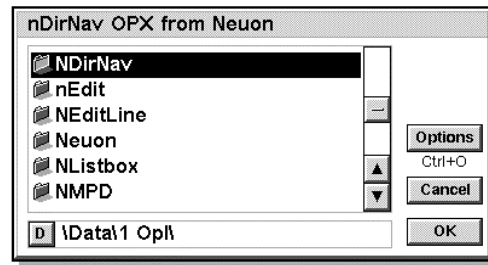


NDirNav.opx

A sample DirNav dialog

- To change drives:
Tap the button or use Tab key
- To change folders:
Select (tap or cursor keys)
and use Enter key, or
tap again
- To select files:
Highlight then use Enter key
- To cancel:
Use Esc



About Neuron



You can learn more about our dynamic 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*

NDirNav.opx: Introduction

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

Please submit any feedback to:

- msdt@neuron.com, or
- via the Neuron homepage <http://www.neuron.com/>

This OPX provides a comprehensive choice of options to display a directory navigator dialog featuring:

- Optional title and buttons
- Optional editor and info bar
- Callbacks to OPL32 functions to provide dynamic interaction
- On-screen positioning and sizing
- Quasi-asynchronous appearance via dynamic functions and callback procedures, whilst the dialog is displayed
- Automatic scrollbars as required

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 **Distributing the opx** which contains important information on how licencees may distribute this opx.

To use this OPX:

- You must include the header file NDirNav.oxh
- The file NDirNav.rsc must be in a \System\Opx\ folder

NDirNav.opx release notes

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

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

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

Release History

v1.01

- Fixed double tap bug on ER5 MARM

Contents

This help file contains the following information:

- About Neuon
- Licence conditions
- Distributing the OPX
- Information about Makesis and shared modules
- Error messages
- Using NDirNav

An overview of the DirNav dialog

Special considerations for a folders-only DirNav dialog

Using callback procedures

The opx commands organised as follows:

- Instructions to construct, configure and display the dialog **DNav...**
- Commands to use while the dialog is displayed and in callbacks **DNavDyn...**
- Commands for use after the dialog has closed **DNavSel...**
- Utility functions for use at any time **DNavU...**

Licence agreement

This opx is copyright (c) Alex Wilbur, Henry Hirst and Neuon 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 NDirNav.sis (for the target machine) and NDirNavWINSxxx opx (Release 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 NDirNav.sis may be granted as follows:
 - Free of charge for freeware applications
 - Free of charge to Neuon 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 Neuon member to use this opx free of charge remains valid only whilst the author is a member of Neuon, 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 Neuon 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 Neuon 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:
"NDirNav.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

NDirNav.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 NDirNav.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 NDirNav.sis file**. You may also include the WINS build versions of NDirNav.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 NDirNav.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 .NDirNav.sis

@ "NDirNav.sis", (0x10004D3E)

- 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 DirNav dialog will raise OPL errors in the following context:

-2 Invalid args

The function arguments are invalid (for example, incompatible flags, flags out of range, no initialPath\$.ext for a single extension dialog)

-9 In use

Using a **DNav...** command in a callback

-20 Failed to start

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

-33 Not exists

Attempting to change or specify non-existent entry data (e.g. buttonId%, file\$)

-85 Structure error

Calling a DirNav dialog construction method before using **DNavInit**

Specified a **DirNav...** command which is incompatible with the flags set in **DirNavInit**

Using an OnEnterEventCB\$ for a folders-only dialog

-99 Procedure not found

Non-existent callbackProcedure\$

-118 Drawable not open

Calling a **DNavDyn...** method when the DirNav dialog is not displayed

DirNav dialog: Overview

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

The DirNav dialog functions are conveniently arranged into four groups

- 1 'Static' functions used to configure and construct the dialog prior to display (**DNav...**)
- 2 Dynamic functions available in OPL32 callback procedures attached to buttons, once the dialog is visible (**DNavDyn...**)
- 3 Enquiry functions to determine selection status when the dialog is dismissed (**DNavSel...**)
- 4 Utility functions which can be used at any time (**DNavU...**)

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

Retrieving DirNav dialog data

The executive command to display a dialog is:

```
ret%=DNavDisplay%:(initialPath$, constructionCB$, OnEnterEventCB$)
```

- 1 **ret%** contains the value of the key used to exit the dialog (13=Enter, 27=Esc) etc.
- 2 **initialPath\$** is the initial path to display in the DirNav dialog.
- 3 **constructionCB\$** is the name of a callback function which will be called before the dialog is displayed. This allows for conditional configurations. See the **Callback** topic.
- 4 **OnEnterEventCB\$** is the name of a procedure which will be called before the dialog is closed, after an 'OnEnter' event has occurred. However, this is subject to qualification - see the **NDirNav dialog: Folders only** topic.
- 5 When the dialog has been dismissed, the currently selected folder(s), and any file(s), are returned from the **DNavSel...** series. Contents of the selection array will depend on how the dialog has been configured (see **DNavInit**)
- 6 If an editor has been specified, the contents of the editor are available (returned) from the function **DNavSelGetEditor\$**
- 7 By setting the **DNavInit** flag **KDNavAllowMultipleSelection%** (value 16), it is possible for a number of items to be selected. The DirNav dialog itself maintains an array of the selections, which can be enquired when **DNavDisplay%** returns.
- 8 **The selection array** is empty if the DirNav dialog is dismissed with the **Esc** key, or no multiple selections were made.
- 9 The selection array is reset upon calling **DNavDisplay**
- 10 Any single item selection is also stored in this array and is retrieved using the **DNavSel...** functions
- 11 Dynamic procedures (**DNavDyn...**) are provided to interrogate and reconfigure the dialog without it being first dismissed. To do this requires the use of callback procedures. See the topic **Callbacks** for more information

