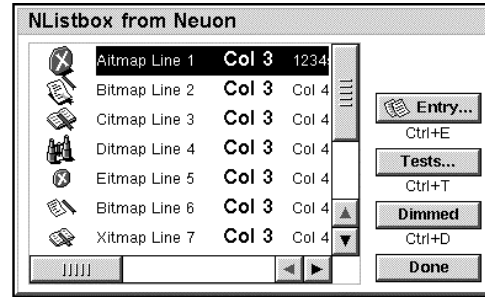


NListBox.opx

```
NListBox test code
-----
Make multiple selections by:
* Dragging
* Ctrl+Click
* Shift+Click for a range
* Toggling with the spacebar
```



About Neuron



You can learn more about our young and growing company, and expanding EPOC software portfolio, from Neuron's web site at <http://www.neuron.com/>

If you are a visionary C++, Java or OPL32 developer, motivated by doing something different, excited by challenges, relish the prospect of working with kindred spirits, and recognise the value of a dedicated support team, Neuron would like to hear from you.

***Neuron** - where innovation and quality are principles,
not an afterthought.*

Introduction

This guide and NListBox.opx are copyright (c) Alex Wilbur, Henry Hirst & Neuron 1999.

This is a dual purpose OPX. It provides you with:

1. Facilities for dynamic string arrays

Creation and manipulation of dynamic string arrays.

Fully featured sorting, editing, enquiries and searching.

Can include column data and bitmap identifiers for use with listboxes

2. Comprehensive choice of options to display a listbox dialog

Optional title and buttons

Callbacks to OPL32 functions

Bitmap entries

Unlimited column entries

Independent column widths and fonts

On-screen positioning and sizing

Auto-management functions for ease-of-use

Dynamic configuration

Automatic scrollbars as required

To contact Neuron regarding this OPX:

- Email msdt@neuron.com
- Opx.registrations@neuron.com if you are required to register this OPX, or
- via the Neuron homepage <http://www.neuron.com/>

Important

Please read the section **Licence agreement**. If you use, or distribute, this opx with your applications, you are agreeing to it. See also the topic **Distribution** which contains important information on how you may distribute this opx.

To use this OPX:

- You must include the header file NListBox.oxh

- The file NListbox.rsc must be in a \System\Opx\ folder

NListbox.opx release notes

This is **version 1.01** of NListbox opx. See **Release History** for details of changes.

Potential enhancements

- Extended column matching to cover column 2 when column 1 has a bitmap
- Addition of SetColumnAlignment methods
- Fully asynchronous operation

Please report any problems or difficulties to Neuron's Member Support & Development Team
msdt@neuron.com

Thank you for your interest. Comments and suggestions are welcomed.

March 2000

Contents

This help file contains the following information:

- **Arrays**

An overview of the array commands

A full listing of all commands, prefixed with **Array...**

- **Listboxes**

An overview of how the listbox commands work

Instructions on the mandatory framework for callback use

Important notes for the use of callback procedures

The listbox commands organised as follows:

- Instructions to construct, configure and display the listbox **LBox...**
- Commands for the creation and manipulation of bitmap arrays **LBoxBArray...**
- Commands to use while the listbox is displayed **LBoxDynamic...**
- Commands for use with multi-item selections **LBoxSelection...**
- Error messages
- Example code
- The shared module problem

Release History

v1.01

- Fix double tap bug on ER5 MARM
- Improved bounds checking on initial line parameter to LBoxDisplay

Licence agreement

This opx is copyright (c) Alex Wilbur, Henry Hirst and Neuron 1999

By installing this opx you are agreeing to the following terms and conditions. Please read them carefully, especially the information about distribution.

Licence Conditions

- 1 This opx is provided under licence for the development of EPOC applications. Only the files NListbox.sis (for the target machine) and NListboxWinsxxx opx (Deb and Rel builds for the emulator) may be distributed to end users of your applications, subject to the conditions below.
- 2 Copyright is retained by the copyright holders declared above.
- 3 A licence to use and distribute NListbox.sis may be granted as follows:

- Free of charge for freeware applications
 - Free of charge to Neuron authors
 - In return for payment of a nominal licence fee where the author receives remuneration, in cash or kind, for the software which uses this opx. This fee is a one-off payment. The fee is twice the sum the author receives for one copy of their software, payable separately for every software title using this opx.
4. The Licence for a Neuron member to use this opx free of charge remains valid only whilst the author is a member of Neuron, or until revoked under the terms of this Licence. On cessation of any membership agreement, the following additional conditions apply to the former member ("ex-member"):
- If the ex-member's Neuron application using this opx is freeware (i.e. the ex-member receives no payment in cash or kind from any person using the software), there are no additional conditions on continuing to use and distribute the opx.
 - If the ex-member's Neuron application is shareware (i.e. the ex-member receives payment in cash or kind from any person using the software), a one off fee is payable. The rate of this fee is twice the full registration fee for the ex-member's application.
5. It is agreed that Neuron may revoke any Licence without warning, and for any reason. On termination, the Licensee may not use nor distribute this opx, whether directly or indirectly by any physical or electronic means. Neuron reserves the right, at its absolute discretion, whether to award any licence.
 6. Licensee agrees to indemnify the copyright holders from any liability arising from the Licensee's reliance on this opx
 7. You may not rename the.sis, opx or the.oxh files.
 8. Any software which is not for the licensee's exclusive personal use, is to contain a credit in the software's "About" dialog (or similar), to the effect:

"NListbox.opx copyright (c) Alex Wilbur, Henry Hirst & Neuron 1999"

Payment of licence fee Please contact opx.registrations@neuron.com, or register via the Neuron web site at <http://www.neuron.com>

Distributing the OPX

NListbox.opx may only be used by Licensees in accordance with the terms of the Licence.

Important

- 1 If you wish to distribute this OPX as part of your Neuron program, you are required to include the **unchanged NListbox.sis** as an embedded .sis in your parent sis .pkg file. Note that **you may not**, under any circumstances, **distribute the unpacked target machine build of the NListbox.sis file**. You may also include the WINS build versions of NListbox.opx.
- 2 If you do not comply with this condition, you disable the EPOC version control over the OPX, and your application may cease to function if an end-user installs an earlier version of the OPX on their machine. Also, removing your application may also remove the unpacked .opx, which will cause all other applications which rely on it, to cease functioning.
- 3 During investigations it has been determined that there is a problem in maintaining version control and shared module usage. A lack of consistency between EPOC Install installations, and those conducted directly on the target machine, means that there is only **one** failsafe distribution method. For a detailed explanation of the problem, see the topic **The shared module** problem
- 4 **The only permitted distribution method for the target machine build of this opx** is to embed NListbox.sis in your .pkg file and to force your application .sis stub, which remains after installation, into C:\system\install\ . To do this, your .pkg file must contain the following:

```
//force the parent .sis stub to be in C:\system\install, by specifying deletion of a drive C .ini file on app removal
```

```
""-“C:\system\myapp\myapp.ini”, FN
```

```
// embed .NListbox.sis
```

```
@“NListbox.sis”,(0x10004D3D)
```

Notes

- Replace 'myapp' with the name of your application
- It does not matter if you do not have an .ini file on drive C, as no error will be raised if this file cannot be found when the app is finally removed.
- Any real .ini file in C:\system\myapp\ folder will not be removed during upgrades, only on final removal of the app.

The shared module problem

For those interested here is an overview of why distributing this shared OPX requires the mandatory method detailed in the topic **Distributing the OPX**

- 1 The embedded method of .sis incorporation has limitations, which are of note for shared modules (typically a shared.opx) which are packaged in their own separate shared module.sis file.
- 2 After an installation, a residual .sis stub is created in \system\install\ for the dependant parent .sis and for each of its embedded shared module.sis. These stubs contain important information for use by EPOC Install and the Control panel option 'Add/Remove'.
- 3 To ensure that
 - (a) any shared module is only removed when no other dependant application requires it, and...
 - (b) the user is warned that removing the shared module directly may break other applications, requires every .sis stub for a dependant parent .sis to be in C:\system\install\.

It does not matter on which drive the .sis stub for any embedded shared module.sis is located, nor does it matter on which drive the dependant applications or shared module(s) are located. The key requirement is that the .sis stub for the dependant parent is in C:\system\install\.

4. Unfortunately, this cannot be guaranteed for two reasons:
 - Unlike installations on the machine itself, which always use C:\system\install\, the PC based EPOC Install program places .sis stubs in the \system\install\ folder dependent on the options originally used in the parent.sis .pkg file. If all files can be installed on an optional drive selected by the user, this same drive is used for the .sis stubs. If this is D, then the .sis stubs are placed in D:\system\install\. The same is true if the .pkg file mandates that all files must be on drive D.
 - Users may move .sis stubs from C:\system\install\ to D:\system\install\, partly prompted by what EPOC Install may have done.

