DOpusARexxTute

The Phantom writer and -)

COLLABORATORS

	TITLE : DOpusARexxTute	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	The Phantom writer and -)	December 28, 2024	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1 DOpusARexxTute **1**

1.1 DOpus Magellan II ARexx Tutorial . . . 1
1.2 Magellan II ARexx Tutorial: Introduction . . . 3
1.3 Magellan II ARexx Tutorial: Requirements . . . 4
1.4 Magellan II ARexx Tutorial: Resources . . . 4
1.5 Magellan II ARexx Tutorial: Format of this tutorial . . . 5
1.6 The Opus Screen/Window . . . 6
1.7 Magellan II ARexx Tutorial: DOpus Front and Back . . . 7
1.8 Magellan II ARexx Tutorial: Dopus Screen . . . 7
1.9 Magellan II ARexx Tutorial: Dopus Query . . . 8
1.10 Magellan II ARexx Tutorial: Dopus Set . . . 9
1.11 Magellan II ARexx Tutorial: Dopus Getdesktop . . . 11
1.12 Magellan II ARexx Tutorial: Dopus Desktoppopup . . . 13
1.13 Version and Errors . . . 14
1.14 Magellan II ARexx Tutorial: DOpus Version . . . 15
1.15 Magellan II ARexx Tutorial: DOpus Error . . . 16
1.16 Getting input from the user . . . 17
1.17 Magellan II ARexx Tutorial: DOpus Request . . . 17
1.18 Magellan II ARexx Tutorial: DOpus Getstring . . . 18
1.19 Magellan II ARexx Tutorial: Lister Request . . . 19
1.20 Magellan II ARexx Tutorial: Lister Getstring . . . 19
1.21 Opening and closing listers . . . 19
1.22 Magellan II ARexx Tutorial: Lister New . . . 20
1.23 Magellan II ARexx Tutorial: Lister Close . . . 21
1.24 Magellan II ARexx Tutorial: Lister Iconify . . . 22
1.25 Magellan II ARexx Tutorial: Lister Read . . . 22
1.26 Magellan II ARexx Tutorial: Lister Copy . . . 23
1.27 Magellan II ARexx Tutorial: Lister Wait . . . 23
1.28 Providing some feedback . . . 24
1.29 Magellan II ARexx Tutorial: Dopus Read . . . 25

1.30 Magellan II ARexx Tutorial: Dopus Progress . . . 26
1.31 Magellan II ARexx Tutorial: Lister Progress . . . 27
1.32 Magellan II ARexx Tutorial: Lister NewProgress . . . 27
1.33 Magellan II ARexx Tutorial: Finding and Setting Lister Attributes . . . 28
1.34 Magellan II ARexx Tutorial: Lister Query . . . 29
1.35 Magellan II ARexx Tutorial: Lister Set . . . 31
1.36 Magellan II ARexx Tutorial: Lister Set Position . . . 32
1.37 Magellan II ARexx Tutorial: Lister Visible . . . 33
1.38 Magellan II ARexx Tutorial: The Phantom is... . . . 33
1.39 Magellan II ARexx Tutorial: Manipulating Lister Entries . . . 34
1.40 Magellan II ARexx Tutorial: Dopus Getfiletype . . . 34
1.41 Magellan II ARexx Tutorial: Lister Query Entry . . . 35
1.42 Magellan II ARexx Tutorial: Lister Select . . . 37
1.43 Magellan II ARexx Tutorial: Lister Remove . . . 37
1.44 Magellan II ARexx Tutorial: Lister Add . . . 38
1.45 Magellan II ARexx Tutorial: Lister Addstem . . . 39
1.46 Magellan II ARexx Tutorial: Command . . . 41
1.47 Magellan II ARexx Tutorial: FTP Commands . . . 42
1.48 Magellan II ARexx Tutorial: Opus v4 functions . . . 46
1.49 Magellan II ARexx Tutorial: Integration . . . 49
1.50 Magellan II ARexx Tutorial: Opus and AWeb II v2.x . . . 49
1.51 Magellan II ARexx Tutorial: Some ideas . . . 50
1.52 Magellan II ARexx Tutorial: Opening a lister with the path of your current shell . . . 50
1.53 Magellan II ARexx Tutorial: Changing the shell path to the same as the lister . . . 51
1.54 Magellan II ARexx Tutorial: Improved DOS-DOpus script (Example 1) . . . 52
1.55 Example 6: Changing the background every 30 seconds . . . 53
1.56 Magellan II ARexx Tutorial: Simple ARexx Module #1 . . . 54
1.57 Magellan II ARexx Tutorial: Simple ARexx Module #2 . . . 55
1.58 Magellan II ARexx Tutorial: Multi-Command ARexx Module . . . 59
1.59 Magellan II ARexx Tutorial: A Simple Custom Handler for a Lister . . . 61
1.60 Magellan II ARexx Tutorial: A Simple Custom Handler for an AppIcon . . . 64
1.61 Magellan II ARexx Tutorial: Improving the inbuilt commands . . . 68
1.62 Magellan II ARexx Tutorial: Cloning source listers . . . 70
1.63 Magellan II ARexx Tutorial: Finding duplicated files in two listers . . . 72
1.64 Magellan II ARexx Tutorial: Adding a bit of Win95 . . . 73
1.65 Magellan II ARexx Tutorial: Adding a directory tree function . . . 74
1.66 Magellan II ARexx Tutorial: An Opus v4 CopyWin replacement . . . 77
1.67 Magellan II ARexx Tutorial: An Opus v4 SwapWin replacement . . . 78
1.68 Magellan II ARexx Tutorial: TroubleShooting . . . 81

1.69 Magellan II ARexx Tutorial: TroubleShooting - The simple things . . . 81
1.70 Magellan II ARexx Tutorial: TroubleShooting - ARexx error codes . . . 82
1.71 Magellan II ARexx Tutorial: TroubleShooting - ARexx tracing . . . 84
1.72 Magellan II ARexx Tutorial: TroubleShooting - The OpusCLI . . . 85
1.73 Magellan II ARexx Tutorial: Credits . . . 86
1.74 Magellan II ARexx Tutorial: Dopus User Position . . . 87
1.75 Magellan II ARexx: Results from commands. . . . 87
1.76 Magellan II ARexx Tutorial: Error Codes . . . 87
1.77 Magellan II ARexx Tutorial: Author . . . 88
1.78 ArcDir.dopus5: Intro . . . 89
1.79 ArcDir.dopus5: Setup . . . 90
1.80 ArcDir.dopus5: Handler . . . 91
1.81 ArcDir.dopus5: Capturing an event . . . 92
1.82 ArcDir.dopus5: Event - Miscellaneous . . . 92
1.83 ArcDir.dopus5: Event - doubleclick . . . 93
1.84 ArcDir.dopus5: Event - drop . . . 94
1.85 ArcDir.dopus5: Event - dropfrom . . . 95
1.86 ArcDir.dopus5: Event - Copy . . . 96
1.87 ArcDir.dopus5: Event - View commands . . . 96
1.88 ArcDir.dopus5: Event - Unsupported . . . 97
1.89 ArcDir.dopus5: Event - Anything else . . . 97
1.90 ArcDir.dopus5: Cleaning Up . . . 98
1.91 ArcDir.dopus5: Parent/Root action . . . 98
1.92 ArcDir.dopus5: Path gadget action . . . 99
1.93 ArcDir.dopus5: Delete action . . . 101
1.94 ArcDir.dopus5: Making new directories . . . 103
1.95 ArcDir.dopus5: Create Directories . . . 104
1.96 ArcDir.dopus5: Listing the archive . . . 105
1.97 ArcDir.dopus5: Extracting from the archive . . . 107
1.98 ArcDir.dopus5: Adding to the archive . . . 110
1.99 ArcDir.dopus5: Viewing a file . . . 112
1.100 ArcDir.dopus5: Getting all the files . . . 114
1.101 ArcDir.dopus5: Patching filenames . . . 115
1.102 ArcDir.dopus5: Getting catalog string . . . 115
1.103 ArcDir.dopus5: Checking for valid handler . . . 116
1.104 ArcDir.dopus5: Syntax error . . . 116
1.105 ArcDir.dopus5: User halts script . . . 117
1.106 ArcDir.dopus5: Displaying errors . . . 117
1.107 ArcDir.dopus5: Displaying a requester . . . 118

Chapter 1

DOpusARexxTute

1.1 DOpus Magellan II ARexx Tutorial

```
       Welcome to the wonderful world of the Directory Opus Magellan II
                             ARexx interface

  Come aboard for a fascinating journey through the twisted mind of an
unqualified person who is trying to impart some knowledge upon other people.

                It'll make you laugh, it'll make you cry.


WARNING: Parts of this guide have been subjected to a 'dry' sense of humour,
         if you find one please return it to the owner.


 Introduction              What is this?
 Requirements              You want what!
 Resources                 Never enough...
 Format                    Nice hard drive ;^)
 Credits                   Uh oh, Martin's in trouble >:-|
 Author                    The character that wrote this tripe.

 Where the results end up          I want results!
 Where the error come from         It's all Opus' fault.

 The Opus Screen/Window
   dopus front / back      A view to a kill.
   dopus screen            You'll get square eyes.
   dopus query             Well, I didn't know that!
   dopus set               in concrete.
   dopus getdesktop        Watch it, that's oak!
   dopus checkdesktop      Quick, what's in the drawers?!
   dopus matchdesktop      A right pair they make.
   dopus desktoppopup      Yoohoo, I'm over here!

 Version and Errors
   dopus version           Next year's model.
   dopus error             A problem? Never!

 Getting information from the user
```

```
  dopus request           I get a choice?
  dopus getstring         I value your input.
  lister request          I can't make up my mind.
  lister getstring        Who asked you?

Opening and Closing Listers
  lister new              Come on, open up.
  lister close            Drink up, it's closing time.
  lister iconify          I feel really small.
  lister read             Every path a story.
  lister copy             We're all clones...
  lister wait             As if we don't do enough already.

Giving the user some feedback
  dopus read              It's learnt to read!
  dopus progress          We're moving!
  lister progress         We progress through this, to...
  lister newprogress      Nothing stands in the way of progress.

Finding and Setting Lister Attributes
  lister query            Hello sailor...
  lister set              A change is as good as a holiday...NOT!
  lister position         Position number 217...
  lister visible          Now you see me, now you don't.

File Manipulation and Information
  dopus getfiletype       Stereotyped again!
  lister query entry      No, you can't come in!
  lister select           I want that one!
  lister remove           You don't get rid of me that easily!
  lister add              1 and 1 is 3...err...2
  lister addstem          Nuffin' 'rong wi' a bit a graft guv'.

Calling the Internal Commands
  command                 "By your command..."

Using OpusFTP Functions

Missing Directory Opus 4 commands?
  CopyWin
  SwapWin

Troubleshooting
  The simple things      in life are very expensive.
  ARexx error codes      Most of it a trial, all of it in error.
  ARexx tracing          We know where you are.
  The Opus CLI           A commanding position.

Integrating Opus and other Programs
  Opus - AWeb-II v2      "Come in to my parlour", said the spider.

Some ideas for you to start with :)     Wouldn't want me to wear my brains out :)

ArcDir.dopus5

Example  1: Opening a lister to the path of your current shell
Example  2: Changing your shell path to the same as the lister
```

Example 3: An improved DOS-DOpus script (Example 1)
Example 4: Copying Source lister to Destination lister
Example 5: Swapping the Source and Destination listers
Example 6: Changing the background every 30 seconds
Example 7: Simple ARexx module #1
Example 8: Simple ARexx module #2
Example 9: Using a module to do other things
Example 10: A Simple Custom Handler for a Lister
Example 11: A Simple Custom Handler for an AppIcon
Example 12: Improving the internal commands
Example 13: Cloning Source listers
Example 14: Comparing files in two listers
Example 15: Adding a bit of Win95   BLAH!!
Example 16: Adding a simple directory tree function

Example: Finding All listers
Example: Finding the Active lister and the State
Example: Finding the current Source lister
Example: Finding the current Destination lister

1.2 Magellan II ARexx Tutorial: Introduction

The idea of this guide is to introduce users to some of the basic functions of Directory Opus' ARexx interface, hoping to allow the user expand upon and create scripts of their own.

This tutorial will not provide examples of every command available, I'd be here from now until Christmas trying to do that, rather it will attempt to show you a general cross-section of the commands available and the context in which these commands are used by using small demonstration scripts that can be executed from within this guide.

The DOpus ARexx command interface has been split into three base commands, with parameters and sub-commands specifying what action should be performed.

From the DOpus manual, the three base commands are:

- dopus   (for access to things not falling into the two categories below).
- lister   (manipulates lister information, attributes, entries, etc).
- command (allows you to call the internal commands of DOpus from ARexx).

The end of this guide will include some simple 'ARexx modules', and a 'custom handler' example to illustrate how these commands can all be used to extend the power of DOpus. Some of the more powerful ARexx commands will be described in this section, as they are uniquely suited to these functions.

All the examples in this tutorial can be found in the DOpus5:Help/TuteRexx directory. In all except a very few cases, they can be run from a shell. They can be used as a basis for your own script, or as a drop-in subroutine for your script.

See the Resources section for more information regarding sources of Directory Opus information and help.

