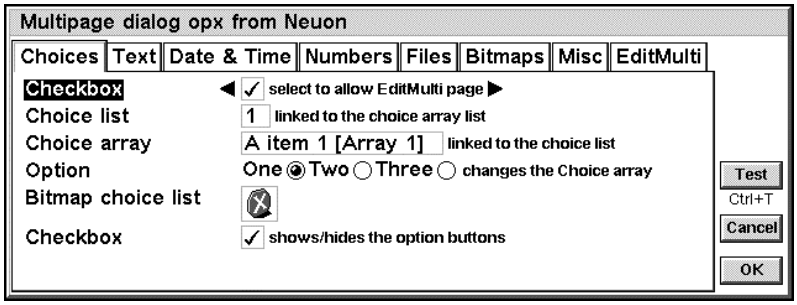
# NMPD.opx




## About Neuon

You can learn more about our dynamic company, and expanding EPOC software portfolio, from Neuon's web site at http://www.neuon.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, Neuon would like to hear from you.

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

## NMPD.OPX release notes

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

Thank you for your interest. Comments and suggestions are always welcome.

**Potential enhancements**

- Inclusion of standard font selector dialog (e.g. **Word | Font**)
- Inclusion of calendar dialog (e.g. **Agenda | View | Calendar**)
- Inclusion of 'special character' dialog (e.g. **Word | Insert | Special character**)
- Fully asynchronous operation

## 1. Introduction

**NMPD.opx provides a comprehensive choice** of options to display a multi-page dialog with a large range of control types. It features dynamic interaction with the dialog whilst it is still displayed, and can additionally be used as a replacement for the existing OPL32 dialog commands for single page dialogs

**This opx also includes** powerful functions for the creation and manipulation of dynamic string and bitmap arrays.

**To use this OPX:**

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

**Please read the section Licence agreement**. If you use, or distribute, this opx with your applications, you are agreeing to it.

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

- msdt@neuon.com
- opx.registrations@neuon.com if you are required to register this opx or via the Neuon homepage at http://www.neuon.com/

## Contents of this help file

This help file contains the following information:

- Licence conditions
- Distributing the OPX
- Information about Makesis and shared modules
- Using NMPD

  An overview of the NMPD dialog

  Using callback procedures

  Error messages

  The opx commands organised as follows:

  - Instructions to construct, configure and display the dialog **NMPD...**
  - Commands to use while the dialog is displayed and in callbacks **NMPDdyn...**
  - Instructions for the use of, and commands for the creation and manipulation of, a text array for use with an array based choice selector (NMPDChoiceByArray) **NMPDArray...**
  - Instructions on the use of, and commands for the creation of, a bitmap array for use with a bitmap choice selector (NMPDBitmapChoice) **NMPDB...**

## Licence agreement



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 NMPD.sis (for the target machine) and any WINS Release versions (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 NMPD.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 Neuon 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. Neuon 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:

   "NMPD.opx copyright (c) Alex Wilbur, Henry Hirst & Neuon 1999"

**Payment of licence fee**

Please contact opx.registrations@neuon.com, or register via the Neuon web site at http://www.neuon.com/

## Distributing the OPX

NMPD.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 program, you are required to include the **unchanged NMPD.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 NMPD.sis file**. You may also include NMPDWins.zip containing the WINS build versions of NMPD.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 NMPD.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 .NMPD.sis

   **@"NMPD.sis",(0x10004D40)**

**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:

   1. 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.
   2. 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

## 2. Overview of NMPD dialogs

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

The NMPD dialog functions are conveniently arranged into two primary groups

1. 'Static' functions used to configure and construct the dialog prior to display (**NMPD...**)
2. Dynamic functions available in OPL32 callback procedures attached to buttons, once the dialog is visible (**NMPDdyn...**)

- All activity between initialisation and eventual return from NMPDDisplay% **must** be within a single procedure, to prevent problems with the scope of variables. This does not apply to any **NMPDdyn...** procedure.

**NMPD makes extensive use of** callback procedures in the host OPL32 program, to provide interaction with the dialog.

Procedures are provided for the creation and manipulation of dynamic string arrays (**NMPDArray...**) and bitmap arrays (**NMPDB...**)

- NMPD can be used to create single page dialogs which can be used instead of the existing OPL32 dialog commands. To do this, specify a page count of 1 in **NMPDInit**. By specifying a null value for caption$ in NMPDPage:(1, “”), no page tab will be created.

## Retrieving dialog data

The executive command to display a dialog is:

Ret%=NMPDDisplay%:(InitControl%, ConstructionCB$, PageChangeCB$)

1. **InitControl%** is the control id of the control which will initially have focus when the dialog is displayed
2. **ConstructionCB$** is the name of a callback function which will be called before the dialog is displayed. This allows for conditional configurations. See the **Callbacks** topic.
3. **PageChangeCB$** is the name of a callback procedure which will be called whenever a page transition occurs. See the **Callbacks** topic
4. **Ret%** contains the value of the key used to exit the dialog (e.g. 13=Enter) or 0 if the dialog was dismissed.using the **Esc** key

Initial values for controls, and the return values when the dialog is closed, are managed using BYREF variables.

Dynamic procedures (**NMPDdyn...**) are provided to interrogate and reconfigure the dialog without it being first dismissed. To do this also requires the use of callback procedures. See the topic **Callbacks** for more information.

For some controls, a required parameter is the address of a string (for example NMPDEdit). In this case, where the declaration is String$(255), the address to be passed is Addr(String$).

## Callbacks

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

- A construction callback
- A page transition callback
- Callbacks attached to buttons
- State changed callbacks, invoked as a result of control interaction by the user.

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 **NMPDdisplay**, this procedure will be called, via the **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 ‘PageChanged’ callback** specified in **NMPDdisplay**, this procedure will be called, via the **callback manager** in the host OPL32 program, when a page transition occurs.

   A page transition event occurs whenever:

   - a page tab is tapped using the pointer
   - the left or right cursor key is pressed when the page selector currently has focus

   The purpose of the page transition callback is to allow the option to dynamically adjust various elements on pages as and when focus is lost or gained.

3. **In the case of callbacks attached to buttons,** when the button is activated, the NMPD dialog calls the associated callback procedure via the **callback manager** in the host OPL32 program.
4. The NMPD framework allows certain controls to have callbacks associated with state change events. For example, it is possible to attach a callback event to a check box control. The callback procedure is then called whenever the check box changes state (e.g. it changes from checked to clear, etc). In these instances, the callback is specified during the construction of the control.

**Note** For these state change callbacks, whilst the callback procedure must utilise the same template as used by all callbacks (see below), the Value% parameter passed to the callback is **always** initialised to a value of zero.

**Important**

**To use any callback procedure,** two conditions must be satisfied:

1. A callback manager is required in the host OPL32 program
2. All callback procedures must conform to a templated design

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

```
PROC NMPDCallBackManager%:(Value%, ProcName$)

    ONERR CallbackError::
    return @%(ProcName$):(Value%)

    CallbackError::
    ONERR OFF

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

ENDP
```

**The required template** for all callback procedures is:

```
PROC ProcNameCB%:(Value%)

    [declare any variables,

    do any processing, display a dialog,

    call other procedures]

    REM If the callback is a dialog button callback only, one of

    RETURN KMPDdialogCanExit% / KMPDdialogCannotExit%

ENDP
```

**Notes**

- In the case of a dynamic state-changed callback, the Value% parameter is always intialised to zero.
- The returned values of **KMPDdialogCanExit%** and **KMPDdialogCannotExit%** are only relevant to callbacks attached to buttons, and do not serve any purpose in the context of the other allowable callback types.
- Please also read the topic **Callback notes**

## Callback notes

See **Callbacks** for an explanation of callback procedures, and the mandatory framework to use them.

1. In the context of button callbacks only, the procedure must only return an integer of either **KMPDdialogCanExit%** or **KMPDdialogCannotExit%** (see NMPDConstants.oxh). These returns dictate whether the dialog will close on returning from the callback procedure.
2. In setting a callback procedure, the name required is the procedure name without the variable parameters. So PROC EditCB%: would be **"EditCB"**
3. Callback procedures may call other procedures in the OPL32 program, but control must always be returned back to the NMPD dialog (via the OPL32 callback manager procedure) so the dialog can properly exit and release all the resources it has in use. OPL32 procedures down the callback chain should not leave, STOP, etc.
4. 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 **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.
5. 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.

6. Whilst other dialogs can be displayed on top of the dialog, the underlying behaviour is still synchronous. While a NMPD 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.
7. It is possible to use synchronous keyboard commands (Get, Get$) in callback procedures.
8. The NMPD dialog automatically sets LOCK ON when it is displayed.
9. 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

## Error values

The NMPD 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, etc.)

-9 **In use**

Using a **NMPD...** command in a callback

Using a controlId% which is already declared

-20 **Failed to start**

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

-33 **Not exists**

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

-76 **Bad number**

Specifying an option button number in NMPDOption which is greater than the total declared in NMPDOptionInit

-85 **Structure error**

Calling a NMPD dialog construction method before using **NMPDInit**

Specified a **NMPD...** command which is incompatible with the flags set in **NMPDInit**

-99 **Procedure not found**

Non-existent callback procedure

-118 **Drawable not open**

Calling a **NMPDdyn...** method when the dialog is not displayed

## 3. Initialisation and display

The **NMPD...** commands are used to initialise, position and display the dialog.

## NMPDINIT:

NMPDInit:(Title$, Flags%, NumPages%)

Prepares for definition of the dialog. Only one NMPD dialog can be displayed at one time.

Any supplied **Title$** will be positioned in a grey box at the top of the dialog

**Flags%** can be any added combination of the following constants to achieve the following effects:

- KMPDButRight% buttons on the right rather than at the bottom = 1
- KMPDNoTitle% no title bar (any title in NMPDInit is ignored) = 2
- KMPDFullScreen% use the full screen = 4
- KMPDNoDrag% don't allow the dialog box to be dragged = 8
- KMPDDensePacking% packs the dialog lines = 16

**NumPages%** specifies the number of pages the dialog contains

- NMPD can be used to create single page dialogs which can be used instead of the exising OPL32 dialog commands. To do this, specify a page count of 1 in **NMPDInit**. By specifying a null value for caption$ in NMPDPage:(1, “”), no page tab will be created.

## NMPDCANCEL:

NMPDCancel:()

Abandons the creation of a new multi-page dialog before it is displayed using NMPDisplay and frees up all the memory resources allocated to it.

## NMPDPAGE:

NMPDPage:(PageNumber%, Caption$)

Prepares an NMPD dialog page for definition.

**PageNumber%** specifies the **unique** page identifier and must be less than or equal to the NumPages% value in NMPDinit. If a page transition callback is specified (see **Callbacks**) this identifer is passed as a parameter to the callback procedure, and can therefore be used as a means of identifying the page.

If **Caption$** is supplied it will be displayed in the page tab on the dialog page.

- NMPD can be used to create single page dialogs which can be used instead of the existing OPL32 dialog commands. To do this, specify a page count of 1 in **NMPDInit**. By specifying a null value for caption$ in NMPDPage:(1, “”), no page tab will be created

## NMPDBUTTON:

NMPDButton:(Caption$, Flags%, Callback$, BitFile$, BitmapId%, BitmapMaskId%,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)

**Callback$**, if specified, sets the procedure in the host OPL32 program which is to be called when this button is activated.