When the last dependant parent.sis stub in C:\system\install\ is removed, or there are no dependant parent .sis stub in C:\system\install\, removing a parent app.sis will mean that shared modules will be deleted, or no warning given, even if a dependant app remains on the machine.

So, what does this mean?

When your app depends on a shared module, in order to ensure that the removal of your application does not break someone else's which also uses the same shared module, this three point plan is recommended:

- 1 You exploit the EPOC Install behaviour which forces the parent .sis stub to be placed in C:\system\install\, even if the drive location for all other files was drive D, by choice or by design. This is done by including one option in the parent .pkg file which has a hard-coded drive of C. Some possibilities are:
 - a. To include a real file which is installed on drive C
"some real file"- "C:\some folder\some real file",FF
 - b. To specify a deletion requirement, when the app is removed, for a real or bogus file on drive C
""- "C:\system\apps\myapp\myapp.ini",FN
""- "C:\bogus stub fix",FN
2. You embed the shared module.sis in the parent sis file
@"shared.sis", (0x00000000)
3. Users are advised not to move any parent .sis stubs from C:\system\install\

Error messages

The listbox will raise OPL errors in the following context:

-2 Invalid args

The function arguments are invalid

-7 Out of range

Attempted to access an invalid array entry, or use an invalid array handle

-9 In use

Attempted to modify the contents of the listbox using the Array... methods whilst the array id& is in use by a currently displayed dialog.

-20 Failed to start

The file \system\opx\nlistbox.rsc could not be found

-33 Not exists

Attempting to change or specify non-existent entry data (e.g. bad column%, or buttonId%)

-71 String too long

Changes to the array entry make the entry string too long

-85 Structure error

Calling a listbox construction method before using **LBoxInit**

-99 Procedure not found

Non-existent callbackProcedure\$

-109 Bad file type


Failure to specify **LBoxSetUsesBitmap** after using the **LBoxInit**: flag KNLBoxUsesBitmapColumn%

-118 Drawable not open

Calling a **LBoxDynamic...** method when the Listbox is not displayed

Arrays: Introduction

The **Array...** series of functions are dual purpose.

- 1 They can be used alone to create and manage dynamic string arrays
 - 2 They must be used to create the required array for use with the **LBox...** commands, for displaying a listbox.
-  When using an array with a listbox, the array may be dynamically enquired and edited. To do this **you must use** the appropriate functions from the **LBoxDynamic...** command series, as the **Array...** commands are not available to use with the item array which is currently in use by a displaying listbox.

A listbox also has an option to use a bitmap array. This is distinct from the **Array...** command series. See the topic **Listbox: Overview** for more information.

ARRAYCREATE&:

id&=ArrayCreate&:

Creates an array object and returns a handle id& for it. id& must have global scope if it is used with the listbox and is accessed by a callback function.

ARRAYFREE:

ArrayFree:(BYREF id&)

Deletes the array object with handle id&, freeing all memory resources used, and sets id& to 0

ARRAYADDITEM:

ArrayAddItem:(id&,item\$)

Adds the string Item\$ to the array with handle id&. The string can be split into columns, by the use of CHR\$(9) as the column separator.

The following string has a 3 column representation:

item\$="Col 1"+CHR\$(9)+"Col 2"+CHR\$(9)+"Col 3"

ARRAYADDBITMAPITEM:

ArrayAddBitmapItem:(id&,bitmapIndex%,item\$)

Adds the string item\$ to the array with handle id&, and associates it with the bitmap numbered index%, with 0 for the first, or only, bitmap in the bitmap array (see **LBoxInit**, **LBoxSetBitmapFile** and the **LBoxBArray...** command series) The string items\$ can have a column representation - see **ArrayAddItem**

- When using an array which has a bitmap item entry within a listbox, during the listbox construction:
 - 1 the LBoxInit flag% must be OR'd with KNLBoxUsesBitmapColumn%
 - 2 the bitmap file must have been set with the function **LBoxSetBitmapFile**
 - 3 the listbox flag KNLListBoxAllowIncrementalMatching% is ignored

ARRAYITEMCOUNT%:

total%=ArrayItemCount%:(id&)

Returns the total of the items in the array with handle id&

ARRAYITEMAT\$:

entry\$=ArrayItemAt\$(id&,position%)

Returns a string entry\$ with the contents of the item at position% in the array with handle id&

ARRAYITEMLENGTH%:

length%=ArrayItemLength%:(id&, position%)

Returns the length of the array item at position%, in the array with handle id&

ARRAYAPPENDCOLUMNSEPERATORTOITEM:

ArrayAppendColumnSeparatorToItem:(id&, position%)

Appends CHR\$(9) to the array entry at position%, in the array with handle id&.

- The total length of the array entry must not exceed 255 bytes.

ARRAYAPPENDTEXTTOITEM:

ArrayAppendTextToItem:(id&, position%, textToAppend\$)

Appends the string textToAppends\$ to the array entry at position%, in the array with handle id&.

- The new length of the array entry must not exceed 255 bytes.

ARRAYREPLACEITEM:

ArrayReplaceItem:(id&, position%, replaceWith\$)

Changes the array item at position% to the string replaceWith\$, in the array with handle id&.

ARRAYINSERTITEM:

ArrayInsertItem:(id&, position%, stringToInsert\$)

Inserts the string stringToInsert\$ at position% in the array with handle id&.

ARRAYINSERTSTRINGINTOITEM:

ArrayInsertStringIntoItem:(id&, position%, insertPos%, insert\$)

Inserts the string insert\$, at character position insertPos%, into the array entry position%, in the array with handle id&.

- The new length of the entry must not exceed 255 bytes.

ARRAYREPLACESTRINGINITEM:

ArrayReplaceStringInItem:(id&, position%,replacePos%, replaceWith\$)

Changes the array item at position%, starting at replacePos%, with the string replaceWith\$, in the array with handle id&.

- The new length of the entry must not exceed 255 bytes.

ARRAYINSERTCOLUMNMNDATA:

ArrayInsertColumnData:(id&, position%,column%,insert\$)

Inserts the string insert\$ into the array entry position%, of the array with handle id&. Column% gives the column number.

- insert\$ should not contain a trailing CHR\$(9)

ARRAYEXTRACTCOLUMNMNDATA\$:

data\$=ArrayExtractColumnData\$(id&,position%,column%)

Returns a string data\$ containing the data for column% in entry position% in the array with handle id&

ARRAYDELETEITEM:

ArrayDeleteItem:(id&, position%)

Deletes the array entry at position% in the array with handle id&

ARRAYDELETEITEMS:

ArrayDeleteItems:(id&, startPosition%, endPosition%)

Deletes the array entries from startPosition% to endPosition% (inclusive) in the array with handle id&

ARRAYDELETEALLITEMS:

ArrayDeleteAllItems:(id&)

Deletes all array entries in the array with handle id&. The array object itself is not deleted - see **ArrayFree**

ARRAYDELETECOLUMNMNDATA:

ArrayDeleteColumnData:(id&, position%, column%)

Deletes the entry for column% in the array entry position%, in the array with handle id&

ARRAYDELETECHARSFROMITEM:

ArrayDeleteCharsFromItem:(id&, position%, deletePos%, length%)

Deletes length% characters, starting at deletePos%, from array item position%, in the array with handle id&

ARRAYSORT:

ArraySort:(id&, flag%)

Sorts the array with handle id&, according to the value of flag%

Constants for flag% are: (see NListbox.oxh)

- KNArrySortNormal% = 1
- KNArrySortFolded% = 2
- KNArrySortCollated% = 4

ARRAYFIND%:

position%=ArrayFind%:(id&, searchString\$, flag%)

Returns the position of the array item, in the array with handle id&, where the string searchString\$ has been found. searchString\$ can contain wildcards. position% returns 0 if no match is found.

flag% sets the type of search, where the constants (see NListbox.oxh) are

- KNArryFindSequential% (value 8) is a sequential search
- KNArryFindBinarySearch%(value 16) is a binary search

■ If using binary search, the array must first be sorted

Listbox: Overview

The listbox functions have similar form to the OPL32 commands. A dialog is initialised, configured and then displayed. In addition, the listbox dialog can remain on the screen whilst processing any button presses, and can be dynamically changed whilst still visible.

