

SBList v2.0 KnowledgeBase

SBList offers many features which are not supported by standard list boxes. Below is a list of topics which should help you to discover what you can do with an SBList. Sample code is provided in most topics.

[How to achieve the fastest results with SBList](#)

[How to clip, wrap, tab and truncate](#)

[How to create a list of check boxes](#)

[How to drag and drop items between two lists](#)

[How to drag and drop multiple list items](#)

[How to reposition list items using drag and drop](#)

[How to use the AddPacket method](#)

[How to use the RightButton property](#)

[How to use the Sorted property](#)

[How to use the SortedOn property](#)

[Problems with pictures?](#)

[Compatibility issues](#)

All code samples have been written and tested using Visual Basic 4.0 32-bit Professional Edition.
Topics devised by Andy Groom. Last revision 16/2/97.

How to achieve the fastest results with SBList

If you have used the SBList demo program you will have seen that SBList can out-perform the standard Visual Basic list box by almost 100%. This is achieved by using the AutoDraw property which gives you control over the drawing of the list box.

A standard list box refreshes itself automatically after you add or remove an item, and so does SBList when the AutoDraw property is set to true, which is the default setting. However, the nature of a list box means that typically you would add more than just one item to the list at a time, and so there is little point in refreshing the list until it contains all the items that you want to add.

By setting the AutoDraw property to false before you add or remove list items you are allowing SBList to update its contents without having to refresh after each change. For only a few items this doesn't make much difference to the speed, but for longer lists the change is dramatic. Also, because SBList offers you the ability to include images and font changes on each line, it does take slightly longer to populate a list if you take advantage of these features. The AutoDraw property minimises this delay.

The code below will add 224 items to a list taking advantage of the AutoDraw feature:

```
SBList1.AutoDraw = False
For A% = 32 to 256
    SBList1.AddItem "Char " & A% & " is a " & Chr$(A%)
Next A%
SBList1.AutoDraw = True
```

This gives you powerful control over your list. For example, you might have a list showing the results of a database search. When the user starts a new search you can set AutoDraw to false, clear your SBList and add the new results to it while the previous search information remains visible in the list. Once the new search is complete, just update the list by setting AutoDraw back to true. The example below demonstrates this principle:

```
Form1_Click()
    Static B%
    SBList1.AutoDraw = False
    SBList1.Clear
    For A% = 1 to 500
        SBList1.AddItem "This is item " & (A% + B%)
    Next A%
    B% = B% + A%
    SBList1.AutoDraw = True
End Sub
```

Each time you click the form, the list is updated but only refreshed once the updates are completed.

When the visible property of an SBList is set to false, the AutoDraw feature is ignored and the list does not refresh until it is made visible.

Note: The standard Refresh method does not redraw the list items when AutoDraw is set to false; you must use the AutoDraw method as shown above.

How to use the RightButton property

Windows95 and NT4 are much improved with the use of popup menus, and SBList has a RightButton property to help you take advantage of this feature. A standard list box does not react to right-mouse clicks in the same way it does to left-mouse clicks, so you must first select a list item by left-clicking it, and then right-click it to pop up a menu. SBList allows you to select list items with the right mouse button, which makes life much easier!

When you show a popup menu, it is a Windows convention that any irrelevant menu items are hidden in order to make the menu as compact as possible. To do this, you need to know whether the user has right-clicked on a list item or in the space which follows the last item in the list. To do this, you can use this formula:

```
Sub SBList1_MouseUp(Button%, Shift%, X&, Y&)*
  If ((Y& / Screen.TwipsPerPixelY) > ((SBList1.ListCount -
    SBList1.TopIndex) * SBList1.ItemHeight)) then
    ' We have not clicked on a list item
  Else
    ' We have clicked on a list item
  End If
End Sub
```

The formula assumes that you have set the ItemHeight property, which fixes the pixel height of each list item. The default value is 0, so you need to make sure that it has been set or this formula will not work. As a guideline, MS Sans Serif at 8.25 point has an ItemHeight of about 14.

Note: If the user right-clicks on the space at the bottom of a list instead of on a list item, it does not invalidate the current ListIndex. This can be potentially confusing, because the user might assume that the menu which appears ought to be relevant to the selected item, when in fact it won't be. To avoid this, you could set the ListIndex to -1 as the first line of code after a right-click on empty space is detected.

(* Data type declarations have been used to shorten this line)

How to drag and drop items between two lists

"Drag and drop" is a standard Windows feature which most programs support in some form or other. Dragging items from one list into another list is not particularly an SBList innovation - the code shown below will work equally well with a standard list box.

For this demonstration you will need to draw a form which has an SBList control array of two list boxes called SBList1(0) and SBList1(1).



Copy this bitmap (somehow!) into SDKPaint as a cursor and set the magenta pixels as being the screen colour. Remember to set the hotspot at the tip of the pointer arrow. Save the image as a cursor and then load it as the DragIcon image for your two SBLists.

1. Insert the following code into your **General Declarations** section:

```
Dim OrigX As Integer
Dim OrigY As Integer
Dim OrigIndex As Integer
```

These variables will be used to keep track of where the drag began and which list item we are dragging.

2. Insert the following code into your **Form_Load** event:

```
For A% = 1 To 10
    SBList1(0).AddItem "List item " & A%
Next A%
```

This is just to give us some items to drag around.

3. Insert the following code into your **SBList1_MouseDown** event:

```
OrigX = X
OrigY = Y
OrigIndex = SBList1(Index).ListIndex
```

We are storing the X and Y position of the mouse so that we can measure how far it has moved in the MouseMove event, and only start the drag operation when it's clear that the user is dragging rather than just clicking badly.

4. Insert the following code in your **SBList1_MouseMove** event:

```
If (Button And 1) = 1 Then
    If Abs(OrigX - X) > 45 Or Abs(OrigY - Y) > 45 Then
        ' Drag this list item...
        SBList1(Index).ListIndex = OrigIndex
        SBList1(Index).Drag
    End If
End If
```

The user has the mouse button held down, and in the second line we are measuring how far it has moved since it was held down. The current tolerance is 45 twips in any direction (roughly 3 pixels). Once we are happy that the user is dragging, we set the list index to which ever item the MouseDown event was fired on (the mouse may have strayed onto

the item above or below the original choice) and commence dragging.

5. Insert the following code in your **SBList1_DragDrop** event:

```
If Source.hWnd <> SBList1(Index).hWnd Then
    SBList1(Index).AddItem Source.List(OrigIndex)
    Source.RemoveItem OrigIndex
Else
    Beep
End If
Source.Drag 0
```

This is the final step of the operation. If the item has been dropped into the same list as it was dragged from, we beep, otherwise we copy the item across and delete it from the source list.

This technique only works for dragging single list items. It's more complicated to drag multiple list items (like you can do in Explorer) but it is possible. This can only be done with an SBList, though. The general principle behind the code above can also be enhanced to allow the user to move list items around within the same list. Click the topics below for more information on either of these techniques.

[How to drag and drop multiple list items](#)

[How to reposition list items using drag and drop](#)

How to drag and drop multiple list items

If you cast your mind back to Windows 3.1, you will probably remember File Manager. Dinosaur that it was, it did introduce one cool feature which was at that time new to Windows - multiple item drag and drop. You could select a range of files from the file list and then drag them into a different subdirectory or drive.

You can't achieve this with a standard list box because as soon as you highlight a range of list items and then try to begin a drag operation, all the selected items are cleared. SBList has an extra MultiSelect property to allow you to create a multiple item drag and drop list.

For this demonstration you will need to draw a form which has an SBList control array of two list boxes called SBList1(0) and SBList1(1). Set the MultiSelect property of both lists to setting "**3 - Special**".



Copy this bitmap (somehow!) into SDKPaint as a cursor and set the magenta pixels as being the screen colour. Remember to set the hotspot at the tip of the pointer arrow. Save the image as a cursor and then load it as the DragIcon image for your two SBLists.

1. Insert the following code into your **General Declarations** section:

```
Dim OrigX As Integer
Dim OrigY As Integer
```

These variables will be used to keep track of where the drag began.

2. Insert the following code into your **Form_Load** event:

```
For A% = 1 To 10
    SBList1(0).AddItem "List item " & A%
Next A%
```

This is just to give us some items to drag around.

3. Insert the following code into your **SBList1_MouseDown** event:

```
OrigX = X
OrigY = Y
```

We are storing the X and Y position of the mouse so that we can measure how far it has moved in the MouseMove event, and only start the drag operation when it's clear that the user is dragging rather than just clicking badly.

4. Insert the following code in your **SBList1_MouseMove** event:

```
If (Button And 1) = 1 Then
    If Abs(OrigX - X) > 45 Or Abs(OrigY - Y) > 45 Then
        ' Begin the Drag operation...
        SBList1(Index).Drag
    End If
End If
```

The user has the mouse button held down, and in the second line we are measuring how

far it has moved since it was held down. The current tolerance is 45 twips in any direction (roughly 3 pixels). Once we are happy that the user is dragging we commence the drag operation.

5. Insert the following code in your **SBList1_DragDrop** event:

```
If Source.hWnd <> SBList1(Index).hWnd Then
    Source.AutoDraw = False
    SBList1(Index).AutoDraw = False
    ' Move the selected items across...
    Do While Source.SelCount
        If Source.Selected(A&) = True Then
            SBList(Index).AddItem Source.List(A&)
            Source.RemoveItem A&
        Else
            A& = A& + 1
        End If
    Loop
    Source.AutoDraw = True
    SBList1(Index).AutoDraw = True
Else
    Beep
End If
Source.Drag 0
```

This is the final step of the operation. If the item has been dropped into the same list as it was dragged from, we beep. Otherwise, we go through each item in the source list and copy across any selected items into the target list. When there are no more selected items we exit the loop. We turn off AutoDraw before we start moving items around so that (a) the operation is faster, and (b) the lists update simultaneously.

How to reposition list items using drag and drop

It's much easier for the user to set the order of list items by dragging them into position rather than providing clumsy up and down buttons to shift things around. For example, if you have a user who wants to customise the order in which a list of database fields are displayed on screen, then you could create a list containing those fields and simply tell the user to drag them around until they're in the desired order.

For this demonstration you will need to draw a form which has a single SBList on it. Set the ItemHeight property of the list to 16.



Copy this bitmap (somehow!) into SDKPaint as a cursor and set the magenta pixels as being the screen colour. Set the hotspot anywhere on the top edge of the rectangular block. Save the image as a cursor and then load it as the DragIcon image for your SBList.

1. Insert the following code into your **General Declarations** section:

```
Dim OrigX As Integer
Dim OrigY As Integer
Dim OrigIndex As Integer
```

These variables will be used to keep track of where the drag began and which list item we are dragging.

2. Insert the following code into your **Form_Load** event:

```
For A% = 1 To 10
    SBList1.AddItem "List item " & A%
Next A%
```

This is just to give us some items to drag around.