The OPL32 dButtons concept of a negative flag% to specify a cancel operation is not supported (e.g. -(%H OR $100))

**If a bitmap is required** on the button:

- **BitFile$** specifies the file containing the bitmap
- **BitmapId%** specifies the index for the required bitmap in bitFile$, 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%** specifes the layout of text and / or any bitmap on the button, and can be one of the following constants. Where text is on the right, any bitmap is on the left; where text is at the top, any bitmap is at the bottom etc.

- KMPDButTextRight% = 0
- KMPDButTextBottom% = 1
- KMPDButTextTop% = 2
- KMPDButTextLeft% = 3

## NMPDPOSITION:

NMPDPosition:(X%, Y%)

Use this procedure to set the top left corner of the dialog to the position **X%**,**Y%**.

- This procedure is ignored if the NMPDInit flag “use the full screen” is set.

- By default the dialog is positioned centrally

## NMPDDISPLAY%:

Ret% = NMPDDisplay%:(InitControlId%, ConstructionCB$, PageChangeCB$)

Displays the dialog, setting **InitControlId%** as the focused control, where InitControlId% is the user supplied unique identifier for the control or the page.

**Optional Callbacks**

If **ConstructionCB$** is specified, this callback will be called before the dialog is displayed.

If **PageChange$** is specified, this is called after a page transition event, and allows dynamic changes to be made to the dialog structure etc.

**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.
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 any **Esc** key callback. However, 0 will always be returned if **Esc** was used, regardless of whether there is a button defined to take the **Esc** keycode.
4. **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 KNMPDialogCanExit% (=1)) Otherwise the dialog will not be dismissed.
5. **The various BYREF variables** used with the individual controls contain the value of the relevant control when the dialog closed. If the escape key was pressed (and hence the dialog was cancelled) the BYREF values will not be updated.

## 4. Control types

The following commands list the controls which may be added to a dialog page.

The general syntax of these commands is:

- **Page%** specifies the relevant dialog page
- **Id%** is a user supplied unique identifier used to identify the control in callback functions and dynamic procedures. Note that **no two controls can utilise the same unique identifier value**, otherwise an 'in use' error will be raised. Id must be a natural number (i.e. > 0)
- **Prompt$** appears on the left of any control, in KFontArialNormal15& (see Const.oph)
- **Trailer$** specifies text to appear after the control on the right hand side, in KFontArialNormal10&. (see flags for NMPDInit and Const.oph)
- **StateChangeCB$** specifies a state change callback procedure, called via the mandatory callback manager. State change callbacks are called each time the value of the control changes. This allows for dynamic configuration of the dialog depending on the choices made. See **Callbacks** for an explanation of callback procedures and the mandatory requirements for their use.
- **BYREF** [variable] is the variable that contains the initial value for a control and the returned value when a dialog is closed.

## NMPDBITMAPCHOICE:

NMPDBitmapChoice:(Page%, Id%, Prompt$, Trailer$, BYREF Choice%, BYREF BitmapArray&, StateChangedCB$, Flags%)

Defines a bitmap choice editor, which operates in the same manor as a dCHOICE except that only images may be shown.

**Choice%** specifies which choice should initially be shown: 1 for the first choice, 2 for the second, etc.. When the dialog is dismissed, Choice% is set to the value of the choice selected: 1 for the first choice, and so on.

**BitmapArray&** is the handle to a dynamic bitmap array. See **Bitmap arrays** for an explanation. BitmapArray& must be a valid handle, and must contain at least one choice item, otherwise an 'invalid arguments' error will be raised.

**StateChangedCB$** is an optional callback procedure in the OPL host program which is called whenever the user selects a new choice in this control.

See **Callbacks** for an explanation of callback procedures and the mandatory requirements for their use.

**Flags%** may be either

- KMPDChoiceDefault% (=0) for default behaviour
- KMPDChoiceFreeArrayOnClose% (=2)

With this flag set, all the resources used to create the bitmap array will be automatically freed when the dialog closes. Hence, there is no need to call **NMPDBFree** in this instance. However, without this flag set, it is **very important** that any bitmap array is deleted using **NMPDBFree** to prevent orphaning and memory leaks. See **Bitmap arrays**

## NMPDBITMAPVIEWER:

NMPDBitmapViewer:(Page%, Id%, Prompt$, Trailer$, WindowId%, File$, Index%, Width%, Height%)

Defines a bitmap previewer.

There are a two potential sources for the image to be previewed - either an image from a file, or the contents of an OPL window.

If a non-zero **WindowId%** value is specified, WindowId% is set as the source of the bitmap. Any **File$** and **Index%** values are ignored.

If **WindowId%** is zero, and File$ is not null, the image at Index% (with 0 as the first or only image) in File$ is set as the source of the preview bitmap and will be loaded. **Note** File$ should contain the full path name of the source file.

**Width%** and **Height%** specify the view size of the bitmap viewer. If a Width% or Height% value of 0 is used, the respective dimension of the viewer will be set to the dimensions of the source image. Using a Width% or Height% value greater than the width or height of the source image will have no effect: the view size will be set to the actual size of the source image.

## NMPDCHECKBOX:

NMPDCheckBox:(Page%, Id%, Prompt$, Trailer$, BYREF State%, StateChangedCB$)

Creates a dialog checkbox entry.

This is similar to a choice list with two items, except that the list is replaced by a checkbox with the tick either on or off. The state of the checkbox is maintained across calls to the dialog.

**State%** may contain one of three initial values:

- KMPDChkBoxStateClear% (=0) to specify that the checkbox should appear initially 'un-ticked'
- KMPDChkBoxStateSet% (=1) to specify that the checkbox should appear initially 'ticked'
- KMPDChkBoxStateIndeterminate% (=2) to specify that the checkbox should appear initially 'indeterminate' (i.e. neither set nor clear)

**StateChangedCB$** is an optional callback procedure in the host OPL program. This is called whenever the user selects a new choice in this control.

See **Callbacks** for an explanation of callback procedures and the mandatory requirements for their use.

## NMPDCHAREDITOR:

NMPDCharEditor:(Page%, Id%, Prompt$, Trailer$, BYREF Char%)

Provides a single character edit box which responds to the key pressed when this line has focus. An example use is for prompting the user for a customisable hotkey.

**Char%** should initally be set to the specified character. When the dialog is dismissed, Char% will contain the value specified by the user.

## NMPDCHOICEBYARRAY:

NMPDChoiceByArray:(Page%, Id%, Prompt$, Trailer$, BYREF Choice%, BYREF ChoiceArray&, StateChangedCB$, Flags%)

This control is the equivalent of the OPL32 dCHOICE command. However, rather than using fixed length strings, it uses a dynamic string array for the choice items. A comprehensive range of commands for the creation and manipulation of dynamic string arrays is in the **NMPDArray...** series of commands. See **String arrays** for an explanation.

**Choice%** specifies which choice should initially be shown: 1 for the first choice, 2 for the second, and so on. When the dialog is closed, Choice% is set to the value of the choice selected:1 for the first choice, and so on.

**ChoiceArray&** is the handle to a dynamic string array. **ChoiceArray&** must be a valid handle, and it must contain at least one choice item, otherwise an 'invalid arguments' error will be raised

**Flags%** may be either:

- KMPDChoiceDefault% (=0) for default behaviour
- KMPDChoiceUseIncMatching% (=1) if incremental matching is to be allowed on any popout
- KMPDChoiceFreeArrayOnClose% (=2) if the memory utilised by the ChoiceArray& is to be freed, and all choices destroyed when the dialog closes. There is no need need to call **NMPDArrayFree** in this instance.

**StateChangedCB$** is an optional callback procedure in the host OPL program. This is called whenever the user selects a new choice in this control.

See **Callbacks** for an explanation of callback procedures and the mandatory requirements for their use..

## NMPDCHOICE:

NMPDChoice:(Page%, Id%, Prompt$, Trailer$, BYREF Choice%, ChoiceList$, StateChangedCB$, Flags%)

This control is the equivalent of the OPL32 dCHOICE command.

**Choice%** specifies which choice should initially be shown: 1 for the first choice, 2 for the second, and so on. When the dialog is closed, Choice% is set to the value of the choice selected, with 1 for the first choice, and so on.

**ChoiceList$** utilises the same syntax notation as specified in the OPL manual. The NMPD framework does, like the OPL dChoice command, support the use of "..." to specify that a further choice continuation will be specified.

**Flags%** may be either

- KMPDChoiceDefault% (=0) for default behaviour
- KMPDChoiceUseIncMatching% (=1) if incremental matching is to be allowed on any popout

**StateChangedCB$** is an optional callback procedure in the OPL host program, which will be called whenever the user selects an new choice in this control.

See **Callbacks** for an explanation of callback procedures and the mandatory requirements for their use.

## NMPDDATE:

NMPDDate:(Page%, Id%, Prompt$, Trailer$, BYREF Date&, Min&, Max&)

Defines an edit box for a date.

**Date&**, which must be a variable, specifies the date to be shown initially. Although it will appear on the screen like a normal date, for example 15/03/92, date& must be specified as "days since 1/1/1900".

**Min&** and **Max&** give the minimum and maximum values which are to be allowed. Again, these are in days since 1/1/1900. An error is raised if Min& is higher than Max&.

When the dialog is dismissed, the date entered is returned in Date&, in days since 1/1/1900.

- The system locale setting determines whether years, months or days are displayed first.

## NMPDEDIT:

NMPDEdit(Page%, Id%, Prompt$, Trailer$, StringAddress&, EditBoxLength%, MaxLength%, Flags%)

Defines a string edit box.

**StringAddress&** is the address of the string variable to edit. Any initial content will appear in the dialog.

For example, to pass the address of the string variable String$, the address& parameter would be **Addr(String$)**

**EditBoxLength%**, is the width of the edit box allowing for widest possible character in the font. The string will scroll inside the edit box if necessary.

**MaxLength%** is the maximum length of the **String$** variable passed in **StringAddress&**.

For example, LOCAL String$(255) would require a MaxLength% value of 255.

**Flags%** are optional configuration flags which can be set to

- **KMPDEditorDefault%** (=0) for default behaviour
- **KMPDEditorReadOnly%** (=1) to limit the edit window to read-only.
- **KMPDEditorHex%** (=16) will accept numerical integer input (Base 16) of 1-9, A-F (**Note:** the number is returned as a string)

## NMPDEDITMULTI:

NMPDEditMulti:(Page%, Id%, Prompt$, BufferAddr&, EditWindowWidth%, MaxLength%, LineCount%, Flags%)

Defines a multi-line edit box.

**BufferAddr&** is the address of a buffer to take the edited data. Its initial contents will appear in the dialog. For example, for an integer array buffer, if:

LOCAL maxlength%, buffer&(101)

Maxlength%=399

REM 101=1+(399+3)/4 in integer arithmetic

Then BufferAddr& would be Addr(Buffer&())

See the OPL manual for full details of dEDITMULTI specification.

**EditWindowWidth%**, is the width of the edit box allowing for widest possible character in the font. Content will wrap inside the edit box if necessary.

**MaxLength%** is the maximum number of characters the buffer can hold

**LineCount%** is the number of dialog lines that the editor will use

**Flags%** specifies the following options

- KMPDEditorDefault% (=0)

  The default behaviour is read & write content, no vertical scroll bar displayed until required, and a horizontal scroll bar will never be displayed.

