

iList Help



Version 1.0
[25/9/92]

Written by and Copyright (c) Ian R Taylor, 1992.

Contents

Overview

Properties

Licencing

Licensing

This custom control is provided as FreeWare. It is NOT public domain and it may not be modified or altered. You may use the control as you wish, and you may use it in any application, whether personal, corporate or commercial, without reference to me.

If you do use it in an application that you distribute, I would appreciate it if you would mention somewhere that you are using the iList.vbx control, say in the docs, about box or help file. This is NOT required, but is just a request.

However, it is provided as-is and without any guarantee of it's performance or reliability. If you use it, as you are welcome to do, then anything that happens is your responsibility, not mine !

Any questions that may arise as to the use of the control I will try to answer if they are brought to my notice. Posting a message in the VB/Win section of the MSBASIC forum on CompuServe, or email to me on CompuServe are probably the best methods. However, I am specifically not responsible for providing any support whatsoever.

Similarly, if anyone has any requests or feedback regarding ways in which the control could be improved, I would be delighted to hear them.

If you distribute the control then, except when bundled as part of an application, you must include this help file with it, unaltered.

Source code to the control is available, if you would like to see it. The cost for providing the source code is \$25 US. Please send this to me, in an international money order or cash by registered delivery (no cheques drawn on US banks please), at the following address:

Ian R Taylor,
Moorgate East Farm,
Broad Lane,
Rochdale,
England
OL16 4QL

I can be contacted on CompuServe, my ID Number is 100025,557

Control Properties Overview

iList.VBX is an enhanced VB listbox control. It is enhanced in two main ways - it can accept and process multiple selections, and it can accept another data item which is linked to a list box entry.

The additional data item is a long integer, and can be used to represent anything the programmer wishes - a database index number or an offset into an array come to mind. This integer stays with the data item even if the listbox has items deleted or its contents reordered in another way.

The classname of the control is "iList" for the purposes of the VB TypeOf function.

iList supports the following properties in exactly the same way as the standard VB list box. Properties unique to this control are marked with a * below:

NAME
INDEX
PARENT
BACKCOLOR
FORECOLOR
LEFT
TOP
WIDTH
HEIGHT
ENABLED
VISIBLE
MOUSEPOINTER
CAPTION
FONTNAME
FONTSIZE
FONTBOLD
FONTITALIC
FONTSTRIKE
FONTUNDER
DRAG
DRAGICON
TABINDEX
TABSTOP
TAG
LIST
LISTCOUNT
LISTINDEX
*LISTSELCOUNT
*FINDITEM
*FOUNDITEM
*RESET
*SELECTED
*TOPINDEX
*ITEMDATA

iList also has two additional functions, which can be called to get and set multiple list selections in one call, by passing an array to the function. These functions are:

GetSelected()

SetSelected()

The only limitation of the control at this point (that I am aware of) is that the maximum length of string you can add to the list is 1K.

iList Special Properties

LISTSELCOUNT

FINDITEM

FOUNDITEM

SELECTED

TOPINDEX

ITEMDATA

ListSelCount

This property returns the total number of selected items in the listbox.

```
x% = iList.ListSelCount
```

Selected.

This is a boolean property.

iList.Selected(n) returns TRUE if the item at index n is currently selected.

```
x% = iList.Selected(n)
```

The Selected property is both read and write. You can therefore toggle the selected state of a property by:

```
iList.Selected(n) = x%
```

ANY value for x% will cause the selection state to toggle.

Reset

The Reset property is used to clear the listbox of all items. The property simply needs to be set to any number in order to carry out the resetting action.

```
iList.Reset = 1
```


ItemData.

This can be both set and read at runtime.

`iList.ItemData(n) = x&` will set the data item for the listbox entry at index n to the value of x&.

`iList.ItemData(n) = x&`

`x& = iList.ItemData(n)` will read the data item for the listbox entry at index n and set x& to this value.

`x& = iList.ItemData(n)`

TopIndex.

This can be both set and read at runtime.

TopIndex the index of the entry which is at the top of the listbox window currently, or scrolls the listbox so that the selected entry is at the top of the visible list.

`x% = iList.TopIndex`

`iList.TopIndex = x%`

FindItem.

FindItem will match the index of the first entry in the listbox to match the supplied string. The match is on the basis of the length of the supplied string - i.e. supplying "abc" will match an entry of "abcdef". FindItem works by setting the .FoundItem property.

```
a$ = "abc"  
iList.FindItem = a$
```

FoundItem.

After setting FindItem, FoundItem will reflect the index of the matching entry.

`x% = iList.FoundItem`

If the last time that FindItem was set a matching entry could not be found, FoundItem will return -1

GetSelected Function()

This function allows the retrieval of multiple items in one call. Because an array is passed to the function, and the size is not known until runtime, It is essential that the array is correctly sized before passing it to the function. To find the size of the array check the ListSelCount property. The array will be returned, filled with the index values of the selected entries. The indexes are integer values, and so the array should be dimensioned as an integer array.

The declare is:

```
Declare Function GetSelected% Lib "ilist.vbx" (Ctrl As Control, ByVal arrSize As Integer, arrSel as Integer)
```

To use the functions you need to pass the control name, the size of the array that you are passing, and the FIRST element of the array. The size of the array should be determined by the value of iList.ListSelCount.

eg:

```
Sub Command1_Click()
```

```
    'Get the size of the array
    arrSize% = iList1.ListSelCount
    'ReDim the array to the correct size
    ReDim arrSel%(arrSize%)
    'Call GetSelected() to fill the array
    b% = GetSelected ( iList1, arrSize%, arrSel%(0) )

    For i% = 0 to (arrSize% - 1)
        'Print the index numbers
        Picture1.Print Str$(i%); Str$(arrSel%(i%))
    Next i%
```

```
End Sub
```

This function returns -1 (TRUE) for success, and 0 (FALSE) if a listbox error occurs.

SetSelected() Function

This function allows the setting of multiple items in one call. Because an array is passed to the function, and the size is not known until runtime, It is essential that the array is correctly sized before passing it to the function. To find the size of the array check the ListSelCount property. The indexes are integer values, and so the array should be dimensioned as an integer array.

You must Dim the array, populate it with the index values of the items that you wish to select, then pass the array to the Setselected function. All of the listbox index entries passed in the array will be selected in one action.

The declare is:

```
Declare Function SetSelected% Lib "ilist.vbx" (Ctrl As Control, ByVal arrSize As Integer, arrSel as Integer)
```

To use the function you need to pass the control name, the size of the array that you are passing, and the FIRST element of the array. The size of the array should be determined by the value of iList.ListSelCount.

eg:

```
Sub Command1_Click()
```

```
    'Lets select three items in the listbox  
    'so we need an array holding 3 elements  
    ReDim arrSel%(3)
```

```
    'We will select entries at index 2,5 and 7  
    arrSel%(0) = 2  
    arrSel%(1) = 5  
    arrSel%(2) = 7
```

```
    'Call SetSelected() to select the listbox items  
    b% = SetSelected ( iList1, 3, arrSel%(0) )
```

```
End Sub
```

This function returns -1 (TRUE) for success, and 0 (FALSE) if a listbox error occurs.