3. Insert the following code into your **SBList1_MouseDown** event:

```
OrigX = X
OrigY = Y
OrigIndex = SBList1.ListIndex
```

We are storing the X and Y position of the mouse so that we can measure how far it has moved in the MouseMove event, and only start the drag operation when it's clear that the user is dragging rather than just clicking badly.

4. Insert the following code in your **SBList1_MouseMove** event:

```
If (Button And 1) = 1 Then
    If Abs(OrigX - X) > 45 Or Abs(OrigY - Y) > 45 Then
        ' Drag this list item...
        SBList1.ListIndex = OrigIndex
        SBList1.Drag
    End If
End If
```

The user has the mouse button held down, and in the second line we are measuring how far it has moved since it was held down. The current tolerance is 45 twips in any direction

(roughly 3 pixels). Once we are happy that the user is dragging, we set the list index to which ever item the MouseDown event was fired on (the mouse may have strayed onto the item above or below the original choice) and commence dragging.

5. Insert the following code in your **SBList1_DragOver** event:

```
NewIndex = Int(SBList1.TopIndex + ((Y / Screen.TwipsPerPixelY) /  
SBList1.ItemHeight))  
If NewIndex < SBList(Index).ListCount Then SBList1.ListIndex = NewIndex
```

We calculate which list item the mouse is over in the first line of code, and select it in the second line (provided that it exists). This is so that as you drag the mouse around, the list item which you are going to insert before is highlighted and the user doesn't have to guess where the dragged list item will be dropped.

6. Insert the following code in your **SBList1_DragDrop** event:

```
N$ = Source.List(OrigIndex)  
LI% = Source.ListIndex  
If LI% > OrigIndex Then  
    Source.RemoveItem OrigIndex  
    Source.AddItem N$, LI% - 1  
ElseIf LI% < OrigIndex Then  
    Source.RemoveItem OrigIndex  
    Source.AddItem N$, LI%  
End If  
Source.ListIndex = LI%
```

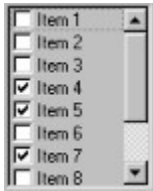
In this code we remove the item which was dragged and re-add it to the list in the relevant position. If the dragged item is being dropped lower down the list than it started, we need to take that into account when we remove it and re-add it, because after we remove it all the list items will have shifted up one place. If it was dropped above itself this isn't a concern. If the item was dropped on itself, we simply ignore it.

In the last line of code we select the item which we originally dragged.

How to create a list of check boxes

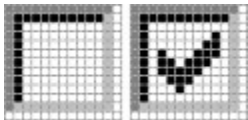
Once you have begun to experiment with SBList you will come up with new and imaginative ways of using it - one of the first may well be creating a list of check boxes. Standard check boxes are very useful in situations where you will always have the same number of options, but there are situations where you simply don't know how many you might need.

For example, suppose you wanted to display a list of users and tick the ones which had administrator privileges. There could be 5 or 500 users; you have no way of knowing. You could create a control array of check boxes and load as many new check boxes as required, but you may end up needing a huge form just to show them all! A far easier solution is to use an SBList to simulate an array of check boxes.



This is how your list might appear. Each entry in the list looks like a normal check box and will act like a normal check box when clicked, but there is virtually no limit to the number of boxes you can display.

For this demonstration you will need to draw a form which has a single SBList on it. Set the ItemHeight property of the list to 15, set the BackColor to light-grey, and set the TextLeft property to 18. You will also need a control array of two Image controls - Image1(0) will hold the un-ticked box, Image1(1) will hold the ticked box (shown below).



We will be using the ItemData property of each list item to hold the ticked status - true or false.

1. Insert the following code into your **Form_Load** event:

```
For A% = 1 To 10
    SBList1.AddPacket -1, "Item " & A%, "", , Image1(0).Picture
Next A%
```

This will place 10 items in the list, all with empty tick boxes. The ItemData parameter is not specified in the AddPacket statement, and so the default value of 0 is used. This is fine for us, because 0 (or False) indicates an empty tick box.

2. Insert the following code into your **SBList1_DbClick** event:

```
With SBList1
    NewSetting% = Not .ItemData(.ListIndex)
    Set .ItemImage(.ListIndex) = Image1(-NewSetting%).Picture
    .ItemData(.ListIndex) = NewSetting%
End With
```

On the first line we toggle the ItemData value for the list item which has been double-clicked. We then change the image in the list to reflect the new setting. Finally, we write the toggle value back to the ItemData property.

3. Insert the following code into your **SBList1_Click** event:

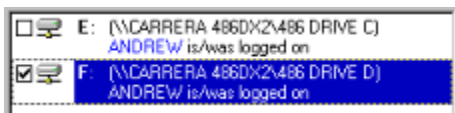
```
If X <> -1 and Y <> -1 then SBList1_DblClick 1
```

Here we test whether the bitmap itself has been clicked, and if it has we react as though a double-click had taken place. The SBList Click event returns the X,Y position of the cursor if it is clicked on the bitmap element of a list item, and it returns -1 for both values if the bitmap element of the line was not clicked.

This bit of code is optional, depending on whether you feel it is logical or not. The idea behind it is that in order to change the tick mark the user must either double-click the text of the list item, **or** single-click the bitmap part of the list item.

You can't use the normal Click event to toggle the tick mark, otherwise it would change state whenever you select any list item with just a normal click, which is why we have used double-click in the example.

To give you another idea of what is possible, below is a screen-shot of an SBList as it is used in one of my programs for choosing which machines on a network can have access to a particular file. I've used the HighLeft property to prevent the highlight bar from intruding onto the image, and also various tab and font settings to enhance the layout of the information which the list is displaying.



Tweaks

There are a couple of ways in which you can tweak this routine depending on how much like a list of check boxes you want your list to appear. For example:

You can set the HighForeColor and HighBackColor properties to the same colours as the ForeColor and BackColor properties so that the highlight bar does not show up.

You can set the list Border style to False so that there is no obvious outline to the list.

You can set the HighLeft property to specify an indent from the left-hand side of the list where the highlight bar begins.

Those who seek perfection will have noticed that the list's BackColor has to be set to light grey in order for this example to appear correctly. To avoid this limitation you can use an ImageList control to hold the ticked and unticked images and then mask them against the default background color of the list.

How to use the Sorted property

The Sorted property of an SBList can be changed at runtime, something which a standard list box does not offer. This makes displaying list items exactly how you want them much easier.

For example, suppose you wanted to show a list of users in your program, sorted into alphabetical order, but have a special user called "Visitor" which allowed access into the program but with reduced privileges. It would be much clearer if this Visitor name appeared at the top of the list, with the names listed alphabetically below, but how would you normally program that? The easiest way would be to have a sorted list box hidden away on the form somewhere; dump the users into it so that they get sorted, then copy them into your visible list and add the "Visitor" name at the top? Sound familiar? Here's how to do it with just one SBList:

For this demonstration you will need to draw a form which has a single SBList on it.

1. Insert the following code into your **Form_Load** event:

```
Names$ = "Fred, Steve, Andy, Cindy, Clive, Sandra, George, "  
SBList1.Sorted = True  
Do While Names$ <> ""  
    P& = Instr(Names$, ",")  
    SBList1.AddItem Left$(Names$, P& - 1)  
    Names$ = Mid$(Names$, P& + 1)  
Loop  
SBList1.Sorted = False  
' Add this item at the top of the list...  
SBList1.AddItem "Visitor", 0
```

As you can see, once the Sorted property is turned off (at the end of the code) you can add any items to the list exactly where you want them.

If you turn the Sorted property on and off, adding items here and there, the final sort order will probably not be what you expected. Once you have turned off sorting and added more items where you wanted them to go, turning Sorted back on and adding more items may not place them in the order you expected. In fairness, you would be missing the objective behind a controllable Sorted property:

Normally, when you draw a list box on a form you have to choose whether it will be sorted or not, and then you're stuck with it. The Sorted property of the SBList is changeable but the intention was that you would only alter the setting when the list is empty. In other words, you would decide at runtime whether you wanted to display items sorted or in their original order, and set the property accordingly before populating the list.

The sample code above shows you how to take advantage of this feature to give you control over the order of list items, and you may discover even further refinements, but be warned that changing the Sorted property too often may give you sleepless nights!

How to use the SortedOn property

The SortedOn property gives you control over the criteria which SBList uses to sort list items when you have the Sorted property set to true. You can base your sort on the text which appears in the list (just like a standard list box), or on the .ItemText value for each list item, or on the .ItemData value for each item.

A good way of visualising this feature is to think about a list containing all the files in your Windows directory. The list always has the same columns of information; File name, Size, and Date. When you add the files to a standard list box, the only way to sort them would be on File name, unless you displayed the columns in a different order so that Size was the first column, and then they would appear in order of size. There would be no way of keeping the columns in the same order because the sort method only works on the text of each list item. With an SBList you can sort your items on the .ItemText value, which is transparent to the user, and always have your columns in the same order.

For this demonstration you will need to draw a form which has a single SBList on it. Set the Sorted property to true. You will also need a control array of three command buttons - set the caption of Command1(0) to "By Name", Command1(1) to "By Size" and Command1(2) to "By Extension".

1. Insert the following code into your **Form_Load** event:

```
With SBList1
    .TabWidth(0) = 100
    .TabWidth(1) = 140
End With
```

This sets up our list columns. The first column (from pixel 0 to 99) will hold the file name less the file extension. The second column (from pixel 100 to 139) will hold the file extension, and the third column (from pixel 140 onwards) will hold the file size.

2. Insert the following code into your **Command1_Click** event:

```
SBList1.Clear
' When Index is 0, we sort on Common$ (.List)
' When Index is 1, we sort on Size& (.ItemData)
' When Index is 2, we sort on Ext$ (.ItemText)
SBList1.SortedOn = Index

P$ = "C:\"
D$ = Dir$(P$ & "*..*")
Do While D$ <> ""
    ' Does the current file have an extension?
    If Instr(D$, ".") Then
        Ext$ = Mid$(D$, Instr(D$, ".") + 1)
        File$ = Left$(D$, Instr(D$, ".") - 1)
    Else
        Ext$ = ""
        File$ = D$
    End If
    Size& = FileLen(P$ & D$)
    ' The \t code represents the Tab character
    Common$ = File$ & "\t" & Ext$ & "\t" & Size&
    SBList1.AddPacket -1, Common$, Ext$, Size&
```

```
D$ = Dir$  
Loop
```

When you run this program and click the three buttons, the list will fill up sorted on either the file name, extension or size. The point here is that (a) we didn't have to write three routines to fill the list in three different ways, (b) we didn't have to change the order of the columns in order to display the list items in a different order, and (c) we didn't need two lists; one list to sort the items and a second list to display them.

Note: When you sort on the .ItemData value, the list is sorted into correct numeric order (1, 2, 3, 4 ...) as opposed to ANSI sort order (1, 10, 11, 2 ...).