  This can be modified by specifying any added combination of the following:

  - **KMPDEditorReadOnly%** (=1) to specify a read-only edit window
  - **KMPDEditorForceHorizontalSB%** (=2) to force the editor to display a horizontal scroll bar
  - **KMPDEditorForceVerticalSB%** (=4) to force the editor to display a vertical scroll bar
  - **KMPDEditorNoWrap%** (=8) to prevent word wrapping

- KMPDEditorHex% (=16) is for use **only** with nMPDEdit: and will raise an error if used with nMPDEditMulti

## NMPDFILE:

NMPDFile:(Page%, Id%, Prompt$, Flags%, Uid1&, Uid2&, Uid3&, FileStringAddr&)

Defines a filename edit box or selector.

- Specifying multiple NMPDFile controls will result in a longer time for the NMPD dialog to display, as each instance of NMPDFile builds its own index to the folders and files on the target machine – as occurs with the OPL32 dFile command.

**A 'folder' and 'disk' selector** are automatically added on the following lines

**By default no prompts are displayed** for the file, folder and disk selectors. A comma-separated prompt list should be supplied. For example, for a filename editor with the standard prompts use:

```
NMPDFile:(Page%, Id%, 0,"File,Folder,Disk",0,0,0,Addr(File$))
```

**flags%** controls the type of file editor or selector, and the kind of input allowed. You can add together any of the following values:

- 0 use a selector
- 1 use an edit box
- 2 allow directory names
- 4 directory names only
- 8 disallow existing files
- 16 query existing files
- 32 allows null string input
- 128 obey/allow wildcards
- 256 allow ROM files to be selected
- 512 allow files in the System folder to be selected

**The first of the list is the most crucial**. If you add 1 into flags%, you will see a file edit box, as when creating a new file. If you do not add 1, you will see the 'matching file' selector, used when choosing an existing file.

If performing a 'copy to' operation, you might use 1+2+16, to specify a file edit box, in which you **can** type the name of a directory to copy to, and which will produce a query if you type the name of an existing file.

If asking for the name of a directory to remove, you might use 4, to allow an existing directory name only.

'Query existing' is ignored if 'disallow existing' is set. These two, as well as 'allow null string input', only work with file edit boxes, not 'matching file' selectors.

**For file selectors, NMPDfile supports file restriction by UID**, or by **type** from the user's point of view. Documents are identified by three UIDs which identify which application created the document and what kind of file it is. Specifying all three UIDs will restrict the files as much as is possible, and specifying fewer will provide less restriction.

You can supply 0 for uid1& and uid2& if you only want to restrict the list to uid3&. This may be useful when dealing with documents from one of your own applications: you can easily find out the third UID as it will be the UID you specified in the APP statement. Note that UIDs are ignored for editors. For example, if your application has UID KUidMyApp&, then the following will list only your application-specific documents:

```
NMPDFile:(Page%, Id%,Prompt$,Flags%,0,KUidOplDoc&,KUidMyApp&,Addr(File$))
REM KUidOplDoc& for OPL docs
```

Some OPL-related UIDs are given in Const.oph.

**FileStringAddr&** is the address of the string variable to edit, where:

```
LOCAL File$(255)
```

File$="C:\Documents\Test.txt"

and the parameter passed to this function is ADDR(File$)

**Important**

The variable File$ to which FileStringAddr& points **must always be declared to be 255 bytes long,** whether it is a string variable, or part of an array of strings.

This is because the filename path returned may be up to this length. If the File$ declaration is shorter than the returned path there will be memory corruption

**The contents of File$ always control** the initial drive and directory used, and contains the selection after the dialog has been dismissed.

**Any filename part is shown** initially in the filename box. For a 'matching file' selector, you can use wildcards in the filename part (such as *.tmp) to control which filenames are matched. To do this,

you must add 128 to flags%. 128 also allows wildcard specifications to be **entered** (returned in File$), for both 'matching' and 'new file' selectors.

With a **matching** file selector (as opposed to an edit box) the value 8 restricts the selection to files which match the filename/extension in file$.

**Matching file selectors** can also use 64, in which case files with the same extension as that in File$ are shown without this extension.

- **You can always press Tab** to produce the full file selector with a NMPDfile item.

## NMPDFLOAT:

NMPDFloat:(Page%, Id%, Prompt$, Trailer$, BYREF Value, Min, Max, Places%)

Defines an edit box for a floating-point number.

**Value**, which must be a variable, specifies the floating point number to be shown initially.

**Min** and **Max** give the minimum and maximum values which are to be allowed. An error is raised if min is higher than max.

**Places%** specifies the number of decimal places that the floating point editor will utilise. If Places% is greater than 0, then the editor will utilise a fixed number of decimal places, otherwise a Places% value of zero will present a normal full length floating point value for editing.

When the dialog is dismissed, the floating point value entered is returned in Value.

## NMPDFONTPREVIEWER:

NMPDFontPreviewer:(Page%, Id%, Prompt$, Trailer$, BYREF Uid&, Preview$)

Defines a single line, read-only text item, which displays the text in Preview$ in the font specified by Uid&.

**Uid&** should be one of the standard OPL font constants (see Const.oph), e.g. KFontArialNormal15&

**Preview$** will be displayed aligned left in the specified font.

## NMPDGRAYSELECTOR:

NMPDGraySelector:(Page%, Id%, Prompt$, Trailer$, BYREF Red%, BYREF Green%, BYREF Blue%, Flags%)

Provides a drop-down list of grey shades from which the user can select a colour.

**Red%**, **Green%** and **Blue%** specify the inital values for each of the respective channels.

**Flags%** must be either

- KMPDGray4Colour% (=2) to specify that a four colour selector is required, or
- KMPDGray16Colour% (=4) if a sixteen colour selector is required.

## NMPDLATITUDEEDITOR:

NMPDLatitudeEditor:(Page%, Id%, Prompt$, Trailer$, BYREF Degrees%, BYREF Minutes%, BYREF Seconds%, BYREF Direction%, Flags%)

Defines a latitude editor.

**Degrees%**, **Minutes%**, **Seconds%** all specify standard longitudinal details.

**Direction%** must be either

- KMPDCompassNorth% (=$1000) or
- KMPDCompassSouth% (=$1001)

**Flags%** must be either

- **KMPDLatLongNoSeconds%** (=0) if the seconds field of the editor is to be omitted, or
- **KMPDLatLongAddSeconds%** (=8) if the seconds field is required.
- **KMPDLatLongUsesDecimals%** (=1024) to allow the 3rd field to contain values in the range of 0-99. The value is set and returned in BYREF seconds%

- **KMPDLatLongFlagsUseDecPrompt%** (=2048) changes the prompt format for the editor from 0°0'0"N to 0°0.0'

## NMPDLINEBUTTON:

NMPDLineButton:(Page%, Id%, Prompt$, Caption$, Flags%, Callback$, BitFile$, BitmapId%, BitmapMaskId%, Flags%, Layout%)

Defines a button to be inserted as a dialog line.

See **NMPDButton** for a description of the variables.

For an example of this control on the Series 5, see **Control panel|International|Date|Workdays**

## NMPDLONG:

NMPDLong:(Page%, Id%, Prompt$, Trailer$, BYREF Value&, Min&, Max&)

Defines an edit box for a long integer.

**Min&** and **Max&** give the minimum and maximum values which are to be allowed. An error is raised if Min& is higher than Max&.

**Value&** must be a variable, not a value. It is initialised to the value to be shown initially. When the dialog is dismissed, the value entered is returned in Value&.

## NMPDLONGITUDEEDITOR:

NMPDLongitudeEditor:(Page%, Id%, Prompt$, Trailer$, BYREF Degrees%, BYREF Minutes%, BYREF Seconds%, BYREF Direction%, Flags%)

Defines a longitude editor.

**Degrees%**, **Minutes%**, **Seconds%** all specify standard longitudinal details. The value for Seconds% is usually in the range of 0-59 unless the flag KMPDLatLongUsesDecimals% is set

**Direction%** must be either

- KMPDCompassWest% (=$2001) or
- KMPDCompassEast% (=$2000)

**Flags%** must be either:

- **KMPDLatLongNoSeconds%** (=0) if the seconds field of the editor is to be omitted, or
- **KMPDLatLongAddSeconds%** (=8) if the seconds field is required.
- **KMPDLatLongUsesDecimals%** (=1024) to allow the 3rd field to contain values in the range of 0-99. The value is set and returned in BYREF seconds%
- **KMPDLatLongFlagsUseDecPrompt%** (=2048) changes the prompt format for the editor from 0°0'0"N to 0°0.0'

## NMPDOPTIONINIT:

NMPDOptionInit:(Page%, Id%, Prompt$, Trailer$,OptCount%, BYREF Choice%, StateChangedCB$)

Prepares a horizontal option list. This is part one (the parent) of a two-phase option list construction process. The second part is **NMPDOption**

**OptCount%** specifies the number of options

**Choice%** specifies the initial option to be selected, and returns the value of which option was selected when the dialog was closed

**StateChangedCB$** is an optional callback procedure in the host OPL program. This is called whenever the user selects a new choice in this control.

See **Callbacks** for an explanation of callback procedures and the mandatory requirements for their use.

## NMPDOPTION:

NMPDOption:(ParentId%, Item%, Caption$)

Adds the actual (child) options to the previously defined horizontal option list (see **NMPDOptionInit**).

**ParentId%** is the unique identifier of the parent option list. If ParentId% is not the control id of an Option list, then a 'type violation' error will be raised.

**Item%** is the unique identifier of this option. If Item% already exists, an 'in use' error will be raised. If there are already the correct number of items specified in the parent, then a -76 error (Bad number) will be raised.

**Caption$** will be displayed before the option radio button.

## NMPDRANGE:

NMPDRange:(Page%, Id%, Prompt$, Trailer$, BYREF Lower&, BYREF Upper&, Min&, Max&)

Allows the user to enter two numbers, separated by a hyphen, to establish a range.

**Lower&** and **Upper&** specify the inital lower and upper values of the editor.

**Min&** and **Max&** specify the limits which the user supplied choices must fall between. Min& must not be greater than Max& otherwise an error will be raised.

When the dialog is dismissed, Lower& and Upper& will contain the values entered by the user.

## NMPDTEXT:

NMPDText:(Page%, Id%, Prompt$, Body$, Flags%)

Defines a line of text to be displayed.

**Prompt$** will be displayed on the left side of the line, and **Body$** on the right side.

**To display a single string**, use a null string ("") for **Prompt$,** and pass the desired string in **Body$**. Body$ will then have the whole width of the dialog to itself. An error is raised if Body$ is a null string.

**Body$** is normally displayed left aligned (although usually in the right column). This can be overriden by specifying **Flags%**:

0=left align Body$

1=right align Body$

2=centre Body$

**However**, alignment of Body$ is only supported when Prompt$ is null, with the body being left aligned otherwise.

**In addition**, you can add any or all of the following values to Flags%, for these effects:

$800 specify this item as a text separator

$1600 use annotation font for Body$ (when Prompt$ is null only)

**A line separator** can be displayed between any dialog items by setting the flag $800 on an item which has null Prompt$ and Body$. If Prompt$ and/or Body$ are not null, then the flag is ignored and no separator is drawn.

## NMPDTIME:

NMPDTime:(Page%, Id%, Prompt$, Trailer$, BYREF Value&, Flags%, Min&, Max& )

Defines an edit box for a time.

