Scrollable Lists

Scrollable list controls must be of type "List or Menu", have solid bodies, and be linked to an STR# resource. The STR# string list can be created and edited with ResEdit, and then optionally manipulated directly by a program using utility commands (see "String Lists" in utility topics). The current list "value" (= the selected item(s)) is set or read through data linking using SetVal and GetVal.

List Items

The strings in the linked STR# can be either plain text or references to PICT, ICON, SICN, cicn, PAT , PAT#, CURS, acur, or clut resources. The following string, for example, would display the 4th pattern in PAT# 1000 followed by "Ronald":

PAT#,1000,4,Ronald

Items can be made inactive by preceding the string with a "-" character:

-PICT,1005,hello

Note how string elements are separated by commas, and that SICN, PAT#, acur, and clut types require an index value in addition to the resource ID.

Options

The following bit values can be added to VarCode to set various list options:

2,4,8,16,32,64,128 = "selFlags" defined by List Manager in Volume 4 of Inside Macintosh:

2 = lNoNilHilite = do not select empty cells
4 = lUseSense = clicked cell determines action
8 = lNoRect = (not used by 1-dimensional lists)
16 = lNoExtend = do not extend Shift selections
32 = lNoDisjoint = deselect others on mouse click
64 = lExtendDrag = allow dragging w/o Shift key
128 = lOnlyOne = one selection at a time

Adding 128 to the VarCode, for example, creates a single-selection type of list, and setting none of the above flags creates a typical multi-selection list.

256 = display list horizontally (default is vertical)
512 = do not show vertical or horizontal scroll bar
1024 = use solid frame rectangle to hilite list items (instead of inverting items)
2048 (requires data linking) = always update list with current contents of linked STR# resource when SetVal is called, and update STR# with list contents when GetVal is called
4096 = ignore references to non-text resources in list items and display as text only (do not use this with horizontal lists or option 8192)
8192 = show text below non-text resource (instead of to its right (vertical) or not at all (horizontal))
16384 (requires option 2048)= create a separate STR#-type handle from a copy of the linked STR# resource, and use this handle to transfer strings to and from the list on SetVal and GetVal. The handle to the STR#-like block can be found in cLoData after a call to GetCtl to get control info.

Horizontal lists have rectangular cells and work best when displaying non-text items. Vertical lists have cells as wide as the list control and as tall as the list's font, and work best when displaying text items, or text items preceded by small icons. If the "text below" option is used (8192 in VarCode), then both horizontal and vertical lists have cells that are as tall or wide as the control, with space for displaying text below the non-text resource.

STR# Link

BaseCt uses the Macintosh List Manager to create and display scrollable lists. The ListHandle for each list can be found in cHiData (after calling GetCtl) and used with List Manager toolbox calls. WARNING: We make use of both the "userHandle" and "refCon" fields in the ListHandle, so you won't be able to use these to store program-specific info (use the control's own refCon to store such info, as discussed in "ID/RefCon" in the "Controls" topic of the ViewIt guide).

• Fixed Lists

The strings from the linked STR# are copied into the data handle maintained by the List Manager for each list, meaning that later changes to the STR# list will not affect the displayed list. This works well in cases where the list is being used to display a fixed set of items.

• Dynamic Lists

In cases where the contents of the list are not fixed, and the list is data-linked to a program variable, then you can use VarCode option 2048 to force the list to "track" the current contents of the linked STR# on calls to GetVal and SetVal. In this case, the utility command SetStr is typically used to set new strings in the linked STR#, and then SetVal called to update the list in the window.

• Multiple Instances

A limitation of using the "tracking" option is that multiple instances of the same list control (as might occur if opening multiple windows based on the same FWND) are linked to the same STR#, meaning that this STR# resource can't be used to save list info for more than one list control. A workaround for this limitation is to dynamically allocate an STR#-type block for each such list control (using the SetStr command), and then use DupLst, GetVal, & SetVal to move strings between this STR#-type block and the list control:

block <-DupLst-> STR# <-Get/SetVal-> control

Since it would be a burden for the programmer to allocate and manage such a block for each control, VarCode option 16384 is available to force BaseCt to allocate such an STR#-type block automatically, and to use the block (instead of the STR# resource) when responding to calls to GetVal and SetVal:

block <- Get/SetVal -> control

The initial contents of the dynamically allocated block will be a copy of the linked STR# resource, and the block's handle can be found in cLoData after a call to GetCtl to get control info. This handle can be passed directly to the SetStr and GetStr utility commands to set and get the strings in the block associated with each such list control.