DirNav dialog: Folders only

When the DirNav dialog is restricted to showing folders only (a **contentFlags%** option for **DNavInit**), the following applies:

- It is mandatory to use a button to accept any selected folder(s). There cannot be a buttonless folder-only dialog
- The **Enter** key (13) cannot be used for this button in **DNavButton**, as **enter** is used to navigate the folder tree.

- Because no 'OnEnter' events can take place, no OnEnterEventCB\$ must be specified in **DNavDisplay**. If one is, a 'structure error' will be raised by **DNavDisplay**

An 'OnEnter' event occurs whenever:

- the enter key is used when a **file** is highlighted in a buttonless dialog.
 - when a double tap selection of a **file** occurs.
 - a specified enter key button is activated in a dialog with buttons
- However, it is still possible to conduct pre-closure validation by specifying a callback attached to the selection button. And, this can be the same procedure as the OnEnterEventCB\$ parameter of **DNavDisplay** would point to.

DirNav dialog: Callbacks

Optional callback procedures are used to provide interaction with the dialog. Three types of callback are used.

- A construction callback
- An 'OnEnterEvent' callback
- Callbacks attached to buttons

Callback procedures are **always** called via a **callback manager** procedure in the host OPL32 program. This is detailed below.

1. **For any construction callback** specified in **DNavDisplay**, this procedure will be called, via a **callback manager** in the host OPL32 program, before the dialog displays.

The purpose of a construction callback is to allow the option to conditionally configure the dialog. An example use would be the dimming or hiding of buttons depending on the status of other factors (such as non-availability of a CF drive).

2. **For any 'OnEnterEvent' callback** specified in **DNavDisplay**, this procedure will be called, via a **callback manager** in the host OPL32 program, when an OnEnter event occurs.

An 'OnEnter' event occurs whenever:

- the enter key is used when a **file** is highlighted in a buttonless dialog.
- when a double tap selection of a **file** occurs.
- a specified enter key button is activated in a dialog with buttons

The purpose of the OnEnterEvent callback is to allow the option to verify dialog input without having to dismiss the dialog first. An example use be to verify that any input in an editor contains valid characters.

- Restrictions apply to the use of the OnEnterEvent callback. See the **NDirNav dialog: Folders only** topic.

3. **In the case of callbacks attached to buttons**, when the button is activated, the DirNav dialog calls the associated callback procedure **via a callback manager** in the host OPL32 program.

- A callback procedure specified for a button with key code of 13 (the enter key) will raise an 'invalid args' error. The enter key is one of the 'OnEnter' events, and the callback for an 'OnEnter' event can only be specified in the OnEnterEventCB\$ parameter of **DNavDisplay**. This is to facilitate an 'OnEnterEvent' callback for buttonless dialogs.

See also the topic **NDirNav dialog: Folders only**.

Important

To use any callback procedure, 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 DNavCallBackManager%:(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%)
[declare any variables,
do any processing, display a dialog,
call other procedures]
RETURN KDNVDialogCanExit% / KDNVDialogCannotExit%
(as appropriate)
```