**Value&**, which must be a variable and not a value, specifies the time to be shown initially. Although it will appear on the screen like a normal time, for example 18:27, Value& must be specified as seconds after 00:00. A value of 60 means one minute past midnight; 3600 means one o'clock, and so on.

**Min&** and **Max&** give the minimum and maximum values which are to be allowed. Again, these are in seconds after 00:00. An error is raised if Min& is higher than Max&.

When the dialog is dismissed, the time entered is returned in Value&, in seconds after 00:00.

**Flags%** specifies the type of display required, as follows:

- 0 absolute time no seconds
- 1 absolute time with seconds
- 2 duration no seconds

- 3 duration with seconds
- 4 time without hours
- 8 absolute time in 24 hours

### NMPDXINPUT:

NMPDXinput:(Page%, Id%, Prompt$, Trailer$, StringAddr&)

Defines a secret string edit box, such as for a password.

**StringAddr&** is the address of the string variable to take the input string. So where:

Local String$(15)

The parameter passed to this procedure is Addr(String$)

**String$** must be less than 16 characters.

Initially the dialog does not show any characters for the string; the initial contents of String$ are ignored. A special symbol will be displayed for each character you type, to preserve the secrecy of the string.

## 5. Dynamic functions

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

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

Some of the dynamic functions are for setting or getting the current value of a particular control type. Others, are applicable to all controls, irrespective of type (see **State change functions**)

Where relevant, in each of the dynamic functions, **Id%** is the unique control identifier specified when the control was initially constructed. If the Id% value specified does not correspond to a control of the requisite type, a 'type violation' error will be raised.

- Any control whose state changes due to the use of a function inside a callback, will automatically be redrawn upon return from the callback procedure.

### NMPDDYNBUTTON:

NMPDdynSetButton:(Id%, State%)

Changes the state of the specified dialog button. This command is only applicable to **button panel buttons** as specified in the initialisation phase using **NMPDButton**. To change the state of a NMPDLineButton see **NMPDdynSetControlState**.

State% may be one of:

- KMPDButtonStateNotDimmed% (=1)
- KMPDButtonStateDimmed% (=2)
- KMPDButtonStateInvisible% (=3)
- KMPDButtonStateVisible% (=4)

### NMPDDYNSETCHARACTER:

NMPDdynSetCharacter:(Id%, Char%)

Sets the current ASCII character value of a character editor.

### NMPDDYNGETCHARACTER%:

char%=NMPDdynGetCharacter%:(Id%)

Returns the current ASCII character value of the character editor Id%.

### NMPDDYNSETCHECKBOX:

NMPDdynSetCheckBox(Id%, State%)

**State%** is the state that the checkbox should be set to, and must be one of:

- KMPDChkBoxStateClear% (=0) to make the checkbox clear

- KMPDChkBoxStateSet% (=1) to make the checkbox set
- KMPDChkBoxStateIndeterminate% (=2) make the checkbox neither set nor clear

## NMPDDYNGETCHECKBOX%:

state%=NMPDdynGetCheckBox%:(Id%)

Returns the checkbox state of the specified control. See NMPDdynSetCheckbox for the returned values.

## NMPDDYNSETCHOICEITEM:

NMPDdynSetChoiceItem:(Id%, Item%)

Sets the current item in the choice list or bitmap choice list to **Item%**.

## NMPDDYNGETCHOICEITEM%:

choice%=NMPDdynGetChoiceItem%:(Id%)

Returns the current item in the choice list or bitmap choice list Id%.

## NMPDDYNSETCHOICESBYSTRING:

NMPDdynSetChoicesByString:(Id%, Choices$)

Sets the options in the choice list id% to the item with text Choice$, which can be a comma separated list ("Item1, Item2, Item3"). The continuation notation (...) is not supported.

- The purpose of this command is to allow dynamic changing of a **nMPDChoice** control as an alternative to using text arrays and **nMPDChoiceByArray**.

## NMPDDYNGETCHOICEITEMTEXT$:

item$=NMPDdynGetChoiceItemText$:(Id%)

Returns the text of the current item in the text choice list Id%.

- This command will raise a 'type violation' error if the Id% parameter is a bitmap choice list.

## NMPDDYNSETCHOICES:

NMPDdynSetChoices:(Id%, ChoiceArrayId&)

Changes the contents of a choice list or bitmap choice list to the specified array ChoiceArrayId&. Id% must be a valid text choice list, or bitmap choice list.

**Important** The border around the choice items does not redraw when the array is swapped. This means that there may not be full viewing of a item in the new array which is wider than the widest item in the array used when the control was first constructed. For example:

If the initial text array widest item is **Neuon**, the new array item **Longer item** would show **Longe**

The solution is to make sure that the one item in the array used to construct the control is the same width as the widest anticipated item.

- .
  - ChoiceArrayId& will replace the contents of the existing control
  - ChoiceArrayId& must contain at least one choice otherwise an Invalid Arguments error will be raised
  - The current item will be reset to the first item in the choice list
  - If in the original specification of the choice list or bitmap choice list the **KMPDChoiceFreeArrayOnClose%** flag was specified, any such value will be ignored after the array has been changed. The developer is now responsible for the dynamic array's destruction.

## NMPDDYNSETDATE:

NMPDdynSetDate:(Id%, Date&)

Sets the date of a date editor. See the OPL manual for the specification of **Date&**.

## NMPDDYNGETDATE&:

date&=NMPDdynGetDate&:(Id%)

Returns the current date value specified in the date editor Id%.

## NMPDDYNSETEDIT:

NMPDdynSetEdit(Id%, Value$)

Sets the current value of a text editor.

- **This function cannot be used** with NMPDEditMulti or NMPDXInput dialog controls. It only works with NMPDEdit

## NMPDDYNGETEDIT$:

text$=NMPDdynGetEdit$:(Id%)

Returns the current text of a text or secret editor. If Id% is not a dXInput or dEdit line a 'type violation' error will be raised.

## NMPDDYNSETEDITMULTI:

NMPDdynSetEditMulti:(Id%, BufferAddr&, MaxLength%)

Changes the contents of a dEditMulti to that specified by BufferAddr& (see **NMPDEditMulti** for a description of this parameter).

- The current contents of the dEditMulti are discarded.
- MaxLength% is the new maximum length of the buffer passed to this procedure.
- When the dialog is dismissed, the dEditMulti contents will be placed in the **new** buffer (as specified by this procedure) **not the original** buffer.

## NMPDDYNGETEDITMULTI:

NMPDdynSetEditMulti:(Id%, BufferAddr&, MaxLength%)

Retrieves up to MaxLength% bytes of the contents of the dEditMulti Id% and places into the buffer address specified.

- The current contents of the buffer passed as a parameter are overwritten if they exist.
- If the current dEditMulti contains more than MaxLength% characters, only MaxLength% characters will be copied to BufferAddr&.
- The current dEditMulti contents are unaffected.

## NMPDDYNSETFONT:

NMPDdynSetFont:(Id%, Font&)

Sets the font of a font preview label. **Font&** must be a valid font unique identifier (see Const.oph).

## NMPDDYNGETFONT&:

uid&=NMPDdynGetFont&:(Id%)

Returns the current font utilised by a font preview label in the form of a unique identifier value (see Const.oph).

## NMPDDYNSETFLOAT:

NMPDdynSetFloat:(Id%, Value)

Sets the floating point value of Id%.

## NMPDDYNGETFLOAT:

value=NMPDdynGetFloat:(Id%)

Returns the floating point value currently contained within Id%.

## NMPDDYNSETGRAY:

NMPDdynSetGray:(Id%, Red%, Green%, Blue%)

Sets the current value of the gray selector Id%.

**Red%**, **Green%**, and **Blue%** specify the red, green and blue values of each channel respectively. The minimum value for each channel is 0, and the maximum is 255.

- If the values specified to not map directly onto a colour in the current palette, the nearest gray value will be selected.

## NMPDDYNGETGRAY:

NMPDdynGetGray:(Id%, BYREF Red%, BYREF Green%, BYREF Blue%)

Returns the current value of the gray selector Id%.

**Red%**, **Green%**, and **Blue%** specify the red, green and blue values of each channel respectively, and upon return from this function will be updated to the correct channel value.

## NMPDDYNSETLATITUDE:

NMPDdynSetLatitude:(Id%, Deg%, Min%, Sec%, Dir%)

Sets the current value of the latitudinal editor Id%.

**Deg%**, **Min%**, and **Sec%** all specify the standard values of such an editor. **Dir%** must be one of the direction constants specified in NMPDConstants.oxh:

- KMPDCompassNorth% (=$1000)
- KMPDCompassSouth% (=$1001)

## NMPDDYNGETLATITUDE:

NMPDdynGetLatitude:(Id%, BYREF Deg%, BYREF Min%, BYREF Sec%, BYREF Dir%)

Returns the current value of the latitudinal editor Id%. On return from the function call, the BYREF variables are updated to reflect the current state of the editor.

See **NMPDDynSetLatitude** for a description of the Dir% constant value.

## NMPDDYNSETLONG:

NMPDdynSetLong:(Id%, Value&)

Sets the long integer value of Id%.

## NMPDDYNSETLONGRANGE:

NMPDdynSetLongRange:(Id%, Min%, Max%)

Resets the range of a dLong control to Min%, Max%.

## NMPDDYNGETLONG&:

long&=NMPDdynGetLong&:(Id%)

Returns the long integer value of Id%, an NMPDLong control.

## NMPDDYNSETLONGITUDE:

NMPDdynSetLongitude:(Id%, Deg%, Min%, Sec%, Dir%)

Sets the current value of the longitudinal editor Id%.

**Deg%**, **Min%**, and **Sec%** all specify the standard values of such an editor. **Dir%** must be one of the direction constants specified in NMPDConstants.oxh:

- KMPDCompassEast% (=$2000)
- KMPDCompassWest% (=$2001)

## NMPDDYNGETLONGITUDE:

NMPDdynGetLongitude:(Id%, BYREF Deg%, BYREF Min%, BYREF Sec%, BYREF Dir%)

Returns the current value of the longitudinal editor Id%.

On return from the function call, the BYREF variables are updated to reflect the current state of the editor.

See **NMPDDynSetLongitude** for a description of the Dir% constant value.

## NMPDDYNSETOPTION:

NMPDdynSetOption:(Id%, Option%)

Sets the currently selection option in a horizontal option list.

**Option%** must be a valid child option as specified in NMPDOption.

## NMPDDYNGETOPTION%:

option%=NMPDdynGetOption%:(Id%)

Returns the currently selected option in the option list Id%. The returned value will be one of the child options specified during the construction of the option list itself (see **NMPDOption**).

## NMPDDYNSETRANGE:

NMPDdynSetRange:(Id%, Upper&, Lower&)

Sets the upper and lower range value of a range editor.

## NMPDDYNGETRANGE:

NMPDdynGetRange:(Id%, BYREF Upper&, BYREF Lower&)

Returns the current upper and lower values utilised by a range editor.

## NMPDDYNSETTEXT:

NMPDdynSetText:(Id%, Value$)

Sets the current value of a text label.

- This function is only for use with **NMPDText** dialog lines.

## NMPDDYNSETTIME:

NMPDdynSetTime:(Id%, Time&)

Sets the time of a time editor. See the OPL manual for the specification of **Time&**.

## NMPDDYNGETTIME&:

time&=NMPDdynGetTime&:(Id%)

Returns the current time value specified in the time editor Id%.