- 1 This version of NListbox is not intended as a listbox form (i.e. a permanent application window which looks like a listbox). Whilst some simple applications will be able to use NListbox in this manner, it was not designed with this purpose in mind; it is a dialog.
- 2 The listbox requires an array of entries to work with. See the **Array...** series of functions.
- 3 The listbox can be configured to display bitmaps in column 1. The bitmap array commands are prefixed **LBoxBArray...**
- 4 **The listbox functions** are conveniently arranged follows:
 - Commands to construct, configure and display the listbox **LBox...**
 - Commands to construct and configure a bitmap array **LBoxBArray...**
 - Commands to use while the listbox is displayed **LBoxDynamic...**
 - Commands for use with multi-item selections **LBoxSelection...**

□ As the listbox construction commands do not have to be in the same procedure, it is possible to have a number of 'templated' listbox formats in an OPL32 program.

5. Listbox data

ret%=LBoxDisplay%:(BYREF id&, BYREF choice&, initialLine%, constructionCB\$)

- id& is the handle of the array to display in the listbox
- choice& contains the value of the highlighted item when the listbox is closed. If the listbox is dismissed using the **esc** key, choice&=0
- initialLine% determines which array item is to have initial listbox focus
- constructionCB\$ is the name of an optional construction callback procedure, which is called before the listbox displays.
- ret% contains the value of the key used to exit the dialog (13=Enter, 27=Esc) etc.

Procedures are provided to interrogate any entries in the listbox's internal selection array, when the listbox is configured for multiple selections (see **LBoxSelection...** functions and flags for **LBoxInit**).

Listbox: Callbacks

NListbox uses two types of callback procedures:

- A construction callback called before the dialog is displayed (see **LBoxDisplay**)

- Callback procedures optionally attached to any buttons displayed in the dialog (e.g. callBackProc\$ in **LBoxAddButton**). When the button is activated, the listbox calls the associated callback procedure **via a callback manager** in the host OPL32 program.

If a button does not have a callback, the listbox dialog closes and the value of the button is returned as normal by LBoxDisplay%: in the same manner as the OPL32 DIALOG command.

Important

To use a callback, two conditions must be satisfied:

- 1 a callback manager is required in the host OPL32 program
- 2 the callback procedure must conform to a templated design

The callback manager must be **exactly** the same as the following in all respects (names, structure etc.)

```
PROC LBoxCallBackManager%:(item%, procedure$)
    ONERR CallbackError::
    return @%(procedure$):(item%)

    CallbackError::
    ONERR OFF

    ALERT("Callback "+ERRX$,ERR$(err)+" (" +GEN$(err,4)+"")")
ENDP
```

The required template for any callback procedure attached to a button, and called by the callback manager, is:

```
PROC Procedure%:(item%)
    [decare any variables,
    do any processing, display a dialog,
    call other procedures]

    RETURN KNLBoxDialogCanExit% / KNLBoxDialogCannotExit%
    (as appropriate)
ENDP
```

Please also read the topic **Listbox: Callback notes**

Listbox: Callback notes

See **Listbox: Callbacks** for an explanation of callbacks, and the mandatory framework to use them.

- 1 A templated callback procedure must take a single variable of type integer (item%) which is automatically initialised to the value of the current focused item of the array with handle id&, as used in the command **LBoxDisplay**.
- 2 The callback procedure must only return an integer of either **KNLBoxDialogCanExit%** or **KNLBoxDialogCannotExit%** (see NListbox.oxh). These returns dictate whether the listbox will close on returning from the callback procedure.
- 3 In setting a callback procedure for a button, the name required is the procedure name without the variable parameters. So PROC Edit%:(item%) would be **"Edit"**
- 4 Callback procedures may call other procedures in the OPL32 program, but control must always be returned back to the listbox (via the OPL32 callback manager procedure) so the listbox dialog can properly exit and release all the resources it has in use. OPL32 procedures down the callback chain should not leave, STOP, etc.
- 5 There is a need for an error handling harness to workaround a problem of incorrect error values being returned to the OPX by the OPL runtime. This error harness is provided by the OPL32 host program's **callback manager** (see **Listbox: Callbacks**), and ensures OPL error messages are correctly notified to the user for errors in any templated callback procedures, and procedures called from there. It is **very** important that any such sub-procedures do not have any additional error handling of their own.

- 6 It is good practice to verify that callback procedures do not raise errors, before they are run inside your callback manager framework. This will make it much easier to debug your code.
- 7 Whilst other dialogs can be displayed on top of the listbox dialog, the underlying behaviour is still synchronous. While a listbox is displaying, it is not currently possible to receive any system or pointer events (e.g. Getevent, Getevent32, GeteventA32, Testevent). Such instructions should not occur within any callback routine or those called from it.
- 8 It is possible to use synchronous keyboard commands (Get, Get\$) in callback procedures.
- 9 The listbox automatically sets LOCK ON when it is displayed.
- 10 When displayed, the listbox receives all keyboard events. So, whilst a callback procedure can display a menu, key events will not be reported even though EPOC will route pen events to the menu window.

Listbox: Prior to display

The **LBox..** topics are all used to construct, configure and display the listbox. (See the **LBoxDynamic...** topics for functions once the listbox is displayed)

LBOXINIT:

LBoxInit:(title\$,flags%,numCols%)

Important

This function must be called before any other **LBox...** functions are used or else a "Structure fault" error (-85) will be raised.

Prepares for definition of a dialog. Only one listbox dialog can be displayed at one time.

1. If **title\$** is supplied, it will be displayed at the top of the dialog.
2. **flags%** can be any added combination of the following constants to achieve the following effects. Constants for these values are included in the header file NListbox.oxh

- Buttons on the right rather than at the bottom (value 1)
- No title bar (any title in LBoxInit: is ignored) (value 2)
- Allow multiple item selections (value 4)
- Do not allow dialog to be dragged (value 8)
- Allow incremental matching (value 16)

Normally, pressing a key will move the listbox selection to the first, or next, item beginning with that character. With this flag set, in the following array:

"AABBCC"

"ABCD"

"ABBCD"

"AX"

"AAABB"

Pressing "A" would take you to "AABBCC", pressing "A" again would find the next "A", in the same item i.e. "AABBCC", as this has an "AA" combination.

Pressing A again would jump to "AAABB" (because you've entered 3 A's). Likewise if you started again and pressed A followed by B you would jump to item 2, ("ABCD") and if you pressed B again you would jump to item 3 ("ABBCD").

- Manual column sizing (value 32)

Normally the listbox automatically sizes columns based on the widest column entry and the setting of any column font (**LBoxSetColumnFont**). With this flag set, sizing will be based on the values of the width settings in **LBoxSetColumnWidth** or **LBoxSetColumnFontAndWidth**. The listbox calculates widths using the size of the widest character for the particular font.

- Uses bitmap column(value 64)

Only the first column in the listbox may contain a bitmap. Use this flag to configure the dialog to interpret the bitmap data stored in an array using **ArrayAddBitmapItem**

- Listbox to clean up array (value 128)

The listbox will automatically call **ArrayFree**: for the array used with the dialog on closure (see **LBoxDisplay**)

Tip

Without this flag set, it is possible to populate an array once, and re-use it with any number of subsequent listbox constructions. However, in this case the OPL32 program must itself call **ArrayFree** before exiting.

- Listbox to close on item double tap (value 258)

Using the pen, double tapping an item, or selecting an already highlighted item, will cause **LBoxDisplay** to return.

3. **numCols%** configures the maximum columns the listbox dialog may contain.

LBOXCANCEL:

LBoxCancel:()

Abandons the creation of a new listbox dialog before it is displayed using **LBoxDisplay** and frees up all memory resources allocated to it.