1.3 Magellan II ARexx Tutorial: Requirements

This guide assumes the user has a basic knowledge of ARexx, (that is, ARexx is correctly installed and available to the system, you listening Ash? :) and Directory Opus Magellan II is installed, [as Workbench Replacement (WBR) or standalone].

Note: Most of the information in this guide will refer to listers in Name mode, and Opus as a Workbench Replacement.

It will also assume that the Opus ARexx port name is 'DOPUS.1'.

If you find that the examples in this guide don't work it could be because Opus isn't running, or you have closed the first Opus interface and are using another one, (ie. you've run the Opus program more than once, and closed the original).

Pressing this button will run an ARexx script that will check your system, prompting you if there is anything that will prevent the examples from running correctly.

References will be given back to DOpusM2_ARexx.guide and DOpus5.guide , both should be available in the directory DOpus5:Help/, for command parameters and format, if it isn't there then the links to them will not work.
They should have been installed there as part of the default Directory Opus installation procedure.

1.4 Magellan II ARexx Tutorial: Resources

Here I will just mention some of the other information resources for creating ARexx scripts for Directory Opus.

As always there is the unrivalled Aminet software archive. Scripts, patches, configurations, etc specific to Directory Opus can be found in the directory biz/dopus in the Aminet directory tree.
The best way in which to learn about the Opus ARexx interface is to download scripts and look at what they do and how they do it.

There is also the Directory Opus v5 Mailing List available to anyone who wants to be bombarded with lots of useless email :^)

You can subscribe to the list by sending an email to:

listserv@lss.com.au

with the following as the message text:

subscribe dopus5 <your email address>

You will recieve a welcome message detailing the commands available, and then the cra...eeerrr, the help should start rolling in ;^)

Updates, announcements, and other sundry items of interest can be found on

the GPSoftware WWW page, located at http://www.gpsoft.com.au.

Bug reports and requests for help can also be be sent to Dr Greg Perry at greg@gpsoft.com.au, you will need to send your registration number in the message if you expect a response. (But try the mailing list first, we're a much more forgiving bunch, heh heh heh |^)

For general ARexx programming information there is, of course, Aminet. Try the util/rexx directory for lots of examples.

Also available, is an excellent ARexx tutorial, written by Robin Evans, it is available from http://www.halcyon.com/robin/www/arexxguide/main.html.

1.5 Magellan II ARexx Tutorial: Format of this tutorial

This guide is designed so that the user starts at the beginning and then just steps forward through it. It is not designed for jumping from one topic to another, but the Contents are provided as a shortcut to a specific section.

The format of commands will not be extensively covered, as this tutorial is more concerned with teaching the basic commands and the context they're used in.

Each description of a command will provide a link to the main Opus ARexx reference, the DOpusM2_ARexx.guide. The DOpusM2_ARexx.guide will provide you with the command parameters and format required for that particular command.

A typical description will look like this:

The first commands we'll look at are the dopus front and dopus back .

They can be used within an ARexx script like so:

```
/* DOpus front/back test */
address 'DOPUS.1'             /* Open DOpus' ARexx port */
dopus front                   /* Bring the DOpus display to the front */
address command wait 2        /* Wait for 2 seconds */
dopus back                    /* Send the DOpus display to the back */
exit                          /* Exit the script */
```

You will notice that dopus front and dopus back are buttons, clicking on them will load the DOpusM2_ARexx.guide reference, open at the correct page for that command.

The ARexx comment for the example script is also a button, clicking on that will run that particular example script.

With some commands you will also see a button labelled ArcDir , hitting this button will take you to a section of Edmund's ArcDir script to show you

where this command is used as a practical example.

All of the scripts are generally written in lower case for clarity, excessive use of upper case characters tends to make texts hard to read because all the letters are the same height and the eye tends to start skipping over them.

There are no TABs used in this guide, TABs suck :) If there are TABs, who told you, you could put them in? >:-|

Indenting is set to two spaces to get as much as possible on screen, and not lose the readability it provides.

What the script looks like in this guide will be what it looks like in reality, with a few exceptions, for example, the commands 'Lister Query' and 'Lister Set'. So if you prefer you can cut and paste from within this guide using PowerSnap or something similar.

Now that I've inflicted upon you my sense of style, on with the show...

1.6 The Opus Screen/Window

This section will look at commands that have to do with the Opus Desktop, whether it be a screen or window.

There are only nine commands that either influence or obtain information from Opus regarding the Desktop. They are:-

```
dopus back
dopus front
```

These two commands just move the Opus Desktop to the front or back, useful for showing a user that there is something to see or that requires their attention on the Desktop.

```
dopus screen
```

Returns information regarding Opus' screen name, screen size/depth and default lister size.

```
dopus query
```

Let's you get information on the current backgrounds, sounds, palette, pens and fonts. It is the complement to the dopus set command.

```
dopus set
```

This command is used extensively for the Opus themes system. If you have some themes installed you will find plenty of additional examples of the usage of this command in the D5Themes: directory. Every (name).theme file is an ARexx script containing many dopus set commands.

```
dopus getdesktop
dopus checkdesktop
dopus matchdesktop
```

These three commands allow you to manipulate the Opus Desktop to a limited degree. It is not the same as the dopus set command, they do not affect the look of the Desktop, rather they can help you to add programs to the Desktop.

dopus desktopopup

This command causes the Desktop popup menu to appear under the mouse, allowing the user to choose from one of the choices.

1.7 Magellan II ARexx Tutorial: DOpus Front and Back

The dopus front and dopus back are self-explanatory, all they do is move the Opus screen or window to the front or back of your display.

They can be used within an ARexx script like so:

```
/* DOpusFrontBack.dopus5 */
address 'DOPUS.1'             /* Address the Opus ARexx port */
dopus front                   /* Bring the DOpus display to the front */
address command wait 2        /* Wait for 2 seconds */
dopus back                    /* Send the DOpus display to the back */
exit                          /* Exit the script */
```

If you were running this tutorial as a window on an Opus screen which is set to Backdrop, (Opus menu), then all you would have noticed is that the window would have become deactivated.

These would have to be the simplest commands in the Opus ARexx interface.

1.8 Magellan II ARexx Tutorial: Dopus Screen

The dopus screen command allows you to find out the dimensions of the Opus display, this can be useful for positioning of listers, viewers, etc, by using a simple algorythm to provide a position that will fit on screen.
It also returns the current Opus screen name, and will set RC to 5 if Opus happens to be iconified at the time.
The default size of your listers is also returned.

Example:

```
/* DopusScreen.dopus5 */
options results
lf = '0a'x
address 'DOPUS.1'
dopus front
dopus screen
if rc = 5 then do
  address command 'RequestChoice "Opus is iconified" "OK"'
  exit
```

```
  end
dimensions = result
text = 'The Opus screen name is: 'word(dimensions,1)
dopus request '"'text'" OK'
text = 'The Opus screen is 'word(dimensions,2)' pixels wide'||lf||,
       'and 'word(dimensions,3)' high.'
dopus request '"'text'" OK'
text = 'The titlebar is 'word(dimensions,5)' pixels high, and'||lf||,
       'it is a 'word(dimensions,4)' bit screen.'
dopus request '"'text'" OK'
x = trunc(word(dimensions,2) / 2) - trunc(word(dimensions,6) / 2)
y = trunc((word(dimensions,3) - word(dimensions,5)) / 2) - trunc(word(dimensions ↩
   ,7) / 2)
position = x''y''word(dimensions,6)''word(dimensions,7)
lister new position
handle = result
text = 'This lister is in the screen centre,',
        ||lf||'taking the titlebar height into account',
        ||lf||'and is the default size.'
lister request handle '"'text'" OK'
lister close handle
dopus back
exit
```

1.9 Magellan II ARexx Tutorial: Dopus Query

New for Magellan II

The dopus query command complements the dopus set command, allows you to read various Opus settings such as; backgrounds, fonts, palette, pens, and sounds.

For instance, while running your script you might want to change the background of the lister to a picture displaying your name, you can find out what the current background is, change it with dopus set then restore it when the script exits.

Example:

```
/* DopusQuery.dopus5 */
options results
lf = '0a'x
address 'DOPUS.1'
dopus front
dopus query font screen
text = 'Current Opus screen font:'||lf||word(result,1)||lf||'Size: 'word(result,2)
dopus request '"'text'" OK'
dopus query background lister
text = 'Current icon lister background: '||lf||word(result,1)||lf||'Options: ' ↩
   subword(result,2)
dopus request '"'text'" OK'
dopus query sound Startup
text = 'Sound you hear when Opus starts:'||lf||word(result,1)||lf||'Volume: 'word( ↩
   result,2)
dopus request '"'text'" OK'
```

```
dopus query pens gauge
text = 'Current fuel gauge colours when:'||lf||'Full: 'word(result,2)||lf||'Not  ↩
    full: 'word(result,1)
dopus request '"'text'" OK'
dopus back
exit
```

1.10 Magellan II ARexx Tutorial: Dopus Set

```
   New for Magellan II

  To complement the  dopus query  command there is the  dopus set
command.  This command primarily allows you to change aspects of the
Opus GUI, that is, the Desktop and listers.

  It allows you to customise your interface, and as such is used extensively
by the Opus Themes system.  If you have any themes installed, then the
D5Themes: directory will contain a number of files labelled  ????.theme .
  Each one of these files is actually an ARexx script that contains a number
of  dopus set  commands that change the GUI in some way.
  For example; fonts, palette, pens, backgrounds, sounds.

/* DopusBackground.dopus5 */
options results
address 'DOPUS.1'
address command 'Copy TuteRexx/Extras/2.iff RAM:'
dopus front
dopus set background 'RAM:2.iff' desktop tile custom        /* Set the new  ↩
   background */
dopus refresh background custom                             /* Refresh the display ↩
    */
address command wait 10                                     /* Wait 10 seconds */
dopus refresh background                                    /* Refresh the display ↩
    */
dopus back
address command 'Delete RAM:2.iff QUIET FORCE'
exit

/* DOpusSound.dopus5 */
options results
address 'DOPUS.1'
dopus front
address command 'Copy TuteRexx/Warning RAM: QUIET'
dopus query sound "'Open Lister'"                   /* What's the old sound? */
oldsound = result                                   /* Save it */
dopus set sound "'Open Lister'" "'RAM:warning'" 64  /* Set the new sound */
lister new                                          /* Open a lister */
handle = result                                     /* Store it's handle */
address command wait 3                              /* Wait 3 seconds */
lister close handle                                 /* Close the lister */
dopus set sound "'Open Lister'" oldsound 64         /* Restore the old sound */
call delete('RAM:Warning')
exit
```

Below is the theme file for Trevor Morris' excellent Aliens theme, as you see, it consists of mostly dopus set commands.

```
/* D5THEME

   Alien.theme

   Directory Opus 5.7 Theme File
*/

parse arg dopus_port apply_flags
if dopus_port='' then
        dopus_port='DOPUS.1'
address value dopus_port

if apply_flags='' then
        apply_flags='PFBS'
else
        apply_flags=upper(apply_flags)

options results
options failat 21

/* Set background pictures */
if index( apply_flags , "B") ~= 0 then do
        dopus  set  background on
        dopus  set  background "'D5THEMES:Alien/Screens/Alien1.iff'" desktop tile  ↩
           precision icon border off
        dopus  set  background "'D5THEMES:Alien/Screens/Marble.iff'" lister tile  ↩
           precision image border off
        dopus  set  background "'D5THEMES:Alien/Screens/WhiteMarble.iff'" req tile ↩
            precision exact border off
end

/* Set sound events */
if index( apply_flags , "S") ~= 0 then do
        dopus  set  sound "'Bad disk inserted'" "'D5THEMES:Alien/Sounds/ ↩
           Alien_Default.snd'" 64
        dopus  set  sound "'Close buttons'" "'D5THEMES:Alien/Sounds/Alien_Open.snd ↩
           '" 64
        dopus  set  sound "'Close group'" "'D5THEMES:Alien/Sounds/Alien_Open.snd'" ↩
            64
        dopus  set  sound "'Close lister'" "'D5THEMES:Alien/Sounds/Alien_Open.snd ↩
           '" 64
        dopus  set  sound "'Disk inserted'" "'D5THEMES:Alien/Sounds/Alien_Twinkle. ↩
           snd'" 64
        dopus  set  sound "'Disk removed'" "'D5THEMES:Alien/Sounds/Alien_Drama.snd ↩
           '" 64
        dopus  set  sound "'FTP close connection'" "'D5THEMES:Alien/Sounds/ ↩
           Alien_Beep.snd'" 64
        dopus  set  sound "'FTP connect fail'" "'D5THEMES:Alien/Sounds/ ↩
           Alien_Suspence.snd'" 64
        dopus  set  sound "'FTP connect success'" "'D5THEMES:Alien/Sounds/ ↩
           Alien_Ring.snd'" 64
        dopus  set  sound "'FTP copy fail'" "'D5THEMES:Alien/Sounds/ ↩
           Alien_Exclamation.snd'" 64
```

```
        dopus  set  sound "'FTP copy success'" "'D5THEMES:Alien/Sounds/ ↩
           Alien_Elevator.snd'" 64
        dopus  set  sound "'FTP error'" "'D5THEMES:Alien/Sounds/Alien_Error.snd'"  ↩
           64
        dopus  set  sound "'Hide'" "'D5THEMES:Alien/Sounds/Alien_Hiss.snd'" 64
        dopus  set  sound "'Open buttons'" "'D5THEMES:Alien/Sounds/Alien_Restore. ↩
           snd'" 64
        dopus  set  sound "'Open group'" "'D5THEMES:Alien/Sounds/Alien_Restore.snd ↩
           '" 64
        dopus  set  sound "'Open lister'" "'D5THEMES:Alien/Sounds/Alien_Restore. ↩
           snd'" 64
        dopus  set  sound "'Reveal'" "'D5THEMES:Alien/Sounds/Alien_Warning.snd'"  ↩
           64
        dopus  set  sound "'Shutdown'" "'D5THEMES:Alien/Sounds/Alien_ShutDown.snd ↩
           '" 64
        dopus  set  sound "'Startup'" "'D5THEMES:Alien/Sounds/Alien_Startup.snd'"  ↩
           64
end

/* Set fonts */
if index( apply_flags , "F") ~= 0 then do
        dopus set font screen "'XHelvetica.font'" 11
        dopus set font listers "'XEN.font'" 8
        dopus set font iconsd "'XEN.font'" 8
        dopus set font iconsw "'RSansSerif.font'" 8
end

/* Set colour settings*/
if index( apply_flags , "P") ~= 0 then do
        dopus set pens icons 8 1 3 1 12 2
        dopus set pens files 1 0
        dopus set pens dirs 10 0
        dopus set pens selfiles 1 8
        dopus set pens seldirs 1 9
        dopus set pens devices 10 0
        dopus set pens assigns 1 0
        dopus set pens source 1 11
        dopus set pens dest 1 12
        dopus set pens gauge 11 13
        dopus set pens user 5
        dopus set palette 0x969696 0x000000 0xFFFFFF 0x3C65A2 0x7000C0 0x804000 0 ↩
           x07000C 0x01058C 0xB4E494 0xC8FC00 0x004C94 0xFCFCD4 0xD8D8D8 0x183454  ↩
           0xE8E8E8 0x505050
end

/* Refresh Opus */
dopus refresh all
```