How to use the AddPacket method

The AddPacket method was borne out of laziness, and simply provides a code-efficient way of adding a list item. The two code samples below both achieve the same result:

```
SBList1.AddItem "Hi!"  
SBList1.ItemText(SBList1.NewIndex) = "XYZ"  
SBList1.ItemData(SBList1.NewIndex) = 123  
SBList1.ItemPicture(SBList1.NewIndex) = Pic1.Picture  
SBList1.ItemBackColor(SBList1.NewIndex) = QBColor(12)
```

is the same as:

```
SBList1.AddPacket -1, "Hi!", "XYZ", 123, Pic1.Picture, QBColor(12)
```

In short, AddPacket saves you having to reference the most recent list item (NewIndex) when you want to set the most commonly used item values. The first three parameters of AddPacket are the minimum requirement; after all, if you only want to set the list text and position you might as well just use AddItem. An easy mistake to make is to accidentally omit the ItemText value (the third value in the sequence) when you are specifying the picture value. For example:

```
SBList1.AddPacket -1, "Hi!", , , Pic1.Image
```

will generate an "Argument not optional" error message at design time. The correct syntax is:

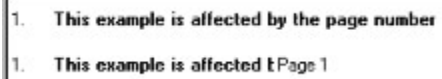
```
SBList1.AddPacket -1, "Hi!", "", , Pic1.Image
```

You can also use indirect references in place of standard references for the Picture element of AddPacket. For example:

```
SBList1.AddPacket -1, "Hi!", "XYZ", 123, imagelist1.ListImages(5)
```

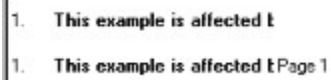
However, you cannot link an ImageList control to an SBList in the same way as you can with the Microsoft TreeView control, for example. You must use an indirect reference as shown above.

*AutoClip = **True**, ClipText = False*

- 
- 1. This example is affected by the page number
 - 1. This example is affected tPage 1

With AutoClip set to true, the addition of the page number in the second line prevents the text from continuing, so over-printing is avoided.

*AutoClip = False, ClipText = **True***

- 
- 1. This example is affected t
 - 1. This example is affected tPage 1

With ClipText set to true, the 'intelligence' is taken out of clipping so that text is truncated whether it would cause over-printing or not.

The Ellipsis property can tidy up clipping by truncating a word to fit as many complete characters as possible in the space available (so that you don't get half a letter as above), and then adds three periods (...) to show that truncation has taken place.

Problems with pictures?

Enter the sample code below. You will need to draw an SBList on your form and set its TextLeft property to 15. You will also need a Picture box 210x210 twips in size, with its **ScaleMode** property set to 3 (pixels) and its **AutoRedraw** property set to True.

1. Insert the following code into your **Form_Click** event:

```
For A% = 0 To 15
    Picture1.Line (0, 0)-(10, 10), QBColor(A%), BF
    SBList1.AddPacket -1, "QBColor" & A%, "", , Picture1.Image
Next A%
```

The logic of it should produce a list of 16 items showing the QBColor shade and value of each item when you click the form:



However, as you will see when you run it you actually end up with a list where all the images are white. To see exactly what is going on, modify the code above to read:

```
For A% = 0 To 15
    Picture1.Line (0, 0)-(10, 10), QBColor(A%), BF
    SBList1.AddPacket -1, "QBColor" & A%, "", , Picture1.Image
    T# = Timer + .5
    Do While Timer < T#
        DoEvents
    Loop
Next A%
```

Now run the program again, and you will see that the picture box is in fact changing colour correctly, but after each change it affects all the images already displayed in the list. When this feature was discovered we decided to leave it in just in case it had some useful application! Besides, it's easy to program around it - simply add the line:

```
Picture1.Cls
```

at the beginning of the loop and the problem is solved.

Why does this happen? When you add a picture to SBList, the picture is linked to the list item by a reference number. When you perform a *destructive operation* on the source picture (like Cls or LoadPicture), the image and reference number are frozen and stored in memory so that your list image doesn't change when the source image is destroyed. In the first block of code above, the same reference number is being used for all 16 list items because the Line operation is not a destructive one - if it was, you could only ever draw one line on a picture control!

Compatibility issues

Can you replace your existing list boxes with SBLists? Of course you can, but first you need to check that you have considered the points below.

Messages

SBList reacts to standard list messages like `lb_FindStringExact` only while it is visible, but ignores them if it is hidden (its `Visible` property is set to `False`).

The following messages can be replaced as shown:

```
Li& = SendMessage(SBList1.hWnd, lb_FindString, -1, F$)
```

becomes

```
Li& = SBList1.FindString(F$)
```

```
Li& = SendMessage(SBList1.hWnd, lb_FindString, 6, F$)
```

becomes

```
Li& = SBList1.FindString(F$, 6)
```

```
Li& = SendMessage(SBList1.hWnd, lb_FindStringExact, 2, F$)
```

becomes

```
Li& = SBList1.FindString(F$, 2, 1)
```

See [FindString](#) for more information.

Methods

All of the methods for this control are listed in the following table.

<u>AddItem</u>	<u>AddPacket</u>	<u>AddPicture</u>	<u>Clear</u>
<u>Drag</u>	<u>FindData</u>	<u>FindString</u>	<u>LoadList</u>
<u>Refresh</u>	<u>RemoveItem</u>	<u>SaveList</u>	<u>SelectAll</u>
<u>SelectNone</u>	<u>SetFocus</u>		

AddItem Method

[See Also](#)

[Examples](#)

Adds an item to the list box.

Syntax

SBList1.AddItem *item*, [*index*]

item Required. String expression specifying the item to add to the list box.

Index Optional. Long integer specifying the position within the list box where the new item is placed. For the first item in a list box, index is 0.

Remarks

If you supply a valid value for *index*, *item* is placed at that position within the list box. If *index* is omitted, *item* is added at the proper sorted position (if the **Sorted** property is set to True) or to the end of the list (if **Sorted** is set to False). If the Sorted property is set to True the *index* property is ignored. Note that unlike a standard list box, the **Sorted** property may be changed at run time.

If you are entering many items in succession, faster loading will be achieved by setting **AutoDraw** to False before using **AddItem**, then set **AutoDraw** to True afterwards. If you do not use **AutoDraw** the list box will attempt to draw each item as it is added.

Examples

```
SBList1.AddItem "France"
```

```
SBList1.AddItem "Italy", 5
```

See Also

[AddItem Method](#)

[AddPicture Method](#)

[AutoDraw Property](#)

[Clear Method](#)

[RemoveItem Method](#)

[Sorted Property](#)

AddPacket Method

[See Also](#)

[Examples](#)

[Knowledge Base](#)

Adds an item to the list box, complete with all extra attributes.

Syntax

SBList1.**AddPacket** *index*, *item*, *itemtext*, [*itemdata*], [*pic*], [*color*]

index Required. Long integer specifying the position within the list box where the new item is placed. For the first item in a list box, *index* is 0. To place an item at the end of the list set *index* to -1.

item Required. String expression specifying the item to add to the list box.

itemtext Required. String expression placed in item's associated **ItemText** value.

itemdata Optional. Long numerical value placed in item's associated **ItemData** value.

pic Optional. Picture to be displayed with item.

color Optional. Color value used as background for item, stored as **ItemBackColor**.

Remarks

If the Sorted property is set to True the *index* property is ignored. Note that unlike a standard list box, the Sorted property may be changed at run time.

If you are entering many items in succession, faster loading will be achieved by setting **AutoDraw** to False before using **AddItem**, then set **AutoDraw** to True afterwards. If you do not use **AutoDraw** the list box will attempt to draw each item as it is added.

See Also

[AddPacket Method](#)

[AddPicture Method](#)

[AutoDraw Property](#)

[Clear Method](#)

[RemoveItem Method](#)

[Sorted Property](#)

Examples

SBList1.AddPacket -1, "Germany", "Berlin"

SBList1.AddPacket 23, "Italy", "Rome", 2435, , RGB(255, 0, 0)

SBList1.AddPacket 0, "England", "London", 1, Image1.Picture

SBList1.AddPacket -1, "Zaire", "", , , RGB(0, 255, 0)

SBList1.AddPacket 5, "USA", "Washington", 1776, Image1.Picture, &H8000000F&

Knowledge Base

[How to create a list of check boxes](#)

[How to use the AddPacket method](#)

[Problems with pictures ?](#)

AddPicture Method

[See Also](#) [Examples](#) [Knowledge Base](#)

Adds a new item consisting of only a picture.

Syntax

SBList1.AddPicture *picture* [, *index*]

picture A picture

index Optional. Integer value

Remarks

This method will add a new item to the list box consisting of just the picture specified with the *picture* value. The item will be inserted at the position specified with *index*, or if omitted the item will be added to the end of the list. If *index* is set to -1 the picture will also be added to the end of the list.

See Also

[AddItem Method](#)

[AddPacket Method](#)

[ItemPicture Property](#)

[PicLeft Property](#)

[PicTop Property](#)

Examples

```
SBList1.AddPicture Picture1.Picture  
SBList2.AddPicture LoadPicture("c:\mypic.bmp"), 6  
SBList1.AddPicture frmGraphics.Image1.Picture, -1
```

Clear Method

[See Also](#) [Example](#)

Removes all items from the list box.

Syntax

SBList1.Clear

Remarks

If the **AutoDraw** property is set to False, the items will not appear to have been deleted although in fact they will have been. No update will occur until **AutoDraw** is set back to True.

See Also

[AutoDraw Property](#)

Example

SBList2.Clear

Drag Method

[See Also](#) [Knowledge Base](#)

Begins, ends, or cancels a drag operation.

Syntax

SBList1.Drag *action*

action Optional. A constant or value that specifies the action to perform, as described in Settings. If action is omitted, the default is to begin dragging the object.

Remarks

The settings for action are:

<u>Constant</u>	<u>Value</u>	<u>Description</u>
vbCancel	0	Cancels drag operation.
vbBeginDrag	1	Begins dragging object.
vbEndDrag	2	Ends dragging and drop object.

See Also

[DragDrop Event](#)

[DragIcon Property](#)

[DragOver Event](#)

FindData Method

[See Also](#)

[Examples](#)

Returns index of list item with matching item data value.

Syntax

SBList1.FindData *value* [, *index*]

value A long value to search on.

index Optional. Index of item to begin searching from.

Remarks

This method searches through all the list items, starting at the position specified by *index*, until the item data value of an item matches the number in *value*. Once the search reaches the end of the list box, searching will resume at index 0 until all items have been searched. The number returned is the index of the matching item, or -1 if no match was found.

See Also

[AddPacket Method](#)

[FindString Method](#)

[ItemData Property](#)

Examples

```
ndx& = SBList1.FindData 1997, 45  
i% = SBList1.FindData 56633
```

FindString Method

[See Also](#)

[Examples](#)

Returns index of item matching string search criteria.

Syntax

SBList1.FindString *value* [, *mode*] [, *index*]

value String value to perform search on.

mode Optional integer.Type of search, default is 1.

index Optional. Index of item to begin searching from.

