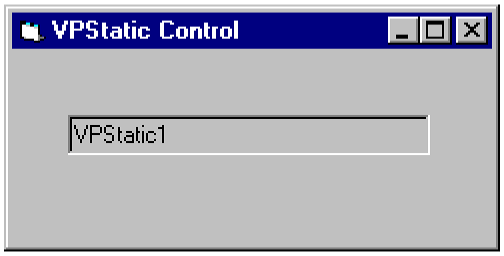
```
Sub LostFocusDefProc (Frm As Form, ctl As Control, rsDataVal)
    'Check if LostFocus event is valid
    If Frm.VPFocus1.ActiveControl = ctl.hWnd Then
        'Check if next target control is cancel button
        If Forms(Frm.VPFocus1.FormIndex).controls(Frm.VPFocus1.ControlIndex) Is Frm.cmdCancel
Then
            'Send focus on to Trapped control
            Frm.VPFocus1.SendFocus
        Else
            Select Case UCase$(rsDataVal)
                Case "BAD"
                    Beep
                    MsgBox "Invalid Data - Please enter again."
                    'Reset data
                    ctl.Text = ""
                    'Invalid data so send focus back to itself
                    Frm.VPFocus1.ReturnFocus
                    Exit Sub
                Case "MSG"
                    'Having a MsgBox normally eats up any GotFocus causing real problems
                    Beep
                    MsgBox "This is a valid entry - Data accepted."
                    'Send focus on to Trapped control
                    Frm.VPFocus1.ForceGotFocus = True
                    Frm.VPFocus1.SendFocus
                Case Else 'GOOD or any other input
                    'Send focus on to Trapped control
                    Frm.VPFocus1.SendFocus
            End Select
        End If
        'Turn off focus highlight color
        If TypeOf ctl Is VPTextBox Then
            ctl.BackColor = glBackColor
            ctl.ForeColor = glForeColor
        End If
    End If
End Sub
```

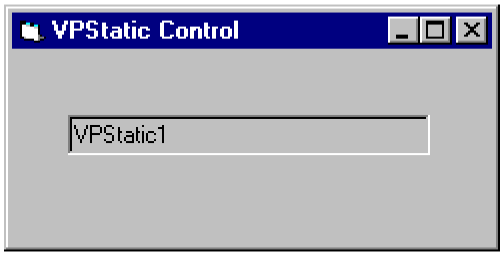
See Also

FocusAction Property
ForceGotFocus Property
TargetControl Property
ReturnFocus Method
TrapFocusOff Method
TrapFocusOn Method
VPFocus Control

SendFocus Method Example







Rather than placing code in each **GotFocus** or **LostFocus** event procedure, one generalized procedure can be created and called from each event procedure type. In this example, we will define two generalized procedures, one named *GotFocusDefProc* and the other *LostFocusDefProc*. In the first generalized procedure the **ActiveControl** property is first used to check if processing for a **GotFocus** event is valid, and if valid, the **ActiveControl** property is assign the "current" control as the "active" control through the **TrapFocusOn** method.   This is done by passing the window handle of the current control as an argument to the **TrapFocusOn** method.

In the *GotFocusDefProc* procedure we first check if the **GotFocus** event is valid. A **GotFocus** event will be valid if no control is currently "active". We can test this by seeing if the **ActiveControl** property is set to zero (0).

If we have a valid **GotFocus** event we record the current control as the "active" control by passing the control's Window handle when we execute the **TrapFocusOn** method. This will also start the **VPFocus** control trapping for **Focus** events.

```
Sub GotFocusDefProc (Frm As Form, ctl As Control)
    'Check if GotFocus event is valid
    If Frm.VPFocus1.ActiveControl = 0 Then
        'Set Active control and start trapping for focus events
        Frm.VPFocus1.TrapFocusOn ctl.hWnd
        'If Text Box control then change Colors to help show where focus is
        If TypeOf ctl Is VPTextBox Then
            glBackColor = ctl.BackColor
            glForeColor = ctl.ForeColor
            ctl.BackColor = glHLBackColor
            ctl.ForeColor = glHLForeColor
        End If
    End If
End Sub
```

In the *LostFocusDefProc* procedure we first check if the **LostFocus** event is valid. A **LostFocus** event will be valid if the object reference in the **ActiveControl** property is equal to the current control. We can test this by seeing if the **ActiveControl** property is equal to window handle of the current control.