```
ENDP
```

 Please also read the topic **DNav dialog: Callback notes**

DirNav dialog: Callback notes

See **DirNav dialog: 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 as follows:
 - to zero (0) in the case of any specified construction callback
 - to zero(0) in the case of any specified OnEnterEvent callback
 - to the value of the current focused item in the selection array, for all callbacks attached to buttons.
2. The callback procedure must only return an integer of either **KDNVDialogCanExit%** or **KDNVDialogCannotExit%** (see NDirNav.oxh). These returns dictate whether the DirNav dialog will close on returning from the callback procedure.
4. In setting a callback procedure, the name required is the procedure name without the variable parameters. So PROC Edit%:(item%) would be **"Edit"**
5. Callback procedures may call other procedures in the OPL32 program, but control must always be returned back to the DirNav dialog (via the OPL32 callback manager procedure) so the DirNav dialog can properly exit and release all the resources it has in use. OPL32 procedures down the callback chain should not leave, STOP, etc.
6. There is a need for an error handling harness to enable correct mapping of error values returned to the OPX by the OPL runtime. This error harness is provided by the OPL32 host program's **callback manager** (see **DirNav dialog: 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.
7. 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.
8. Whilst other dialogs can be displayed on top of the DirNav dialog, the underlying behaviour is still synchronous. While a DirNav dialog 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.
9. It is possible to use synchronous keyboard commands (Get, Get\$) in callback procedures.
10. The DirNav dialog automatically sets LOCK ON when it is displayed.

11. When displayed, the dialog 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.

DirNav dialog: Prior to display

The **DNav..** topics are all used to construct, configure and display the DirNav dialog.

DNAVINIT:

DNavInit:(title\$, setupFlags%, contentFlags%)

Important

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

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

See NDirNav.oxh for constants for all the various ...flags%

1. If **title\$** is supplied, it will be displayed at the top of the dialog.
2. **setupFlags%** can be any added combination of the following constants to achieve the following effects.
 - Buttons on the right rather than at the bottom = 1
 - No title bar (any title in **DNavInit** is ignored) = 2
 - Full screen dialog = 4
 - No drag dialog = 8
 - Allow multiple selections = 16
3. **contentFlags%** specify the sorting order and content of the dialog and can be an added combination of the three groups sort order, display type and options:
 - **Sort order** for directory contents. Select **one** from:
 - By name = 0 (default)
 - By size = 1
 - By date = 2
 - By type = 4
 - **Display type.** Select one from:
 - content only = 8
 - Only the content of the specified path-on any drive-is displayed
 - folders only = 16

Note Using this flag has implications. See the topic **NDirNav dialog: Folders only**

Display only files matching any specified extension (provided in the initialPath\$ variable of **DNavDisplay**) = 32
- **Options.** Select any added combination of the following:
 - include system files/folders = 64
 - include hidden files/folders = 128
 - show the size = 256
 - show the date = 512
 - allow ROM to be selected = 1024
 - allow drive change & info bar = 2048
 - allow edit box = 4096

- The drive change button on the info bar can also be activated by pressing the tab key

DNAVCANCEL:

DNavCancel:()

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

DNAVBUTTON:

DNavButton:(caption\$, flags%, callbackProc\$, bitmapFile\$, bitmapIndex%, bitmapMask%, buttonLayout%)

Adds a button to the dialog.

1. **caption\$** is the text to display on the button.
2. **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))
3. **callbackProc\$**, if specified, sets the procedure in the host OPL32 program which is to be called when this button is pressed (see **DNav dialog: Overview** for more details).
 - If the **Enter** key is specified, an 'invalid args' error will be raised if a callback procedure is specified. The only way to specify a callback for the **enter** key, is the OnEnterEventCB\$ parameter of **DNavDisplay**. This is to allow a pre-closure callback option for buttonless dialogs.
 - There are button implications for any DirNav dialog which is restricted to showing folders only (a contentFlags% option for **DNavInit**). See the topic **DirNav dialog: Folders only** for more information.
4. **bitmapFile\$**, if specified, contains a bitmap to display on the button.
5. **bitmapIndex%** specifies which bitmap to use in an MBM file (with 0 as the first, or only, bitmap).
6. **bitmapMask%** specifies the bitmap mask, with -1 if none is required.
7. **buttonLayout%** specifies the relative positions of text and bitmaps on a button, with **one** from:
 - KNavTextRight% = 0
 - KNavTextTop% = 1
 - KNavTextBottom% = 2
 - KNavTextLeft% = 3

DNAVPOSITION:

DNavPosition:(x%,y%)

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

DNAVSIZE:

DNavSize:(width%,heightInItems%)

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

DNAVEXTENT:

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

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

DNAVSETEDITOR:

DNavSetEditor:(prompt\$,initialText\$)