1.11 Magellan II ARexx Tutorial: Dopus Getdesktop

New for Magellan II

Supposing in your ARexx script you wanted to add a file/dir to the Desktop:

Where do you copy the file to?

Do you copy or move it?
How do you tell Opus to add it to the Desktop display?

Supposing you wanted to Drag'n'Drop a file from the Desktop to a custom handler:

How can you be sure the source path is the Opus Desktop directory?

This is where these three commands, dopus getdesktop , dopus checkdesktop and dopus matchdesktop are useful.

When you copy or move a file to the Desktop, they are copied or moved to this directory. You can check this out for yourself very easily, open two listers, one with a path of S:, the other with DOpus5:Desktop. Now copy a file from S: to DOpus5:Desktop, after a few seconds you will see the file appear on the Desktop, usually with whatever default icon for that type of file.
If you then use the icons popup menu to delete the file, and rescan the DOpus5:Desktop lister, you will see that the file has gone.

So by copying things to that directory you can add them to the Desktop, but how do you find out the directory that the Desktop uses?

dopus getdesktop returns the path of the Opus Desktop directory as defined in the Environment - Desktop settings, usually the default is DOpus5:Desktop.

NOTE: The path returned will be the absolute path, for example, HD0:Opus5/Desktop.

The DOPUSRC variable will contain a value that will indicate what the default action is for files dropped onto the Desktop. See the example for the values available.

Example:

```
/* DopusGetdesktop.dopus5 */
options results
address 'DOPUS.1'
dopus front
dopus getdesktop
text = 'Current Desktop path is: 'result
dopus request '"'text'" OK'
text = 'Popup disabled'                        /* dopusrc = 0 */
if dopusrc = 1 then text = 'No default action'
if dopusrc = 2 then text = 'Create left-out'
if dopusrc = 3 then text = 'Move to Desktop'
if dopusrc = 4 then text = 'Copy to Desktop'
dopus request '"'text'" OK'
dopus back
exit
```

If you did the experiment above you noticed that Opus automatically added the program to the Desktop after a few seconds. This is because Opus uses file notification on the Desktop directory, when a file gets copied there, Opus is notified, it rescans the directory and adds any not already on the Desktop.

However if you copy a directory to the Desktop directory, Opus won't add it to the Desktop because it doesn't recieve notification, this is where the dopus checkdesktop command is useful.
You give it a path to check, if it matches the one in the environment settings, it forces Opus to go and check the Desktop directory for any new items to add to the Desktop.

Example:

```
/* DopusCheckdesktop.dopus5 */
options results
address 'DOPUS.1'
dopus getdesktop
deskpath = result
address command 'MakeDir 'deskpath'T'
dopus checkdesktop deskpath
text = 'There should be an icon for ''T'' on the Desktop'
dopus request '"'text'" OK'
address command 'Delete 'deskpath'T FORCE QUIET'
dopus checkdesktop deskpath
exit
```

Supposing you had a custom handler for a lister and you dragged a file from the Desktop to your lister, Opus would give you the full file and path, for example, HD0:Desktop/some.file

After seperating the path from the file, how could you check that the path given was the Desktop path that Opus uses?
This is what the dopus matchdesktop command is for, you can give it a path and it will check it against the path specified for the Desktop in the environment settings.

Example:

```
/* DopusMatchdesktop.dopus5 */
options results
address 'DOPUS.1'
dopus front
dopus matchdesktop 'SYS:Prefs'
if result = 1 then
  text = 'Your Desktop path is SYS:Prefs, very strange :-/'
else
  text = 'Your Desktop path isn''t SYS:Prefs, good :)'
dopus request '"'text'" OK'
dopus back
exit
```

1.12 Magellan II ARexx Tutorial: Dopus Desktoppopup

New for Magellan II

Supposing in your nice, new script you want to give the user a choice when they drop a file on the Desktop of either; creating a left-out or either copying or moving to the Desktop.

The dopus desktoppopup command will allow you to popup the Desktop menu, letting you choose any one of these, the value of the choice will be returned in RC .

The options can be limited by adding a flag to the command, so you could, for example, limit the options to just 'Create left-out'. The value of the flags can be added together to disable more than one options.

Flag	Option	Result Code
2	Create Left-out	1
4	Copy to Desktop	2
8	Move to Desktop	3

So to disable the Copy and Move to Desktop options, you would specify a flag of 12.

Example:

```
/* DopusDesktoppopup.dopus5 */
options results
address 'DOPUS.1'
dopus front
text = 'The mouse will need to be over the'||lf||,
       'Desktop, not a lister or button bank.'
dopus request '"'text'" OK'
address command wait 1
dopus desktoppopup 4
text = 'Nothing chosen'
if rc = 1 then text = 'Create Left-out'
if rc = 3 the ntext = 'Move to Desktop'
dopus request '"'text'" OK'
dopus back
exit
```

1.13 Version and Errors

This section deals with just finding the version information for Opus and how to get more information from an Opus ARexx error code.

The two commands that deal with this are:-

dopus version

Every Opus has a version and revision number, as per the normal Style Guide recommendations. This is not the number that's normally associated with the release, for example: Directory Opus Magellan had a release of 5.661, (final release after patches were applied), the actual version/revision was 5.1500.
Generally with every release there are either more or changes to existing ARexx commands available. Obtaining the version/revision information helps you in tailoring your scripts to a particular release of Opus.

dopus error

Generally when an ARexx script terminates, the variable RC , (Result Code), will get set to a value. A value of 0 means that there was no

problems and the script terminated normally, anything other than 0 means there was a problem. Obviously a number isn't going to impart much information, that's where the dopus error command is useful, it will provide a short message telling you what's wrong.

1.14 Magellan II ARexx Tutorial: DOpus Version

ArcDir

The dopus version command can be used to check that your script will only run on specific versions of Opus.

More ARexx functions have been added since Opus v5 was first created, if you create a script that uses commands that are in Opus Magellan but not in Opus v5.5, you can use dopus version to ensure that the script will only run on Opus Magellan or later.

The result is returned as two numbers representing the version and the revision, separated by a space.

Example:

```
/* DOpusVersion.dopus5 */
options results
address 'DOPUS.1'
dopus version                              /* Ask for version information */
text = "Your DOpus version is: "result     /* Format result into string */
dopus request '"'text'" OK'                /* Output to a requester */
dopus version
newopus = result ~= 'RESULT' & translate(result,'.',' ') ~< 5.1215
if newopus then
  text = 'You can use the ''lister request'' command.'
else
  text = 'You can''t use the ''lister request'' command.'
dopus request '"'text'" OK'
exit
```

In the example above, we reduced the version information to a boolean value, (0 or 1), by the line:

```
newopus = result ~= 'RESULT' & translate(result,'.',' ') ~< 5.1215
```

We can break it down to make it a bit more understandable this way:

```
result ~= 'RESULT'        <- This will give true or false, (1 or 0).
```

So, if the command worked, (dopus version), then RESULT won't equal 'RESULT', so this is true, (1).

```
translate(result,'.',' ') <- Change v rrrr to v.rrrr, (eg. 5 1215 -> 5.1215).

~< 5.1215                 <- Then compare the result to 5.1215.
```

If the version/revision information was not less than, (that is, greater than, or equal to), 5.1215 then this part will also be true, (1).

The final part is just the boolean addition of the two results of these two seperate expressions. If both are true, then you have the following:

```
newopus = 1 & 1
```

If you had an early version of Opus then:

```
newopus = 1 & 0
```

It's just an easier way of making comparisons later in the script, you won't need to do:

```
if versioninfo = '5 1215' then....
```

or similar.

1.15 Magellan II ARexx Tutorial: DOpus Error

The next command, dopus error , is used to provide more meaningful error messages, rather than just a code like 1, 5, 10, etc.

If you have created a script that keeps failing because you have provided an incorrect handle to a lister query command, all that is reported to you is the error code: 10.

You could have a short script in your DOpus5:ARexx/ directory like the following:

```
/* DOpusError.dopus5 */
address 'DOPUS.1'
dopus getstring '"Please enter error code:" 2 "" OK|Cancel' /* Ask for code */
if dopusrc = 0 then exit                          /* If Cancel then exit */
ec = result                                       /* Save error code entered */
dopus error ec               /* Pass error code to DOpus for interpretation */
text = "Error code "ec": "result             /* Format result into a string */
dopus request '"'text'" OK'              /* Display error text in requester */
exit
```

Then when you get an error code running a Opus script, you could execute the above script:

```
'rx DOpus5:ARexx/DOpusError.dopus5'
```

It would prompt you for the error code, then tell you what it meant.

The command could also be used to inform users of your scripts that there was a problem, this could then be refered back to you, providing you with important fault finding information.

NOTE: This is only true for Opus ARexx error codes, if you give it a normal ARexx error code then you might end up with the completely wrong answer.

You can see a list of valid Opus ARexx error codes @" here " link gen_errors}.

1.16 Getting input from the user

Opus provides basically two ways of getting direct input from the user, each has two variations:

dopus request

This presents the user with a window with some text and a choice of one or more buttons to select. These are similar to the normal System requesters. Requesters will appear in the center of the Opus screen.

dopus getstring

This displays a string gadget in which the user can type some text and also have a choice of one or more buttons. The string gadget will appear in the center of the Opus screen.

The two commands below provide the same functions as the two above except that the requester/string gadget will appear centered over the nominated lister.
The two main advantages of this is that:

- it provides an immediate indication of which lister the input is going to be used for.
- you don't need to keep moving the mouse to the center of the screen if you are using the lister that's running the script.

The disadvantage is, of course, you need a lister open :)

lister request
lister getstring

1.17 Magellan II ARexx Tutorial: DOpus Request

ArcDir

The dopus request . This command is used to provide interaction with the user, similiar to the normal System requesters like 'Please insert volume xxxx', where you can click on one of a number of buttons, eg. OK or Cancel.

The dopus request command will open it's window in the centre of the Opus screen, as will the following command dopus getstring .

The value of the button you chose will be returned in the ARexx variable RC .

An example:

```
/* DOpusRequest.dopus5 */
```

```
options results
lf = '0a'x
address 'DOPUS.1'
dopus request '"Please select a button below:" Replace|Skip|Rename|Abort'
if rc = 1 then button = 'Replace'
if rc = 2 then button = 'Skip'
if rc = 3 then button = Rename'
if rc = 0 then button = 'Abort'
text = 'You chose button: 'button||lf||'The value of the button you chose was: 'rc
dopus request '"'text'" OK'
exit
```

If you chose the button 'Abort' in the example, you will note that the value returned in the ARexx variable RC was 0. This is because the last button, (right-most), of the requester is designated the Cancel button.
Values returned in the above example, according to button order, will then be 1, 2, 3, and 0.

1.18 Magellan II ARexx Tutorial: DOpus Getstring

ArcDir

An extension to 'dopus request', is the dopus getstring command, which allows the user to input a string instead of just choosing a button.

The following example asks you for your name, only allows you to input 15 characters maximum, and displays a default name of 'Jon Potter', which you can use backspace to erase and then input your own.