## Miscellaneous dynamic functions

The following series of miscellaneous dynamic functions are solely for use in callback procedures.

## NMPDDYNTOPLEFT:

NMPDdynTopLeft:(BYREF X%, BYREF X%)

returns the co-ordinates in pixels of the top left corner of the dialog

### NMPDDYNWIDTH%:

Width%=NMPDdynWidth%:

returns the width in pixels of the dialog

### NMPDDYNHEIGHT%:

Height%=NMPDdynHeight%:

returns the height in pixels of the dialog

### NMPDDYNBUTTONCORNER:

NMPDdynButtonCorner:(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 NMPDButton **but without any OR'd modifier** ($100 or $200)

  **For example**, if the NMPDButton keycode% parameter was %E OR $100, the Id% parameter would be just %E

- **corner%** is one of

  KMPDTopLeft% = 0

  KMPDBottomLeft% = 1

  KMPDTopRight% = 2

  KMPDBottomRightt% = 3

## State change dynamic functions

The following series of functions are solely for use in callback procedures, and are applicable to all controls (regardless of type). In addition, some functions may also be applied to pages themselves.

### NMPDDYNSETPROMPT:

NMPDdynSetPrompt:(Id%, Prompt$)

Changes the prompt for the control Id% to Prompt$

- If Prompt$ is longer than the current prompt length, it will be truncated to the current length.

### NMPDDYNSETFOCUS:

Result=NMPDdynSetFocus:(Id%)

Makes Id% the currently focused control. **Note that Id% must correspond to** a control or a page id% otherwise a 'not found' error will be raised.

If it was not possible to set focus to the specified control (for instance, if the control is dimmed, invisible or non-focusing by default) then this function returns –1.0

### NMPDDYNGETFOCUS%:

controlId%=NMPDdynGetFocus%:

Returns the currently focused control or page identifier.

### NMPDDYNSETCONTROLSTATE:

NMPDdynSetControlState:(Id%, State%)

Sets the state of **Id%** to the value specified by **State%**, where Id% may be either a page identifier or a control identifier.

**If Id% is a page identifier** then valid values for State% are:

- KMPDCtrlStateDimmed%(=1)
- KMPDCtrlStateNotDimmed% (=2)

**If Id% is a control identifier** then State% may additionally be:

- KMPDCtrlStateVisible% (=3)
- KMPDCtrlStateNotVisible% (=4)

### NMPDDYNGETCONTROLSTATE%:

State%=NMPDdynGetControlState%:(Id%)

Returns the current state of the control or page identifier Id%.

### NMPDDYNGETCONTROLTYPE%:

type%=NMPDdynGetControlType%:(Id%)

Returns the control type of the control identifier Id%. A listing of valid control types is also specified in NMPDConstants.oxh

KMPDCtrlUnknown% = -1

KMPDCtrlBitmapChoice% = 1

KMPDCtrlBitmapViewer% = 2

KMPDCtrlButton% = 3

KMPDCtrlCharEditor% = 4

KMPDCtrlCheckBox% = 5

KMPDCtrlChoice% = 6

KMPDCtrlDate% = 7

KMPDCtrlEdit% = 8

KMPDCtrlEditMulti% = 9

KMPDCtrlFloat% = 10

KMPDCtrlFontPreviewer% = 11

KMPDCtrlGraySelector% = 12

KMPDCtrlLatitudeEditor% = 13

KMPDCtrlLineButton% = 14

KMPDCtrlLong% = 15

KMPDCtrlLongitudeEditor% = 16

KMPDCtrlOption% = 17

KMPDCtrlRange% = 18

KMPDCtrlText% = 19

KMPDCtrlTime% = 20

KMPDCtrlXInput% = 21

## 6. String arrays

In order to facilitate simple management of choice lists, NMPD.opx provides the ability to manage text-based string arrays.

The **NMPDArray...** series of functions describe the available dynamic string array facilities.

### NMPDARRAYCREATE&:

Id& = NMPDArrayCreate&:

Creates an array object and returns a handle **Id&** for it.

## NMPDARRAYFREE:

NMPDArrayFree:(BYREF Id&)

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

## NMPDARRAYADDITEM:

NMPDArrayAddItem:(Id&, Item$)

Adds the string **Item$** to the array with handle **Id&**.

## NMPDARRAYITEMCOUNT%:

Count% = NMPDArrayItemCount%:(Id&)

Returns a count of the items in the array with handle **Id&**

## NMPDARRAYITEMAT$:

Item$ = NMPDArrayItemAt$:(Id&, Position%)

Returns a string **Item$** with the contents of the item at **Position%** in the array with handle **Id&**

## NMPDARRAYITEMLENGTH%:

Length% = NMPDArrayItemLength%:(Id&, Position%)

Returns the length of the array item at **Position%**, in the array with handle **Id&**

## NMPDARRAYREPLACEITEM:

NMPDArrayReplaceItem:(Id&, Position%, ReplaceWith$)

Changes the array item at **Position%** to the string **ReplaceWith$**, in the array with handle **Id&**.

## NMPDARRAYINSERTITEM:

NMPDArrayInsertItem:(Id&, Position%, StringToInsert$)

Inserts **StringToInsert$** at **Position%** in the array with handle **Id&**.

## NMPDARRAYDELETE:

NMPDArrayDelete:(Id&, Position%)

Deletes the array entry at **Position%** in the array with handle **Id&**

## NMPDARRAYDELETEITEMS:

NMPDArrayDeleteItems:(Id&, StartPosition%, EndPosition%)

Deletes the array entries from **StartPosition%** to **EndPosition%** (inclusive) in the array with handle **Id&**

## NMPDARRAYDELETEALL:

NMPDArrayDeleteAll:(Id&)

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

## NMPDARRAYSORT:

NMPDArraySort:(Id&, Flag%)

Sorts the array with handle **Id&**, according to the value of **Flag%**

Constants for **Flag%** are:

- KMPDArraySortNormal% = 0
- KMPDArraySortFolded% = 2

- KMPDArraySortCollated% = 4

## NMPDARRAYFIND%:

Position% = NMPDArrayFind%:(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 are

- KMPDArrayFindSequential% (=8) is a sequential search
- KMPDArrayFindBinarySearch%(=16) is a binary search

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

## 7. Bitmap arrays

The **NMPDB...**series of functions create and manage an array of bitmaps. There is no limit to the size of the array (subject to available memory). The bitmap array can be populated from, and saved to, an mbm file.

Bitmap arrays are used solely as the source of any choice list bitmaps in the bitmap choice list control.

## NMPDBARRAYCREATE&:

Id&=NMPDBArrayCreate&:

Creates a bitmap array and returns a handle to it.

- See **NMPDBFree** for details of how to recover all the resources used to store the array.

## NMPDBFREE:

NMPDBFree:(BYREF Id&)

Deletes any bitmaps in the bitmap array with handle **Id&**, and releases all resources used. Id& is set to 0.

This is the **only** way to release all the resources used by the bitmap array object. It is **very** important that this function is called before an OPL32 program leaves. If not, bitmaps may be orphaned resulting in a memory leak.

## NMPDBADD:

NMPDBAdd:(Id&, WindowId%, File$, MbmIndex%)

Adds a bitmap to the bitmap array with handle **Id&**

If **WindowId%** is specified (as returned by gCreate or gCreateBit) adds the window/bitmap contents.

If **File$** (a bitmap or MBM) is specified, adds that bitmap. For an mbm, **MbmIndex%** specifies the required bitmap, with 0 as the first, or only, bitmap. **WindowId%** must be 0 if file$ is specified.

## NMPDBINSERT:

NMPDBInsert:(Id&, Position%, WindowId%, File$, MbmIndex%)

Inserts a bitmap into the bitmap array with handle **Id&** at **Position%**.

If **WindowId%** is specified (as returned by gCreate or gCreateBit) adds the window/bitmap contents.

If **File$** (a bitmap or MBM) is specified, adds that bitmap. For an mbm, MbmIndex% specifies the required bitmap, with 0 as the first, or only, bitmap. WindowId% must be 0 if file$ is specified.

## NMPDBDELETE:

NMPDBDelete:(Id&, Position%)

Deletes the bitmap at **Position%** from the bitmap array with handle **Id&**

Deleting all bitmaps in the bitmap array **does not** delete the array object itself. See **NMPDBFree**

## NMPDBGET:

NMPDBGet:(Id&, Position%, WindowId%, PosX%, PosY%, File$ ) :

Gets the bitmap at **Position%** from the bitmap array with handle **Id&**

If the drawable **WindowId%** is specified, as returned by gCreate, the bitmap is drawn with the top left corner at **PosX%**, **PosY%**

If **File$** is specified, the bitmap is saved to File$, which will be overwritten if it exists. WindowId% must be 0

## NMPDBCOUNT%:

total%=NMPDBCount%:(Id&)

Returns the total bitmaps in the bitmap array with handle **Id&**.

## NMPDBSIZE:

NMPDBSize:(Id&, Position%, BYREF Width%, BYREF Height%)

Sets the BYREF variables to the dimensions of the bitmap at **Position%** in the bitmap array with handle **Id&**.

```
rem nMPDConstants.OXH
rem Constants Header File for use with nMPD.OPX
rem © Copyright Alex Wilbur, Henry Hirst and Neuon 2000

rem ================================================
rem Bitwise dialog flags for NMPDInit:
rem ================================================
const KMPDButRight%                 = 1
const KMPDNoTitle%                  = 2
const KMPDFullScreen%               = 4
const KMPDNoDrag%                   = 8
const KMPDDensePacking%             = 16

rem ================================================
rem Button state constants
rem ================================================
const KMPDButtonStateNotDimmed%     = 1
const KMPDButtonStateDimmed%        = 2
const KMPDButtonStateInvisible%     = 3
const KMPDButtonStateVisible%       = 4

rem ================================================
rem Lat/Long Editor Direction Constants
rem ================================================
const KMPDCompassNorth%             = $1000
const KMPDCompassSouth%             = $1001
const KMPDCompassEast%              = $2000
const KMPDCompassWest%              = $2001
const KMPDLatLongNoSeconds            = 0
const KMPDLatLongFlagsAddSeconds%   = 8
const KMPDLatLongFlagsUseGPSRange% = 1024

rem ================================================
rem Button layout constants
rem ================================================
const KMPDButTextRight%             = 0
const KMPDButTextBottom%            = 1
const KMPDButTextTop%               = 2
const KMPDButTextLeft%              = 3

rem ================================================
rem Control state constants
rem ================================================
```

```
const KMPDCtrlStateDimmed%          = 1
const KMPDCtrlStateNotDimmed%       = 2
const KMPDCtrlStateVisible%         = 3
const KMPDCtrlStateNotVisible%      = 4

rem ================================================
rem Check box state constants
rem ================================================
const KMPDChkBoxStateClear%         = 0
const KMPDChkBoxStateSet%           = 1
const KMPDChkBoxStateIndeterminate% = 2

rem ================================================
rem Dialog exit constants for use in callback procedures
rem ================================================
const KMPDDialogCannotExit%         = 0
const KMPDDialogCanExit%            = 1

rem ================================================
rem Editor flag constants
rem For use with NMPDEdit & NMPDEditMulti
rem See command lsiting for applicability
rem ================================================
const KMPDEditorDefault%            = 0
const KMPDEditorReadOnly%           = 1
const KMPDEditorFoceHorizontalSB%   = 2
const KMPDEditorForceVerticalSB%    = 4
const KMPDEditorNoWrap%             = 8
const KMPDEditorHex%               = 16

rem ================================================
rem Grey scale selector constants
rem ================================================
const KMPDGray4Colour%              = 2
const KMPDGray16Colour%             = 4

rem ================================================
rem Control type constants
rem ================================================
const KMPDCtrlBitmapChoice%         = 1
const KMPDCtrlBitmapViewer%         = 2
const KMPDCtrlButton%               = 3
const KMPDCtrlCharEditor%           = 4
```

```
const KMPDCtrlCheckBox%          = 5
const KMPDCtrlChoice%            = 6
const KMPDCtrlDate%              = 7
const KMPDCtrlEdit%              = 8
const KMPDCtrlEditMulti%         = 9
const KMPDCtrlFloat%             = 10
const KMPDCtrlFontPreviewer%     = 11
const KMPDCtrlGraySelector%      = 12
const KMPDCtrlLatitudeEditor%    = 13
const KMPDCtrlLineButton%        = 14
const KMPDCtrlLong%              = 15
const KMPDCtrlLongitudeEditor%   = 16
const KMPDCtrlOption%            = 17
const KMPDCtrlRange%             = 18
const KMPDCtrlText%              = 19
const KMPDCtrlTime%              = 20
const KMPDCtrlXInput%            = 21
const KMPDCtrlEditHex%           = 22

rem ================================================
rem  Choice control constants
rem  NMPDChoice, NMPBChoiceByArray & NMPDBitmapChoice
rem  see Command listing for applicability
rem ================================================
const KMPDChoiceDefault%         = 0
const KMPDChoiceUseIncMatching%  = 1
const KMPDChoiceFreeArrayOnClose% = 2

rem ================================================
rem  String array constants
rem ================================================
const KMPDArraySortNormal%       = 0
const KMPDArraySortFolded%       = 2
const KMPDArraySortCollated%     = 4
const KMPDArrayFindSequential%   = 8
const KMPDArrayFindBinarySearch% = 16
```