LBOXADDBUTTON:

LBoxAddButton:(caption\$, flags%, callbackProc\$, bitFile\$, bitmapId%, bitmapIdMask%, Layout%)

Adds a button to the dialog.

caption\$ is the text to display on the button. **flags%** is the keycode of the shortcut key which can be OR'd with the following:(see OPL32 **dBUTTONS** command)

- display a button with no shortcut label underneath it = 256 (\$100)
- use the key alone (without the Ctrl modification) = 512 (\$200)

The **dBUTTONS** concept of a negative flag% to specify a cancel operation is not supported (e.g. **-(%H OR \$100)**)

callbackProc\$, if specified, sets the procedure in the host OPL32 program which is to be called when this button is pressed (see **Listbox: Overview** for more details).

bitFile\$, if specified, is the path of the file containing the bitmap for the button

bitmapId% specifies the index for the required bitmap in **bitFile\$**, with 0 as the first, or only, item

bitmapMaskId% specifies the index for the required bitmap mask, with a value of -1 if none is required

layout% specifies relative positions of the text and icon on a button

text right = 0

text bottom = 1

text top = 2

text left = 4

LBOXSETPOSITION:

LBoxSetPosition:(x%,y%)

Sets the top left corner of the dialog to x%,y%

LBOXSETSIZE:

LBoxSetSize:(width%,heightInItems%)

Sets the width in pixels, and height in number of items

■ The actual number of lines visible may be different if the listbox array items contain bitmaps

LBOXSETEXTENT:

LBoxSetExtent:(x%,y%,width%,heightInItems%)

Sets the dialog's top left corner to x%,y% and sizes the dialog to width% pixels, heightInItems%

■ The actual number of lines visible may be different if the listbox array items contain bitmaps

LBOXSETCOLUMNFONT:

LBoxSetColumnFont:(column%,fontUid&)

Sets the font for column% to the uid value fontUid& (see Const.oph)

■ The default widths and fonts are 3 chars wide and KFontArialNormal15&

LBOXSETCOLUMNWIDTH:

LBoxSetColumnWidth:(column%,numChars%)

Sets the width of column% to numChars% characters. Has no effect unless the **LBoxInit** flag **KNLBoxManualColumnSizing%** (value 32) has also been set.

The listbox calculates actual widths based on numChars% x maximum character width of fontUid&.

■ The default widths and fonts are 3 chars wide and KFontArialNormal15&

LBOXSETCOLUMNFONTANDWIDTH:

LBoxSetColumnFontAndWidth:(column%,fontUid&,numChars%)

Sets the font for column% to the uid value fontUid& (see Const.oph)

Sets the width of column% to numChars% characters. Has no effect unless the **LBoxInit** flag **KNLBoxManualColumnSizing%** (value 32) has also been set.

The listbox calculates actual widths based on numChars% x maximum character width of fontUid&.

■ The default widths and fonts are 3 chars wide and KFontArialNormal15&

LBOXSETUSESBITMAPS:

LBoxSetUsesBitmaps:(Id&)

Notifies the Listbox of the bitmap array (id&) to use when displaying an **ArrayAddBitmapItem** entry. The bitmap array must first have been created and configured using the **LBoxBArray...** commands.

The **LBoxInit** flag **KNLBoxUsesBitmapColumn%** (value 64) must also be set.

Bitmap arrays

The following **LBoxBArray...** commands are for the creation and manipulation of a bitmap array for use in a bitmap column

LBOXBARRAYCREATE&:

id&=LBoxBArrayCreate&:()

Creates a bitmap array object and returns a handle id& for it.

LBOXBARRAYADDITEM:

LBoxBArrayAddItem:(id&, windowId%, bitmapFile\$, Index%)

Adds an item to the bitmap array with handle id&.

If windowId% is supplied, as returned by gCREATE or gCREATEBIT, the item added will be the contents of windowId%

If bitmapFile\$ is supplied, the bitmap at position index% in the file bitmapFile\$, will be added. index%=0 for the first, or only, item.

LBOXBARRAYINSERT:

LBoxBArrayInsert:(id&, position%, windowId%, bitmapFile\$, index%)

Inserts a new item into the bitmap array with handle id& at position%

If windowId% is supplied, as returned by gCREATE or gCREATEBIT, the item added will be the contents of windowId%

If bitmapFile\$ is supplied, the bitmap at position index% in the file bitmapFile\$, will be added.
index%=0 for the first, or only, item.

LBOXBARRAYDELETE:

LBoxBArrayDelete:(id&, position%)

deletes the item at position% in the bitmap array with handle id&

LBOXBARRAYFREE:

LBoxBArrayFree:(BYREF Id&)

Deletes the bitmap array object with handle id&, freeing all memory resources used, and sets id& to 0

LBOXBARRAYGET:

LBoxBArrayGet:(Id&, Position%, WindowId%, posX%, posY%, File\$)

Retrieves the bitmap at position% in the bitmap array with handle id&

If windowId% is supplied, as returned by gCREATE, the bitmap will be drawn in the window, with the top left corner at posX%, posY%.

If File\$ is supplied, the bitmap will be saved to the file specified as File\$. Any existing File\$ will be replaced. WindowId% must be set to 0.

LBOXBARRAYCOUNT%:

Total%=LBoxBArrayCount%:(Id&)

returns the total number of bitmaps in the bitmap array with handle id&

LBOXBARRAYSIZE:

LBoxBArraySize:(Id&, Position%, BYREF width%, BYREF Height%)

returns the size of the bitmap, at position% in the bitmap array with handle id&, in the BYREF variables

LBOXDISPLAY%:

ret%=LBoxDisplay%:(BYREF id&, BYREF choice&, initialLine%, ConstructionCB\$)

Displays the list box, using the items in the array with handle id&, setting the item initialLine% as the focused item.

ConstructionCB\$ is the name of a callback procedure which is called prior to the dialog being drawn. Its use allows, for example, conditional configuration of dialog buttons, depending on the status of other factors.

Notes

- 1 On return, ret% contains the keycode of the event which closed the dialog.
- 2 **For a buttonless dialog**, 0 is returned if the dialog was cancelled, 13 if **enter** was pressed, or the dialog was closed by a double tap event (with the **LBoxInit** flag KNLBoxReturnOnDoubleTap% (value 258) specified.)
- 3 **If any buttons are specified**, 13 will only be returned if there is a button set with a value of 13. If a button using 13 has a callback, the button will animate and the callback will be called. The same applies to the **esc** key value 27
- 4 The flag KNLBoxReturnOnDoubleTap% maps to the **enter** keycode 13,

- 5 If a button has an associated callback procedure, `ret%` will contain the button code only if the callback procedure allows the dialog to close (i.e. the callback procedure returns `KLBoxDialogCanExit%` (value 1).) Otherwise the dialog will not be dismissed.
- 6 **choice&** contains the value of the item highlighted. 0 will always be returned if **esc** was used, regardless of whether there is a button defined to take the **esc** keycode.
- 7 If the **LBoxInit** flag `KNLBoxAllowMultipleSelection%` was set, the contents of the selection array can be enquired using the **LBoxSelection...** series of commands.
- 8 If the **LBoxInit** flag `KNLBoxFreeMemoryAfterDisplay%` (value 128) was set, the memory for array `Id&` will be released and `id&` will equal 0 upon return from the dialog.

Listbox: While displayed

The **LBoxDynamic...** series of functions are for use inside callback procedures only. Where relevant, they take effect on the array with handle `id&` used in the function **LBoxInit**. See the **LBox...** functions for pre-display construction and configuration.

Dynamic changes can only be conducted within the scope of the flag settings used to initialise, construct and display the dialog (For example, number of columns, whether an array bitmapitem can be added, etc).

Notes

- After any use of a **LBoxDynamic...** function which could have altered the column widths, the function **LBoxDynamicUpdateColumnWidths** may be called. If it is not used, the dialog may not resize the scrollbars correctly after an addition, insertion or deletion.
- The listbox is automatically refreshed on return from a callback procedure. However, the dialog can be refreshed earlier if required, by using the command **LBoxDynamicDrawListboxNow**
- When a change has occurred which affects the underlying listbox data (such as an addition or deletion), the cursor is automatically re-set to the first item in the listbox. It can be moved back to any desired location using the **LBoxDynamicSetCurrentItem** call.
- Dynamically sorting the array resets any selection indices.

LBOXDYNAMICSCROLLTOITEM:

`LBoxDynamicScrollToItem:(position%)`

Scrolls the array item at `position%`, into the listbox view. `position%` is not selected.

LBOXDYNAMICADDITEM:

`LBoxDynamicAddItem:(item$)`

Adds the entry `item$` to the array.

LBOXDYNAMICADDBITMAPITEM:

`LBoxDynamicAddBitmapItem:(bitmapIndex%, Item$)`

Adds the bitmap entry `item$`, which uses `bitmapIndex%` in the previously declared bitmap array, to the listbox item array.

LBOXDYNAMICINSERTITEM:

`LBoxDynamicInsertItem:(position%, insert$)`

Inserts the entry `insert$` into the array before `position%`.

LBOXDYNAMICREPLACEITEM:

`LBoxDynamicReplaceItem:(itemToReplace%, replaceWithString$)`

Replaces the array entry at `itemToReplace%` with `replaceWithString$`.

LBOXDYNAMICDELETEITEM:

`LBoxDynamicDeleteItem:(itemToDelete%)`

Deletes the array entry itemToDelete%.