If we have a valid **LostFocus** event we next check if the control where **Focus** will next be sent is a special "Cancel" command button control. In this case, any processing of our **LostFocus** event should be by-passed. If **Focus** is going to any other control we check the validity of the data entered into the current control. If it fails any validation checks the **ReturnFocus** method is executed, returning focus back to the current control. If the data is acceptable then the **SendFocus** method is executed, sending **Focus** on to the next control.

```
Sub LostFocusDefProc (Frm As Form, ctl As Control, rsDataVal)
    'Check if LostFocus event is valid
    If Frm.VPFocus1.ActiveControl = ctl.hWnd Then
        'Check if next target control is cancel button
        If Forms(Frm.VPFocus1.FormIndex).controls(Frm.VPFocus1.ControlIndex) _
        Is Frm.cmdCancel Then
            Frm.VPFocus1.SendFocus 'Send focus on to Trapped control
        Else
            Select Case UCase$(rsDataVal)
                Case "BAD"
                    Beep
                    MsgBox "Invalid Data - Please enter again."
                    'Reset data
                    ctl.Text = ""
                    Frm.VPFocus1.ReturnFocus 'Invalid data so send focus back to itself
                    Exit Sub
                Case "MSG"
                    'Having a MsgBox normally eats up any GotFocus causing real problems
                    Beep
                    MsgBox "This is a valid entry - Data accepted."
                    Frm.VPFocus1.ForceGotFocus = True
                    Frm.VPFocus1.SendFocus 'Send focus on to Trapped control
                Case Else 'GOOD or any other input
                    Frm.VPFocus1.SendFocus 'Send focus on to Trapped control
            End Select
        End If
        'Turn off focus highlight color
        If TypeOf ctl Is VPTextBox Then
            ctl.BackColor = glBackColor
            ctl.ForeColor = glForeColor
        End If
    End If
End Sub
```

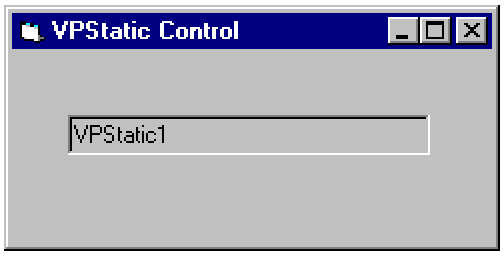
See Also

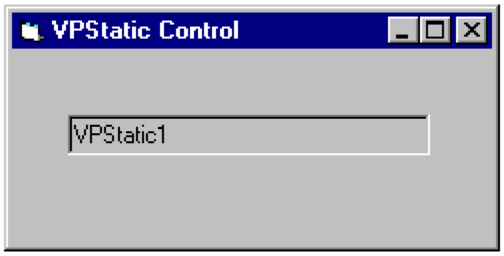
FocusAction Property
ReturnFocus Method
SendFocus Method
TrapFocusOn Method
VPFocus Control

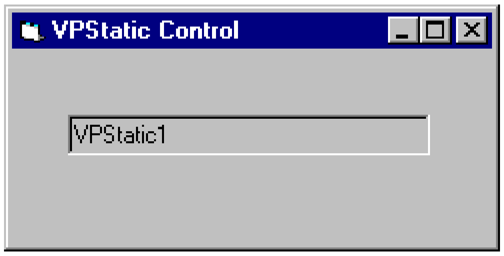
See Also

FocusAction Property
ReturnFocus Method
SendFocus Method
TrapFocusOff Method
VPFocus Control

TrapFocusOn Method Example







The **TrapFocusOn** method is usually used in the **GotFocus** event of a control.

Rather than placing code in each **GotFocus** event procedure, one generalized procedure can be created and called from each event procedure. In this example, we will name the generalized procedure *GotFocusDefProc*. The **ActiveControl** property is first used to check if processing for a **GotFocus** event is valid, and if valid, the **ActiveControl** property is assigned the "current" control as the "active" control through the **TrapFocusOn** method. This is done by passing the window handle of the current control as an argument to the **TrapFocusOn** method.

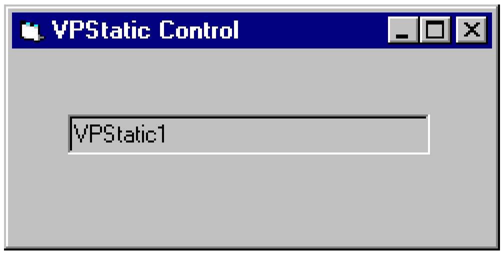
In the *GotFocusDefProc* procedure we first check if the **GotFocus** event is valid. A **GotFocus** event will be valid if no control is currently "active". We can test this by seeing if the **ActiveControl** property is set to zero (0).
If we have a valid **GotFocus** event we record the current control as the "active" control by passing the control's Window handle when we execute the **TrapFocusOn** method. This will also start the **VPFocus** control trapping for **Focus** events.