**NMPD Demonstration code**

```
include "NMPD.oxh"
include "NMPDConstants.oxh"

rem ================================
rem Demonstration code for NMPD.opx
rem
rem Requires bitmaps.mbm and neuon.mbm
rem in the same folder as this test code
rem
rem All callback procedures end with CB%
rem Control id convention is [page][line]
rem where 21 = page 2, line 1
rem and 6 = page 6
rem ================================

PROC Main:
        rem Misc vars
                LOCAL flags%, pageCount%, i%, ret%, c%, ptext&
                LOCAL off%(6), j%

        rem Page 1 vars (choices)
                LOCAL cbState%, chItem%, opt%,cb1State%
                GLOBAL arrayId&(3), bitmapId&

        rem Page 2 vars (text)
                LOCAL edString$(255), secString$(10)

        rem Page 3 vars (date & time)
                LOCAL dtValue&, durValue&, timeValue1&, timeValue2&

        rem Page 4 vars (numbers)
                LOCAL fpValue, fxValue, long&, lower&, upper&

        rem Page 5 vars (file)
                LOCAL file$(255), file2$(255)
                REM IMPORTANT  These variables must ALWAYS be max length
                REM strings.

        rem Page 6 vars (advanced/custom controls)
                LOCAL bmChoice%, r%, g%, b%
```

```
	rem Page 7 vars (misc)
		LOCAL degreesLong%, minutesLong%, secondsLong%, dirLong%
		LOCAL degreesLat%, minutesLat%, secondsLat%, dirLat%
		LOCAL fontId&, char%

	rem Page 8 vars (multi edwin)
		LOCAL dataAddr&, buffer&(101)

	rem check the required bitmap files are present
		IF NOT(EXIST(PARSE$("bitmaps.mbm",CMD$(1),off%())))
			ALERT(ERR$(-33),PARSE$("bitmaps.mbm",CMD$(1),off%()))
			STOP
		ENDIF
		IF NOT(EXIST(PARSE$("neuon.mbm",CMD$(1),off%())))
			ALERT(ERR$(-33),PARSE$("neuon.mbm",CMD$(1),off%()))
			STOP
		ENDIF

	rem ============================
	rem Create some data to work with
		print "Welcome to Neuon's NMPD.opx"
		print "_____________________________"
		print
		print CHR$(34)+"Creating specimen array data"+CHR$(34)
		print "Normally this would be done long"
		print "before the dialog was required"
		print
		print CHR$(34)+"Building text arrays"+CHR$(34)

	rem Create 3 string arrays
		arrayId&(1)			= NMPDArrayCreate&:
		arrayId&(2)			= NMPDArrayCreate&:
		arrayId&(3)			= NMPDArrayCreate&:

	rem populate the 3 string arrays
		DO
			j%=j%+1
			i%=0
			DO
				NMPDArrayAddItem:(ArrayId&(j%),	CHR$(65+i%)+" item
"+GEN$(i%+1,3) + " [Array "+NUM$(j%,1)+"]")
				i%=i%+1
			UNTIL i%=26
		UNTIL j%=3
```

```
	print "Done"
	print
	print CHR$(34)+"Building the bitmap array"+CHR$(34)
	print "It takes time to load all the bitmaps from file"

rem create bitmap array
	bitmapId&			= NMPDBArrayCreate&:

rem populate the bitmap array
	i% = 0
	DO
		giprint "Adding bitmap "+NUM$(i%+1,3),2
		NMPDBAdd:(bitmapId&, 0, PARSE$("bitmaps.mbm",CMD$(1),off%()), i%)

		i%=i%+1
	UNTIL i%=20
	print "Done"
	print
	giprint "Completed bitmaps",2

rem end of creating test data
rem ==============================

	print CHR$(34)+"Creating controls"+CHR$(34)

rem NMPDInitSetup
	pageCount%	= 8
	edString$	= "Test edit window"
			flags%			= KMPDButRight%
	rem	flags%			= flags% OR KMPDNoTitle%
			flags%			= flags% OR KMPDFullScreen%
	rem	flags%			= flags% OR KMPDNoDrag%
			flags%			= flags% OR KMPDDensePacking%

rem Initialise...
	NMPDInit:("Multipage dialog opx from Neuon", flags%, pageCount%)

rem Prepare pages...
	NMPDPage:(1, "Choices")
	NMPDPage:(2, "Text")
	NMPDPage:(3, "Date & Time")
	NMPDPage:(4, "Numbers")
	NMPDPage:(5, "Files")
```

```
        NMPDPage:(6, "Bitmaps")
        NMPDPage:(7, "Misc")
        NMPDPage:(8, "EditMulti")

    rem Prepare buttons...
        NMPDButton:("Test", %T, "TestsCB", "", 0, 0, KMPDButTextRight%)
        NMPDButton:("Cancel", 27 OR $100, "", "", 0, 0, KMPDButTextRight%)
        NMPDButton:("OK", 13 OR $100, "", "", 0, 0, KMPDButTextRight%)

    rem Setup page 1 controls (choices)
        cbState% = 1
        NMPDCheckBox:(1, 11, "Checkbox", "select to allow EditMulti page", cbState%,
"DynamicCB")
        chItem% = 1
        NMPDChoice:(1, 12, "Choice list", "linked to the choice array list",
chItem%,"1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26", "DynamicCB",
KMPDChoiceUseIncMatching%)
        NMPDChoiceByArray:(1, 13, "Choice array", "linked to the choice list", chItem%,
arrayId&(1), "DynamicCB", 0)
        opt% = 131      REM controlId% for the initial option
        NMPDOptionInit:(1, 14, "Option", "changes the Choice array", 3, opt%, "DynamicCB")
        NMPDOption:(14, 131, "One")
        NMPDOption:(14, 132, "Two")
        NMPDOption:(14, 133, "Three")
        NMPDBitmapChoice:(1, 15, "Bitmap choice list", "", bmChoice%, bitmapId&,
"DynamicCB", 0)
        cb1State%=KMPDChkBoxStateSet%
        NMPDCheckBox:(1, 16, "Checkbox", "shows/hides the option buttons", cb1State%,
"DynamicCB")

    rem Setup page 2 controls (text)
        NMPDEdit:(2, 21, "Edit", "", ADDR(edString$), 25, 255, 0)
        NMPDXInput:(2, 22, "Secret", "(hidden)", ADDR(secString$), 10)
        NMPDText:(2, 23, "", "Test - normal font (left)", 0)
        NMPDText:(2, 24, "", "", $800)
        NMPDText:(2, 25, "", "Test - normal font (right)", $201)
        NMPDText:(2, 26, "", "Annotation font (centered)", $1002)

    rem Setup page 3 controls (date & time)
        timeValue1& = (INT(HOUR) * 3600) + (MINUTE * 60) + SECOND
        dtValue& = DAYS(16,8,1978)
        NMPDDate:(3, 31, "Date", "(a date)", dtValue&, 0, DAYS(1,1,2000))
        NMPDTime:(3, 32, "Time", "(no secs)", timeValue1&, 0, 1, (INT(23)*3600)+(59*60)+59)
        timeValue2& = 200
        NMPDTime:(3, 33, "Time", "(24hr)", timeValue2&, 8, 0, 3600)
```

```
        durValue& = 61
        NMPDTime:(3, 34, "Duration", "", durValue&, 6, 60, 360)

    rem Setup page 4 controls (numbers)
        fpValue = PI : NMPDFloat:(4, 41, "Float", "Normal length rules apply", fpValue, 0, 2*PI,
0)
        fxValue = PI : NMPDFloat:(4, 42, "Fixed", "Max. 4 decimal places", fxValue, 0, 2*PI, 4)
        long& = 200  : NMPDLong:(4, 43, "Long", "Max. val is 4000", long&, 0, 4000)
        lower& = 1
        upper& = 200
        NMPDRange:(4, 44, "Range", "", lower&, upper&, -1000, 1000)

    rem Setup page 5
        file$ = "C:\"
        file2$ = "C:\System\"
        NMPDFile:(5, 51, "File,Folder,Disk", 512, 0, 0, 0, ADDR(file$))
        NMPDText:(5, 52, "", "", $800)
        NMPDFile:(5, 53, "Folder,Disk", 516, 0, 0, 0, ADDR(file2$))

    rem Setup page 6 (advanced/custom controls)
        NMPDBitmapViewer:(6, 61, "Bitmap view", "", 0,
PARSE$("neuon.mbm",CMD$(1),off%()), 0, 382, 120)
        NMPDGraySelector:(6, 62, "Grey selector", "", r%, g%, b%, KMPDGray16Colour%)

    rem Setup page 7 (misc)
        degreesLong% = 15
        minutesLong% = 30
        secondsLong% = 45
        dirLong%                    = KMPDCompassEast%
        NMPDLongitudeEditor:(7, 71, "Longitude", "", degreesLong%, minutesLong%,
secondsLong%, dirLong%, 8)

        degreesLat%   = 15
        minutesLat%   = 30
        secondsLat%   = 45
        dirLat%             = KMPDCompassNorth%
        NMPDLatitudeEditor:(7, 72, "Latitude", "", degreesLat%, minutesLat%, secondsLat%,
dirLat%, 8 OR KMPDLatLongFlagsUseGPSRange%)

        fontId&             = 268436069
        NMPDFontPreviewer:(7, 73, "Font", "", fontId&, "Font preview label text")

        char%                       = 64 rem @
        NMPDCharEditor:(7, 74, "Character", "", char%)
```

```
		NMPDLineButton:(7, 75, "Button", "", "Button", "ButtonTapCB",
PARSE$("bitmaps.mbm",CMD$(1),off%()), 0, 1, %W, KMPDButTextRight%)


	rem Setup page 8
		PutString:(ADDR(buffer&()), "Some text to edit. You can copy text in Word and paste it
here using Ctrl+V.")
		NMPDEditMulti:(8, 81, "", ADDR(buffer&()), 30, 399, 7, 0)


	rem end of intialisation and configuration
		print "Done"


	rem Run dialog
		print
		cls
		ret% = NMPDDisplay%:(11, "ConstructionCB", "PageChangeCB")


	rem IMPORTANT!!!!!
	rem	Don't forget to free any array resources used
		NMPDArrayFree:(arrayId&(1))
		NMPDArrayFree:(arrayId&(2))
		NMPDArrayFree:(arrayId&(3))
		NMPDBFree:(bitmapId&)


	rem process the results
		cls
		if	(ret% = 0) REM Esc pressed
			return
		endif


		PRINT "Time: ",timeValue1&,timeValue2&
		PRINT "EditString:",edString$
		PRINT "EditMulti contents"
		PRINT "Length:";buffer&(1)
		PRINT "Text: ";


		PrintBuffer:(ADDR(buffer&()))


		print "File1 is:",file$
		print "File2 is:",file2$
		print
		print "Press any key to quit"
		get
ENDP
```

```
REM ================================================================
REM This is the mandatory callback manager procedure

PROC NMPDCallBackManager%:(Param%, ProcName$)
        ONERR CallbackError::

        return @%(ProcName$):(Param%)

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

REM ================================================================
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

REM ================================================================
PROC PageChangeCB%:(aPageId%)
        giprint "Page change callback: page "+NUM$(aPageId%,3),2
ENDP

REM ================================================================
PROC TestsCB%:(NotUsed%)
        LOCAL x%, y%

        NMPDDynSetDate:(31, DAYS(17,8,1978))
        NMPDDynSetTime:(32, 3600 * 5)
        NMPDdynButtonCorner:(%T, 0, x%, y%)

        NMPDdynSetFocus:(3)

        dINIT "Tests callback"
        dTEXT "","Pressing this button has invoked a callback procedure attached to"
        dTEXT "","the button. In this example, PROC TestsCB%: does the following:"
        dTEXT "," "
```

```
        dTEXT "",CHR$(149)+" Sets the focus to the 'Date & Time' page"
        dTEXT "",CHR$(149)+" Dynamically changes the date and time"
        dTEXT "",CHR$(149)+" Reports the top left corner of the button as x%="+GEN$(x%,3)+" y%="
+GEN$(y%,3)
        dBUTTONS "Continue", 13 OR $100
        DIALOG
        return KMPDDialogCannotExit%
ENDP

REM ================================
PROC ButtonTapCB%:(NotUsed%)
        GIPRINT "Button callback",2
ENDP

REM ========================================================================
PROC DynamicCB%:(CtrlId%)
        LOCAL type%

        type% = NMPDDynGetControlType%:(CtrlId%)

        if              (type% = KMPDCtrlBitmapChoice%)
                giprint "Bitmap choice changed to "+NUM$(NMPDdynGetChoiceItem%:(CtrlId%),3),2

        elseif (type% = KMPDCtrlCheckBox%) AND (CtrlId%=11)
                giprint "Check box with id="+NUM$(CtrlId%,3)+" changed",2
                if              NMPDDynGetCheckbox%:(CtrlId%) = KMPDChkBoxStateSet%
                        REM Un-dim the dEditMulti page
                        NMPDdynSetControlState:(8, KMPDCtrlStateNotDimmed%)

                else
                        REM Dim the dEditMulti page
                        NMPDdynSetControlState:(8, KMPDCtrlStateDimmed%)
                endif

        elseif (type% = KMPDCtrlCheckBox%) AND (CtrlId%=16)
                if              NMPDDynGetCheckbox%:(CtrlId%) = KMPDChkBoxStateSet%
                        REM show the option list
                        NMPDdynSetControlState:(14,KMPDCtrlStateVisible%)
                else
                        REM hide the option list
                        NMPDdynSetControlState:(14,KMPDCtrlStateNotVisible%)
                endif

        elseif (type% = KMPDCtrlChoice%)
```

```
        REM link the two choice lists
        REM checking that they only change if they are not equal
        REM or a deathful loop will be created as the controls
        REM are changed.
        REM The two choice lists are of equal number
        if              CtrlId% = 12
                if NMPDDynGetChoiceItem%:(13)<>NMPDDynGetChoiceItem%:(12)
                        NMPDDynSetChoiceItem:(13, NMPDDynGetChoiceItem%:(CtrlId%))

                endif
        elseif CtrlId%=13
                if NMPDDynGetChoiceItem%:(12)<>NMPDDynGetChoiceItem%:(13)
                        NMPDDynSetChoiceItem:(12, NMPDDynGetChoiceItem%:(CtrlId%))

                endif
        endif
elseif (type% = KMPDCtrlOption%)
        giprint "Choice array swapped",2
        REM swop the choice array
        NMPDdynSetChoices:(13, ArrayId&(NMPDDynGetOption%:(CtrlId%)-130))

        REM Set it to same value as the Choice list
        if NMPDDynGetChoiceItem%:(13)<>NMPDDynGetChoiceItem%:(12)
                NMPDDynSetChoiceItem:(13, NMPDDynGetChoiceItem%:(12))

        endif

elseif (type% = KMPDCtrlLineButton%)
        giprint "Line button with id="+NUM$(CtrlId%,3)+" changed",2
elseif (type% = KMPDCtrlBitmapViewer%)
elseif (type% = KMPDCtrlButton%)
elseif (type% = KMPDCtrlCharEditor%)
elseif (type% = KMPDCtrlDate%)
elseif (type% = KMPDCtrlEdit%)
elseif (type% = KMPDCtrlEditMulti%)
elseif (type% = KMPDCtrlFloat%)
elseif (type% = KMPDCtrlFontPreviewer%)
elseif (type% = KMPDCtrlGraySelector%)
elseif (type% = KMPDCtrlLatitudeEditor%)
elseif (type% = KMPDCtrlLong%)
elseif (type% = KMPDCtrlLongitudeEditor%)
elseif (type% = KMPDCtrlRange%)
elseif (type% = KMPDCtrlText%)
elseif (type% = KMPDCtrlTime%)
```

```
        elseif (type% = KMPDCtrlXInput%)
        else
                giprint "Unrecognised control type",2
        endif
ENDP

PROC PutString:(BufferAddr&, String$)
LOCAL Count%, Addr&
        Addr& = BufferAddr& + 4
        POKEB BufferAddr&, LEN(String$)
        Do
                Count% = Count% + 1
                POKEB Addr& + Count% - 1, ASC(Mid$(String$, Count%, 1))
        Until Count%=LEN(String$)
ENDP

PROC PrintBuffer:(BufferAddr&)
LOCAL i%, pText&, length%
                i%=0
                length%         = PEEKL(BufferAddr&)
                pText&          = BufferAddr& + 4
                WHILE i%<length%
                        PRINT CHR$(PEEKB(pText&+i%));
                        i%=i%+1
                ENDWH
                print
ENDP
```