Initialises the editor with prompt\$ on the left, and initialText\$ in the edit window (per the OPL32 command dEDIT)

- The inclusion of an editor must have been specified as an optional contentFlags% in **DNavInit**

DNAVDISPLAY%:

ret%=DNavDisplay%:(initialPath\$,constructCB\$, OnEnterEventCB\$)

Displays the list box, setting **initialPath\$** as the focused folder. On return, **ret%** contains the keycode of the key used to close the dialog

constructionCB\$ specifies a callback procedure which will be called before the dialog displays

OnEnterEventCB\$ specifies a callback procedure to call when an 'OnEnter' event occurs. An 'OnEnter' event occurs whenever:

- the **Enter** key is used when a **file** is highlighted in a buttonless dialog.
- when a double tap selection of a **file** occurs.
- a specified enter key button is activated in a dialog with buttons

The purpose of the OnEnterEvent callback is to allow the option to verify dialog input without having to dismiss the dialog first. An example use could be to verify that any input in an editor contains valid characters.

- It is not possible to specify an OnEnterEventCB\$ for a DirNav dialog which is restricted to showing folders only. See the topic **DirNav dialog: Folders only**

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

Notes

- 1 **For a buttonless dialog**, the default keys are Enter and Esc (with Tab to change drives if an info bar is specified). 0 is returned if the dialog was cancelled, 13 if **enter** was pressed, or the dialog was closed by a double tap event on a file.
- 2 **If any buttons are specified**, 13 will only be returned if there is a button set with a value of 13. **Esc** always returns 0.
- 3 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 KNavDialogCanExit% (value 1).) Otherwise the dialog will not be dismissed.
- 4 NDirNav maintains an internal array of any selections made. These are retrieved when the dialog is closed using the **DNavSel...** series of commands. In addition, the contents of this array can be enquired from within a callback procedure using the relevant **DNavDyn...** series command.

DirNav dialog: While displayed & in callbacks

The **DNavDyn...** series of functions are for use inside callback procedures only.

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

DNAVDYNTOPARENT:

DNavDynToParent()

Moves up the directory tree.

DNAVDYNITEMCOUNT%:

total%=DNavDynItemCount%:()

Returns the number of items currently selected

- Unless the dialog has been configured for multiple selections, by setting the **DNavInit** flag KNavAllowMultipleSelection% (value 16), total% will contain one item.

DNAVDYNITEMDETAILSAT\$:

details\$=DNavDynItemDetailsAt\$:(index%, BYREF size&, BYREF Attrs&, BYREF Uid&)

Returns the details for the selected item at index% in the selection array. The BYREF variables are updated with the specified information.

The value returned in Attribs& will be an added combination of the following:

- Normal=0
- ReadOnly=1
- Hidden=2
- System=4
- Volume=8
- Dir=16
- Archive=32

Uid& is the UID of any parent application, or 0 if the file does not have one. Folders always return 0.

Any returned UID& can be used with the appropriate utility functions in the **DNavU...** series

DNAVDYNGETPATH\$:

path\$=DNavDynGetPath\$:()

Returns the current displayed path

DNAVDYNSETPATH:

DNavDynSetPath:(path\$)

Changes the dialog to display the supplied path\$

DNAVDYNCHANGESORT:

DNavDynChangeSort:(order%)

Re-configures the dialog sort order according to order%, where:

- By name = 0 (default)
- By size = 1
- By date = 2
- By type = 4

DNAVDYNISPARENTDIR%:

result%=DNavDynIsParentDir%:(index%)

If the item at position index% in the selection array is a parent directory, returns -1, else returns 0

DNAVDYNSETITEM:

DNavDynSetItem:(index%)

Displays the open directory for the item at position% in the selection array

DNAVDYNREFRESH:

DNavDynRefresh:

Refreshes the dialog. Typically, this call would be used after any action which alters the content of the dialog (such as renaming or deleting a file)

DNAVDYNGETEDITOR\$:

details\$=DNavDynGetEditor\$(Type%)

Retrieves the current content of the editor, according to the value of type%

Return just the contents of the editor = 1

Return the full path = 2

■ The use of the editor must have been specified in the setupflags% in **DNavInit**

DNAVDYNSETEEDITOR:

DNavDynSetEditor:(text\$)

Changes the current contents of the editor to text\$

■ The use of the editor must have been specified in the setupflags% in **DNavInit**

DNAVDYNSETBUTTON:

DNavDynSetButton:(buttonId%, state%)

Changes the state of the DirNav dialog button with keycode buttonId%, according to the value of state%

- KNavStateNotDimmed% = 1
- KNavButtonStateDimmed% = 2
- KNavButtonStateInvisible% = 3
- KNavButtonStateVisible% = 4

DNAVDYNSETTITLE:

DNavDynSetTitle:(title\$)

Sets the dialog title to the string title\$

DNAVDYNTOPLEFT:

DNavDynTopLeft:(BYREF x%, BYREF y%)

Returns the position of the top left corner of the dialog (relative to the screen) in the BYREF variables.

DNAVDYNWIDTH%:

width%=DNavDynWidth%:()

Returns the width of the dialog.

DNAVDYNHEIGHT%:

height%=DNavDynHeight%:()

Returns the height of the dialog.

DirNav dialog: After dismissal

The **DNavSel...** series of commands are for use once the dialog has been dismissed.

DNAVSELCOUNT%:

total%=DNavSelCount%:

Returns the total number of the selected items

DNAVSELAT\$:

details\$=DNavSelAt\$(index%)

Returns the name of the item at position index% in the selection array

Note

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

LOCAL total%, arrayPosition%