Remarks

This method searches through all the list items, starting at the position specified by *index*, until the list text or item text value of an item matches the string in *value*. Once the search reaches the end of the list box, searching will resume at index 0 until all items have been searched. The number returned is the index of the matching item, or -1 if no match was found. The searching is not case sensitive, and ignores any escape codes.

The type of search is specified with the *mode* parameter :

Constant	Value	Description
sbFindListTextStart	0	Searches the list text for a string that starts with <i>value</i> .
sbFindListTextExact	1	Searches the list text for a string that exactly matches <i>value</i> .
sbFindListTextWild	2	Searches the list text for a string that contains <i>value</i> within it.
sbFindItemTextStart	4	Searches the item text for a string that starts with <i>value</i> .
sbFindItemTextExact	5	Searches the item text for a string that exactly matches <i>value</i> .
sbFindItemTextWild	6	Searches the item text for a string that contains <i>value</i> within it.

See Also

[AddPacket Method](#)

[FindData Method](#)

[ItemText Method](#)

Examples

```
i% = SBList1.FindString "Belgium", 1, 34  
x& = SBList1.FindString "tion", 2  
j& = SBList1.FindString "Cairo", 4  
h& = SBList1.FindString "Canada", , 99
```

```
i% = SBList1.FindString "Belgium", sbFindListTextExact, 34  
x& = SBList1.FindString "tion", sbFindListTextWild  
j& = SBList1.FindString "Cairo", sbFindItemTextStart
```

LoadList Method

[See Also](#)

[Examples](#)

Loads list box contents from a file

Syntax

SBList1.LoadList *filename*

filename A file name.

Remarks

Loads a list box with the contents previously saved using the **SaveList** method. As well as the actual list item the item data, item text and background color for each list item is also loaded. The list box is cleared before loading, and the list is loaded in the same order as it was previously saved with, regardless of the **Sorted** property.

See Also

[SaveList Method](#)

Examples

```
SBList1.LoadList "c:\mylist.lst"
```

```
FrmMain.SBList2.LoadList "appdata.lst"
```

Refresh Method

Forces a complete repaint of a list box.

Syntax

SBList1.Refresh

Remarks

Generally, painting a list box is handled automatically while no events are occurring. However, there may be situations where you want the list box updated immediately.

RemoveItem Method

[See Also](#)

[Examples](#)

Removes a single item from the list box.

Syntax

SBList1.RemoveItem value

value A number between 0 and **ListCount** – 1

Remarks

Removes an item from a list box. If **AutoDraw** is set to false the list box will not change visually until **AutoDraw** is set back to true

See Also

[AutoDraw Property](#)

[Clear Method](#)

Examples

SBList1.RemoveItem 10

SaveList Method

[See Also](#) [Examples](#)

Saves list box contents to file.

Syntax

SBList1.**SaveList** *filename*

Filename A file name.

Remarks

Saves the complete contents of a list box, including each items item data, item text and back color attributes. The **LoadList** method is used to load data back to the list box.

See Also
[LoadList Method](#)

Examples

```
SBList1.SaveList "c:\appdata.lst"
```

SelectAll Method

[See Also](#)

Selects all items.

Syntax

SBList1.**SelectAll**

Remarks

Selects every item in the list box. Only available if **MultiSelect** is not set to 0-None.

See Also

[MultiSelect Property](#)

[Selected Property](#)

[SelectNone Method](#)

SelectNone Method

[See Also](#)

De-selects all items

Syntax

SBList1.**SelectNone**

Remarks

Removes selection from all items in a list box.

See Also

[MultiSelect Property](#)

[SelectAll Method](#)

[Selected Property](#)

SetFocus Method

[See Also](#)

Moves the focus to the list box

Syntax

SBList1.**SetFocus**

Remarks

The **ShowFocus** property determines if a focus rectangle is drawn or not.

See Also

[GotFocus Event](#)

[LostFocus Event](#)

[ShowFocus Property](#)

Events

All of the events for this control are listed in the following table.

<u>Click</u>	<u>DbClick</u>	<u>DragDrop</u>	<u>DragOver</u>
<u>GotFocus</u>	<u>KeyDown</u>	<u>KeyPress</u>	<u>KeyUp</u>
<u>LostFocus</u>	<u>MouseDown</u>	<u>MouseMove</u>	<u>MouseUp</u>

Click Event

[See Also](#)

Syntax

Private Sub *SBList1_Click* (*[Index As Integer,] ByVal Button As Integer, ByVal X As Integer, ByVal Y As Integer*)

Index An integer that uniquely identifies a control if it's in a control array.

Button Value is 1 for left mouse button, 2 for right. If event was fired by setting the **ListIndex** property, then *Button* is -1.

X The horizontal position of the cursor over an items' image. If the cursor is not over the image, or no image exists, -1 is returned.

Y The vertical position of the cursor over an items' image. If the cursor is not over the image, or no image exists, -1 is returned.

Remarks

This event is fired when :

- A user clicks on an item with the mouse
- The **ListIndex** property is set, provided **FireClick** is True.
- The **Selected** property is changed, provided **FireClick** is True

The *X* and *Y* parameters are only used when an item contains a picture. If a picture exists and the mouse was clicked with the cursor on the picture then the co-ordinates are returned, otherwise they each return -1.

See Also

[FireClick Property](#)

[ListIndex Property](#)

[Selected Property](#)

DbIClick Event

Syntax

Private Sub *SBList1_DbIClick* (*[Index As Integer,]* **ByVal** *Button As Integer*)

Index An integer that uniquely identifies a control if it's in a control array.

Button Value is 1 for left mouse button, 2 for right.

Remarks

This event is fired when the user double clicks a list item.

DragDrop Event

[See Also](#) [Knowledge Base](#)

Occurs when a drag-and-drop operation is completed as a result of dragging a control over a form or control and releasing the mouse button or using the **Drag** method with its action argument set to 2 (Drop).

Syntax

Private Sub *SBList1_DragDrop*(*[Index As Integer,* *Source As Control, X As Single, Y As Single*)

Index An integer that uniquely identifies a control if it's in a control array.

Source The control being dragged. You can include properties and methods with this argument-for example, `Source.Visible = 0`.

X, Y A number that specifies the current horizontal (x) and vertical (y) position of the mouse pointer within the target form or control. These coordinates are always expressed in terms of the target's coordinate system as set by the `ScaleHeight`, `ScaleWidth`, `ScaleLeft`, and `ScaleTop` properties.

Remarks

Use a **DragDrop** event procedure to control what happens after a drag operation is completed. For example, you can move the source control to a new location or copy a file from one location to another.

See Also

[Drag Method](#)

[DragIcon Property](#)

[DragOver Event](#)

DragOver Event

[See Also](#)

Occurs when a drag-and-drop operation is in progress.

Syntax

```
Private Sub SBList1_DragOver([Index As Integer,] Source As Control, X As Single, Y As Single,  
State As Integer)
```

Index

Source The control being dragged. You can refer to properties and methods with this argument-for example, `Source.Visible = False`.

X, Y A number that specifies the current horizontal (x) and vertical (y) position of the mouse pointer within the target form or control. These coordinates are always expressed in terms of the target's coordinate system as set by the `ScaleHeight`, `ScaleWidth`, `ScaleLeft`, and `ScaleTop` properties.

State An integer that corresponds to the transition state of the control being dragged in relation to a target form or control:

0 = Enter (source control is being dragged within the range of a target).

1 = Leave (source control is being dragged out of the range of a target).

2 = Over (source control has moved from one position in the target to another).

Remarks

Use a **DragOver** event procedure to determine what happens after dragging is initiated and before a control drops onto a target.

See Also

[Drag Method](#)

[DragDrop Event](#)

[DragIcon Property](#)

GotFocus Event

[See Also](#)

Occurs when the list box receives the focus, either by user action, such as tabbing to or clicking the object, or by changing the focus in code using the **SetFocus** method

Syntax

```
Private Sub SBList1_GotFocus([Index As Integer])
```

Index An integer that uniquely identifies a control if it's in a control array.

Remarks

By default the list box will display a focus rectangle around an item if the list box has focus. To turn this focus rectangle off use the **ShowFocus** property.

See Also

[LostFocus Event](#)

[SetFocus Method](#)

[ShowFocus Property](#)

KeyDown, KeyUp Events

[See Also](#)

Syntax

Private Sub *SBList1_KeyDown*(*[Index As Integer,] KeyCode As Integer, Shift As Integer*)

Private Sub *SBList1_KeyUp*(*[Index As Integer,] KeyCode As Integer, Shift As Integer*)

Index An integer that uniquely identifies a control if it's in a control array.

KeyCode A key code

Shift An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys at the time of the event. 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. 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 are pressed, the value of shift is 6.

Remarks

For both events, the object with the focus receives all keystrokes.

See Also

[KeyPress Event](#)

KeyPress Event

[See Also](#)

Syntax

Private Sub *SBList1_KeyPress*([*Index As Integer*,] *KeyAscii As Integer*)

Index An integer that uniquely identifies a control if it's in a control array.

KeyAscii An integer that returns a standard numeric ANSI keycode. *Keyascii* is passed by reference; changing it sends a different character to the object. Changing *keyascii* to 0 cancels the keystroke so the object receives no character.

Remarks

Occurs when the user presses and releases an ANSI key.

See Also

[KeyDown Event](#)

[KeyUp Event](#)

LostFocus Event

[See Also](#)

Syntax

Private Sub *SBList1*_**LostFocus**([*Index As Integer*])

Index An integer that uniquely identifies a control if it's in a control array.

Remarks

Occurs when an object loses the focus, either by user action, such as tabbing to or clicking another object, or by changing the focus in code using the **SetFocus** method.

See Also

[GotFocus Event](#)

[SetFocus Method](#)

[ShowFocus Property](#)

MouseDown, MouseUp Events

[See Also](#)

Syntax

Private Sub *SBList1_MouseDown*(*Index As Integer*, *Button As Integer*, *Shift As Integer*, *X As Single*, *Y As Single*)

Private Sub *SBList1_MouseUp*(*Index As Integer*, *Button As Integer*, *Shift As Integer*, *X As Single*, *Y As Single*)

Index An integer that uniquely identifies a control if it's in a control array.

Button Returns an integer that identifies the button that was pressed (**MouseDown**) or released (**MouseUp**) 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 that corresponds to the state of the SHIFT, CTRL, and ALT keys when the button specified in the button argument is pressed or 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 that specifies the current location of the mouse pointer. The x and y values are always expressed in terms of the coordinate system set by the ScaleHeight, ScaleWidth, ScaleLeft, and ScaleTop properties of the object.

Remarks

Use a **MouseDown** or **MouseUp** event procedure to specify actions that will occur when a given mouse button is pressed or released. Unlike the **Click** and **DbClick** events, **MouseDown** and **MouseUp** events enable 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.

See Also

[Click Event](#)

[DbClick Event](#)

[MouseMove Event](#)

MouseMove Event

[See Also](#)

Syntax

Private Sub *SBList1_MouseMove*(*Index As Integer*, *Button As Integer*, *Shift As Integer*, *X As Single*, *Y As Single*)

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

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.

