Vantage Control Set 2.0 Demo

The **CTLDEMO** project demonstrates the different user interface aspects of the **VPTextBox**, **VPStatic**, **VPComboBox**, **VPListBox**, and **VPForm** controls and how they can be manipulated through code during run time. It also demonstrates some of the advanced features of the **VPComboBox** and **VPListBox** controls, such as extended data awareness, match entry operations (similar to automatic searching found in Access, MS Money, and Quicken), custom sorting which supports multiple sort key columns, and new flexible data locate functions and methods.

The main form of the **CTLDEMO** project displays four sample controls representing a **VPTextBox**, **VPStatic**, **VPComboBox**, and **VPListBox** control class. Associated with each control are a number of other option button and check box controls which you can use to manipulate the sample controls.

Within this project you can manipulate each control's appearance and alignment. For the **VPComboBox** and **VPListBox** controls, you can manipulate a number of properties which determine the look and operations of the list portion of these controls. These list properties are maintained in a separate dialog, available by clicking on the appropriate List Properties command button. These properties include:

GridLines	**MaxDrop** (VPComboBox only)
GridAppearance	**MaxWidth** (VPComboBox only)
Heading	

You can also review and set many of the array properties which define the columns within the list portion of these controls. Reviewing and setting column properties are also handled through a separate dialog, available by clicking on the appropriate Column Properties command button. These properties include:

MaxCols	**ColLink**
ColWidth	**ColHeading**
ColAlign	**ColHeadAlign**

From the main form you can also manipulate the appearance of the demo form itself, and optionally set a custom background brush to be applied to the form's background. At the bottom of the main form is a command button which will take you to a form that displays the additional features of the **VPComboBox** and **VPListBox** controls including data binding, extended data matching features of the **VPComboBox** control, and other custom sorting and search capabilities associated with Lists.


New features of the **VPComboBox** and **VPListBox** controls


For more information on **Vantage Control Set**

New features

A number of new features have been added to the **VPComboBox** and **VPListBox** controls of **Vantage Control Set**. These include the ability to select case sensitive or non case sensitive operations, custom sorting options, flexible data locate functions and methods, and match entry operations which allow the **VPComboBox** control to operate like the Windows Help Search Engine or the interactive matching found in Access, Quicken, or Microsoft Money. There are also new column binding properties which allow selected values to be bound to a second data source. For the **VPComboBox** control, binding through the **ColBound**, **ColDataSource**, and **ColDataField** properties can be used in addition to, or in place of the normal Edit or Static portion binding through the standard **DataSource** and **DataField** properties. This allows for the flexibility to display one column of information linked to the Edit control through the **ColLink** property, and have a different column of data be bound to a data source. These new features even include a powerful **LocateText** method (ActiveX only) or the **VLocateText** exported DLL function for searching a **VPComboBox** or **VPListBox** control for data values within the columns of a List. Many of these new capabilities are demonstrated in the form loaded by clicking on the "Other features..." command button at the bottom of the main form of the **CTLDEMO** project.

The following are some of the new properties, events, methods, and functions of the **VPComboBox** and **VPListBox** controls:

CaseSensitive Property
CellText Property
ColBound Property
ColDataField Property
ColDataSource Property
ColLink Property
ColListField Property
ColSortBy Property
ColSortOrder Property
MatchEntry Property
RowSource Property

CloseUp Event

LocateText Method (ActiveX version)
VLocateText Function (Exported DLL function for VBX version)

CaseSensitive Property

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.

Applies To

VPComboBox Control **VPListBox** Control

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 that 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

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

Applies To

VPComboBox Control **VPListBox** Control

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.

Example

This example uses the **CellText** property to retrieve the contents of the currently selected row and 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
```

ColSortBy Property

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

Applies To

VPComboBox Control **VPListBox** Control

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

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

Applies To

VPComboBox Control **VPListBox** Control

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 columns **ColSortBy** property is set to **True**.

MatchEntry Property

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**.

Applies To

VPComboBox Control

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.

CloseUp Event

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

Applies To

VPComboBox Control

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).

LocateText Method

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.

Applies To

VPComboBox Control **VPListBox** Control

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 that specifies the type of search to be conducted, as described in Settings. Optional argument.
direction	A value or constant that specifies the direction to search within the List, as described in Settings. Optional argument.
scase	A value or constant that 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 OCX version.

Example

This first example searches the items of a **VPListBox** control for the sub-string elect, locating the first row that 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 that 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
```

VLocateText Function

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.

Applies To

VPComboBox Control **VPListBox** Control

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** note: function also found in VPCOMB.VBX

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 that specifies the type of search to be conducted, as described in Settings.
direction	A value or constant that specifies the direction to search within the List, as described in Settings.
scase	A value or constant that 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.

Example

This first example searches the items of a **VPListBox** control for the sub-string elect, locating the first row that 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 that 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
```

ColBound Property

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.

Applies To

VPComboBox Control **VPListBox** Control

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

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

Applies To

VPComboBox Control **VPListBox** Control

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

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.

Applies To

VPComboBox Control **VPListBox** Control

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.

ColLink Property

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.

Applies To

VPComboBox 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

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.

Applies To

VPComboBox Control **VPListBox** Control

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 which 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.

RowSource Property

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.

Applies To

VPComboBox Control **VPListBox** Control

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.