```

LOCAL item$(255)
LOCAL Size&, Attrs&, Uid&
arrayPosition%=1
total%=DNavSelCount%:
WHILE arrayPosition% <= total%
    item$=DNavSelAt$(arrayPosition%)
    DNavUItemDetails:(item$, Size&, Attrs&, Uid&)
    ...
    <process the results>
    ...
    arrayPosition%=arrayPosition%+1
ENDWH

```

DNAVSELSORT:

DNavSelSort()

Sorts the selection index array into ascending order

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

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)

DNAVSELGETEDITOR\$:

details\$=DNavSelGetEditor\$():

Retrieves the current content of the editor.

■ The use of the editor must have been specified in the setupflags% in **DNavInit**

NDirNav dialog: At any time

The **DNavU...** series of utility functions can be used at any time before, during and after using a NDirNav dialog.

DNAVUIITEMDETAILS:

DNavUItemDetails:(item\$, BYREF Size&, BYREF Attrs&, BYREF Uid&)

Returns details for the item specified in item\$ (which can be a file or a directory), in the BYREF variables.

The value returned in Attrs& will be an added combination of the following:

- Normal=0
- ReadOnly=1
- Hidden=2
- System=4
- Volume=8
- Dir=16
- Archive=32

Uid& is the UID of any parent application, or 0 if the file does not have one. Folders always return 0.

Any returned UID& can be used with the appropriate utility functions in the **DNavU...** series.

DNAVUISFILEOPEN%:

Result%=DNavUlsFileOpen%:(File\$)

Checks to see if file\$ is in use. Returns -1 for yes, 0 for no

Raises an error if file\$ does not exist.

DNAVUGETUID&:

Uid&=DNavGetUid&:(File\$, index%)

Returns the index% Uid& for file\$, where index%

- 0=File store type
- 1=Document type
- 2=Parent app uid

▣ Uid&=0 if file\$ does not have a valid Uid (e.g. a text file)

▣ Raises an error if File\$ does not exist

▣ See **Const.opb** "OPL related Uids for dFile"

DNAVUGETAPPPATH\$:

appPath\$=DNavUGetAppPath\$:(Uid&)

Returns the full path to the application .app file for the app with uid Uid& (where Uid& is index%=2 in the function **DNavUGetUid&**)

appPath\$ is a null string if the application could not be found

▣ Raises an error if Uid& is not a valid Uid

DNAVUGETAPPCAPTION\$:

caption\$=DNavUGetAppCaption\$: (appPath\$)

Returns the caption for the application at appPath\$ (e.g. as returned from **DNavGetAppPath\$**)

▣ appPath\$ must be the full path to the application's .app or .aif file

DNAVUISDIR%:

result%=DNavUlsDir%:(path\$)

Checks whether the directory in the specified path exists. Returns -1 for yes, 0 for no or if the disk was not present.

▣ path\$ can be a directory path or the full path to a file.

DNAVUISVALID%:

result%=DNavUlsValid%:(text\$)

Confirms if text\$ contains valid characters. Returns -1 for yes, 0 for no.

Use this function to get whether a filename and path are syntactically correct. The following restrictions apply to the path and to its components: —

- Wildcards are not allowed in any path component, including the filename and extension.
- Double backslashes are not allowed anywhere in the path.
- The following characters cannot occur in the path: < > : " / |

All other characters are acceptable.

- Either or both of a filename or extension must be present, or the function will return false.
- The entire component following the final backslash (the filename and extension) may not consist solely of space characters, or of a single or double dot.

- Spaces between the drive, if specified, and the first directory in the path are illegal, although there may be spaces between other path components, for instance between directories. For example, the following code will return false:

```
result%=DNavIsValid%("C: \Dir\")
```

DNAVUCHECKDISK%:

Result%=DNavUCheckDisk%:(driveletter\$)

Use this function to check the integrity of the disk driveletter\$ (a single character).

- If the drive specified is empty, returns -18. Any other value except 0 indicates that the disk is corrupt.

rem **NDirNav.OXH**

rem

rem © Copyright Alex Wilbur, Henry Hirst and Neuron 1999

rem =====

rem **DirNav dialog initialisation constants**

rem used in DNavInit:(title\$, setupFlags%, contentFlags%)

rem SETUPFLAGS%

rem select any added combination from

const KDNNavButRight%	= 1
const KDNNavNoTitle%	= 2
const KDNNavFullScreen%	= 4
const KDNNavNoDrag%	= 8
const KDNNavAllowMultipleSelection%	= 16

rem =====

rem CONTENTFLAGS%

rem an added combination of sort order,display type & options

rem

rem 1.Sort order: select one from

const KDNNavSortByName%	= 0
const KDNNavSortBySize%	= 1
const KDNNavSortByDate%	= 2
const KDNNavSortByType%	= 4

rem 2. Display type: select one from:

const KDNNavDisplayContentOnly%	= 8
const KDNNavDisplayFoldersOnly%	= 16
const KDNNavDisplaySingleExt%	= 32

rem 3. Options: select any combination of

const KDNNavContentSystemFiles%	= 64
const KDNNavContentHiddenFiles%	= 128
const KDNNavDisplayShowSize%	= 256
const KDNNavDisplayShowDate%	= 512
const KDNNavContentAllowROM%	= 1024
const KDNNavHasInfobar%	= 2048
const KDNNavHasEditor%	= 4096

rem =====

rem Button state flags% for DNavButton: & DNavDynButton:

const KDNNavStateNotDimmed%	= 1
-----------------------------	-----


```

const KDNNavStateDimmed%           = 2
const KDNNavStateInvisible%        = 3
const KDNNavStateVisible%          = 4

```

```

rem =====
rem Dialog exit constants for use in callback procedures

```

```

const KDNNavDialogCannotExit%      = 0
const KDNNavDialogCanExit%         = 1

```

```

rem =====
rem Button layout constants

```

```

const KDNNavTextRight%             = 0
const KDNNavTextBottom%            = 1
const KDNNavTextTop%               = 2
const KDNNavTextLeft%              = 3

```