See Also

[MouseDown Event](#)

[MouseUp Event](#)

Text Formatting

Unlike a standard list box, each list box item in SBList may contain text comprised of different font styles and colors, and each list box item may comprise text wrapped onto 2 or more lines. To change the attribute or color, embed an escape character followed by a single character, which determines the style of the following text. These characters are:

B , b	Bold on/off
U , u	Underline on/off
I , i	Italic on/off
S , s	Strikethru on/off
0	Change color to <u>ForeColor</u> .
1 , 2 , 3 , 4	Change color to <u>ForeColor</u> <i>n</i> .
n	Throw a new line, equivalent to Chr\$(10).
t	Tab character, equivalent to Chr\$(9).

Example:

“Hello \2\UWorld” appears as “Hello World”.

If you need to display an escape character then enter 2 consecutive ones, for instance \\ will display as \.

Credits

Special thanks to Andrew Groom for providing the accompanying demonstration program, the knowledge base, and for his invaluable contributions during the development of this control.

Support

If you require any assistance, contact us at:

E-mail: **GlobalCom@pobox.com**

Address: **Global Components
41 Milton Street
Northampton
NN2 7JG
United Kingdom**

WWW: **<http://ds.dial.pipex.com/globalcom/>**

Release notes

2.0 Major speed increase

- New FindData & FindString methods
- New AutoDraw property
- New SortedOn property
- Sorted property can be set at run-time
- ItemHeight can be set at run-time
- New SaveList & LoadList methods
- New MultiSelect property
- New SelCount property
- New Selected property
- New SelectAll & SelectNone properties
- New AutoClip property
- Property pages removed
- Bug fixes

1.2 HighLeft property

- IntegralHeight property
- Border property
- Ellipsis property
- Double escape character displays one escape chr.
- Added \n for new line & \t for tab
- Bug fixes

1.1 MouseIcon and Mousepointer properties added.

- Bug fixes

1.0 First release.

Legal Notice

SBLIST OCX VERSION 2.0 (c) 1997 GLOBAL COMPONENTS

THE SOFTWARE IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND. GLOBAL COMPONENTS DISCLAIMS ALL WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING THE WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

IN NO EVENT SHALL GLOBAL COMPONENTS BE LIABLE FOR ANY DAMAGES WHATSOEVER INCLUDING DIRECT, INDIRECT, INCIDENTAL, CONSEQUENTIAL, LOSS OF BUSINESS PROFITS OR SPECIAL DAMAGES, EVEN IF GLOBAL COMPONENTS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES

Properties

All of the properties for this control are listed in the following table.

<u>Appearance</u>	<u>AutoClip</u>	<u>AutoDraw</u>	<u>BackColor</u>
<u>Border</u>	<u>Cell</u>	<u>CenterV</u>	<u>ClipText</u>
<u>DotsHorizontal</u>	<u>DotsVertical</u>	<u>DragIcon</u>	<u>DragMode</u>
<u>Ellipsis</u>	<u>Enabled</u>	<u>EscapeChr</u>	<u>FireClick</u>
<u>Font</u>	<u>ForeColor</u>	<u>ForeColor1</u>	<u>ForeColor2</u>
<u>ForeColor3</u>	<u>ForeColor4</u>	<u>Height</u>	<u>HelpContextID</u>
<u>HighBackColor</u>	<u>HighForeColor</u>	<u>HighLeft</u>	<u>hWnd</u>
<u>Index</u>	<u>IntegralHeight</u>	<u>ItemBackColor</u>	<u>ItemData</u>
<u>ItemHeight</u>	<u>ItemPicture</u>	<u>ItemText</u>	<u>Left</u>
<u>LineColor</u>	<u>LineHorizontal</u>	<u>LineVertical</u>	<u>List</u>
<u>ListCount</u>	<u>ListIndex</u>	<u>ListRaw</u>	<u>MouseIcon</u>
<u>MousePointer</u>	<u>MultiSelect</u>	<u>Name</u>	<u>NewIndex</u>
<u>PicLeft</u>	<u>PicTop</u>	<u>RightButton</u>	<u>SelCount</u>
<u>Selected</u>	<u>ShowFocus</u>	<u>Sorted</u>	<u>SortedOn</u>
<u>TabDots</u>	<u>TabIndex</u>	<u>TabLines</u>	<u>TabStop</u>
<u>TabWidth</u>	<u>Tag</u>	<u>Text</u>	<u>TextLeft</u>
<u>TextTop</u>	<u>Top</u>	<u>TopIndex</u>	<u>Visible</u>
<u>WhatsThisHelpID</u>	<u>Width</u>		

ListIndex is the default value of the control.

Appearance Property

[See Also](#)

[Examples](#)

Returns or sets the paint style of the control.

Syntax

SBList1.**Appearance** = *value*

value 0 (flat) or 1 (3D). Paints control with specified effects.

Remarks

If *value* is set to 1 the list box will appear with a 3D effect, and the **Border** property will be ignored. Unlike the standard list box, this property may be set at run-time. The default value is 1-3D.

See Also

[Border Property](#)

Examples

SBList1.Appearance = 1
i% = SBList2.Appearance

AutoClip Property

[See Also](#)

[Examples](#)

[Knowledge Base](#)

Performs intelligent column clipping

Syntax

SBList1.AutoClip = *value*

value True or False.

Remarks

If *value* is set to True the control will truncate text at each tab position defined with the **TabWidth** property only if there is text in the next 'cell'. If there is no text in the next cell, the current text will be allowed to over run into it without truncation. To specify that no text exists in the next cell, place 2 tab characters (or \t twice) in the text, or an end-of-line character (or \n). When set to True, the **ClipText** and **TabDots** properties is ignored.

See Also

[ClipText Property](#)

[TabWidth Property](#)

Examples

```
SBList1.AutoClip = True  
SBList1.AddItem "Column 1 & 2\t\tColumn 3"
```

AutoDraw Property

[See Also](#)

[Examples](#)

[Knowledge Base](#)

Controls whether new added items are displayed immediately.

Syntax

```
SBList1.AutoDraw = value
```

value True or False

Remarks

The default value is True, which causes the list box to behave as normal. If set to False, items added to the list box will not be made visible until **AutoDraw** is set back to True. This also applies to using the **Clear** method. By setting this property to False before adding many items a huge performance gain will be achieved.

See Also

[AddItem Method](#)

[AddPacket Method](#)

[AddPicture Method](#)

[Clear Method](#)

[RemoveItem Method](#)

Examples

```
SBList1.AutoDraw = False  
SBList1.Clear  
For i% = 1 to 500  
    SBList1.AddItem "Item " & Str$(i%)  
Next i%  
SBList1.AutoDraw = True
```

Knowledge Base

[How to achieve the fastest results with SBList](#)

BackColor Property

[See Also](#)

[Examples](#)

Sets background color

Syntax

```
SBList1.BackColor = value
```

value A color value.

Remarks

This property sets the color used to paint the background of the list box. Individual list items can override this color by use if the **ItemBackColor** property or **AddPacket** Method.

See Also

[AddPacket Method](#)

[ItemBackColor Property](#)

Examples

```
SBList1.BackColor = RGB(128,128,128)  
SBList1.BackColor = &H8000000F&  
SBList1.BackColor = QBColor(4)  
col = SBList2.BackColor
```

Border Property

[See Also](#)

[Examples](#)

Sets or retrieves the border style

Syntax

SBList1.**Border** = *value*

value True or False.

Remarks

Specifies whether a border is drawn around the list box. If **Appearance** is set to 1 (3D) this property is ignored.

See Also

[Appearance Property](#)

Examples

SBList1.Border = True

SBList2.Border = frmMain.SBList1.Border

Cell Property

[See Also](#)

[Examples](#)

Returns unformatted contents of a cell. Read only, and available at run time only.

Syntax

```
str = SBList1.Cell ( num [, index] )
```

str String value which will contain the cell value.

num Integer specifying the cell number. The first cell is 0.

index Optional. Long integer specifying the item number. If omitted and **MultiSelect**=0 the current list index is used. If omitted, and either the list index is -1 or **MultiSelect** > 0 then an error will occur.

Remarks

A cell is the text between tab positions, defined as either ASCII character 9 or the `\t` formatting code. The first cell is the text up to the first tab position. Up to 32 cells may be referenced.

If **MultiSelect** is set to 0 (simple) or 1 (extended) then the *index* parameter must be used.

All formatting codes are removed, including the tab characters.

See Also

[TabWidth Property](#)

Examples

```
Dim a$  
SBList1.AddItem "England\tWales\tScotland"  
SBList1.ListIndex = 0  
a$ = SBList1.Cell(0, 0)    ' England  
a$ = SBList1.Cell(1)     ' Wales  
a$ = SBList1.Cell(2, 0)  ' Scotland
```

CenterV Property

[See Also](#)

[Examples](#)

Specifies whether text is vertically centered.

Syntax

SBList1.CenterV = *value*

value Boolean value, set to True to vertically center the text items, or False otherwise

Remarks

When this property is set to True (default) each text item and picture will be vertically centered within the height allocated for each item. This height is either a size automatically derived from the font size, or is the value set in **ItemHeight**. If **CenterV** is set to False, the top of the text will start at a position set by the **TextTop** property, and the top of any picture will be set by **PicTop**.

See Also

[ItemHeight Property](#)

[PicTop Property](#)

[TextTop Property](#)

Examples

```
SBList1.CenterV = True  
i% = SBList3.CenterV
```

ClipText Property

[See Also](#)

[Examples](#)

[Knowledge Base](#)

Truncates text at tab positions.

Syntax

```
SBList1.ClipText = value
```

value True or False.

Remarks

If this property is set to True, text will be truncated at the positions set by the **TabWidth** property. If the **Ellipsis** property is also set to True, where text is truncated an ellipsis (...) is displayed to indicate further text.

If **ClipText** is set to False and text extends beyond a tab position, the text following the tab will be superimposed on the first.

An alternative to **ClipText** is the **AutoClip** property which will only truncate if there is text after the next tab position.

See Also

[AutoClip Property](#)

[Ellipsis Property](#)

[TabWidth Property](#)

Examples

SBList1.ClipText = True

SBList2.ClipText = frmMain.SBList5.ClipText

DotsHorizontal Property

[See Also](#)

[Examples](#)

Syntax

SBList1.DotsHorizontal = *value*

value True or False.

Remarks

If this property is set to True, and **LineHorizontal** is non-zero, the horizontal line will be dotted instead of solid.

See Also

[LineHorizontal Property](#)

Examples

SBList1.DotsHorizontal = True

SBList2.DotsHorizontal = frmMain.SBList5. DotsHorizontal

DotsVertical Property

[See Also](#)

[Examples](#)

Syntax

SBList1.DotsVertical = *value*

value True or False.

Remarks

If this property is set to True, then any vertical lines as specified with the **LineVertical** or **TabLines** properties will appear dotted, and not solid.