If you then click on the 'Okay' button, any text you typed in will be in the standard ARexx variable RESULT .
The value of the button you used will be returned in the special Opus variable DOPUSRC .

```
/* DOpusGetstring.dopus5 */
options results
lf = '0a'x
address 'DOPUS.1'
dopus getstring '"Please enter your name" 15 "Jon Potter" Okay|Cancel'
if dopusrc = -1 then what = 'pressed Enter'
if dopusrc = 1 then what = 'chose Okay'
if dopusrc = 0 then what = 'chose Cancel'
text = 'You 'what||lf||'The value of DOPUSRC was: 'dopusrc
dopus request '"'text'" OK'
text = "Your name is: "result
dopus request '"'text'" OK'
exit
```

If you clicked on the 'Cancel' button, the RESULT variable will be cleared of any string and DOPUSRC will contain the value 0.

If you clicked the 'Okay' button then RESULT will contain any text, (or none), that was in the string gadget, and DOPUSRC will have the value of 1.

If you hit the Return/Enter key to enter your name, then RESULT will

contain any text typed in, and DOPUSRC variable will be -1.

1.19 Magellan II ARexx Tutorial: Lister Request

ArcDir

The lister request command provides the same capability as the dopus request command, the difference being that the requester will open over the specified lister.

Example:

```
/* ListerRequest.dopus5 */
options results
address 'DOPUS.1'
lister new "SYS:"
handle = result
lister wait handle
lister request handle '"I am positioned over a lister" OK'
lister close handle
exit
```

1.20 Magellan II ARexx Tutorial: Lister Getstring

The lister getstring command is identical in function to the dopus getstring command, the only difference being in that the string requester will open over the specified lister.

Example:

```
/* ListerGetstring.dopus5 */
options results
address 'DOPUS.1'
lister new "SYS:"
handle = result
lister wait handle
lister getstring handle '"How old are you?" 2 "" OK|Cancel'
lister close handle
exit
```

Any button selected from the requester will have it's value returned in the DOPUSRC result variable, please refer to dopus getstring to see what possible values it can hold.

1.21 Opening and closing listers

Listers, the interface through which you manipulate files in Opus v5, are the heart of the system, so it makes sense that we should know how to open and close the things :)

Whenever you successfully open a lister, a handle for that particular

lister will be returned in the RESULT variable. This handle will be required if you need to do any further operations on that lister, for example, close it.

lister new

This is the only ARexx command that can open a lister, and it will let you do it in any mode, position, path, with any toolbar, iconified, invisible and standing on your head, (see the all new dopus user position command).

lister close

This command can close a single lister or it can close them all.

lister iconify

If you want to make a lot of space available on screen fast without closing any listers, lister iconify is the command for you.

lister read

This lets you read another path into the lister.

lister copy

The lister copy command lets you quickly copy the contents of one lister to another.

lister wait

Some lister operations can take a while, for example, reading in a directory with hundreds of files. This command will force the script to wait until the lister is again ready to accept commands, ensuring that none are lost.

1.22 Magellan II ARexx Tutorial: Lister New

ArcDir

The first lister command we'll look at is lister new . This command let's you open a new lister, allowing you to specify it's position, size, initial path, toolbar, status, mode, etc.

The default position for a lister to open is under the mouse, and if no path is specified then the lister will be empty.

An important thing to note is that any PATH you specify must occur at the end of the command line.

In the following lister new examples, the listers will close after 5 seconds.

Examples:

```
/* DOpus lister new */
```

```
options results
address 'DOPUS.1'
 lister new mode name           /* open name mode lister, no initial path */
 lister new mode icon SYS:    /* open icon mode lister, with path of SYS: */
 lister new mode icon action SYS:  /* open in iconaction mode, with path of SYS:  ↩
    */
 lister new mode showall RAM:   /* open icon mode lister showing all files */
 lister new 1/1/200/300         /* open a lister in the top left corner */
 lister new toolbar 'TuteRexx/ListerNew.tb'
 lister new iconify RAM:      /* opens a lister, iconified */
exit
```

Click on the buttons above to see the appropriate lister open.

Any lister you open, will have a handle. This is a number that points to that specific lister out of however many you have open. You will need the handle for any further operations you want to perform on that lister.

This handle is returned in RESULT when you open a lister in ARexx.

Example:

```
/* ListerHandle.dopus5 */
options results
address 'DOPUS.1'
dopus front
lister new
text = "Lister handle is: "result
dopus request '"'text'" OK'
lister close handle
dopus back
exit
```

1.23 Magellan II ARexx Tutorial: Lister Close

The opposite of lister new is the lister close command, which allows you to close a single lister, or all listers.

To close a single lister, you need to know it's handle.

An example:

```
/* ListerClose.dopus5 */
options results
address 'DOPUS.1'
dopus front
lister new               /* Open a new lister */
handle = result          /* Store it's handle */
address command wait 2   /* Wait 2 seconds    */
lister close handle      /* Close the lister  */
exit
```

Or, alternatively, you can close all listers by substituting the word ALL instead of a handle.

```
/* ListerClose2.dopus5 */
options results
address 'DOPUS.1'
dopus front
lister new
"lister new 50/50/200/200"
"lister new 100/100/200/200"
"lister new 200/200/200/200"
dopus request '"Now to close them all" OK'
lister close all
dopus back
exit
```

1.24 Magellan II ARexx Tutorial: Lister Iconify

As it suggests, the lister iconify command allows you to 'iconify', reduce it from a window to an icon, a lister or all listers. This is useful for making other things more visible on the Desktop without actually closing the lister.

Example:

```
/* ListerIconify.dopus5 */
options results
lf = '0a'x
address 'DOPUS.1'
dopus front
lister new mode icon 'SYS:'
handle = result
text = 'When you close this requester'||lf||,
       'the lister will be iconified.'
dopus request '"'text'" OK'
lister iconify handle on                    /* Iconify the lister */
text = 'The lister is now iconified.'||lf||,
       'It will be de-iconified when'||lf||,
       'you close this requester.'
dopus request '"text'" OK'
lister iconify handle off                   /* Uniconify */
address command wait 2
lister close handle
dopus back
exit
```

All listers can be iconified by replacing the handle with the word ALL.

1.25 Magellan II ARexx Tutorial: Lister Read

ArcDir

The lister read command causes the lister to read a different path into the display. When you read a new path into the lister, the old path will be returned in RESULT .

Example:

```
/* ListerRead.dopus5 */
options results
address 'DOPUS.1'
dopus front
lister new mode name "RAM:"
handle = result
lister wait handle
lister request handle '"We will now read ''SYS:'' into this lister" OK'
lister read handle "SYS:"             /* Read the new path */
oldpath = result                      /* Save the old path */
text = 'The old path was: 'oldpath
lister request handle '"'text'" OK'
address command wait 2
lister close handle
dopus back
exit
```

1.26 Magellan II ARexx Tutorial: Lister Copy

The lister copy command lets you quickly copy the display contents, (not the actual files/directories), from one lister to another.

Example:

```
/* ListerCopy.dopus5 */
options results
lf = 0a'x
address 'DOPUS.1'
dopus front
"lister new 1/1/100/100 mode name 'SYS:'"
handle1 = result
"lister new 100/100/100/100 mode name 'RAM:'"
handle2 = result
lister wait handle2
dopus request '"The contents will be copied from one to the other'||lf||,
              'when you close this requester." OK'
lister copy handle1 handle2             /* Copy the contents */
address command wait 4
lister close handle1
lister close handle2
dopus back
exit
```

This command can be used to provide a 'Window Copy' function like in the old Directory Opus 4, in fact someone has done it for you, look here .

1.27 Magellan II ARexx Tutorial: Lister Wait

```
 ArcDir

  The  lister wait  command instructs DOpus to wait until the
lister is idle before continuing with the ARexx script.

  This is sometimes necessary because some of the lister operations can take
a short while to complete and any commands sent to the lister while it isn't
idle will be lost.

  The command will wait for two seconds if the lister is idle, if you don't
want this delay, then you can add the QUICK keyword which will cause the
command to return immediately if the lister is idle.

  Example:

/* ListerWait.dopus5 */
lf = '0a'x
options results
address 'DOPUS.1'
dopus front
lister new "DOpus5:Images/"
handle = result
text = 'We didn''t wait, so the requester'||lf||,
       'is not over the center of the'||lf||,
       'lister.'
lister request handle '"'text'" OK'
address command wait 3
lister close handle
address command wait 1
lister new "DOpus5:Images/"
handle = result
lister wait handle
text = 'We waited this time, so the'||lf||,
       'requester is over the center'||lf||,
       'of the lister.'
lister request handle '"'text'" OK'
address command wait 3
lister close handle
dopus back
exit
```

1.28 Providing some feedback

```
  If you have written a script that provides a useful purpose, it's often
nice to have some indication given to the user that something is happening.
This can be either by audible or visual means.

  Audible feedback in ARexx with Opus is only possible by using the  command
command to call the internal audio commands.

  Visual feedback through Opus can be in the form of:

 - changing the state of the lister, (for example, SRCE -> OFF)
 - changing the lister header or title.
```

```
  See  lister set  for examples of the two above.

 - changing the Desktop somehow, (see  dopus set  for this).

 - progress bars.
 - opening a viewer.

  These two are the one's we will deal with here since Opus provides commands
to directly implement them.

 dopus read

  This command allows you to call the  dopus viewer , a text reader that can
be used for displaying files in ASCII, ANSI or HEX.  It could be used for the
displaying of an intermediate result/file or the final output.

 dopus progress

   dopus progress  will open a progress bar in the middle of the Opus screen.

 lister progress
 lister newprogress

  These two commands will both open progress bars over a nominated lister,
the difference being that the  lister newprogress  version allows you have
more information in the progress bar window.
```

1.29 Magellan II ARexx Tutorial: Dopus Read

```
  The  dopus read  provides ARexx control of the internal
Opus viewer.

  You can use it to view any type of file on your system, using the normal
reader menus to switch between modes, call an editor, etc.

  When you open a viewer from ARexx, a handle will be returned in the
ARexx  RESULT  variable, much the same as for opening listers.  Having a handle
will allow you to load new files into an already open viewer, or close an
open viewer.

  Specifying the delete keyword will cause the file being viewed
to be deleted when you close the reader.  This makes it useful for viewing
temporary files, or intermediate results.

  Example:

/* DopusRead.dopus5 */
options results
address 'DOPUS.1'
address command 'Copy TuteRexx/DOpusRead.dopus5 RAM: QUIET'
"dopus read delete pos 50/50/200/100 RAM:DOpusRead.dopus5"
text = 'The reader''s handle is: 'result
dopus request '"'text'" OK'
```

```
text = 'The file ''RAM:DOpusRead.dopus5'' will be deleted when you close the  ↩
    viewer.'
dopus request '"'text'" OK'
exit
```

NOTE: Because we have included a position for the viewer to open in, (pos 50/50/200/100), it will be necessary to put the whole command within quotes so ARexx doesn't interpret 50/50/200/100 as a mathematical expression.

In the following example, we'll open the viewer in hex mode and then close it using it's handle.

Example:

```
/* DopusRead2.dopus5 */
options results
address 'DOPUS.1'
dopus read hex 'C:LoadWB'
handle = result
text = 'The reader''s handle is: 'handle
dopus request '"'text'" OK'
address command wait 3
dopus read handle quit
exit
```

1.30 Magellan II ARexx Tutorial: Dopus Progress

Similar to the lister progress and lister newprogress commands, the dopus progress command allows you to use a progress bar without having to open a lister.

Useful for indicating to a user that something is happening. As with the lister new command, this command returns a handle in the RESULT variable so that you can control the output of the progress bar, as well as close it or check the abort button. You need to save this handle if you want to do anything with the progress bar.

Example:

```
/* DopusProgress.dopus5 */
options results
address 'DOPUS.1'
dopus progress name info info2 info3 bar abort
handle = result
dopus progress handle title "DOpus Progress Test"
dopus progress handle info "Counting from 10 to 0"
dopus progress handle info2 "Press abort to end"
do i = 10 to 0 by -1
  dopus progress handle bar 10 i
  dopus progress handle info3 i" to go."
  address command wait 1
  dopus progress handle abort
  if result = 1 then leave
end
```

```
dopus progress handle off
exit
```

1.31 Magellan II ARexx Tutorial: Lister Progress

The lister progress command was the first implementation of a progress display ARexx command for listers.
As with the lister newprogress command, this is actually a sub-command of the lister set command.
If you don't need the extra information space and options that the lister newprogress command can supply, then this is a simpler command to set up and use.

When you have finished using the progress bar you can turn it off using the lister clear command.

Example:

```
/* ListerProgress.dopus5 */
options results
address 'DOPUS.1'
dopus front
lister new mode name "RAM:"
handle = result
lister wait handle
lister set handle progress 100 "Lister Progress Demo"
lister clear handle abort
do i = 100 to 0 by -5
  lister set handle progress name i"% left"
  lister set handle progress count i
  lister query handle abort
  if result = 1 then leave
  address command wait 1
end
lister clear handle progress
address command wait 2
lister close handle
dopus back
exit
```

1.32 Magellan II ARexx Tutorial: Lister NewProgress

This command is really lister set newprogress , it is another sub-command of the lister set command but was worth mentioning by itself.