**rem NMPD.OXH**

**rem Header File for NMPD.OPX**

rem



rem Standard OPX constants

const KUidOpxNMPD& = &10004D40

const KNMPDOpxVersion% = $120

DECLARE OPX NMPD, KUidOpxNMPD&, KNMPDOpxVersion%

**rem Initialisation, configuration & display**

NMPDInit:(Title$, Flags%, Pages%) : 1

NMPDCancel:() : 2

NMPDPage:(PageNumber%, Caption$) : 3

NMPDPosition:(TopLeftX%, TopLeftY%) : 4

NMPDDisplay%:(InitControl%, ConstructionCB$, PageChangeCB$) : 5

**rem controls**

NMPDButton:(Caption$, Flags%, Callback$, bitFile$, bitmapId%, bitmapMask%, Layout%) : 21

NMPDCheckBox:(Page%, Id%, Prompt$, Trailer$, BYREF State%, StateChangedCB$) : 22

NMPDChoice:(Page%, Id%, Prompt$, Trailer$, BYREF Choice%, Choices$, StateChangedCB$, Flags%) : 23

NMPDChoiceByArray:(Page%, Id%, Prompt$, Trailer$, BYREF Choice%, BYREF ChoiceArray&, StateChangedCB$, Flags%) : 24

NMPDDate:(Page%, Id%, Prompt$, Trailer$, BYREF Date&, MinV&, MaxV& ) : 25

NMPDEdit:(Page%, Id%, Prompt$, Trailer$, StringAddr&, EdwinWidth%, MaxLength%, Flags%) : 26

NMPDEditMulti:(Page%, Id%, Prompt$, BufferAddr&, EdwinWidth%, MaxLength%, LineCount%, Flags%) : 27

NMPDFloat:(Page%, Id%, Prompt$, Trailer$, BYREF Value, MinV, MaxV, DecPlaces%) : 28

NMPDLong:(Page%, Id%, Prompt$, Trailer$, BYREF Value&, MinV&, MaxV&) : 29

NMPDOptionInit:(Page%, Id%, Prompt$, Trailer$, OptCount%, BYREF Option%, StateChangedCB$) : 30

NMPDOption:(ParentId%, ItemId%, Caption$) : 31

NMPDText:(Page%, Id%, Prompt$, Body$, Flags%) : 32

NMPDTime:(Page%, Id%, Prompt$, Trailer$, BYREF Value&, Flags%, Min&, Max&) : 33

NMPDXInput:(Page%, Id%, Prompt$, Trailer$, StringAddr&, MaxLength%) : 34

NMPDLineButton:(Page%, Id%, Prompt$, Trailer$, Caption$, Callback$, bitFile$, bitmapId%, bitmapMask%, Flags%, Layout%) : 35

NMPDFile:(Page%, Id%, Prompt$, Flags%, Uid1&, Uid2&, Uid3&, StringAddr&) : 36

NMPDBitmapChoice:(Page%, Id%, Prompt$, Trailer$, BYREF Choice%, BYREF BitmapArrayId&, StateChangedCB$, Flags%) : 51

NMPDBitmapViewer:(Page%, Id%, Prompt$, Trailer$, WindowId%, File$, Index%, Width%, Height%) : 52

NMPDLongitudeEditor:(Page%, Id%, Prompt$, Trailer$, BYREF Degrees%, BYREF Minutes%, BYREF Seconds%, BYREF Direction%, Flags%) : 53

NMPDLatitudeEditor:(Page%, Id%, Prompt$, Trailer$, BYREF Degrees%, BYREF Minutes%, BYREF Seconds%, BYREF Direction%, Flags%) : 54

NMPDFontPreviewer:(Page%, Id%, Prompt$, Trailer$, BYREF FontUid&, PreviewText$) : 55

NMPDCharEditor:(Page%, Id%, Prompt$, Trailer$, BYREF Character%) : 56

NMPDRange:(Page%, Id%, Prompt$, Trailer$, BYREF Lower&, BYREF Upper&, Min&, Max&) : 57

NMPDGraySelector:(Page%, Id%, Prompt$, Trailer$, BYREF Red%, BYREF Green%, BYREF Blue%, Flags%) : 58