See Also

[LineVertical Property](#)

[TabLines Property](#)

Examples

SBList1.DotsVertical = True

SBList2.DotsVertical = frmMain.SBList5.DotsVertical

DragIcon Property

[See Also Knowledge Base](#)

Syntax

```
SBList1.DragIcon = icon
```

icon Any code reference that returns a valid icon, such as a reference to a form's icon (Form1.Icon), a reference to another control's DragIcon property (Text1.DragIcon), or the LoadPicture function.

Remarks

You can use the DragIcon property to provide visual feedback during a drag-and-drop operation for example, to indicate that the source control is over an appropriate target. DragIcon takes effect when the user initiates a drag-and-drop operation. Typically, you set DragIcon as part of a MouseDown or DragOver event procedure.

See Also

[DragDrop Event](#)

[DragMode Property](#)

[DragOver Event](#)

DragMode Property

[See Also](#)

Returns or sets a value that determines whether manual or automatic drag mode is used for a drag-and-drop operation.

Syntax

SBList1.**DragMode** = *value*

value An integer that specifies the drag mode, as described below.

Remarks

The settings for value are:

<u>Setting</u>	<u>Description</u>
0	(Default) Manual-requires using the Drag method to initiate a drag-and-drop operation on the source control.
1	Automatic-clicking the source control automatically initiates a drag-and-drop operation. OLE container controls are automatically dragged only when they don't have the focus.

See Also

[DragDrop Event](#)

[DragIcon Property](#)

[DragOver Event](#)

Ellipsis Property

[See Also](#)

[Examples](#)

[Knowledge Base](#)

Syntax

SBList1.**Ellipsis** = *value*

value True or False

Remarks

If set to True, any text that is truncated at tab stops will have an ellipsis (...) drawn to indicate that truncation has occurred. This only applies if **ClipText** or **AutoClip** are set to True.

See Also

[AutoClip Property](#)

[ClipText Property](#)

[TabWidth Property](#)

Examples

SBList1.Ellipsis = True

SBList2.Ellipsis = frmMain.SBList5.Ellipsis

Enabled Property

Examples

Syntax

SBList1.Enabled = *value*

value True or False

Remarks

Specifies whether list box reacts to user intervention with the keyboard or mouse.

Examples

SBList1.Enabled = True

SBList2.Enabled = frmMain.SBList1.Enabled

EscapeChr Property

[See Also](#)

[Examples](#)

Syntax

SBList1.**EscapeChr** = *value*

value Integer value between 0 and 255.

Remarks

This value is the ASCII value of the escape character used to precede a text formatting character. By default this value is 92 which is a backslash. Setting this property to 0 will effectively turn off embedded escape characters. To display the escape character in the list box, use 2 consecutive characters.

See Also

[Text Formatting](#)

Examples

```
SBList1.EscapeChr = 92  
esc% = frmMain.SBList5.EscapeChr
```

FireClick Property

[See Also](#) [Examples](#)

Syntax

SBList1.FireClick = *value*

value True or False

Remarks

If this property is True, a Click event will be fired whenever the **ListIndex** or **Selected** properties are set by code. The default is True.

See Also

[Click Event](#)

[ListIndex Property](#)

[Selected Property](#)

Examples

SBList1.FireClick = False

SBList2.FireClick = frmMain.SBList1.FireClick

Font Property

[See Also](#)

[Examples](#)

Syntax

SBList1.Font[*attrib*] = *value*

attrib Optional. An attribute of a Font object, such as Bold, Size, etc.

value Value for new font object or font attribute.

Remarks

Sets the font object for the list box. Attributes of this font may be changed by inserting the relevant escape codes.

See Also

[EscapeChr Property](#)

[Text Formatting](#)

Examples

```
SBList1.Font.Bold = True  
SBList1.Font.Size = 12  
SBList1.Font.Name = "Arial"  
Set SBList1.Font = frmMain.SBList2.Font
```

ForeColor Property

[See Also](#) [Examples](#)

Syntax

SBList1.ForeColor = *value*

value A color value

Remarks

This value determines the default color of the text within the list box. The color of text may be changed by inserting escape codes.

See Also

[ForeColor1 Property](#)

[ForeColor2 Property](#)

[ForeColor3 Property](#)

[ForeColor4 Property](#)

[Text Formatting](#)

Examples

```
SBList1.ForeColor = RGB(255,0,0)  
SBList1.ForeColor = &H80000012&  
SBList1.ForeColor = QBColor(3)  
col = SBList1.ForeColor
```

ForeColor1, ForeColor2, ForeColor3 and ForeColor4 Properties

[See Also](#)

[Examples](#)

Syntax

SBList1.ForeColor*n* = *value*

value A color value.

Remarks

These color values are used when changing the text color by using embedded escape characters. By default, these four colors are blue, red, green and cyan respectively.

See Also

[ForeColor Property](#)

[Text Formatting](#)

Examples

```
SBList1.ForeColor1 = RGB(128,0,0)
SBList1.ForeColor2 = &H80000010&
SBList1.ForeColor3 = QBColor(1)
col& = SBList1.ForeColor4
```

Height, Width Properties

[See Also](#)

[Examples](#)

Return or set the dimensions of an object.

Syntax

SBList1.Height = *value*

SBList1.Width = *value*

value A numeric expression specifying the dimensions of an object. These properties use the scale units of the object's container.

Remarks

The values for these properties change as the object is resized. If **IntegralHeight** is set to True, the height will adjust to a value that displays a whole number of list items.

See Also

[IntegralHeight Property](#)

[Left Property](#)

[Top Property](#)

Examples

SBList1.Height = 3000

SBList2.Width = 200 * Screen.TwipsPerPixelX

h% = SBList3.Height

HelpContextID Property

[See Also](#)

[Examples](#)

Returns or sets an associated context number for the list box. Used to provide context-sensitive Help for your application.

Syntax

SBList1.HelpContextID = number

number Context number, or 0 for none.

Remarks

For context-sensitive Help on the list box in your application, you must assign the same context number to both list box and to the associated Help topic when you compile your Help file.

See Also

[WhatsThisHelpID Property](#)

Examples

SBList1.HelpContextID = 6

SBList2.HelpContextID = frmMain.SBList1.HelpContextID

HighBackColor Property

[See Also](#)

[Examples](#)

Syntax

SBList1.**HighBackColor** = *value*

value A color value

Remarks

This property specifies the color used as the highlight bar for selected items. By default, it is set to the color chosen by the user in Control Panel. The **ItemBackColor** property is ignored when an item is selected.

See Also

[HighForeColor Property](#)

[HighLeft Property](#)

[ItemBackColor Property](#)

Examples

```
SBList1.HighBackColor = &H8000000D&  
SBList1.HighBackColor = RGB(0,0,255)  
col& = SBList2.HighBackColor
```

HighForeColor Property

[See Also](#)

[Examples](#)

Syntax

SBList1.HighForeColor = *value*

value A color value.

Remarks

This property specifies the color used for text when selected. By default, it is set to the color chosen by the user in Control Panel. Any color changes made by the insertion of escape codes are ignored when an item is selected.

See Also

[HighBackColor Property](#)

[Text Formatting](#)

Examples

```
SBList1.HighForeColor = &H8000000E&  
SBList1.HighForeColor = RGB(0,0,255)  
col& = SBList2.HighForeColor
```

HighLeft Property

[See Also](#)

[Examples](#)

[Knowledge Base](#)

Syntax

SBList1.HighLeft = *value*

value An integer value greater than or equal to 0.

Remarks

This property set the indentation of the highlight bar for selected items, and is measured in pixels. The default value is 0.

See Also

[HighBackColor Property](#)

[HighForeColor Property](#)

Examples

SBList1.HighLeft = 20
i% = SBList2.HighLeft

hWnd Property

Examples

Syntax

SBList1.hWnd

Remarks

Returns the window handle for the list box. Not available at design time. Read only at run time.

Examples

```
h% = SBList1.hWnd
```

Index Property

Returns or sets the number that uniquely identifies the list box in a control array. Available only if the control is part of a control array.

Syntax

SBList1[(*number*)].**Index**

number A numeric expression that evaluates to an integer that identifies an individual list box within a control array.

Remarks

Because control array elements share the same **Name** property setting, you must use the **Index** property in code to specify a particular control in the array. **Index** must appear as an integer (or a numeric expression evaluating to an integer) in parentheses next to the control array name - for example, `SBList(3)`. You can also use the **Tag** property setting to distinguish one control from another within a control array.

When a control in the array recognizes that an event has occurred, Visual Basic calls the control array's event procedure and passes the applicable **Index** setting as an additional argument. This property is also used when you create controls dynamically at run time with the **Load** statement or remove them with the **Unload** statement.

Although Visual Basic assigns, by default, the next integer available as the value of **Index** for a new control in a control array, you can override this assigned value and skip integers. You can also set **Index** to an integer other than 0 for the first control in the array. If you reference an **Index** value in code that doesn't identify one of the controls in a control array, a Visual Basic run-time error occurs.

IntegralHeight Property

[See Also](#)

[Examples](#)

Syntax

```
SBList1.IntegralHeight = value
```

value True or False.

Remarks

If this property is True, the height of the list box will be adjusted to be a size that fits whole lines only. If set to False, it is possible to set the height to any value. The default is True.

See Also

[Height Property](#)

Examples

```
SBList1.IntegralHeight = True  
a% = SBList2.IntegralHeight
```

ItemBackColor Property

[See Also](#)

[Examples](#)

Syntax

SBList1.ItemBackColor(*index*) = *value*

index List item index, between 0 and **ListIndex** – 1.

value Color value.

Remarks

This sets or returns the background color for a particular list item. If this property is not used on an item, its background color will be the **BackColor** property. This value may be set when an item is added by using the **AddPacket** method.

Not available at design time.

See Also

[AddPacket Method](#)

[BackColor Property](#)

Examples

```
SBList1.ItemBackColor(3) = RGB(128,128,128)
SBList1.ItemBackColor(SBList1.ListIndex) = QBColor(4)
col& = SBList2.ItemBackColor(0)
```

ItemData Property

[See Also](#)

[Examples](#)

Syntax

SBList1.ItemData(*index*) = *value*

index Index of item, between 0 and **ListIndex** – 1

value Long integer value

Remarks

Sets or returns a long integer value that may be attached to a list item. This value is not used for any purpose other than a place to store user data associated with an item. Similarly, a string value may be associated with a list item by using **ItemText**. The **ItemData** value may be set when an item is added to the list by using the **AddPacket** method.

Not available at design time.

See Also

[AddPacket Method](#)

[FindData Method](#)

[ItemText Property](#)

Examples

SBList1.ItemData(56) = 1996

a& = SBList1.ItemData(100)

SBList2.ItemData(SBList1.NewIndex) = -100

ItemHeight Property

[See Also](#)

[Examples](#)

Syntax

```
SBList1.ItemHeight = value
```

value Height in pixels of every list item, or 0 to use default value.

Remarks

Sets or returns the height in pixels of every list item. If set to 0 the size will be calculated based on the font size. The text can be positioned within its space by use of **CenterV**, **TextTop** and **TextLeft** properties.