LBOXDYNAMICDELETEITEMS:

LBoxDynamicDeleteItems:(startPosition%, finishPosition%)

Deletes the array entries from startPosition% to finishPosition%.

LBOXDYNAMICDELETEALLITEMS:

LBoxDynamicDeleteAllItems: ()

Deletes all the array entries.

■ The array object itself is not deleted-see **ArrayFree**

LBOXDYNAMICITEMLENGTH%:

length%=LBoxDynamicItemLength%:(position%)

returns the length of the array item at position%

LBOXDYNAMICINSERTSTRINGINTOITEM:

LBoxDynamicInsertStringIntoItem:(position%, insertPos%,insert\$)

Inserts the string insert\$, at character position insertPos%, into the array entry position%

■ The new length of the entry must not exceed 255 bytes.

LBOXDYNAMICREPLACESTRINGINITEM:

LBoxDynamicReplaceStringInItem:(position%,replacePos%,replaceWith\$)

Changes the array item at position%, starting at replacePos%, with the string replaceWith\$

■ The new length of the array item must not exceed 255 bytes

LBOXDYNAMICDELETECHARSFROMITEM:

LBoxDynamicDeleteCharsFromItem:(position%,deletePos%, length%)

Deletes length% characters, starting at deletePos%, from array item position%

LBOXDYNAMICITEMCOUNT%:

total%=LBoxDynamicItemCount%:()

Returns the total items in the array

LBOXDYNAMICITEMAT\$:

entry\$=LBoxDynamicItemAt\$:(position%)

Returns a string entry\$ with the contents of the item at position% in the array

LBOXDYNAMICSORT:

LBoxDynamicSort:(flags%)

Sorts the array according to the value of flag%

Constants for flag% are: (see NListbox.oxh)

- KNAArraySortNormal% = 1
- KNAArraySortFolded% = 2
- KNAArraySortCollated% = 4

■ Dynamically sorting the array resets any selection indicies.

LBOXDYNAMICFIND%:

position%=LBoxDynamicFind%:(searchString\$, flag%)

Returns the position of the array item where the string searchString\$ has been found.
searchString\$ can contain wildcards. position% returns 0 if no match is found.

flag% sets the type of search, where the constants (see NListBox.oxh) are

- KNAryFindSequential% (value 8) is a sequential search
- KNAryFindBinarySearch%(value 16) is a binary search

■ If using binary search, the array must first be sorted

LBOXDYNAMICEXTRACTCOLUMNDATA\$:

data%=LBoxDynamicExtractColumnData\$(position%,column%)

Returns a string data\$ containing the data for column% in entry position% in the array

LBOXDYNAMICSETCURRENTITEM:

LBoxDynamicSetCurrentItem(position%)

Makes the array item position% the focused selection.

LBOXDYNAMICUPDATECOLUMNWIDTHS:

LBoxUpdateColumnWidths:

Updates the listbox view model

■ This function may be called after any use of a **LBoxDynamic...** function which could have altered the column widths. If it is not used, the dialog may not resize the scrollbars correctly after an addition, insertion or deletion

LBOXDYNAMICDRAWLISTBOXNOW:

LBoxDynamicDrawListBoxNow:

Redraws the listbox

LBOXDYNAMICSETBUTTONSTATE:

LBoxDynamicSetButtonState:(buttonId%, state%)

Changes the state of the listbox button with keycode buttonid%, according to the value of state%

- KNDlgButtonStateNotDimmed% = 1
- KNDlgButtonStateDimmed% = 2
- KNDlgButtonStateInvisible% = 3
- KNDlgButtonStateVisible% = 4

LBOXDYNAMICSETDIALOGTITLE:

LBoxDynamicSetDialogTitle:(title\$)

Changes the dialog title to the string title\$

LBOXDYNAMICTOPLEFT:

LBoxDynamicTopLeft(BYREF x%, BYREF y%)

returns the x and y co-ordinates of the top left corner of the listbox in the BYREF variables

LBOXDYNAMICWIDTH%:

width%=LBoxDynamicWidth%:()

returns the width (in pixels) of the listbox

LBOXDYNAMICHEIGHT%:

height%=LBoxDynamicHeight%:()

returns the height (in pixels) of the listbox

LBOXDYNAMICBUTTONCORNER:

LBoxDynamicButtonCorner:(id%, corner%, BYREF x%, BYREF y%)

returns the pixel co-ordinates for the button id% for the location specified by corner% where:

- **id%** is the keycode for the button, as declared in LBoxAddButton **but without any OR'd modifier** (\$100 or \$200)
For example, if the LBoxAddButton keycode% parameter was %E OR \$100, the id% parameter would be just %E
- **corner%** is one of
 - KNLBoxTopLeft% = 0
 - KNLBoxBottomLeft% = 1
 - KNLBoxTopRight% = 2
 - KNLBoxBottomRight% = 3

Listbox: Multiple selection functions

If the listbox is not cancelled using the **esc** key, choice& (see **LBoxDisplay**) returns the position in the array id& of the focused item.

By setting the **LBoxInit** flag KNLBoxAllowMultipleSelection% (value 4), it is possible for a number of items to be selected. The listbox itself maintains an array of the selections, which can be enquired when **LBoxDisplay** returns, and additionally from inside a callback function.

Notes

- **The selection array** is empty if the listbox is dismissed with the **esc** key, or no multiple selections were made.
- The selection array is reset upon calling **LBoxDisplay**

LBOXSELECTIONCOUNT%:

total%=LBoxSelectionCount%:

returns the total number of the selected items

LBOXSELECTIONINDEXAT%:

arrayPosition%=LBoxSelectionIndexAt%:(position%)

returns the array entry index value at position% in the selection array.

Note

This function provides the opportunity to iterate through the selection array as follows:

LOCAL total%, count%, arrayPosition%

count%=1

total%=LBoxSelectionCount%:

WHILE count% <= total%

arrayPosition% = LBoxSelectionIndexAt%:(count%)

...

<process the array item at arrayPosition%>

...

count%=count%+1

LBOXSELECTIONITEMAT\$:

entry\$=LBoxSelectionItemAt\$(id&, position%)

returns the entry details for the item in the array with handle id&, which corresponds to position% in the selected items array.

LBOXSELECTIONEXTRACTCOLUMNDATA\$:

details\$=LBoxSelectionExtractColumnData\$(id&, position%, column%)

returns the column% details for the item in the array with handle id&, which corresponds to position% in the selected items array.

LBOXSELECTIONSORT:

LBoxSelectionSort()

sorts the selection index array

- By default, the selection array is ordered based upon the order in which items were selected in the listbox.

For example, if item 4 was selected, followed by item 2, followed by item 1 then the array contents would be (4,2,1). This function orders the selection indices into ascending order (1,2,4).

```

rem NListbox.OXH
rem Header File for NListbox.OPX
rem
rem © Copyright Alex Wilbur, Henry Hirst and Neuron 1999

rem Bitwise dialog flags for LBoxInit
const KNLBoxButRight%           = 1
const KNLBoxNoTitle%           = 2
const KNLBoxAllowMultipleSelection% = 4
const KNLBoxNoDrag%            = 8
const KNLBoxAllowIncrementalMatching% = 16
const KNLBoxManualColumnSizing% = 32
const KNLBoxUsesBitmapColumn%  = 64
const KNLBoxFreeMemoryAfterDisplay% = 128
const KNLBoxReportDoubleTapEvents% = 256

rem Bitwise flags for ArraySort: & LBoxDynamicArraySort:
const KNLArraySortNormal%       = 1
const KNLArraySortFolded%      = 2
const KNLArraySortCollated%    = 4

rem Bitwise flags for ArrayFind%: & LBoxDynamicArrayFind%:
const KNLArrayFindSequential%   = 8
const KNLArrayFindBinarySearch% = 16

rem Array Entry column separator = CHR$(9)

rem Button layout constants
const KLBoxBtLayoutTextRightPicLeft% = 0
const KLBoxBtLayoutTextBottomPicTop% = 1
const KLBoxBtLayoutTextTopPicBottom% = 2
const KLBoxBtLayoutTextLeftPicRight% = 3

rem Button state constants
const KNLBoxButtonStateNotDimmed% = 1
const KNLBoxButtonStateDimmed%    = 2
const KNLBoxButtonStateInvisible% = 3
const KNLBoxButtonStateVisible%   = 4

rem Button corner constants
const KNLBoxTopLeft%              = 0
const KNLBoxBottomLeft%           = 1
const KNLBoxTopRight%             = 2