This command displays a progress bar over a lister that you can update as an indication to the script users that something is happening. It is similar to the progress bar that appears when you Copy, Delete, Move, etc files from one lister to another.

Example:

```
/* ListerNewprogress.dopus5 */
options results
address 'DOPUS.1'
dopus front
plural. = 's'
plural.1 = ''
lister new mode name 'SYS:'
handle = result
lister set handle newprogress name info info2 bar abort
lister set handle newprogress title 'Newprogress Example'
lister set handle newprogress info2 'Hit abort to end.'
do i = 1 to 10
  text = i' second'||plural.i
  lister set handle newprogress name '"'text'"'
  text = (10 - i)' second'plural.(10 - i)' to go.'
  lister set handle newprogress info '"'text'"'
  lister set handle newprogress bar 10 i
  lister query abort
  if result then leave
  address command wait 1
end
lister clear handle progress
address command wait 2
lister close handle
dopus back
exit
```

The lister newprogress command provides more information than the lister progress command, so it is proportionly more work to set up.

1.33 Magellan II ARexx Tutorial: Finding and Setting Lister Attributes

Well, your lister is sitting there on the Desktop staring at you. You'd like to find out all sorts of intimate information about it, but how?

lister query

The lister query command will let you find out all there is to know about your lister and it's contents, as long as you know the handle.
Things like; it's path, the number of entries, the number of files, the fields currently displayed, the sorting method, etc are yours for the asking.

What's that? You don't like the way your lister currently looks? Have no fear, we even supply a command to let you change it.

lister set

lister set will let you change things such as; the current path, the sort direction, entry seperation method, titlebar, header, state, mode, etc all at the flick of a wrist.

There really is only two commands that let you find and set lister attributes, lister query and lister set , but I thought I'd describe a couple of others just for something to do.

Your lister opened in the wrong spot or perhaps it’s obscuring something on the Desktop that you’d like to see. Maybe you’d just like to have it scale itself relative to the screen size.

```
lister set position
```

And the one below will let you make a lister visible or invisible, the possible use of this is you can set up your lister with the contents, toolbar, mode, etc without the user seeing all this happening, and then just make it visible...magic!

```
lister set visible
```

1.34 Magellan II ARexx Tutorial: Lister Query

```
ArcDir  (There are many more than just this example in ArcDir.)
```

The lister query command allows you to retrieve information on either the lister settings or entries in the lister.

You can retrieve information such as the sorting method, (name, date, etc), the path, (RAM:, etc), total number of entries, (selected or not), just selected files or directories, specific entry information, and much more.

The results of the lister query command will generally be returned in the RESULT variable.

For example:

```
/* ListerQuery.dopus5 */
options results
address 'DOPUS.1'
lister new mode name "S:"
handle = result
call setclip('Lister.test',handle)
exit
```

Run the example and then click on the buttons below to obtain information on the new lister.

```
lister query handle numentries
lister query handle numfiles
lister query handle separate
lister query handle flags
lister query handle path
lister query handle mode
lister query handle display
lister query handle busy
```

Click on this button to close the lister.

```
lister query can also be used to find the handle for the current Active, ↩
    Source or
```
Destination lister. This is analogous to sending the {Ql} and {Qd} parameter to the script, except with the ARexx version you can get the handles of all

```
listers.
  Once you've found the Active lister, you can determine whether it is a
Source or Destination by comparing it to the list of handles retrieved for
all Source or Destination listers.

  Example:

/* Lister.dopus5 */
lf = '0a'x
options results
address 'DOPUS.1'
lister close all
'lister new 1/1/100/100'
handle1 = result
lister wait handle1
lister set handle1 source lock
'lister new 1/110/100/100'
handle2 = result
lister wait handle2
lister set handle2 source lock
'lister new 1/220/100/100'
handle3 = result
lister wait handle3
lister set handle3 dest lock
'lister new 140/1/100/100'
handle4 = result
lister wait handle4
lister set handle4 dest lock
'lister new 140/110/100/100'
handle5 = result
lister wait handle5
lister set handle5 off
'lister new 140/220/100/100'
handle6 = result
lister wait handle6
lister set handle6 busy on

lister query all
text = 'Handles of All non-busy listers,'||lf||,
       'there should be five of them:'||lf||,
        result
dopus request '"'text'" OK'

lister query source var dummy
text = 'Handles of all Source listers,'||lf||,
       'there should be two of them:'||lf||,
        dummy
dopus request '"'text'" OK'

lister query dest var dummy
text = 'Handles of all Dest listers,'||lf||,
       'there should be two of them:'||lf||,
        dummy
dopus request '"'text'" OK'

text = 'You''ll have five seconds to make a non-busy'||lf||,
       'lister active, just click on it''s titlebar,'||lf||,
```

```
       'then I''ll tell you it''s handle and state.'
dopus request '"'text'" OK'

address command wait 5
lister query active
if result = 'RESULT' | result = 0 then
  dopus request '"You didn''t activate one!" OK'
else do
  handle = result
  text = 'The Active lister''s handle is: 'handle
  lister query source
  if pos(handle,result) > 0 then
    text = text||lf||'It is a SOURCE lister.'
  else do
    lister query dest
    if pos(handle,result) > 0 then
      text = text||lf||'It is a DESTINATION lister.'
    else
      text = text||lf||'It''s state is OFF.'
    end
  dopus request '"'text'" OK'
  end

lister set handle6 busy off
lister close all
exit
```

Edmund's SwapListers , Leo's WinCopy and SwapWin and the CDO.dopus5 provide some real-life examples of how this can be used.

There is much more information that can be obtained through the lister query , please refer to the DOpusM2_ARexx.guide for the other options.

1.35 Magellan II ARexx Tutorial: Lister Set

ArcDir (There are many more than just this example in ArcDir.)

You can set various lister attributes, (sorting, title, etc), file displays, (eg. hide icon files, #?.info), with the lister set command.

There are quite a lot of parameters for this command, a full description can be found in the DOpusM2_ARexx.guide.

Versions of lister set that in some way change the lister display require a lister refresh statement issued before the changes become visible.

If the command only changes the entry display of the lister, then only a 'lister refresh <handle>' is required.
If it changes the titlebar, status bar, etc, then you need to issue a 'lister refresh <handle> full'.

Some examples of setting attributes are as follows:

```
/* ListerSet.dopus5 */
options results
```

```
address 'DOPUS.1'
lister new mode name "SYS:"
handle = result                                /* Store the handle */
call setclip('Lister.test',handle)
exit
```

  Run the above example, then click on one of the buttons below to change various options of the lister.

```
 lister set handle dest
 lister set handle source
 lister set handle busy on
 lister set handle busy off
 lister set handle header 'This is the header'
 lister set handle title 'This is the title'
 lister set handle mode icon
 lister set handle mode icon action
 lister set handle mode name
 lister set handle flags reverse
 lister set handle separate filesfirst
 lister set handle toolbar 'RAM:Demo.tb'
```

  Click on this button to close the lister.

1.36 Magellan II ARexx Tutorial: Lister Set Position

  The lister set position command allows you to move listers around the screen, this could be useful for uncovering parts of the display or, as you can see here , Edmund has used it to emulate the Opus v4 SwapWin command.

  Example:

```
/* ListerPosition.dopus5 */
options results
address 'DOPUS.1'
dopus front
dopus screen
width = word(result,4)
"lister new 0/50/200/200"
handle = result
text = 'What''s over there ->'
lister request handle '"'text'" OK'
do forever
  lister query handle position
  pos = result
  parse var pos x''y''w''h
  newx = x + 5
  if newx + w > width then do
    text = 'Looks like the end of the screen!'
    lister request handle '"'text'" OK'
    leave
    end
  x = newx
  lister set handle position x''y''w''h
```

```
end
address command wait 2
lister close handle
exit
```

  I’m sure someone can think of another use for this command :)

1.37 Magellan II ARexx Tutorial: Lister Visible

  You might wonder why you want to make a lister invisible, here’s a couple of possible reasons:

a) You want to clear the Desktop of windows without closing listers.
b) You can set up the lister with a new toolbar and entries and then make it visible, rather than doing it in front of the user.

  The lister set visible command allows you to make a lister visible or not.

  Example:

```
/* ListerVisible.dopus5 */
options results
address 'DOPUS.1'
dopus front
say pragma('d')
address command 'copy TuteRexx/extras/Surprise.info RAM:'
address command 'copy TuteRexx/extras/Example.tb RAM:'
lister new mode name invisible 'SYS:'
handle = result
dopus request '"Ready?" Yep'
lister set handle toolbar 'RAM:Example.tb'
lister set handle title 'Lister visible'
lister set handle visible on
address command wait 3
lister close handle
address command 'delete RAM:Example.tb'
address command 'delete RAM:Surprise.info'
dopus back
exit
```

1.38 Magellan II ARexx Tutorial: The Phantom is...

```
   666          666          666
  666          666          666
 666          666          666
666          666          666
6666666      6666666      6666666
666  666     666  666     666  666
666   666    666   666    666   666
666   666    666   666    666   666
 666  666     666  666     666  666
```

```
          666666        666666        666666


               /* ListerTest.dopus5 */
```

1.39 Magellan II ARexx Tutorial: Manipulating Lister Entries

The main purpose of the listers is to enable you to manipulate files/dirs easily. The commands below are the main ones used for; finding information about a file/directory, selecting entries, removing and adding entries.

dopus getfiletype

This command returns the Filetype of a file or directory as defined in your Filetypes, it does not need a lister to work.

lister query entry

This command provides every bit of information you can get on a file or directory, datestamp, protection, comment, etc, etc.

With the commands below, it is necessary to perform a lister refresh after them in order to have the lister display updated.

lister select

This command lets you select and unselect entries in listers, perhaps your script lets users select files then pick an action, as each file is processed you could unselect it in the lister display giving the user direct visual feedback of the progress.

lister remove

The lister remove command lets you remove entries from the display, it doesn't remove them from the storage medium, just the display.

lister add

This command is simpler than the one below, but as a result, is nowhere near as powerful, lister addstem should be used in preference. It is here for the sake of completeness.

lister addstem

This command allows you to add entries to listers, but it is not restricted to just entries. It can add arbitrary display strings and popup menus as well.

1.40 Magellan II ARexx Tutorial: Dopus Getfiletype

ArcDir

The dopus getfiletype command is useful for just obtaining information on what type an file is, or what the Filetype ID is for the file.

You can use it for determining if your script has been called with the right type of file, that is, if your script is designed to handle LhA archives it's hardly worthwhile having it trying to process a ZIP archive.

It can also be used to identify the Filetype ID of your argument, for example, it will return 'LHA' if used to identify an LhA archive, (providing that's what you use for an LhA Filetype ID of course).

Example:

```
/* DopusGetfiletype.dopus5 */
lf = '0a'x
options results
address 'DOPUS.1'
dopus front
address command 'RequestFile >ENV:testfile S: title "Pick a file"'
address command 'rxset getfiletype '$testfile
file = getclip('getfiletype')
dopus getfiletype file
text = 'The filetype is: 'result
dopus getfiletype file ID
if result ~= 'RESULT' | result ~= '' then
  text = text||lf||lf||'The Filetype ID is: 'result'.'
else
  text = text||lf||lf||'The Filetype ID is unknown.'
dopus request '"'text'" OK'
call delete('ENV:testfile')
dopus back
exit
```

1.41 Magellan II ARexx Tutorial: Lister Query Entry

ArcDir

The lister query entry command allows you to find information about an entry, (file, directory, etc), in a lister.

The output of the command can be displayed in two ways basically, depending on how you format the command.

The first way will output the information in one long string of the form:

<name> <size> <type> <selection> <seconds> <protect> <comment>

This will returned in either RESULT or a variable that you specify if you include the var parameter in the command.

The alternative provides a lot more information which is output in the form of compound variables, the stem of which, you specify.

A simple description of a compound variable:

```
      / name     = info.name
 info.- date     = info.date
      \ type     = info.type
   ^                    ^
   |                    |
 This is             These are the
 the 'stem'.       'compound' variables.
```

  A more involved description of compound variables and stems can be found in the Amiga ARexx manual.

  Example:

```
/* ListerQueryEntry.dopus5 */
options results
address 'DOPUS.1'
dopus front
lf = '0a'x
lister new mode name "S:"
handle = result
lister set handle display name comment version filetype
lister refresh handle full
lister wait handle
lister query handle entry "Startup-Sequence" var output
output = 'This is the output from ''lister query entry'' of'||lf||,
         'the file ''S:Startup-Sequence'' without stem variables:'||lf||,
         lf||output
dopus request '"'output'" OK'
lister query handle entry "Startup-Sequence" stem info.
text = 'The following output is using compound variables'||lf||,
       'with a stem of ''info.'''||lf||,
       'info.NAME       = '||info.NAME||lf||,
       'info.SIZE       = '||info.SIZE||lf||,
       'info.TYPE       = '||info.TYPE||lf||,
       'info.SELECTED   = '||info.SELECTED||lf||,
       'info.DATE       = '||info.DATE||lf||,
       'info.PROTECT    = '||info.PROTECT||lf||,
       'info.DATESTRING = '||info.DATESTRING||lf||,
       'info.PROTSTRING = '||info.PROTSTRING||lf||,
       'info.COMMENT    = '||info.COMMENT||lf||,
       'info.FILETYPE   = '||info.FILETYPE||lf||,
       'info.VERSION    = '||info.VERSION||lf||,
       'info.REVISION   = '||info.REVISION||lf||,
       'info.VERDATE    = '||info.VERDATE||lf||,
       'info.DATENUM    = '||info.DATENUM||lf||,
       'info.TIME       = '||info.TIME
dopus request '"'text'" OK'
lister close handle
dopus back
exit
```

  A description of the various variables above can be found in the DOpusM2_ARexx.guide .