Data Linking

A list control's private "value" is an 8-byte integer that corresponds to the items selected in the list. For single-selection lists (VarCode contains 128), this value is equal to the number of the currently selected list item. For multi-selection lists, the list's value is equal to the sum of the bit values corresponding to the selected items in the list. For example, if the first, third, and fifth items are selected in a multi-selection list, then the list's "value" = 1 + 4 + 16 = 21. If no items are selected, then the list's value = 0 (for both list types).

Scrollable lists can be linked to program variables (typically 2, 4 or 8-byte integers - see "Data Links" in the ViewIt Guide for more info on data linking). On GetVal, an integer value is returned that indicates which items in the list are selected. On SetVal, the selected list items are updated to reflect the value of the linked program variable. As described above, you can also have the list contents track the linked STR# resource or a dynamically allocated STR#-type block on GetVal and SetVal.

The following code, for example, links a 2-byte integer variable named "myInt" to a single-selection list control that is the 3rd control in the 2nd view of the window based on FWND 1005, and then selects the 5th list item:

```
FaceIt(nil,GetCtl,1005,0,2,3);
FaceIt(nil,LnkCtl,ord(cControl), ord(@myInt),2,0);
...
myInt := 5;
FaceIt(nil,SetVal,1005,0,2,3);
```

Long Lists

The internal 8-byte (64-bit) integer used to store a list's "value" will be a problem if a multi-selection list has (or can have) more than 64 items. In this case, the list can be linked to any contiguous block of memory (instead of a single integer) that will be used to get and set selection flags. This is done by setting the Variable Type of the linked block to 1310 to inform BaseCt that it should treat the data address as the address of a block of bits corresponding to the selection status of items in the list.

The linked block of bits must be as large or larger than the number of items in the list, otherwise bits beyond the block will get clobbered on GetVal. The bits in the block should be tested, set, and cleared using the toolbox calls BitTst, BitSet, and BitClr, where the bit offset will be one less than the number of the list item (i.e., zero-based).

The following code, for example, links a 100-byte block of bits named "myBlock" to a multi-selection list control that is the 3rd control in the 2nd view of the window based on FWND 1005, and then selects the 5th and 6th list items. Using Pascal,

```
myBlock : array [1..50] of integer;
...
for i:= 1 to 50 do
  myBlock[i] := 0;
FaceIt(nil,GetCtl,1005,0,2,3);
FaceIt(nil,LnkCtl,ord(cControl), ord(@myBlock),1310,0);
...
BitSet(@myBlock,4);
BitSet(@myBlock,5);
FaceIt(nil,SetVal,1005,0,2,3);
...
```

Or, using FORTRAN,

```
integer*2 myBlock(50)
...
do i = 1,50
  myBlock(i) = 0
end do
call FaceIt(0,GetCtl,1005,0,2,3);
call FaceIt(0,LnkCtl,cControl,
%loc(myBlock),1310,0);
...
call BitSet(myBlock,%val(4));
call BitSet(myBlock,%val(5));
call FaceIt(0,SetVal,1005,0,2,3);
```

Display Tip

Lists work best if there are no partial cells showing. To create a list with a whole number of cells: 1) first create a list control that is slightly smaller than the desired size, 2) scroll to the end of the list, and 3) go back into editing mode and expand the list 1 pixel at a time until the last item "jumps" into position.

Limitations

Although scrollable lists act as single Mac controls, the "GetCtl..." and "SetCtl..." toolbox calls do not get or set the list's state since BaseCt makes no use of the control's "contrlValue", "contrlMin", or "contrlMax".

When using GetVal or SetVal with multi-selection lists linked to integers, the maximum number of list items that can be affected is 64 since the list uses an 8-byte (64-bit) integer to store its "value". This restriction does not apply to single-selection lists or to the "long lists" described above.

Scrollable lists currently do not support right or center justification of list contents. Lists look best with one-pixel frames and no indent. No support for hand scrolling.