```

```

const KNLBoxBottomRight%           = 3

rem Dialog exit constants
const KNLBoxDialogCanExit%         = 1
const KNLBoxDialogCannotExit%      = 0

rem OPX constants
const KOpXUidNLBox&                = &10004D3D
const KOpXVersionNLBox%            = $100

```

```

DECLARE OPX NLISTBOX, KOpXUidNLBox&, KOpXVersionNLBox%

```

```

REM =====

```

```

REM Dynamic string arrays

```

```

    ArrayCreate&:( ) : 1
    ArrayAddItem:(Id&, Item$) : 2
    ArrayAddBitmapItem:(Id&, BitmapNumber%, Item$) : 3
    ArrayInsertItem:(Id&, Position%, StringToInsert$) : 4
    ArrayReplaceItem:(Id&, Index%, ReplaceWith$) : 5
    ArrayDeleteItem:(Id&, Index%) : 6
    ArrayDeleteItems:(Id&, StartIndex%, FinishIndex%) : 7
    ArrayDeleteAllItems:(Id&) : 8
    ArrayItemLength%:(Id&, Index%) : 9
    ArrayAppendTextToItem:(Id&, Index%, TextToAppend$) : 10
    ArrayAppendColumnSeparatorToItem:(Id&, Index%) : 11
    ArrayInsertStringIntoItem:(Id&, Index%, InsertPos%, Insert$) : 12
    ArrayReplaceStringInItem:(Id&, Index%, replacePos%, ReplaceWith$) : 13
    ArrayDeleteCharsFromItem:(Id&, Index%, DeletePos%, length%) : 14
    ArrayItemCount%:(Id&) : 15
    ArrayItemAt$(Id&, Index%) : 16
    ArrayFree:(BYREF Id&) : 17
    ArraySort:(Id&, Flags%) : 18
    ArrayFind%:(Id&, SearchString$, Flags%, BYREF found%) : 19
    ArrayExtractColumnData$(Id&, Index%, Column%) : 20
    ArrayInsertColumnData:(Id&, Index%, Column%, Insert$) : 21
    ArrayDeleteColumnData:(Id&, Index%, Column%) : 22
    ArrayMoveItem:(Id&, FromIndex%, ToIndex%) : 23

```

```

REM =====

```

```

REM LISTBOX

```

```

REM Construction, configuration and display

```

LBoxInit:(Title\$, Flags%, ColumnCount%) : 41
 LBoxCancel:() : 42
 LBoxAddButton:(Caption\$, Flags%, CallBackProc\$, File\$, IndexMain%, IndexMask%,
 Layout%) : 43
 LBoxSetPosition:(TopLeftX%, TopLeftY%) : 44
 LBoxSetSize:(Width%, HeightInItems%) : 45
 LBoxSetExtent:(TopLeftX%, TopLeftY%, Width%, HeightInItems%) : 46
 LBoxSetColumnFont:(Column%, Font&) : 47
 LBoxSetColumnWidth:(Column%, Width%) : 48
 LBoxSetColumnFontAndWidth:(Column%, Font&, Width%) : 49
 LBoxSetUsesBitmaps:(Id&) : 50
 LBoxDisplay%:(BYREF Id&, BYREF Choice&, InitialLine%, ConstructionCB\$) : 51

REM =====

REM Dynamic functions when the listbox is displaying

LBoxDynamicScrollToItem:(Index%) : 61
 LBoxDynamicAddItem:(Item\$) : 62
 LBoxDynamicAddBitmapItem:(bitmapIndex%, Item\$) : 63
 LBoxDynamicInsertItem:(Index%, Insert\$) : 64
 LBoxDynamicReplaceItem:(IndexToReplace%, ReplaceWithString\$) : 65
 LBoxDynamicDeleteItem:(IndexToDelete%) : 66
 LBoxDynamicDeleteItems:(StartIndex%, FinishIndex%) : 67
 LBoxDynamicDeleteAllItems:() : 68
 LBoxDynamicItemLength%:(Index%) : 69
 LBoxDynamicInsertStringIntoItem:(Index%, InsertPos%, Insert\$) : 70
 LBoxDynamicReplaceStringInItem:(Index%, replacePos%, replaceWith\$) : 71
 LBoxDynamicDeleteCharsFromItem:(Index%, DeletePos%, DeleteCount%) : 72
 LBoxDynamicItemCount%:() : 73
 LBoxDynamicItemAt\$:(Index%) : 74
 LBoxDynamicSort:(Flags%) : 75
 LBoxDynamicFind%:(SearchString\$, flags%, BYREF found%) : 76
 LBoxDynamicExtractColumnData\$:(Index%, Column%) : 77
 LBoxDynamicSetCurrentItem:(Index%) : 78
 LBoxDynamicDrawListBoxNow:() : 79
 LBoxDynamicSetButtonState:(ButtonId%, State%) : 80
 LBoxDynamicSetDialogTitle:(Title\$) : 81
 LBoxDynamicUpdateColumnWidths:() : 82
 LBoxDynamicTopLeft:(BYREF X%, BYREF Y%) : 83
 LBoxDynamicWidth%:() : 84
 LBoxDynamicHeight%:() : 85
 LBoxDynamicButtonCorner:(Id%, Corner%, BYREF X%, BYREF Y%) : 86
 LBoxDynamicMoveItem:(FromIndex%, ToIndex%) : 87

LBoxDynamicSelectAll(): 88

REM =====

REM Multi-selection enquiry functions

LBoxSelectionCount%(): 101

LBoxSelectionIndexAt%:(Index%) : 102

LBoxSelectionItemAt\$(Id&, Index%) : 103

LBoxSelectionExtractColumnData\$(Id&, Index%, Column%) : 104

LBoxSelectionSort(): 105

REM =====

REM bitmap array methods

LBoxBArrayCreate&(): 121

LBoxBArrayAddItem:(Id&, WindowId%, File\$, ImageIndex%) : 122

LBoxBArrayInsert:(Id&, Position%, WindowId%, File\$, ImageIndex%) : 123

LBoxBArrayDelete:(Id&, Position%) : 124

LBoxBArrayGet:(Id&, Position%, WindowId%, PosX%, PosY%, Files\$) : 125

LBoxBArrayCount%:(Id&) : 126

LBoxBArraySize:(Id&, Position%, BYREF Width%, BYREF Height%) : 127

LBoxBArrayFree:(BYREF Id&) : 128

END DECLARE

```
INCLUDE "Const.oph"  
INCLUDE "NListBox.oxh"
```

REM NListBox demonstration

```
REM requires NListBox demo.mbm in the same folder  
REM Non-optimised code for illustrative purposes  
REM Creates an 11 item listbox with a graphical column  
REM and illustrates callback routines  
REM In this code, callbacks end with CB
```

PROC Main:

```
GLOBAL id&                REM the entry array handle  
LOCAL bitarray&           REM the bitmap array handle
```

```
GLOBAL NotDimmed%, NotVisible% REM button status flags
```

```
LOCAL i%, count%, selected&  
LOCAL returned%, flags% , off%(6)  
LOCAL mbm$(255)
```

```
rem set up the bitmap file  
mbm$=(PARSE$("nlistbox demo.mbm",CMD$(1),off%()))  
IF NOT(EXIST(mbm$))  
    LOCK ON  
    ALERT(mbm$,ERR$(-33))  
    LOCK OFF  
    STOP  
ENDIF
```

```
rem initialise and populate the bitmap array  
print "Creating bitmap array"
```

```
bitarray&=LBoxBArrayCreate&:  
DO  
    LBoxBArrayAddItem(bitarray&, 0, mbm$, i%)  
    i%=i%+1  
    GIPRINT "Adding bitmap "+NUM$(i%,3),2  
UNTIL i%=21
```

```
rem create an entry array and populate it with column entries  
rem with the first column using a bitmap
```

```
print "Creating dynamic text array"
```

```

id& = ArrayCreate&:
ArrayAddBitmapItem(id&, 0, "Aitmap Line 1"+chr$(9)+"Col
3"+chr$(9)+"123456789#####123456789")
ArrayAddBitmapItem(id&, 2, "Bitmap Line 2"+chr$(9)+"Col 3"+chr$(9)+"Col 4")
ArrayAddBitmapItem(id&, 4, "Citmap Line 3"+chr$(9)+"Col 3"+chr$(9)+"Col 4")
ArrayAddBitmapItem(id&, 6, "Ditmap Line 4"+chr$(9)+"Col 3"+chr$(9)+"Col 4")
ArrayAddBitmapItem(id&, 8, "Eitmap Line 5"+chr$(9)+"Col 3"+chr$(9)+"Col 4")
ArrayAddBitmapItem(id&, 10, "Bitmap Line 6"+chr$(9)+"Col 3"+chr$(9)+"Col 4")
ArrayAddBitmapItem(id&, 12, "Xitmap Line 7"+chr$(9)+"Col 3"+chr$(9)+"Col 4")
ArrayAddBitmapItem(id&, 14, "Bitmap Line 8"+chr$(9)+"Col 3"+chr$(9)+"Col 4")
ArrayAddBitmapItem(id&, 16, "Mitmap Line 9"+chr$(9)+" "+chr$(9)+"Col 4")
ArrayAddBitmapItem(id&, 18, "Qitmap Line 10")
ArrayAddBitmapItem(id&, 20, "Zitmap Line 11")