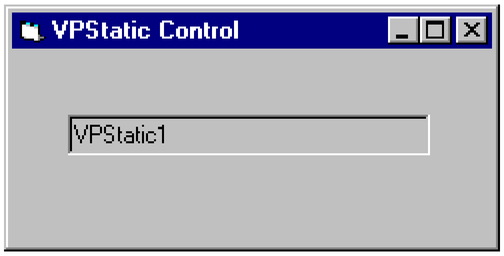
```
Sub GotFocusDefProc (Frm As Form, ctl As Control)
    'Check if GotFocus event is valid
    If Frm.VPFocus1.ActiveControl = 0 Then
        'Set Active control and start trapping for focus events
        Frm.VPFocus1.TrapFocusOn ctl.hWnd
        'If Text Box control then change Colors to help show where focus is
        If TypeOf ctl Is VPTextBox Then
            glBackColor = ctl.BackColor
            glForeColor = ctl.ForeColor
            ctl.BackColor = glHLBackColor
            ctl.ForeColor = glHLForeColor
        End If
    End If
End Sub
```

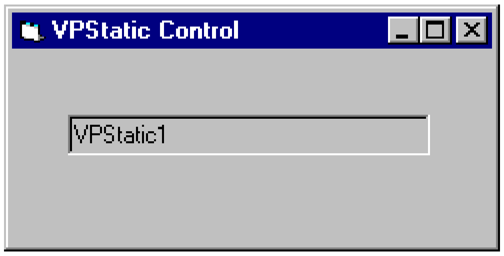
See Also

LocateText Method
VPComboBox Control
VPListBox Control

VLocateText Function Example







This first example searches the items of a **VPListBox** control for the sub-string elect, locating the first row which may have any phrase or word that has the characters elect (such as the word electronics) within the third column, which, in this example, is the company name column. The row index returned is set as the selected row or item.

```
Sub Command1_Click ()
    VPListBox1.ListIndex = VLocateText (VPListBox1, "elec", 3, 0, vxSubStringMatch, _
        vxDown, vxCaseSensitive)
End Sub
```

This second example recursively searches the items of a **VPListBox** control for the sub-string elect, locating any rows which may have any phrase or word that has the characters elect (such as the word electronics) within the third column, which, in this example, is the company name column. In this example, the search is executed in a loop, with the starting position modified each time, continuing the search after each successful locate. When a row is matched, the row is added to a second **VPListBox** control. The loop is terminated upon an unsuccessful match.

```
Sub Command1_Click ()
    Dim iFound As Integer
    iFound = -1
    Do
        iFound = VLocateText (VPListBox1, "elec", 3, iFound, vxSubStringMatch, _
            vxDown, vxCaseSensitive)
        If iFound <> -1 Then
            VPListBox2.AddItem VPListBox1.List(iFound)
        End If
    Loop Until iFound = -1
End Sub
```

See Also

ColAlign Property
ColBound Property
ColFormat Property
ColHeadAlign Property
ColHeading Property
ColLink Property
ColListField Property
ColSortBy Property
ColSortOrder Property
ColWidth Property
VPComboBox Control
VPListBox Control

data-aware

A control is **data-aware** when it can provide access to a specific field in a database through a **Data** control. Typically, a **data-aware** control can be bound to a **Data** control through its **DataSource** and **DataField** properties. A **data-aware** control is also referred to as a **bound** control.

bound control

A control that provides access to a specific field in a database through a **Data** control. A control is bound to a **Data** control through its **DataSource** and **DataField** properties. When a **Data** control moves from on record to the next, all bound controls change to display data from fields in the current record. When users change data in a bound control and then move to a different record, the changes are automatically saved in the database.

drag-and-drop

Features that enable the user to drag a control or its contents and drop it onto a form or another control using the mouse. An object can be a *source* (an item the user drags) or a *target* (an item on which the user drops a source).

design time

The time during which you build an application in the development environment by adding controls, setting control or form properties, and so on. In contrast, during run time, you interact with the application as a user would.

run time

The time when code is running. During run time, you interact with the application as a user would.

control array

A group of controls that share a common name, type, and event procedures. Each control in the array has a unique index number that can be used to determine which control recognizes an event.

focus

In the Microsoft Windows environment, only one window, form, or control can receive mouse clicks or keyboard input. This object "has the focus."   The focus can be set by the user or by the application. Focus is usually indicated by a highlighted caption or border.

object expression

An expression which specifies a particular object. This expression can include any of the objects' containers.

boolean expression

An expression which evaluates to either **True** or **False.**

string expression

Any expression which evaluates to a sequence of contiguous characters. Elements of the expression can include a function that returns a string, a string literal, a string constant, a string variable, a string variant, or a function that returns a string variant (VarType 8).

numeric expression

Any expression which can be evaluated as a number. Elements of the expression can include any combination of keywords, variables, constants, and operators that result in a number.