If **IntegralHeight** is set to True and **ItemHeight** is greater than 0, the list box height will adjust to fit a whole number of list items.,

Unlike in earlier version, this property may be set at run time.

See Also

[CenterV Property](#)

[TextLeft Property](#)

[TextTop Property](#)

Examples

SBList1.ItemHeight = 30
i% = SBList2.ItemHeight

ItemPicture Property

[See Also](#)[Examples](#) [Knowledge Base](#)

Supplies an item with a picture.

Syntax

Set *SBList1*.ItemPicture(*index*) = *picture*

index Numeric value between 0 and **ListIndex** – 1.

picture A picture

Remarks

This property sets the picture to be displayed for a list item. The picture is displayed at the position within the list item as set by the **PicLeft** and **PicTop** properties.

Not available at design time and write only at run time.

See Also

[AddPicture Method](#)

[AddPacket Method](#)

[PicLeft Property](#)

[PicTop Property](#)

Examples

```
Set SBLst1.ItemPicture(12) = Picture2.Picture  
Set SBLst1.ItemPicture(1) = LoadPicture("c:\mypic.bmp")  
Set SBLst1.ItemPicture(2) = LoadPicture()
```

Knowledge Base

[How to create a list of check boxes](#)

[Problems with pictures ?](#)

ItemText Property

[See Also](#)

[Examples](#)

Syntax

SBList1.ItemText(index) = value

index Numerical value between 0 and **ListIndex** – 1.

value A string value

Remarks

This property allows a string value to be attached to any list item, similar to the **ItemData** property. The value is not displayed.

Not available at design time.

See Also

[AddPacket Method](#)

[FindString Method](#)

[ItemData Property](#)

Examples

```
SBList1.ItemText(56) = "Hello World"  
a$ = SBList1.ItemText(100)
```

Knowledge Base

[How to Clip, Wrap, Tab and Truncate](#)

Knowledge Base

[How to create a list of check boxes](#)

Knowledge Base

[How to drag and drop items between two lists](#)

[How to drag and drop multiple list items](#)

[How to reposition list items using drag and drop](#)

Left, Top Properties

[See Also](#)

[Examples](#)

- **Left** - returns or sets the distance between the internal left edge of an object and the left edge of its container.
- **Top** - returns or sets the distance between the internal top edge of an object and the top edge of its container.

Syntax

SBList1.Left = value

SBList1.Top = value

value A numeric expression specifying distance.

Remarks

The **Left** and **Top** properties are measured in units whose size depends on the coordinate system of the object's container. The values for these properties change as the object is moved by the user or by code.

See Also

[Height Property](#)

[Width Property](#)

Examples

SBList1.Left = 300

SBList1.Top = 180

LineColor Property

[See Also](#)

[Examples](#)

Syntax

```
SBList1.LineColor = value
```

value A color value

Remarks

Sets or retrieves the color value used to draw any horizontal or vertical lines as set by the **LineHorizontal**, **LineVertical** and **TabLines** properties

See Also

[LineHorizontal Property](#)

[LineVertical Property](#)

[TabLines Property](#)

Examples

```
SBList1.LineColor = RGB(255,0,0)
SBList1.LineColor = &H000000FF&
col& = SBList2.LineColor
```

LineHorizontal Property

[See Also](#)

[Examples](#)

Sets or retrieves the position of a horizontal line drawn across each list item.

Syntax

SBList1.LineHorizontal = *number*

number Numeric value

Remarks

Gets or retrieves the vertical position of a horizontal line, drawn the width of the list box within each list item. The color of the line is specified with the **LineColor** property. The line will either be drawn solid or as a series of dots, as specified with the **DotsHorizontal** property.

The values that can be assigned to *number* are :

<u>Constant</u>	<u>Value</u>	<u>Description</u>
sbNone	0	No line is drawn
sbLineBelow	-2	Line is drawn across the bottom
sbLineAbove	-1	Line is drawn across the top
	> 0	The vertical position in pixels, starting from the top

See Also

[DotsHorizontal Property](#)

[LineColor Property](#)

[LineVertical Property](#)

Examples

```
SBList1.LineHorizontal = -2  
SBList2.LineHorizontal = 5  
i% = SBList3.LineHorizontal
```

LineVertical Property

[See Also](#)

[Examples](#)

Sets or retrieves the horizontal position of vertical lines drawn within each list item.

Syntax

SBList1.LineVertical(index) = number

index Numeric value between 0 and 31 specifying which line is being set.

number Numeric value. -1 indicates no line.

Remarks

This property allows up to 32 vertical lines to be drawn within a list item. The numeric value in *number* contains the position in pixels from the left edge where the line will be drawn, or -1 for no line. The color of the line is set with the **LineColor** property. The **DotsVertical** property controls whether the lines are drawn solid or as a series of dots. If **TabDots** is set to True the values stored in **LineVertical** are ignored.

Not available at design time.

See Also

[DotsVertical Property](#)

[LineColor Property](#)

[LineHorizontal Property](#)

[TabLines Property](#)

Examples

SBList1.LineVertical(0) = 30

SBList1.LineVertical(1) = 90

i% = frmMain.SBList3.LineVertical(3)

List Property

[See Also](#) [Examples](#)

Sets or retrieves a list item.

Syntax

SBList1.**List**(*index*) = *value*

index Numeric value between 0 and ListIndex – 1.

value A string expression.

Remarks

This property is used to either retrieve the text for a list item, or replace the text with a different value. Not available at design time.

See Also

[Text Property](#)

Examples

SBList1.List(1) = "Mongolia, Ulaanbaatar"
x\$ = SBList1.List(84)

ListCount Property

Examples

Returns the number of items in the list box

Syntax

SBList1.ListCount

Remarks

Returns the number of items in the list box. Not available at design time.

Examples

```
i% = SBList1.ListCount
```

ListIndex Property

Examples

Syntax

SBList1.ListIndex = *number*

number Numeric value.

Remarks

This property sets or retrieves the currently selected item, or -1 if no item is selected. If **MultiSelect** is set to 1, 2 or 3 then **ListIndex** returns the position of the list item containing the focus rectangle.

Examples

```
SBList1.ListIndex = -1  
SBList1.ListIndex = 8  
i% = SBList2.ListIndex
```

ListRaw Property

[See Also Examples](#)

Syntax

SBList1.ListRaw(*index*)

index Numeric value between 0 and **ListCount** – 1

Remarks

Returns the contents of a list item with all text formatting codes removed. Not available at design time and read-only at run time.

See Also

[List Property](#)

[Text Formatting](#)

Examples

```
SBList1.Clear
```

```
SBList1.AddItem "[BVietnam[b\t\\Hanoi]"
```

```
a$ = SBList1.ListRaw(1)      ' a$ = "VietnamHanoi"
```


MouseIcon Property

[See Also](#)

[Examples](#)

Sets a custom mouse icon.

Syntax

```
SBList1.MouseIcon = picture
```

picture A picture, or **LoadPicture** statement.

Remarks

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

See Also

[DragIcon Property](#)

[MouseMove Event](#)

[MousePointer Property](#)

Examples

```
SBList1.MouseIcon = Picture1.Picture
```

```
SBList1.MouseIcon = LoadPicture("c:\hand.cur")
```

```
SBList2.MouseIcon = frmMain.SBList1.MouseIcon
```

MousePointer Property

[See Also](#) [Examples](#)

Returns or sets a value indicating the type of mouse pointer displayed when the mouse is over the list box.

Syntax

SBList1.**MousePointer** = *number*

number An integer specifying the type of mouse pointer displayed.

Remarks

A *number* of 99 will use the cursor stored in **MouseIcon** to be used.

See Also

[DragIcon Property](#)

[MouseIcon Property](#)

[MouseMove Event](#)

Examples

SBList1.MousePointer = 11
SBList2.MousePointer = 99
i% = SBList3.MousePointer

‘ Hourglass
‘ MouseIcon

MultiSelect Property

[See Also](#) [Examples](#) [Knowledge Base](#)

Returns or sets a value indicating whether a user can make multiple selections and how the multiple selections can be made.

Syntax

SBList1.MultiSelect = *number*

number An integer between 0 and 3.

Remarks

The MultiSelect property settings are:

Constant	Value	Description
sbNone	0	(Default) Multiple selection isn't allowed.
sbSimple	1	Simple multiple selection. A mouse click or pressing the SPACEBAR selects or deselects an item in the list. (Arrow keys move the focus.)
sbExtended	2	Extended multiple selection. Pressing SHIFT and clicking the mouse or pressing 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.
sbSpecial	3	Special multiple selection. Similar to 2-Extended but it is not possible to select items by dragging a selection. This is useful for multiple item drag & drop.

See Also

[SelCount Property](#)

[SelectAll Method](#)

[Selected Property](#)

[SelectNone Method](#)

Examples

```
SBList1.MultiSelect = 3  
i% = frmMain.SBList2.MultiSelect
```


Knowledge Base

[How to drag and drop multiple list items](#)

Name Property

[See Also](#)

Returns the name used in code to identify the list box.

Syntax

SBList1.Name

Remarks

The default name for new list boxes is SBList plus a unique integer. For example, the first new list box is SBList1.

Read-only at run-time.

See Also

[Index Property](#)

NewIndex Property

[See Also](#)

[Examples](#)

Returns index of last item added to list box.

Syntax

SBList1.**NewIndex**

Remarks

This returns the index number of the last item added. If an item has been removed, **NewIndex** returns – 1.

Not available at design time, and read-only at run time.

See Also

[AddItem Method](#)

[AddPacket Method](#)

[AddPicture Method](#)

Examples

```
SBList.Sorted = True  
SBList1.AddItem "Morocco"  
i% = SBList1.NewIndex
```

PicLeft Property

[See Also](#)

[Examples](#)

[Knowledge Base](#)

Sets or returns the indentation of each items picture.

Syntax

SBList1.PicLeft = number

number Integer value, specifying number of pixels.

Remarks

This property is used to specify the number of pixels between the left border of the list box and the left edge of an items picture.

See Also

[AddPicture Method](#)

[ItemPicture Property](#)

[PicTop Property](#)

Examples

SBList1.PicLeft = 30

i% = frmMain.SBList2.PicLeft

PicTop Property

See Also [Examples](#) [Knowledge Base](#)

Sets or returns the vertical position of each items picture.

Syntax

SBList1.PicTop = *number*

number Integer value, specifying number of pixels.

Remarks

This property is used to specify the number of pixels between the top of an items space and the top edge of an items picture. Ignored if **CenterV** is set to True.

See Also

[AddPicture Method](#)

[CenterV Property](#)

[ItemPicture Property](#)

[PicLeft Property](#)

Examples

SBList1.PicTop = 5

i% = frmMain.SBList2.PicTop

RightButton Property

Knowledge Base

Boolean value specifying whether list box reacts to the right mouse button.

Syntax

SBList1.RightButton = *value*

value True or False.