```

rem =====
rem returnType% for DNavDynGetEditor$(returnType%)

```

```

const KDNNavReturnSimple%          = 1
const KDNNavReturnFull%            = 2

```

```

rem =====
rem Standard OPX constants

```

```

const KOpXUidNDNav&                = &10004D3E
const KOpXVersionNDNav%            = $101

```

```

rem =====

```

```

DECLARE OPX NDIRNAV, KOpXUidNDNav&, KOpXVersionNDNav%

```

```

REM ***** Initialisation, configuration and display *****

```

```

    DNavInit:(Title$, setupFlags%, contentFlags%) : 1
    DNavCancel:() : 2
    DNavAddButton:(Caption$, Flags%, CallBackProc$, bitmapFile$, bitmapIndex%,
bitmapMask%, buttonLayout%) : 3
    DNavSetPosition:(TopLeftX%, TopLeftY%) : 4
    DNavSetSize:(Width%, HeightInItems%) : 5
    DNavSetExtent:(TopLeftX%, TopLeftY%, Width%, HeightInItems%) : 6
    DNavSetEditor:(Prompt$, initialText$) : 7
    DNavDisplay%:(InitialPath$, ConstructCB$, OnEnterEventCB$) : 8

```

```

REM ***** While displayed / Using a callback *****

```

```

    DNavDynToParent:() : 21
    DNavDynItemCount%:() : 22
    DNavDynItemDetailsAt$(Index%, BYREF Size&, BYREF Attrs&, BYREF Uid&) : 23

```

```

DNavDynSetPath:(Path$) : 24
DNavDynGetPath$:( ) : 25
DNavDynChangeSort:(Order%) : 26
DNavDynIsParentDir%:(Index%) : 27
DNavDynSetItem:(Index%) : 28
DNavDynRefresh:( ) : 29
DNavDynSetEditor:(Text$) : 30
DNavDynGetEditor$:(returnType%) : 31
DNavDynSetButton:(ButtonId%, State%) : 32
DNavDynSetTitle:(Title$) : 33
DNavDynTopLeft:(BYREF x%, BYREF y%) : 34
DNavDynWidth%:( ) : 35
DNavDynHeight%:( ) : 36

REM ***** After the dialog has closed *****
    DNavSelCount%:( ) : 41
    DNavSelAt$:(Index%) : 42
    DNavSelSort:( ) : 43
    DNavSelGetEditor$:( ) : 44

REM ***** At any time *****
    DNavUItemDetails:(File$, BYREF Size&, BYREF Attribs&, BYREF Uid&) : 51
    DNavUlsFileOpen%:(File$) : 52
    DNavUGetUid&:(File$, index%) : 53
    DNavUGetAppPath$:(Uid&) : 54
    DNavUlsDir%:(Path$) : 55
    DNavUlsValid%:(text$) : 56
    DNavUGetAppCaption$:(appPath$) : 57
    DNavUCheckDisk%:(driveLetter$) : 58

END DECLARE

```

```
INCLUDE "Const.oph"  
INCLUDE "NDirNav.oxh"
```

REM Illustrative code to display a DirNav dialog with callbacks

REM All callback procedures end with "CB"

PROC Main:

```
GLOBAL NotVisible%, NotDimmed%, contentFlags%, initialPath$(255)  
  
LOCAL i%, total%  
LOCAL returned%, setupFlags%  
LOCAL Title$(50)  
LOCAL constructionCB$(255), OnEnterEventCB$(255)  
LOCAL details$(255), Size&, Attribs&, Uid&  
  
rem Initialise the dialog  
rem DNavInit:(Title$, setupFlags%, contentFlags%)  
  
setupFlags% = 0  
rem select any added combination of  
  
setupFlags%=setupFlags% OR KNavButRight%  
rem setupFlags%=setupFlags% OR KNavNoTitle%  
  
rem setupFlags%=setupFlags% OR KNavFullScreen%  
  
rem setupFlags%=setupFlags% OR KNavNoDrag%  
  
rem setupFlags%=setupFlags% OR KNavAllowMultipleSelection%  
  
contentFlags% = 0  
rem an added combination of sort order,display type & options  
  
rem 1. Sort order: select one from  
contentFlags% = contentFlags% OR KNavSortByName%  
  
rem contentFlags% = contentFlags% OR KNavSortBySize%  
  
rem contentFlags% = contentFlags% OR KNavSortByDate%  
  
rem contentFlags% = contentFlags% OR KNavSortByType%  
  
rem 2. Display type: Select one from  
rem contentFlags% = contentFlags% OR KNavDisplayContentOnly%
```