**rem Dynamic methods used in callback procedures**

NMPDdynSetCheckBox:(Id%, State%) : 71

NMPDdynGetCheckBox%:(Id%) : 72

NMPDdynSetChoiceItem:(Id%, Item%) : 73

NMPDdynGetChoiceItem%:(Id%) : 74

NMPDdynGetChoiceItemText$:(Id%) : 75

NMPDdynSetOption:(Id%, Option%) : 76

NMPDdynGetOption%:(Id%) : 77

NMPDdynSetDate:(Id%, Date&) : 78

NMPDdynGetDate&:(Id%) : 79

NMPDdynSetTime:(Id%, Time&) : 80

NMPDdynGetTime&:(Id%) : 81

NMPDdynSetFloat:(Id%, Value) : 82

NMPDdynGetFloat:(Id%) : 83

NMPDdynSetLong:(Id%, Value&) : 84

NMPDdynGetLong&:(Id%) : 85

NMPDdynSetEdit:(Id%, Value$) : 86 rem Only works on dEdit

NMPDdynGetEdit$:(Id%) : 87 rem Works on dXInput and dEdit

NMPDdynSetFont:(Id%, FontId&) : 88

NMPDdynGetFont&:(Id%) : 89

NMPDdynSetRange:(Id%, Upper&, Lower&) : 90

NMPDdynGetRange:(Id%, BYREF Upper&, BYREF Lower&) : 91

NMPDdynSetCharacter:(Id%, Char%) : 92

NMPDdynGetCharacter%:(Id%) : 93

NMPDdynSetLongitude:(Id%, Deg%, Min%, Sec%, Dir%) : 94

NMPDdynGetLongitude:(Id%, BYREF Deg%, BYREF Min%, BYREF Sec%, BYREF Dir%) : 95

NMPDdynSetLatitude:(Id%, Deg%, Min%, Sec%, Dir%) : 96

NMPDdynGetLatitude:(Id%, BYREF Deg%, BYREF Min%, BYREF Sec%, BYREF Dir%) : 97

NMPDdynSetGray:(Id%, Red%, Green%, Blue%, Flags%) : 98

NMPDdynGetGray:(Id%, BYREF Red%, BYREF Green%, BYREF Blue%) : 99

NMPDdynSetButton:(ButtonId%, State%) : 100

NMPDdynSetChoices:(Id%, ArrayId&) : 101

NMPDdynSetEditMulti:(Id%, BufferAddr&, MaxLength%) : 102

NMPDdynGetEditMulti:(Id%, BufferAddr&, MaxLength%) : 103

NMPDdynSetText:(Id%, Text$) : 104

NMPDdynSetLongRange:(Id%, Min&, Max&) : 105

NMPDdynSetChoicesByString:(Id%, Choices$) : 106

NMPDdynSetPrompt:(Id%, Prompt$) : 107

**rem Dynamic general methods used in callbacks**

NMPDdynSetFocus:(Id%) : 121

NMPDdynGetFocus%: : 122

NMPDdynSetControlState:(Id%, State%) : 123

NMPDdynGetControlState%:(Id%) : 124

NMPDdynGetControlType%:(Id%) : 125

NMPDdynTopLeft:(BYREF X%, BYREF Y%) : 126

NMPDdynWidth%:() : 127

NMPDdynHeight%:() : 128

NMPDdynButtonCorner:(Key%, Corner%, BYREF X%, BYREF Y%) : 129

**rem bitmap array methods**

NMPDBArrayCreate&:() : 201

NMPDBAdd:(Id&, WindowId%, File$, ImageIndex%) : 202

NMPDBInsert:(Id&, Position%, WindowId%, File$, ImageIndex%) : 203

NMPDBDelete:(Id&, Position%) : 204

NMPDBFree:(BYREF Id&) : 205

NMPDBGet:(Id&, Position%, WindowId%, PosX%, PosY%, Files$) : 206

NMPDBSize:(Id&, Position%, BYREF Width%, BYREF Height%) : 207

NMPDBCount%:(Id&) : 208

**rem string array methods**

NMPDArrayCreate&:() : 301

NMPDArrayAddItem:(Id&, Item$) : 302

NMPDArrayInsertItem:(Id&, Position%, StringToInsert$) : 303

NMPDArrayReplaceItem:(Id&, Index%, ChangeTo$) : 304

NMPDArrayDelete:(Id&, Index%) : 305

NMPDArrayDeleteItems:(Id&, StartIndex%, FinishIndex%) : 306

NMPDArrayDeleteAll:(Id&) : 307

NMPDArrayItemLength%:(Id&, Index%) : 308

NMPDArrayItemCount%:(Id&) : 309

NMPDArrayItemAt$:(Id&, Index%) : 310

NMPDArrayFree:(BYREF Id&) : 311

NMPDArraySort:(Id&, Flags%) : 312

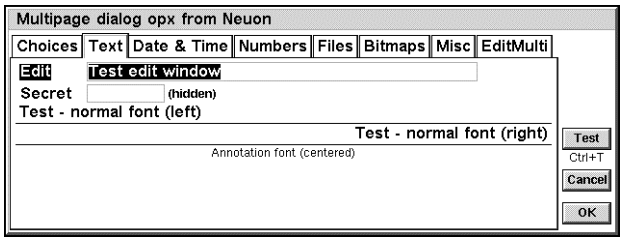
NMPDArrayFind%:(Id&, Flags%, SearchString$) : 313

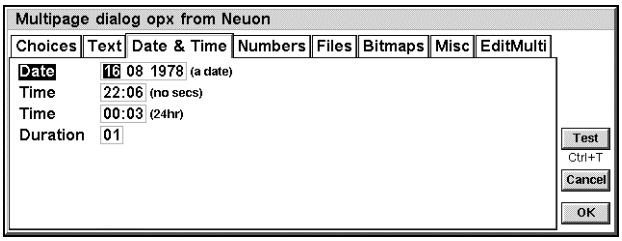
**rem resource methods**

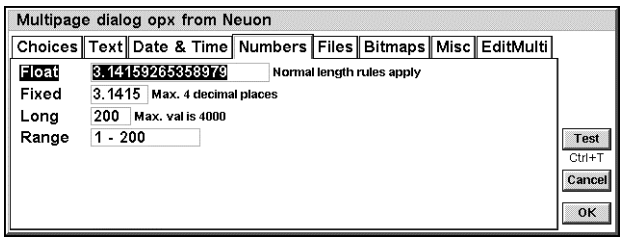
NMPDResourceLoad:() : 351

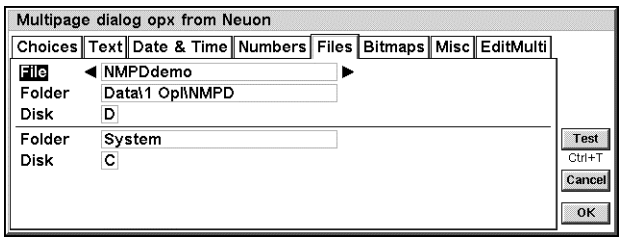
NMPDResourceUnload:() : 352

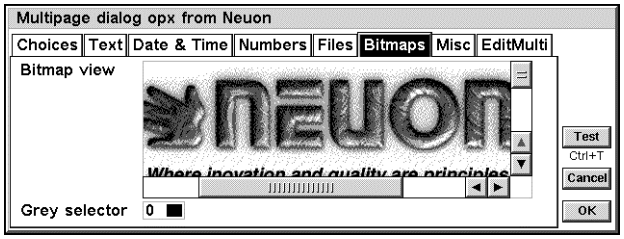
END DECLARE

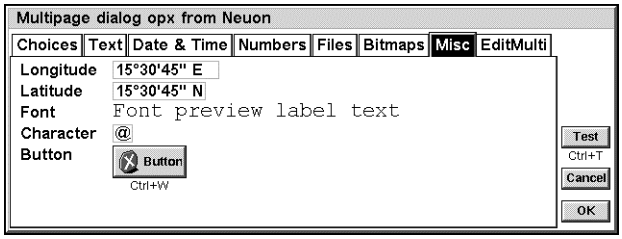

Multipage dialog opx from Neuon
Choices
Text
Date & Time
Numbers
Files
Bitmaps
Misc
EditMulti
Edit
Test edit window
Secret
(hidden)
Test - normal font (left)
Test - normal font (right)
Annotation font (centered)
Test
Ctrl+T
Cancel
OK

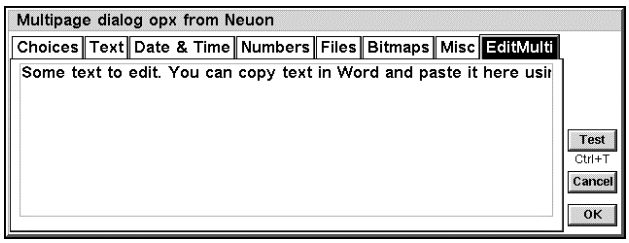

Multipage dialog opx from Neuon
Choices
Text
Date & Time
Numbers
Files
Bitmaps
Misc
EditMulti
Date
16 08 1978 (a date)
Time
22:06 (no secs)
Time
00:03 (24hr)
Duration
01
Test
Ctrl+T
Cancel
OK

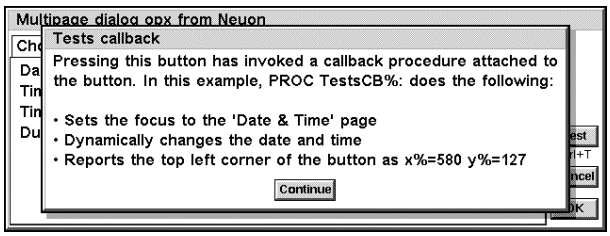

Multipage dialog opx from Neuon
Choices
Text
Date & Time
Numbers
Files
Bitmaps
Misc
EditMulti
Float
3.14159265358979
Normal length rules apply
Fixed
3.1415
Max. 4 decimal places
Long
200
Max. val is 4000
Range
1 - 200
Test
Ctrl+T
Cancel
OK


Multipage dialog opx from Neuon
Choices
Text
Date & Time
Numbers
Files
Bitmaps
Misc
EditMulti
File
NMPDdemo
Folder
Data\1 Opl\NMPD
Disk
D
Folder
System
Disk
C
Test
Ctrl+T
Cancel
OK


Multipage dialog opx from Neuon
Choices
Text
Date & Time
Numbers
Files
Bitmaps
Misc
EditMulti
Bitmap view
Where innovation and quality are principles
Grey selector
0
Test
Ctrl+T
Cancel
OK


Multipage dialog opx from Neuon
Choices
Text
Date & Time
Numbers
Files
Bitmaps
Misc
EditMulti
Longitude
15°30'45" E
Latitude
15°30'45" N
Font
Font preview label text
Character
@
Button
Button
Ctrl+W
Test
Ctrl+T
Cancel
OK


Multipage dialog opx from Neuon
Choices
Text
Date & Time
Numbers
Files
Bitmaps
Misc
EditMulti
Some text to edit. You can copy text in Word and paste it here usi
Test
Ctrl+T
Cancel
OK


Tests callback
Pressing this button has invoked a callback procedure attached to the button. In this example, PROC TestsCB%: does the following:
• Sets the focus to the 'Date & Time' page
• Dynamically changes the date and time
• Reports the top left corner of the button as x%=580 y%=127
Continue