Remarks

If set to True, the list box will react to right mouse clicks in the same way as for the left mouse button. The default setting is False.

Knowledge Base

[How to use the RightButton property](#)

SelCount Property

[See Also](#)

[Examples](#)

Syntax

SBList1.**SelCount**

Remarks

Returns the number of items currently selected in the list box. Not available at design time, and read-only at run time.

See Also

[MultiSelect Property](#)

[Selected Property](#)

Examples

`i% = SBList1.SelCount`

Selected Property

[See Also Examples](#)

Sets or returns the selected status of an item.

Syntax

SBList1.**Selected**(*index*) = *value*

index Integer value between 0 and **ListCount** – 1.

value True or False.

Remarks

Sets or returns the selected status of an item. If **MultiSelect** = 0 then this property affects the **ListIndex** property. Not available at design time.

See Also

[ListIndex Property](#)

[MultiSelect Property](#)

[SelCount Property](#)

Examples

```
SBList1.Selected(5) = True  
i% = SBList2.Selected(10)
```

ShowFocus Property

[See Also](#)

[Examples](#)

Syntax

SBList1.**ShowFocus** = *value*

value True or False

Remarks

Gets or sets whether a focus rectangle is displayed whenever the list box has the focus. The default is True, which matches the behaviour of a standard list box

See Also

[GotFocus Event](#)

[LostFocus Event](#)

[SetFocus Method](#)

Examples

```
SBList1.ShowFocus = False  
i% = SBList3.ShowFocus
```

Sorted Property

[See Also](#)

[Examples](#)

[Knowledge Base](#)

Syntax

SBList1.**Sorted** = *value*

value True or False.

Remarks

Gets or sets a Boolean value determining whether items added to the list box are sorted or not. The **SortedOn** property specifies on what attribute the sorting is performed on. Any escape characters are ignored for the purposes of sorting.

Unlike a standard list box, the **Sorted** property may be changed at run time. However, changing the value will not cause a change to the order of any items already in the list box.

See Also

[AddItem Method](#)

[AddPacket Method](#)

[SortedOn Property](#)

Examples

```
SBList1.Sorted = True  
i% = SBList2.Sorted
```

SortedOn Property

[See Also](#)

[Examples](#)

[Knowledge Base](#)

Syntax

SBList1.SortedOn = *value*

value Integer value between 0 and 2.

Remarks

Determines on what attribute an item is to be sorted on when adding a new item to a list box. If **Sorted** = False this property is ignored. With *value* set to 1 or 2, adding items must be performed using the **AddPacket** method in order to achieve correct placement of items.

Settings for *value* are :

- 0 Sort on list text, the actual text seen in the list box. This is default.
- 1 Sort on item data. Items must be added using AddPacket for this to work.
- 2 Sort on item text. Items must be added using AddPacket for this to work.

See Also

[AddItem Method](#)

[AddPacket Method](#)

[Sorted Property](#)

Examples

```
SBList1.SortedOn = 2  
i% = SBList2.SortedOn
```

TabDots Property

[See Also](#)

[Examples](#)

Syntax

SBList1.**TabDots** = *value*

value True or False.

Remarks

Gets or specifies if dots are to be drawn between columns of text, as specified by the **TabWidth** property. This only has effect when the **ClipText** property is true.

See Also

[ClipText Property](#)

[TabWidth Property](#)

Examples

```
SBList1.TabDots = True  
i% = SBList1.TabDots
```


TabIndex Property

[See Also](#)

Returns or sets the tab order of the list box within its parent form.

Syntax

SBList1.**TabIndex** = *value*

value An integer from 0 to (n-1), where n is the number of controls on the form that have a **TabIndex** property

See Also

[TabStop Property](#)

TabLines Property

[See Also](#)

[Examples](#)

Syntax

SBList1.**TabLines** = *value*

value True or False

Remarks

If *value* is set to True then vertical lines will be drawn at tab positions specified with the **TabWidth** property, and any values stored using **LineVertical** are ignored.

See Also

[DotsVertical Property](#)

[LineColor Property](#)

[TabWidth Property](#)

Examples

```
SBList1.TabLines = True  
i% = SBList1.TabLines
```

TabStop Property

[See Also](#)

Returns or sets a value indicating whether a user can use the TAB key to give the focus to the list box.

Syntax

SBList1.**TabStop** = *value*

value True or False

See Also

[TabIndex Property](#)

TabWidth Property

[See Also](#)

[Examples](#)

[Knowledge Base](#)

Sets or returns the tab positions.

Syntax

SBList1.**TabWidth**(*index*) = *value*

index Integer between 0 and 31.

value Integer.

Remarks

Sets or returns the position in pixels for a particular tab stop. The first tab character (or \t) will start printing from **TabWidth(0)**, the 2nd from **TabWidth(1)** etc. It is therefore possible to have tab positions out of order. When displaying multiple line items, the tab positioning does not go back to 0 for each new line, making it possible to have different tab positions for each line.

Vertical lines can be drawn at tab positions by setting **TabLines** to True. Text can be clipped at tab positions by use of **AutoClip** or **ClipText** properties.

Not available at design time.

See Also

[AutoClip Property](#)

[ClipText Property](#)

[TabLines Property](#)

Examples

```
SBList1.TabWidth(0) = 50  
SBList1.TabWidth(1) = 130  
SBList1.TabWidth(2) = 80  
i% = SBList2.TabWidth(2)
```

Tag Property

Examples

Returns or sets an expression that stores any extra data needed for your program. Unlike other properties, the value of the Tag property isn't used by Visual Basic; you can use this property to identify objects.

Syntax

SBList1.Tag = value

value String expression.

Examples

```
SBList1.Tag = "Listbox 3"  
a$ = frmMain.SBList2.Tag
```

Text Property

[See Also](#)

[Examples](#)

Syntax

SBList1.Text

Remarks

Returns the contents of the currently selected item. If no item is selected, an empty string is returned.

Not available at design time, and read-only at run time.

See Also

[List Property](#)

[ListIndex Property](#)

Examples

a\$ = SBList1.Text

TextLeft Property

[See Also](#) [Examples](#) [Knowledge Base](#)

Sets or returns the indentation of list text.

Syntax

SBList1.TextLeft = *value*

value Integer value

Remarks

Sets or returns the number of pixels between the left border of the list box and the start of the text. If set to -1 (default) the text will be 2 pixels from the left border, or if a picture is present the text will start 2 pixels to the right of it.

See Also

[CenterV Property](#)

[TextTop Property](#)

Examples

SBList1.TextLeft = 10

SBList2.TextLeft = -1

i% = frmMain.SBList1.TextLeft

TextTop Property

[See Also](#)

[Examples](#)

Syntax

SBList1.TextTop = *value*

value Integer value.

Remarks

Gets or sets the number of pixels from the top of an items space to the top of the text within it. If **CenterV** is set to True, this property is ignored.

See Also

[CenterV Property](#)

[TextLeft Property](#)

Examples

```
SBList1.TextTop = 5  
i% = SBList1.TextTop
```

TopIndex Property

Examples

Syntax

SBList1.**TopIndex** = *value*

value Integer value

Remarks

Gets or returns the item index number of the top most visible item in the list box. Not available at design time.

Examples

SBList1.TopIndex = 23
b& = SBList1.TopIndex

Visible Property

Examples

Syntax

SBList1.Visible = *value*

value True or False.

Remarks

Sets or returns the visible state of the list box.

Examples

```
SBList1.Visible = True  
i% = SBList1.Visible
```

WhatsThisHelpID Property

[See Also](#)

[Examples](#)

Syntax

SBList1.WhatsThisHelpID = *value*

value Numeric value

Remarks

Returns or sets an associated context number for an object. Used to provide context-sensitive Help for your application using the What's This popup in Windows 95 Help.

See Also

[HelpContextID Property](#)

Examples

SBList1.WhatsThisHelpID = 101
I% = SBList2.WhatsThisHelpID

Knowledge Base

[How to use the Sorted property](#)

[How to use the SortedOn property](#)

Knowledge Base

Problems with pictures ?



SBList OLE Control (ActiveX) Version 2.0 Shareware version

[Properties](#)

[Methods](#)

[Events](#)

[Formatting](#)

[Knowledge Base](#)

SBList OCX is a 32-bit high performance enhanced list box, incorporating most of the features of the standard Windows list box but with many extra features :

- Add a bitmap to any list item, in any position
- Change font attributes within an item
- Change color within an item
- Click event fires with right mouse button
- Can display horizontal & vertical lines
- Multiple lines per list item
- Powerful tab positioning
- Set background color for individual items
- Set highlight colors
- Assign a string to each item as well as standard long data type
- Sort items on text, ItemData or ItemText
- Load and save contents to disk
- Comprehensive search facilities

[Purchase information](#)

[Legal](#)

[Release notes](#)

[Support](#)

[Credits](#)

Purchase Information

To be able to distribute SBList OCX with your applications, or to use it commercially, or if you just want to continue to use SBList with a clean conscience, you must first register. Remember, this is just an **evaluation** copy only. Once registered you will receive the registered version which may be distributed by you in your applications, with no restrictions on the volume sold. The registered version will not display a message box when used with a compiled VB program.

By registering you will also receive all future upgrades of this product (if you select e-mail as your delivery method). Full lifetime support is also included.

There are two options for registering SBList OCX :

Option 1

Purchase one licence for each of your developers that will be using SBList OCX for application development. The cost of each licence is either 25 GB pounds, 40 US dollars or 55 Canadian dollars. Other currencies are also accepted, but please contact us for up-to-date information.

Option 2

Purchase a corporate licence. This entitles you to unlimited licences for all your company's application developers *and* the full source code to SBList OCX (compiles with Visual C++ 4.0). The cost is 100 GB pounds, 160 US dollars or 220 Canadian dollars. Other currencies are also accepted, but please contact us for up-to-date information.

The methods available to pay are:

Check. Personal and company checks are accepted, and should be made payable to 'Global Components'. The currency used on the check must match the country of the bank printed on it. For example, do *not* send a Canadian check made out in US dollars, not unless it has a US bank address printed on the front. Euro-cheques are accepted but these *must* be in your own currency.

Cash. When sending cash through the post you do so at your own risk. Please ensure it is not visible from outside of the envelope.

Bank transfer. Please contact us for bank details.

Credit card. These are not currently accepted, but we are in the process of obtaining the facilities to accept Visa, Mastercard and Eurocard. By the time you are reading this we may be able to accept them, so please contact us to verify.

With your registration, please supply :

1. Your name & company name (if applicable)
2. Your postal address.
3. Your e-mail address (if applicable)
4. The product name : SBList OCX.
5. The purchase option, either 1 or 2. If option 1, then specify the number of licences.
6. How you would like to receive the registered files, either e-mail or post.

Please send to :

Global Components
41 Milton Street
Northampton
NN2 7JG

United Kingdom

If you have any queries/comments/bug-reports you can e-mail us at :

GlobalCom@pobox.com

Visit our web site at :

<http://ds.dial.pipex.com/globalcom/>