```

rem contentFlags% = contentFlags% OR KDNNavDisplaySingleExt%

rem contentFlags% = contentFlags% OR KDNNavDisplayFoldersOnly%
    rem when displaying folders only,
    rem • the dialog must have a button
    rem   which cannot take the enter key (13)
    rem • no OnEnterEventCB$ can be specified

rem 3. Options: select any combination of
    contentFlags% = contentFlags% OR KDNNavContentSystemFiles%

    rem contentFlags% = contentFlags% OR KDNNavContentHiddenFiles%

    rem contentFlags% = contentFlags% OR KDNNavDisplayShowSize%

    rem contentFlags% = contentFlags% OR KDNNavDisplayShowDate%

    rem contentFlags% = contentFlags% OR KDNNavContentAllowROM%

    contentFlags% = contentFlags% OR KDNNavHasInfobar%

    rem contentFlags% = contentFlags% OR KDNNavHasEditor%

rem other initialisation data
rem In this demonstration, the dialog title is set as
rem part of the Construction callback

rem initialise the dialog
DNavInit:(title$, setupFlags%, contentFlags%)

rem add buttons
DNavAddButton:("Example", %E OR $100,"ExampleCB","",0,-1,0)
DNavAddButton:("Options",%O, "OptionCB","",0,-1,0)
DNavAddButton:("Cancel", 27 OR $100, "", "",0, -1, 0)
IF contentFlags% AND KDNNavDisplayFoldersOnly%
    DNavAddButton:("Done", %D ,"PrecloseCB","",0, -1, 0)
    REM
    REM Note: For a folder-only dialog, enter cannot
    REM be used to leave the dialog, as it is also
    REM used to navigate the folder structure;
    REM so an alternative button has to be used.
    REM Here, Ctrl+D is linked to the same callback which
    REM is used for OnEnterEventCB$ (PrecloseCB) so the same
    REM pre-closure code is used.
    REM There must be at least one button for a folder-only

```

```

        REM dialog
ELSE
    DNavAddButton:("OK", 13 OR $100, "", "", 0, -1, 0)
    REM
    REM Enter keys never have a callback specified
    REM Instead one is specified as the OnEnterEventCB$
    REM variable of DirNavDisplay%:
    REM This is so that buttonless dialogs can also
    REM have a callback attached to an OnEnter event
    REM when selecting files.
ENDIF

rem Set dialog position
rem Is ignored by the OPX if setupFlags% AND KNavFullScreen%
DNavSetPosition:((gWIDTH/2)-90, 5)

rem set up the screen
DoScreenInfo:

rem Display dialog
initialpath$="C:\\"
    REM If contentFlags% AND KNavDisplaySingleExt%
    REM the initialpath$ must specify an extension
    REM for example "C:\*.txt", "C:\*.*"

constructionCB$="ConstructCB"

If contentFlags% AND KNavDisplayFoldersOnly%
    OnEnterEventCB$=""
    REM Cannot specify an OnEnter callback
    REM when showing folders only.
    REM But the same procedure can be specified for the callback
    REM attached to the mandatory dialog button
ELSE
    OnEnterEventCB$="PrecloseCB"
ENDIF

returned% = DNavDisplay%:(initialPath$,constructionCB$, OnEnterEventCB$)

rem process the results
CLS
print "returned% =
DNavDisplay%:(\"+CHR$(34)+Initialpath$+CHR$(34)+\", \"+CHR$(34)+constructionCB$+CHR$(34)+\", \"
+CHR$(34)+OnEnterEventCB$+CHR$(34)+\")"

```

```

print
print "returned% =", returned%
IF returned% = 0
    print "Dialog was dismissed"
ELSEIF (returned%=13) OR (returned%=%D)
    IF ContentFlags% AND KNavHasEditor%
        print "The editor contained:",CHR$(34)+DNavSelGetEditor$:+CHR$(34)
    ENDIF
    total% = DNavSelCount%:
    print "Number of selected items =",total%
    DNavSelSort: REM sort selections into ascending order
    i%=1
    WHILE i% <= total%
        details$ = DNavSelAt$(i%)
        DNavUItemDetails:(details$, size&, attribs&, uid&)
        print "Item "+Num$(i%,3)+".",CHR$(34)+details$+CHR$(34), "Size:", size&,
"Attribs:",attribs&, "Uid:", "&"+HEX$(uid&)
        i%=i%+1
    ENDWH
ELSE
    print "Dialog not closed by Enter, Ctrl+D or Esc"
ENDIF
print
print "press a key to quit"
GET
ENDP

```

```

PROC DoScreenInfo:
    cls
    print "A sample DirNav dialog"
    print REPT$(" ",24)
    print
    IF contentFlags% AND KNavHasInfobar%
        print CHR$(149)+" To change drives:"
        print " Tap the button or use Tab key"
    ENDIF
    print CHR$(149)+" To change folders:"
    print " Select (tap or cursor keys)"
    print " and use Enter key, or"
    print " tap again"
    If contentFlags% AND KNavDisplayFoldersOnly%
        print CHR$(149)+" To select folder:"
        print " Highlight then use 'Done'"
    ELSE

```

```

        print CHR$(149)+" To select files:"
        print " Highlight then use Enter key"
    ENDIF
    print CHR$(149)+" To cancel:"
    print " Use Esc"
ENDP