```

rem Initialise and configure the listbox

rem set the required flags

```

flags% = KNLBoxAllowMultipleSelection%
flags% = flags% OR KNLBoxUsesBitmapColumn%
flags% = flags% OR KNLBoxButRight%

```

rem Initialise the dialog

print "Initialising the listbox"

LBoxInit("NListbox from Neuon", flags%, 4)

rem Add the required buttons

```

LBoxAddButton("Entry"+CHR$(133),%E,"EntryCB",mbm$,20,21,0)
LBoxAddButton("Tests"+CHR$(133),%T,"TestCB","",0,0,0)
LBoxAddButton("Dimmed",%D,"DimCB","",0,0,0)
LBoxAddButton("Done",13 OR $100,"","",0,0,0)

```

rem Set the column widths and fonts

```

LBoxSetColumnFontAndWidth:(1, KFontArialBold13&, 2)
LBoxSetColumnFontAndWidth:(2, KFontArialNormal11&, 14)
LBoxSetColumnFontAndWidth:(3, KFontArialNormal15&, 5)
LBoxSetColumnFontAndWidth:(4, KFontArialNormal11&, 5)

```

rem set size and number of lines to show

LBoxSetSize:(gwidth/2-50,8)

rem Set the bitmap array

LBoxSetUsesBitmaps:(bitarray&)

rem Set dialog position

LBoxSetPosition:(gwidth/2-85,0)

rem set up screen information

cls

print "NListbox test code"

print REPT\$("-",20)

about:

rem Display dialog with line 1 focused

rem listbox automatically sets LOCK ON when it is displaying

returned%=LBoxDisplay%:(id&, selected&, 1, "ConstructionCB")

rem report results

cls

print "NListbox test code"

print REPT\$("-",20)

print "returned%=LBoxDisplay%:(id&,choice&,initialLine%,constructionCB%)"

print

print "returned% =", returned%

print "Selected line is:",selected&, "(Choice&)"

count% = LBoxSelectionCount%:

LBoxSelectionSort:

i% = 1

print "Multiple selection count =",count%

print

while i%<=count%

 print "Multiple selection ";i%;" is array item:", LBoxSelectionIndexAt%:(i%)

 print "which is:",CHR\$(34)+LBoxSelectionItemAt\$(id&, i%)+CHR\$(34)

 print

 i%=i%+1

endwh

print

rem delete the bitmap array

if bitarray&

```

        LBoxBArrayFree:(bitarray&)
    endif

    rem delete the entry array
    if id&
        ArrayFree:(id&)
    endif

    print "press any key to quit"
    get
ENDP

PROC About:
    print "Make multiple selections by:"
    print " * Dragging"
    print " * Ctrl+Click"
    print " * Shift+Click for a range"
    print " * Toggling with the spacebar"
ENDP

REM *****
REM This next procedure is the MANDATORY callback manager
REM if you use callback procedures
REM It must be identical to this
REM If not you will get an "Unknown error"
REM reported
REM *****

PROC LBoxCallBackManager%:(item%, procedure$)
    ONERR CallbackError::

    return @%(procedure$):(item%)

    CallbackError::
        ONERR OFF
        ALERT("Callback "+ERRX$,ERR$(ERR)+" ("+"GEN$(ERR,4)+"")")
ENDP

PROC ConstructionCB%:(NotUsed%)
    dINIT "Construction callback"
    dTEXT "", "This dialog was displayed using", 2
    dTEXT "", "the construction callback procedure", 2
    dTEXT "", "This is called before the dialog is displayed", 2

```

```

dtext "" and allows for conditional dialog configurations",2
dbuttons "Continue",13 OR $100
dialog

```

```

ENDP

```

```

PROC DimCB%:(index%)

```

```

    GIPRINT "Use "+CHR$(34)+"Tests"+CHR$(34)+" and select 'Dimmed'",KBusyTopRight%
    return KNLBoxDialogCannotExit%

```

```

ENDP

```

```

PROC TestCB%:(index%)

```

```

    LOCAL ret%, total%, NoGoto%
    LOCAL key%(4)
    LOCAL x%,y%, corner%
    LOCAL oldwin%

```

```

    REM *****

```

```

    REM This is a templated callback proc

```

```

    REM which calls other procedures

```

```

    REM It is in the form ProcedureName%:(index%)

```

```

    REM Must take an integer only (the current focus item)

```

```

    REM And return ONLY KNLBoxDialogCannotExit% or

```

```

    REM KNLBoxDialogCanExit% as appropriate

```

```

    REM

```

```

    REM All error handling in the callback, or procedures

```

```

    REM down the callback chain, are handled by the

```

```

    REM callback manager

```

```

    REM

```

```

    REM *****

```

```

    rem Get test button positioning details

```

```

    LBoxDynamicButtonCorner:(%T, KNLBoxTopLeft%, x%, y%)

```

```

    rem confirm which side of the button to

```

```

    rem display the pop-up

```

```

    oldwin%=gidentity

```

```

    guse 1

```

```

    if x% >=gwidth/2

```

```

        corner%=KNLBoxTopRight%

```

```

        x%=x%-5

```

```

    else

```

```

        LBoxDynamicButtonCorner:(%T, KNLBoxTopRight%, x%, y%)

```

```

        corner%=KNLBoxTopLeft%
        x%=x%+5
    endif
    guse oldwin%

    rem do check to configure the popup
    total%=LBoxDynamicItemCount%: REM check item array total
    If total%<2
        NoGoto%=1
    ENDIF

    rem configure the popup key values
    key%(1)=1 OR $0800 REM has a checkbox
    key%(2)=2 OR $0800 REM has a checkbox
    key%(3)=3
    key%(4)=4

    ret%=mPOPUP(x%,y%,corner%,"Dimmed",key%(1) OR ($2000 *
NotDimmed%),"Invisible",key%(2) OR ($2000 * NotVisible%),"Raise
error",key%(3),"Goto"+CHR$(133),key%(4) OR ($1000 * NoGoto%))
    IF ret%<>0
        @("Choice"+GEN$(ret%,2)):(index%)
    ENDIF
    return KNLBoxDialogCannotExit%
ENDP

PROC Choice1:(index%)
    REM This procedure makes no use of index%, which is
    REM the currently focused entry in the array

    IF NotDimmed%=0
        LBoxDynamicSetButtonState:(%D, KNLBoxButtonStateDimmed%)
    ELSE
        LBoxDynamicSetButtonState:(%D, KNLBoxButtonStateNotDimmed%)
    ENDIF
    NotDimmed%=1-NotDimmed%
ENDP

PROC Choice2:(index%)
    If NotVisible%=0
        LBoxDynamicSetButtonState:(%D, KNLBoxButtonStateInvisible%)
    ELSE
        LBoxDynamicSetButtonState:(%D, KNLBoxButtonStateVisible%)
    ENDIF

```

NotVisible%=1-NotVisible%

ENDP

PROC Choice3:(index%)

RAISE -123

ENDP

PROC Choice4:(index%)

local max% , newLocation&

local new%, ret%

REM An example of another dialog, but

REM called via a callback procedure

max% = LBoxDynamicItemCount%:

newLocation& = index%

if max%=0

return

endif

dINIT "Goto"

dLONG newLocation&, "Entry",1,max%

dBUTTONS "Cancel",- (27 OR \$100), "OK",13 OR \$100

IF DIALOG=13

new% = newLocation&

LBoxDynamicSetCurrentItem:(new%)

ENDIF

ENDP

PROC EntryCB%:(index%)