NOTE: To obtain some variables, that particular information has to be displayed in the lister, for example: you will not get the version

information if you have not enabled the version display in the lister.

1.42 Magellan II ARexx Tutorial: Lister Select

ArcDir

The lister select command allows you to select or unselect various entries in a lister according to their name, or a cardinal number that is allocated due to their position in the lister.

Example:

```
/* ListerSelect.dopus5 */
options results
address 'DOPUS.1'
dopus front
lister new "RAM:"             /* Open a new lister with path = RAM: */
handle = result               /* Store it's handle */
lister wait handle            /* Wait until the lister is free */
lister select handle "T" on   /* Select the T: directory */
lister refresh handle         /* Refresh, T: should show as selected */
address command wait 3        /* Wait 3 seconds */
lister select handle "T" off  /* Unselect the T: directory */
lister refresh handle         /* Refresh, T: shouldn't be selected */
address command wait 3        /* Wait 3 seconds */
lister close handle           /* Close the lister */
exit

/* ListerSelect2.dopus5 */
options results
address 'DOPUS.1'
dopus front
lister new "SYS:"             /* Open a new lister with path = SYS: */
handle = result               /* Store it's handle */
lister wait handle            /* Wait until the lister is free */
lister select handle #0 on    /* Select the first entry */
lister refresh handle         /* Refresh, should show as selected */
address command wait 3        /* Wait 3 seconds */
lister select handle #0 off   /* Unselect the first entry */
lister refresh handle         /* Refresh, shouldn't be selected */
address command wait 3        /* Wait 3 seconds */
lister close handle           /* Close the lister */
exit
```

1.43 Magellan II ARexx Tutorial: Lister Remove

ArcDir

The lister remove command allows you to remove entries from a lister. It doesn't delete the file or directory, just the entry in the display.

This could be useful for use in a custom handler where you could be adding

and removing entries as the result of a user's actions.

  Example:

```
/* ListerRemove.dopus5 */
options results
address 'DOPUS.1'
dopus front
lister new mode name 'SYS:'
handle = result
lister wait handle
text = 'We will now remove the ''Devs'' directory from the display'
lister request handle '"'text'" OK'
lister remove handle 'Devs'
lister refresh handle
address command wait 3
lister close handle
dopus back
exit
```

1.44 Magellan II ARexx Tutorial: Lister Add

 ArcDir

 lister add

  The lister addstem command allows you to add far more information and is more flexible than the lister add command, and is discussed below.
  Here we'll just see what the lister add command can do.

  The first difference between the two commands, is that with the lister add command you must specify all parameters before an entry is added. So on the command line you must include:

  <filename> <size> <type> <seconds from 1-1-1978> <protection bits> <comment>

  Example:

```
/* ListerAdd.dopus5 */
options results
address 'DOPUS.1'
lister new mode name
lister set handle display name size date protect comment
lister refresh handle full
handle = result                              /* Store the handle */
call setclip('Lister.test',handle)
exit
```

  Run the above example, then click on one of the buttons below to add an entry.

```
 lister add handle 'Directory1' 2 '1' 1000 rwed 'A Directory'
 lister add handle 'File1' 546 '-1' 60000 rwd 'A File'
 lister add handle 'Directory2' 1234 '2' 500000 d 'in assign colour'
 lister add handle 'File2' 64554 '-2' 734398 rw 'in device colour'
```

```
 lister add handle 'Directory3' 54654 '3' 87657 hwd 'in bold (link)'
 lister add handle 'File3' 245 '-3' 87656 hsparwed 'in bold (link)'
 lister add handle 'Directory4' 45 '4' 435435 har 'in assign colour and bold'
 lister add handle 'File4' 436565 '-4' 76567 pwed 'in device colour and bold'
```

Click on this button to close the lister.

This is equivalent to the AddFile command in Opus v4.

1.45 Magellan II ARexx Tutorial: Lister Addstem

With the lister addstem command, you need to set up the compound variables before you can add the entry to the lister. (See the lister query entry command for a simple explanation of compound variables.)

To be able to add an entry using stems you must specify at the very least a filename, the other information will be set to the defaults, (for example, current date/time, no comment, rwed protection bits).

Example:

```
/* ListerAddstem1.dopus5 */
options results
address 'DOPUS.1'
dopus front
lister new mode name
handle = result
fileinfo.NAME = 'TestFile'
lister addstem handle fileinfo.
lister refresh handle
address command wait 3
lister close handle
dopus back
exit
```

This is equivalent to the AddFile command in Opus v4.

You can also get the information from a lister query entry command and use that for adding an entry without modification.

Example:

```
/* ListerAddstem2.dopus5 */
options results
address 'DOPUS.1'
dopus front
lf = '0a'x
lister new mode name "S:"
handle = result
lister set handle display name comment version filetype
lister refresh handle full
lister wait handle
address command wait 2
lister query handle entry "Startup-Sequence" stem info.
```

```
text = 'We will get the information for the'||lf||,
       '''Startup-Sequence'' file, and then'||lf||,
       'change the path in the lister and'||lf||,
       'add an entry with the same data.'
dopus request '"'text'" OK'
text = 'The following information is for the'||lf||,
       'file ''S:Startup-Sequence'':'||lf||lf||,
       'NAME = '||info.NAME||lf||,
       'SIZE = '||info.SIZE||lf||,
       'TYPE = '||info.TYPE||lf||,
       'SELECTED = '||info.SELECTED||lf||,
       'DATE = '||info.DATE||lf||,
       'PROTECT = '||info.PROTECT||lf||,
       'DATESTRING = '||info.DATESTRING||lf||,
       'PROTSTRING = '||info.PROTSTRING||lf||,
       'COMMENT = '||info.COMMENT||lf||,
       'FILETYPE = '||info.FILETYPE||lf||,
       'VERSION = '||info.VERSION||lf||,
       'REVISION = '||info.REVISION||lf||,
       'VERDATE = '||info.VERDATE||lf||,
       'DATENUM = '||info.DATENUM||lf||,
       'TIME = '||info.TIME
dopus request '"'text'" OK'
text = 'We will now add this entry'||lf||,
       'to the lister after changing'||lf||,
       'it''s path to ''RAM:'''
dopus request '"'text'" OK'
lister read handle "RAM:" force
lister wait handle
lister addstem handle info.
lister refresh handle
text = 'This hasn''t copied the file from ''S:'''||lf||,
       'it has just copied the entry details for'||lf||,
       '''Startup-Sequence'' into the lister display.'
dopus request '"'text'" OK'
text = 'If we reread the directory the entry'||lf||,
       'will disappear because the file doesn''t'||lf||,
       'exist in ''RAM:'''
dopus request '"'text'" OK'
lister read handle "RAM:" force
address command wait 3
lister close handle
dopus back
exit
```

While listers are mainly file manipulation interfaces, you can make them display other information. There is a DISPLAY field that can be used to display any text you want, it can contain up to 256 characters.

Two good examples of what you can use the DISPLAY field for are shown in my Grep.dopus5, (used for displaying text search output), and Awari.dopus5 scripts, (a game in a lister). Both of these scripts should be available on either Aminet by the time you read this, or on the DOpus Plus CD.

Example:

```
/* ListerAddstem3.dopus5 */
options results
address 'DOPUS.1'
dopus front
lister new '0/11/150/200' mode name
hande = result
fileinfo.DISPLAY = 'Some really silly text demonstrating the DISPLAY field !@#$ ↩
    %^&*()_+|:?><,./][{}\=-`~'
lister addstem handle fileinfo.
lister refresh handle
address command wait 5
lister close handle
dopus back
exit
```

This is equivalent to using AddCustEntry command in Opus v4.

1.46 Magellan II ARexx Tutorial: Command

ArcDir

The command command is used for calling one of Opus' commands that is in the internal command list, whether they be an 'original' or a 'replaced' version.

Since internal commands act primarily between a SOURCE lister and a DESTINATION lister, this is the way the command command works since it is only calling internal commands. If there aren't any, then the command is ignored. You can tell the command to act specifically on a SOURCE and DESTINATION lister by specifying their handles in the command line, like so:

```
command source <source_handle> dest <dest_handle> .......
```

In the following example, we will only use one SOURCE lister, to ensure this all other listers will be closed, then a new one opened and locked as SOURCE.

Example:

```
/* Command.dopus5 */
options results
address 'DOPUS.1'
lister close all
lister new mode name "DOpus5:Icons"
handle = result
lister set handle source lock
call setclip('Lister.test',handle)
exit
```

```
WARNING: The following buttons allow you to select files/dirs,
         if you then use the last button, those files will be DELETED!!!
         I have used the DOpus5:Icons directory as an example because it
         generally contains non-critical files that you can re-install
         from the distribution disks.
```

```
          Please use the commands in the order they are shown, OR make
          certain before you push the last one that the path in the lister
          is the correct one!!!!!!
```

Run the example above, then click on the buttons below to see what happens.

In the first two command buttons below, we've added the parameter wait . This is just telling Opus to wait until that command has been executed, normally the commands will be done asynchronously, that is, it doesn't wait to see the result.

```
command wait all
command wait none
command flash
command beep
```

Supposing you have replaced the internal Copy command with an enhanced version, (See Example 12: Improving the internal commands), to ensure his script runs the same on everybodies setup, the writer could use the original parameter to tell Opus to use the original Copy command, not your enhanced version.

```
command original copy DOpus5:Icons TO RAM:

command ScanDir RAM:Icons
command select name Opus.info
command play
command select name Group.info
command read
command doubleclick Settings.info
command parent
command select name Icons

Check your path before using!!!!!

command delete
```

Click on this button to close the lister.

1.47 Magellan II ARexx Tutorial: FTP Commands

OpusFTP doesn't have any ARexx commands, but Greg wanted something simple, so here I am :)

The number of commands in the OpusFTP module has been reduced in the latest version because the need for some of them has been made obsolete.

In previous versions of OpusFTP you needed commands that could allow you access to so-called 'hidden' files or directories. For example, on Aminet you have a file named 'RECENT', and also a directory named 'recent'. Both appear in the Aminet 'root' directory, but because they had the same name only one would be displayed in a lister, (remember, AmigaDOS is case-insensitive, even though the target filesystem might not have been).

Due to the fact that listers can now be case-sensitive, there is now no need for the following commands: FTPCD, FTPCopy, FTPDelete, and FTPRename.

The OpusFTP module has no ARexx interface, but the commands it adds to Opus can be accessed as though they were normal Opus commands like All, None, etc.

To access them you just call them as you would for any other normal Opus internal command, that is, you preceed them with the command command.

The following commands are all that are available through the OpusFTP module interface.

```
FTPAdd            FTPAddressBook    FTPCommand        FTPConnect
FTPOptions        FTPQuit           FTPSetVar
```

The ftpaddressbook command doesn't do anything amazing, all it does is open the FTP address book so that you can edit or connect to an entry.

Example:

```
/* FTPAddressbook.dopus5 */
options results
address 'DOPUS.1'
command ftpaddressbook
address command wait 3
command ftpquit
exit
```

This is probably even easier than the dopus back and dopus front commands.

The FTPConnect command instructs OpusFTP to connect to the specified site, using a username/password if provided, changing to a specified directory, and listing it, if asked.

NOTE: The following example will require a TCP/IP connection to the InterNet in order to execute, and it will try to download the latest Aminet INDEX of directory biz/dopus to RAM:. Do not execute it if you don't have a stack running or believe it will cause financial hardship.

With this example, because we know there will be a file called INDEX in the biz/dopus directory, (standard Aminet directory index), there is no need to actually have the lister scan the directory in. This can save time, (and save you some money too ;-) and we can do this by adding the flag noscan to the ftpconnect command.

Example:

```
/* FTPConnect.dopus5 */
lf = '0a'x
options results
address 'DOPUS.1'
'command wait ftpconnect host wuarchive.wustl.edu dir /pub/aminet/biz/dopus noscan ↩
    recon'
handle = result
```

```
text = 'We should now be connected to Aminet'
dopus request '"'text'" OK'
text = 'I like this directory, in fact I''ll think'||lf||,
       'I''ll add it to your address book :)'
dopus request '"'text'" Yes?|No!'
if rc ~= 0 then command ftpadd
lister set handle source
'command original wait copy INDEX TO RAM:'
lister close handle
command ftpquit
exit
```

Directory Opus is fully multitasking and multithreading, just because you have closed the lister that was your FTP connection does not mean that the FTP process has been ended.

If you no longer require the FTP module loaded, then issuing an FTPQuit command will remove it from the task list. This will also stop a requester opening when you quit your TCP/IP stack.