REM *****
REM This next procedure is the MANDATORY callback manager
REM required if you use any callback procedures.
REM It must be identical to this.
REM If not you will get an "Unknown error" reported
REM
REM Provides a trap harness for errors in any callback procedure
REM and for any procedures called from callback procedures.
REM *****

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

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

REM *****
REM The next procedure is a templated callback proc
REM which is called via the callback manager.
REM It is in the form ProcedureName%:(index%)
REM Must take an integer only (normally the current focus item)
REM and return ONLY KNavDialogCannotExit% or
REM KNavDialogCanExit% as appropriate
REM
REM All error handling in the callback, or procedures
REM down the callback chain, is handled by the
REM callback manager
REM
REM *****

PROC TemplateCB%:(index%)
    rem declare variables
    rem do the required work

```

```

        return KDNavigateDialogCannotExit%
        rem return KDNavigateDialogCanExit%
ENDP

PROC ConstructCB%:(index%)
    LOCAL title$(50),off%(6)
    REM The optional construction callback

    REM hide the example button
    DNavDynSetButton:(%E, KDNavigateStateInvisible%)
    NotVisible%=1-NotVisible% REM variable used with the Options button pop-up

    REM set the title according to the contentFlags% set
        title$="nDirNav OPX from Neuron"
        IF contentFlags% AND KDNavigateDisplayFoldersOnly%
            title$=title$+" - folders only"
        ELSEIF contentFlags% AND KDNavigateDisplayContentOnly%
            title$=title$+" - this folder only"
        ELSEIF contentFlags% AND KDNavigateDisplaySingleExt%
            PARSE$(" ",initialPath$,off%())
            title$=title$+" - matching *. "+RIGHT$(initialPath$,LEN(initialPath$)-off%(5))
        ENDIF

    DNavDynSetTitle:(title$)

    dINIT "Construction callback"
    dTEXT "", "This dialog was specified in the Construction callback.",2
    dTEXT "", "This callback is executed after a request to display the",2
    dTEXT "", "DirNav dialog, but before the dialog actually appears.",2
    dTEXT "", "This callback can be used, for example, to configure",2
    dTEXT "", "the availability of button options.",2
    dBUTTONS "Continue",13 OR $100
    DIALOG
    return KDNavigateDialogCannotExit%
ENDP

PROC PrecloseCB%:(index%)
    dINIT "A Pre-closure callback"
    dTEXT "", "This dialog was specified in PROC PrecloseCB%:",2
    If contentFlags% AND KDNavigateDisplayFoldersOnly%
        dTEXT "", "and was called as a button callback for 'Done'",2
    ELSE

```



```

        dTEXT "", "and was called as the DNavDisplay%: 'OnEnterEventCB$' callback", 2
    ENDIF
    dTEXT "", "This callback can be used, for example, for input", 2
    dTEXT "", "confirmation before the dialog closes.", 2
    dBUTTONS "Close", -(27 OR $100), "Continue", 13 OR $100
    IF DIALOG
        return KNavDialogCannotExit%
    ENDIF
    return KNavDialogCanExit%
ENDP

PROC OptionCB%:(index%)
    LOCAL key%(3), ret%
    LOCAL x%, y%, width%, height%
    LOCAL posX%, posY% , corner%

    guse 1

    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

    rem Get dialog positioning details
    DNavDynTopLeft:(x%, y%)
    width%=DNavDynWidth%:
    height%=DNavDynHeight%:

    REM Position the pop-up
    IF (x%+width%-100 > (gwidth/2))
        posX%=(x%+width%)-70
        corner%=1
    ELSE
        posX%=(x%+width%)
        corner%=0
    ENDIF
    posY%=y%+(height%/2)

    REM Display the pop-up
    ret%=mPOPUP(posX%, posY%, corner%, "Dimmed", key%(1) OR ($2000 *
NotDimmed%), "Invisible", key%(2) OR ($2000 * NotVisible%), "Raise error", key%(3))
    if ret%<>0
        @("Choice"+GEN$(ret%, 2)):(index%)
    endif

```

```

        return KDNavigateDialogCannotExit%
ENDP

PROC Choice1:(index%)
    REM This procedure makes no use of index%
    IF NotDimmed%=0
        DNavDynSetButton:(%E, KDNavigateStateDimmed%)
    ELSE
        DNavDynSetButton:(%E, KDNavigateStateNotDimmed%)
    ENDIF
    NotDimmed%=1-NotDimmed%
ENDP

PROC Choice2:(index%)
    If NotVisible%=0
        DNavDynSetButton:(%E, KDNavigateStateInvisible%)
    ELSE
        DNavDynSetButton:(%E, KDNavigateStateVisible%)
    ENDIF
    NotVisible%=1-NotVisible%
ENDP

PROC Choice3:(index%)
    RAISE -123
    return KDNavigateDialogCannotExit%
ENDP

PROC ExampleCB%:(index%)
    REM If the 'Example' button did not have a callback
    REM the dialog would exit, if it returned KDNavigateDialogCanExit%,
    REM reporting the button keycode.

    GIPRINT "Use 'Options' button menu",KBusyTopRight%
    return KDNavigateDialogCannotExit%
ENDP

```