LOCAL total%, ret%, NoOptions%

LOCAL x%, y%, corner%

LOCAL oldwin%

rem Get entry button positioning details

LBoxDynamicButtonCorner:(%E, KNLBoxTopLeft%, x%, y%)

rem confirm which side of the button to

rem display the pop-up

oldwin%=gidentity

guse 1

if x% >=gwidth/2

corner%=KNLBoxTopRight%

```

        x%=x%-5
    else
        LBoxDynamicButtonCorner:(%E, KNLBoxTopRight%, x%, y%)
        corner%=KNLBoxTopLeft%
        x%=x%+5
    endif
guse oldwin%

total%=LBoxDynamicItemCount%: REM check if item array is empty
if total%=0
    NoOptions%=1
endif

    ret%=mPOPUP(x%,y%,corner%,"Add"+CHR$(133),%a,"Edit"+CHR$(133),%e OR ($1000
* NoOptions%),"Delete"+CHR$(133),%d OR ($1000 * NoOptions%))
    if ret%<>0
        @("Entry"+CHR$(ret%)):(index%)
    endif
    return KNLBoxDialogCannotExit%
ENDP

```

```

PROC EntryA:(index%)
    LOCAL entry$(255)
    LOCAL a$(255),b$(255),c$(255),d$(255),bitindex&

    REM Add an entry

    WHILE 1
        dINIT "Add entry"
        dLONG bitindex&,"Bitmap index",0,21
        dEDIT b$,"Col 2",20
        dEDIT c$,"Col 3",20
        dEDIT d$,"Col 4",20
        dBUTTONS "Cancel",- (27 OR $100),"OK",13 OR $100
        IF DIALOG
            a$=NUM$(bitindex&,3)

            rem check the length of the entry
            IF TooLong:(a$,b$,c$,d$,4)
                GIPRINT ERR$(-112),KBusyTopRight%
                CONTINUE
            ENDIF

```

rem check for CHR\$(9) in text which will cause column problems

```

IF HasTab:(b$,c$,d$)
    GIPRINT ERR$(-78),KBusyTopRight%
    CONTINUE
ENDIF

```

```

entry$=a$+CHR$(9)+b$+CHR$(9)+c$+CHR$(9)+d$
LBoxDynamicAddItem:(entry$)

```

REM in case new entry is now the widest entry, reset the col widths

LBoxDynamicUpdateColumnWidths:

REM set the listbox focus on the newly added item

REM without this line, focus is set to line 1

```

LBoxDynamicSetCurrentItem:(LBoxDynamicItemCount%:)

```

```

BREAK

```

```

ELSE

```

```

    BREAK

```

```

ENDIF

```

```

ENDWH

```

```

ENDP

```

```

PROC EntryE:(index%)

```

```

    LOCAL entry$(255)

```

```

    LOCAL a$(255),b$(255),c$(255),d$(255)

```

```

    LOCAL bitindex&, d%, selection%

```

rem Edit the entry

rem As this is a multi-selection listbox

rem get the selection information:

```

selection%=LBoxSelectionCount%:

```

```

IF selection%=0

```

```

    GIPRINT "No item selected",KBusyTopRight%

```

```

    return

```

```

endif

```

rem edit the item

rem extract the array item data

```

a$=LBoxDynamicExtractColumnData$(index%,1)
bitindex&=INT(VAL(a$))
b$=LBoxDynamicExtractColumnData$(index%,2)
c$=LBoxDynamicExtractColumnData$(index%,3)
d$=LBoxDynamicExtractColumnData$(index%,4)

```

```

WHILE 1

```

```

    rem dialog to edit the data
    dINIT "Edit entry "
        dLONG bitindex&,"Bitmap index",0,21
    dEDIT b$,"Col 2",20
    dEDIT c$,"Col 3",20
    dEDIT d$,"Col 4",20
    dBUTTONS "Cancel",- (27 OR $100),"OK",13 OR $100
    IF DIALOG

```

```

        a$=NUM$(bitindex&,3)
        rem check the length
        IF TooLong:(a$,b$,c$,d$,4)
            GIPRINT ERR$(-112),KBusyTopRight%
            CONTINUE
        ENDIF

```

```

        rem check for CHR$(9) in text which will cause column problems
        IF HasTab:(b$,c$,d$)
            GIPRINT ERR$(-78),KBusyTopRight%
            CONTINUE
        ENDIF

```

```

        entry$=a$+CHR$(9)+b$+CHR$(9)+c$+CHR$(9)+d$
    LBoxDynamicReplaceItem:(index%, entry$)

```

```

    REM in case new entry is now the widest entry, reset the col widths
    LBoxDynamicUpdateColumnWidths:

```

```

    REM set focus on the edited item
    REM without this line, focus is set to line 1

        LBoxDynamicSetCurrentItem:(index%)
    BREAK
    ELSE
        BREAK
    ENDIF
ENDWH

```


ENDP

PROC EntryD:(index%)

LOCAL total%, newpos%, selection%, selectionCount%

REM Delete item(s)

rem As this is a multi-selection listbox

rem get the selection information:

selectionCount%=LBoxSelectionCount%:

IF selectionCount%=0

GIPRINT "No item selected",KBusyTopRight%

return

ENDIF

rem find out if multiple delete required

IF selectionCount%>1

dINIT "Delete "+NUM\$(selectionCount%,3)+" items?"

ELSE

dINIT "Delete item "+NUM\$(index%,3)+"?"

ENDIF

dbuttons "No",-(%N OR \$100 OR \$200), "Yes",%Y OR \$100 OR \$200

IF DIALOG

IF selectionCount%>1

REM must sort the selection array before deleting items. Once
REM the array is sorted, we then must delete items from the
REM end of the listbox first. If we delete from the front of the
REM listbox, then after the first item has been deleted, all other
REM indices will be incorrect.

LBoxSelectionSort:

selection% = selectionCount%

WHILE selection%

LBoxDynamicDeleteItem(LBoxSelectionIndexAt%:(selection%))

selection%=selection%-1

ENDWH

ELSE

LBoxDynamicDeleteItem(index%)

ENDIF

REM in case deleted entry was the widest entry

REM re-calculate the column widths

rem LBoxDynamicUpdateColumnWidths:

total%=LBoxDynamicItemCount%: REM get the item array total

REM make sure the new item focus is within bounds

newpos%=MIN(index%,total%)

IF newpos%<>0 REM item array is not empty

REM set focus on the newpos% item

REM without this line, focus is set to line 1

LBoxDynamicSetCurrentItem:(newpos%)

ENDIF

ENDIF

ENDP

PROC TooLong:(a\$,b\$,c\$,d\$,numcols%)

REM check that the new entry plus column spacers is not too long

IF LEN(a\$)+LEN(b\$)+LEN(c\$)+LEN(d\$)+(numcols%-1) >255

return ktrue%

ENDIF

ENDP

PROC HasTab:(b\$,c\$,d\$)

REM check there is no tab character in the entry

IF LOC(b\$,CHR\$(9))

RETURN ktrue%

ELSEIF LOC(c\$,CHR\$(9))

RETURN ktrue%

ELSEIF LOC(d\$,CHR\$(9))

RETURN ktrue%

ENDIF

ENDP

Example code: Simple pop-up

```
INCLUDE "Const.oph"
INCLUDE "NListBox.oxh"
REM Demonstrate a simple pop-up
REM using auto-values for size
PROC Main:
    GLOBAL id&
    LOCAL i%, selected&, returned%, flags%
rem create an array for the entries
    id& = ArrayCreate&:
rem populate the array
    DO
        ArrayAddItem:(id&, CHR$(65+i%)+ " item "+GEN$(i%+1,3))
        i%=i%+1
    UNTIL i%=26
    print "Array contains", ArrayItemCount%:(id&), "items"
    print
rem set the dialog flags
    flags% = KNLBoxNoTitle%
    flags% = flags% OR KNLBoxReportDoubleTapEvents%
    flags% = flags% OR KNLBoxFreeMemoryAfterDisplay%
rem Initialise the dialog
    LBoxInit:("", flags%, 1)
rem Set dialog position
    LBoxSetPosition:(0,20)
rem Display dialog
    returned%=LBoxDisplay%:(id&, selected&, 1, "")
    print "Return key was:", returned%
    If returned%=0
        print "Dialog was dismissed"
    else
        if (id&<>0)
            rem only executes if the flag KNLBoxFreeMemoryAfterDisplay% is not set
            print "Selected array item=", selected&
            print "which is:", CHR$(34)+ArrayItemAt$:(id&, selected&)+CHR$(34)
        endif
    endif
    print
    if(id&<>0)
        rem with the flag KNLBoxFreeMemoryAfterDisplay% set, the next commands will not
        execute
        print "Array now contains ", ArrayItemCount%:(id&), "items"
```

```
    ArrayFree:(id&)  
else  
    print "Array is empty"  
endif  
print  
print "press any key to quit"  
get  
ENDP
```