For example, supposing you had a script that did the following:

1) Start Miami.
2) Log on to ISP.
3) Start OpusFTP.
4) Log in to an Aminet site and download the 'RECENT' file.
5) Close the FTP connection.
6) Tell Miami to log off and quit.
7) Analyse RECENT file for specific keywords indicating interesting stuff, and present them in the dopus viewer for you.

If you didn't tell the OpusFTP module to quit, then when you asked Miami to quit, Miami would put up a requester similar to the following:

```
The following applications
are still using Miami:

 ftp_addressbook

Proceed anyway?

   Yes   No
```

Your script would halt, waiting for the requester to be acknowledged. This would mean that step 7, which analyses the 'RECENT' file for interesting stuff would never happen, until you acknowledged Miami's requester.

In fact while we're at it, why don't we create a simple script to do the above steps. Since this will be simple, practically no error checking will be used, you could expand upon this script to add it and any other features, like downloading the interesting files as well.

If you're not using Miami v3, which provides the assign 'Miami:', then change the line:

```
address command 'Miami:Miami'
```

to point to where you keep it.

```
/* ParseAminetRecent.dopus5 */
lf = '0a'x
interesting = 'biz/dopus hard/hack util/dtype sex beer money'

options results
address 'DOPUS.1'

if ~show('p','MIAMI.1') then do
  address command 'Miami:Miami'
  WaitForPort 'MIAMI.1'
  if ~show('p','MIAMI.1') then do
    dopus request '"Unable to run Miami!" OK'
    exit
    end
  end

flag = 0
address 'MIAMI.1' 'ISONLINE'
if rc then flag = 1             /* see if Miami is already online */
do while flag = 0
  address 'MIAMI.1' 'ONLINE'
  address 'MIAMI.1' 'ISONLINE'
  if rc then flag = 2
end

lister new
handle = result
lister set handle source
'command wait ftpconnect lister 'handle' host ftp.livewire.com.au dir /pub/aminet  ↩
   noscan recon'

command source handle original wait 'copy name=RECENT to=RAM:'

lister close handle
command ftpquit
if flag = 2 then do
  address 'MIAMI.1' 'OFFLINE'  /* Leave Miami online if it */
  address 'MIAMI.1' 'QUIT'     /* was already              */
  end

if ~open('infile','RAM:RECENT','R') then do
  dopus request '"Cannot open RECENT file!" OK'
  exit
  end

text = 'Found these interesting files:'
do while ~eof('infile')
  aline = readln('infile')
  do i = 1 to words(interesting)
    if index(aline, word(interesting,i)) > 0 then do
      text = text||lf||aline
      leave
      end
  end
end
```

```
call close('infile')

dopus request '"'text'" OK'
exit
```

1.48 Magellan II ARexx Tutorial: Opus v4 functions

Some of you may have noticed that some of the commands from Opus v4.x are missing from Opus v5.x.

While there really is no need for some of these commands in Opus v5, some users have decided they couldn't live without them and have created scripts that emulate these commands.

Below is a list of the Opus v4 internal and ARexx commands, together with their equivalent in Opus v5. The commands in the Opus v4 column in white are the ARexx commands that were available in v4.

Directory Opus V4	Directory Opus V5
AbortPrint	Available from the Print requester .
About	Menu - Opus About
AddCustEntry	lister addstem
AddCustHandler	lister set handler
AddFile	lister add lister addstem
AddIcon	AddIcon
Alarm	Alarm
All	All
AnsiRead	AnsiRead dopus read
ARexx	Can be selected via the Function Editor .
Assign	Assign
Beep	Beep
BufferList	CacheList
Busy	lister set busy Now that the listers go busy, there is really no need to busy the pointer. Remember Opus v5 is fully multitasking and multithreading.
ButtonIconify	Available through each button banks popup menu.
CD	
CheckAbort	dopus progress lister query abort
CheckFit	CheckFit
ClearBuffers	FreeCaches lister clearcaches
ClearSizes	ClearSizes
ClearWin	lister clear
Clone	Duplicate
Comment	Comment
Configure	All settings are available through the Settings menu.
ContST	

Copy	Copy
CopyAs	CopyAs
CopyWindow	A script to emulate this function lister copy
DateStamp	DateStamp
Defaults	You can load any environment through the Environment Editor .
Delete	Delete
DirTree	Listers do not support displaying directories as a tree, but there is no reason why you could not write a script or module to do it :)
DiskCopy	DiskCopy
DiskCopyBG	No longer required because the DiskCopy process multitasks.
DiskInfo	DiskInfo
DisplayDir	ScanDir lister refresh
DOpusToBack	dopus back
DOpusToFront	dopus front
Encrypt	Encrypt
ErrorHelp	
Execute	Can be selected via the Function Editor.
FileInfo	lister query entry
FinishSection	FinishSection
Format	Format
FormatBG	No longer required because the Format process multitasks.
GetAll	lister query entries
GetDevices	DeviceList
GetDirs	lister query dirs
GetEntry	lister query entry
GetFileType	dopus getfiletype
GetFiles	lister query files
GetNextSelected	lister query firstsel
GetSelectedAll	lister query selentries
GetSelectedDirs	lister query seldirs
GetSelectedFiles	lister query selfiles
GetSizes	GetSizes
Help	Simply press the Help key, or use the Opus - Help! menu item.
HexRead	Read HexRead dopus read
Hunt	FindFile
Iconify	Hide
IconInfo	IconInfo
Install	Available through the Format process.
InstallBG	Available through the Format process.
LastSaved	Available through the Environment Editor .
LoadConfig	LoadEnvironment
LoadStrings	
LoopPlay	
MakeDir	MakeDir
Modify	Modify items using the Environment Editor then choose Use.
Move	Move

MoveAs	MoveAs
NewCLI	CLI
NextDrives	DeviceList
None	None
OtherWindow	Since you are not restricted to two windows like v4, there is no equivalent to this function. The easiest way to swap listers is to activate them with the mouse.
Parent	Parent
PatternMatch	
Play	Play
PlayST	Play will handle some formats but it is better to use a dedicated module player like DeliTracker, etc using filetypes.
Print	Print
PrintDir	PrintDir
Protect	Protect
Query	lister query
Quit	Quit
Read	Read dopus read
Redraw	Use the Icons - Reset menuitem, to redraw Opus' Desktop.
Relabel	Rename
Remember	
RemoveEntry	lister remove
RemoveFile	lister remove
Rename	Rename
Request	dopus request dopus getstring lister request lister getstring
Rescan	ReReadDir
Reselect	ReSelect
Restore	
Root	Root
Run	Run
SaveConfig	Available through the Environment Editor .
ScanDir	ScanDir
ScrollH	
ScrollToShow	You can scroll the listers while operations are happening.
ScrollV	
Search	Search
Select	Select
SelectEntry	lister select
SelectFile	lister select
SetWinTitle	lister set title
Show	Show
SmartRead	SmartRead
Status	lister query lister set
StopST	The Play command provides an abort button to end playing, provided you haven't used the QUIET switch.

```
SwapWindow                          A script to emulate this function
TechSupport                         Read the manual!
Toggle                              Toggle
TopText                             lister set header
                                    lister set title
Uniconify                           Reveal
User1 - User4                       User
                                    User1
                                    User2
                                    User3
                                    User4
Verify                              Confirm
Version                             dopus version
```

1.49 Magellan II ARexx Tutorial: Integration

Opus can be used to provide functions not available in other programs, such as the 'Opus and AWeb' link below, or perhaps just a convenient interface to another program.

Opus and AWeb

1.50 Magellan II ARexx Tutorial: Opus and AWeb II v2.x

Here is a simple example of integrating OpusFTP functions into AWeb II, which prior to v3.x had no native code to handle ftp:// sites.

When you click on a ftp:// link in AWeb II, the site will be passed to OpusFTP as well as the initial directory if one exists.

If the port 'DOPUS.1' doesn't exist the script will exit with an error, otherwise OpusFTP will assume an anonymous login and try to connect to that site changing to the initial directory if one was passed.

```
/*
OpusAWebFTP.dopus5
$VER: OpusAWebFTP.dopus5 1.2 (14.5.96) Andrew Dunbar

Copy to your Dopus5:Arexx directory
Add these lines to AWeb's Setting/Network 3: External programs

   Command:   Sys:Rexxc/RX
   Arguments: DOpus5:Arexx/OpusAWebFTP.dopus5 %s %s
*/

if ~show('P','DOPUS.1') then exit 20
address 'DOPUS.1'
parse arg host dir .
if host ~= '' then do
  dopus front
  if dir ~= '' then command ftpconnect host 'dir' dir
  else command ftpconnect host
```

```
  end
exit
```

1.51 Magellan II ARexx Tutorial: Some ideas

```
  Here's where I try to inspire you into bashing away on that keyboard :)

1) A script to make updating your WWW page simple, should be easy.

   After you've logged on to your site via OpusFTP, open a lister with your
   local WWW page directory structure and run the script.

 a. Get all the file entries in the local lister with  lister query files .
 b. Do a  lister query entry  on the ftp lister.
 c. If it doesn't exist, copy it.
 d. If the creation date is earlier than the local version, copy over it.
 e. After you've done the files, then get the list of directories and check
    their creation dates.
 f. If the dates are earlier, change path to the directory in both listers
    and go to step a. (Recursion)
 g. When you've finished that directory, restore to the initial path and do
    the next.

  I'll watch for it on Aminet ;-)

  Of course you could just use the options for the internal Copy command, but
that's so boring :^)

2) We need more games in listers, Opus is much too serious.  I've done one,
   but I want to see OpusInvaders in a lister.

3) Archive handler for ZIP, TAR, GZIP and other archives.  (Beat me to it,
   please :)

4) Link it into your home automation, get it to boil that egg.

  Use your imagination, I'm sure there's always been something you wanted
Opus to do or do differently.  Opus is the most configurable program that's
been written on the Amiga, how it works and what it does is all up to you.

                              USE THE FORCE!
```

1.52 Magellan II ARexx Tutorial: Opening a lister with the path of your current shell

```
  Even with the few commands we have seen so far, we can create some useful
ARexx scripts, here is a simple example:

/*
 $VER: DOS-DOpus.dopus5 1.0 (28.7.98)
 Loads current shell directory into a new DOpus5 lister.
*/
options results                         /* Enable results */
```

```
if ~show('P','DOPUS.1') then do        /* Check for DOPUS.1 ARexx port */
  Say "DirectoryOpus is not running."  /* Warn user if DOpus not running */
  Exit 5                               /* Exit with result code = 5 (Warn) */
  end                                  /* End this loop */
Address 'DOPUS.1'                      /* Address DOpus */
dopus front                            /* Bring DOpus to the front */
dir = pragma('d')                      /* Returns current shell in 'dir' */
lister new dir                         /* Open a new lister, path 'dir' */
exit                                   /* Exit */
```

Pressing this button will put this script into the DOpus5:ARexx/ directory, then from an AmigaDOS shell type:

```
rx DOpus5:ARexx/DOS-DOpus.dopus5
```

What will happen is the script will bring Opus to the front of your display, then open a new lister with the shell's path, the script will then end. You can then select files in that directory to perform operations on without having had to open a new lister and work your way through the directory structure.

We don't need to store the lister's handle for such a simple script, because we don't perform any further operations on that lister.

Hint: Rather than having to type in 'rx DOpus5:ARexx/DOS-DOpus.dopus5' every time you want to use the script, edit your S:Shell-Startup file and insert something similar to the following line:

```
Alias DD "rx DOpus5:ARexx/DOS-DOpus.dopus5"
```

Now when you type 'DD' in a shell it will call the above script.

1.53 Magellan II ARexx Tutorial: Changing the shell path to the same as the lister

Using the lister query command can now allow us to do almost the opposite of the first example given previously.

This example will allow you to change your current path in a shell by entering a simple command, to that of the source, destination or any lister, in that order.
That is, if there is a source lister you will change to it's path, if there isn't a source lister, then the first destination lister will be used, if there isn't a destination lister, then the first non-busy lister will be used.
If no non-busy listers are found then a requester will open informing you.

```
/*
 $VER: CDO.dopus5 1.0 (6.8.98)
 Changes your shell path to lister path
*/
options results
if ~show('P','DOPUS.1') then do        /* Check for DOPUS.1 ARexx port */
  Say "Directory Opus is not running." /* Warn user if Opus not running */
  Exit 5                               /* Exit with result code = 5 (Warn) */
  end                                  /* End this loop */
```

```
address 'DOPUS.1'
lister query source                          /* Look for a Source lister */
if result = '' | result ~= 'RESULT' then call gotone /* We've found one */
lister query dest                       /* Look for a Destination lister */
if result = '' | result ~= 'RESULT' then call gotone /* We've found one */
lister query all                         /* Look for any non-busy lister */
if result = '' | result ~= 'RESULT' then call gotone /* We've found one */
dopus request '"No non-busy listers found" OK'     /* We didn't find any */
exit 5                                               /* suitable listers */

gotone:
parse var result handle .     /* Parse the first lister handle */
lister query handle path      /* Ask for it's current path */
path = strip(result,'b',"'")  /* Store the path, strip "'s */
push path                     /* Push the result into the input stream */
say pragma('d')
exit
```

Press this button to copy this script to the DOpus5:ARexx/ directory, open a lister, then in a shell type:

```
rx DOpus5:ARexx/CDO.dopus5
```

The shell's current path should now be the same as the lister.

Hint: Rather than having to type in 'rx DOpus5:ARexx/CDOpus.dopus5' every time you want to use the script, edit your S:Shell-Startup file and insert something similar to the following line:

```
Alias CDO "rx DOpus5:ARexx/CDOpus.dopus5"
```

Now when you type 'CDO' in a shell it will call the above script.

1.54 Magellan II ARexx Tutorial: Improved DOS-DOpus script (Example 1)

Now that we have seen what the command command does, we can improve the functionality of our original example script, DOS-DOpus.dopus5.

```
/*
 $VER: DOS-DOpus.dopus5 1.1 (28.7.98)
 Loads current shell directory into a new DOpus5 lister, selects files
 according to the passed filespec.

*/
options results                         /* Enable results */
parse arg pattern .                     /* Parse a given pattern into a variable */
if ~show('P','DOPUS.1') then do         /* Check for DOPUS.1 ARexx port */
  Say "Directory Opus is not running." /* Warn user if Opus not running */
  Exit 5                                /* Exit with result code = 5 (Warn) */
  end                                   /* End this loop */
Address 'DOPUS.1'                       /* Address Opus */
dopus front                             /* Bring Opus to the front */
dir = pragma('d')                       /* Returns current shell in 'dir' */
lister new dir                          /* Open a new lister, path 'dir' */
handle = result                         /* Store the handle */
```

```
lister wait handle                      /* Wait for the lister to finish */
if pattern ~= '' then
  command source handle select name='"'pattern'"'  /* Select the files */
lister refresh handle                 /* Refresh the display */
exit                                    /* Exit */
```

  Now when you enter:

  rx DOpus5:ARexx/DOS-DOpus.dopus5 #?.info

  a lister will open with your shell's current path, and any icon files will show as selected.

  It's also possible to have Opus perform some function on the selected files by adding a couple of lines.

```
if function ~= '' then
  command source handle original wait function
```

  If you add these two lines after the command source handle select... line, then when you enter:

  rx DOpus5:ARexx/DOS-DOpus.dopus5 #?.info delete

  A lister will open, all icon files will be selected then they will be deleted.

1.55 Example 6: Changing the background every 30 seconds

  This is just a diversion from serious scripts, all it does is change the Desktop background every 30 seconds.

```
/* RandomBackground.dopus5 */
options results
signal on halt
address command 'Copy Extras/#?.iff T:'
if ~show('l','rexxsupport.library') then
  call addlib('rexxsupport.library',,-30)
address 'DOPUS.1'
dopus query background main
oldmain = result
pattern = 'T:#?.iff'
do forever
  dopus set background '"'pattern'"' custom
  dopus refresh background custom
  call delay(1500)
end
halt:
dopus refresh background
address command 'Delete T:#?.iff QUIET'
exit
```

  Hit this button to stop it, your background will revert to normal when it goes to change it again.

1.56 Magellan II ARexx Tutorial: Simple ARexx Module #1

This simple module adds a command to the internal command list called Ports. When you call this command it will use the ARexx Show command to get a list of message ports from the system, then format them and present the output in an Opus requester.

This is a very simple example of a module in that it does not need to get any information from Opus and only uses two Opus ARexx commands in total.

By removing lines 5, 8-11, and changing line 6 to: address 'DOPUS.1' you can change this script so that it isn't a module, and can be executed from a button in Opus as an ARexx script rather than an internal command, or even from a shell.
It just serves to illustrate how easy it is to turn a standalone script into an Opus internal command.

```
 1/*
 2$VER: Ports.dopus5 1.2 (10.8.98)
 3Show all system message ports in an DirectoryOpus requester
 4*/
 5parse arg portname function source dest arguments .
 6address value portname
 7options results

 8if function = 'init' then do
 9   dopus command  "Ports" program "Ports" desc "'Display all system message  ↩
    ports'"
10  exit
11  end

12lf = '0a'x
13port = ''
14names = SHOW(ports,,'^')

15do while names ~= ""
16  parse var names currport "^" names
17  if left(currport,4) = '|WSH' then currport = '<'||substr(currport,2)||'>'
18  port = port||currport||lf
19end
20port = "Message Ports:"||lf||lf||port
21 dopus request  '"'port'" Ok'
22exit
```

Lines:
1-4 These are just the version information for the script and a short description of what it does.

5 Opus will send these five arguments to any ARexx module:
portname - The Opus ARexx port, eg. DOPUS.1
function - What function has been called, in case the module provides more than one.
source - The handle of the source lister.
dest - The handle of the destination lister.
arguments - Any extra arguments that you have specified.

```
  6-7  Here we address the parsed portname, and tell ARexx to return results.

    8  For any ARexx module to be added to the command list it must
       provide a function 'init'.  This function is initially called
       when the DOpus5:Modules/ directory is scanned and the module
       detected, this function will add your commands to the command
       list.

    9  The  dopus command  is the line that actually adds your command to
       the list.  It breaks down as follows:

         dopus command     - The Opus ARexx command.
         "Ports"           - The command name that will appear in the
                             command list, in quotes.
         program "Ports"   - This tells Opus what program to call in the
                             DOpus5:Modules/ directory when your function
                             is called.  The script named must have the
                             suffix '.dopus5', eg. the above example will
                             have the filename 'Ports.dopus5'.
         desc "'Display all system message ports'"
                           - This is a brief description that will appear
                             next to the command 'Ports' when it is added
                             to the internal command list.

       If you are adding more than one command, you will require the relevant
       number of  dopus command  lines.  When your script is called, you
       would then use 'if...then', or 'select...case' statements to action
       the required function.

       Since this script only adds one function, no other 'if function...then'
       statements are required except the one required to differentiate the
       'init' function.

   10  We exit because the 'init' function is finished once you have
       specified your commands.

12-13  We define a couple of variables: lf    = a linefeed character
                                       ports = empty string

   14  We ask for all the system message ports returned in variable 'names',
       seperated with the character '^'.

15-19  We start looping, grabbing the first port each time, if the port =
       '|WSH', (an idle WShell), we change the output so it will be <WSH_x>,
       (because it looks better :)  Then add the port to the variable 'port'
       together with a linefeed to seperate them.

20-21  Once finished we add a header to the output, and then call the
        dopus request  command to display the results.
```

1.57 Magellan II ARexx Tutorial: Simple ARexx Module #2

This module will allow you to select a files and then set their comment to their version information. It is a bit more complex than the preceding module in that we need to retrieve information from the lister and from

any selected files within that lister.

If you want to use this module on your computer, simply click on this button and it will be copied to your DOpus5:Modules/ directory.

All you need to do to use it is to create a button in your toolbar as below:

```
Command       Ver2Com
```

NOTE: Because this example is simple, there is virtually no error checking incorporated into it.

```
 1  /*
 2  $VER: Ver2Com.dopus5 1.0 (3.8.98) D.Clarke
 3  Copies file version information to comment field of file.
 4  */
 5  parse arg portname function source dest arguments
 6  address value portname
 7  options results

 8  if function = 'init' then do
 9     dopus command "Ver2Com" program "Ver2Com" desc "'Copies file version to  ↩
    comment'" 'source'
10    exit
11    end

12   lister query source display
13  if pos('version',result) = 0 then do
14     lister request  '"Lister must have Version display" OK'
15    exit 5
16    end
17   lister query source path
18  path = strip(result,B,'"')
19   lister query source selfiles  stem files
20  if files.count = 0 | files.count = '' | files.count = 'RESULT' then do
21     lister request  '"No files selected" OK'
22    exit (5)
23   end
24  do i = 0 to files.count - 1
25     lister query source entry  '"'files.i'"' stem fileinfo.
26    info = fileinfo.version||'.'fileinfo.revision' ('fileinfo.verdate')'
27    if info ~= '. ' then do
28       command  wait original comment '"'path||files.i'"' '"'info'"'
29      fileinfo.comment = info
30       lister addstem  source fileinfo.
31      end
32     lister select  source files.i off
33    lister refresh  source
34  end
35  exit
```

Lines:

1-4 These are just the version information for the script and a short description of what it does.

5 Please refer to the explanation of this line given in

Example 7: Simple ARexx Module #1

6-7 Here we address the parsed portname, and tell ARexx to return results.

8 Please refer to the explanation of this line given in
Example 7: Simple ARexx Module #1

9 The dopus command is the line that actually adds your command to the list. It breaks down as follows:

```
dopus command     - The Opus ARexx command.
"Version2Comment" - The command name that will appear in the
                    command list, in quotes.
program "Ver2Com" - This tells Opus what program to call in the
                    DOpus5:Modules/ directory when your function
                    is called.  The script named must have the
                    suffix '.dopus5', eg. the above example will
                    have the filename 'Ver2Com.dopus5'.
desc "'Copies file version to comment'"
                  - This is a short description of what your command
                    does.  It will appear in the command list next
                    to your command.
'source'          - This specifies to Opus that there must be a
                    SOURCE lister present, and for it's handle to be
                    sent to the function.  If no SOURCE lister is
                    available, nothing will happen when this script
                    is called.  Because you have indicated that you
                    only require the 'source' parameter, the
                    destination handle sent to the module will be 0.
```

If you are adding more than one command, you will require the relevant number of dopus command lines. When your script is called, you would then use 'if...then', or 'select...case' statements to action the required function.

Since this script only adds one function, no other 'if...then' statements are required except the one required to differentiate the 'init' function.

10 We exit because the 'init' function is finished once you have specified your commands.

12-16 This particular example requires that the lister display has been set to show a file's version, (see Opus manual on Environment/Lister Display). If it hasn't, you will get a requester over the lister informing you, and the script will then exit with a return code of 5, (WARN).

17-18 We query the lister's path, strip any leading/trailing quotes, and assign the variable 'path' to it.

19-23 We query the number of selected files only, (no version numbers for directories), and instruct Opus to return the result in a compound variable with a stem of 'files'. So you will get the following, assuming that there are 3 selected files.

```
files.count = 3
```

```
files.0     = First filename
files.1     = Second filename
files.2     = Third filename
```

If files.count equals zero, an empty string, or the word 'RESULT', we open a requester saying no files are selected and exit with a return code of 5, (WARN).

24 We start a 'do' loop that will increment from 0, (first filename), to files.count - 1, (last filename).

25 Query the lister for information on the first filename, putting the results in to a compound variable with the stem of 'fileinfo'.

The results of this will look like the following:

```
fileinfo.NAME      = filename
fileinfo.SIZE      = size in bytes
fileinfo.TYPE      = (<0 = file, >0 = dir)
fileinfo.SELECTED  = 0 or 1
fileinfo.DATE      = seconds since 1/1/78
fileinfo.PROTECT   = protection bits (hsparwed)
fileinfo.DATSTRING = datestamp in ASCII
fileinfo.COMMENT   = the filecomment (if any)
fileinfo.FILETYPE  = file type (if any)
fileinfo.VERSION   = version number
fileinfo.REVISION  = revision number
fileinfo.VERDATE   = version date (numerical dd.mm.yy format)
fileinfo.DATENUM   = file date in numerical dd.mm.yy format
fileinfo.TIME      = file time in hh:mm:ss 24 hour format
fileinfo.DISPLAY   = any user defined display information
```

More information on this command can be found here .

26 We take the relevant file information and format it into a variable 'info'.

27 If 'info' equals '. ', ie. no version information available, we skip the next statements, unselect the file, (lines 32-33), and loop to the next filename.

28 Here we use the command command to call the 'original' Opus Comment command, (just in case someone has replaced the original with their own version), sending the full path and filename, and our reformatted version information, 'info'. It will 'wait' for the command to return before allowing the script to continue.

29-30 Now that we have provided the file with a new filecomment, we need to update the lister display with the new information. We make the compound variable 'fileinfo.comment' equal our new filecomment, (info), then using the command 'lister addstem' we add the updated entry back into our lister.

32-33 Finally, we unselect the file, then refresh the lister display so that it displays the file unselected and with our new filecomment information.

```
   34  Loop back to the start of the 'do' loop.

   35  Exit when we've done them all.
```

1.58 Magellan II ARexx Tutorial: Multi-Command ARexx Module

A module doesn't have to do anything in itself, in this example we are just using it as a convenient way to add scripts/commands to the popup menu for certain filetypes.

This script is taken straight from my DOpus5:Modules/ directory, so a bit of explaining about my filetypes setup might be necessary first, before you can see how it can work.

I have filetypes for many different archives, for example: LhA, LZX, DMS, GZIP, HPACK, TAR, ARJ, RAR, OWS, ZIP, ARC...
For the archives I just mentioned, most people would have the filetype IDs as: LHA, LZX, DMS, GZIP, HPACK, TAR, ARJ, RAR, OWS, ZIP, ARC...
The way my filetype IDs for the above are set up is: aLHA, aLZX, aDMS, aGZP, aHPK, aTAR, aARJ, aRAR, aOWS, aZIP, aARC... That is, a lower case 'a' followed by three letters signifying the archive type.

You might think this is a bit strange, but where it is extremely useful is in the adding of menuitems and similar. For example, for all of my archive filetypes I have 'User4' defined as the 'Test archive' function. So that whenever I call 'User4' on any type of archive, I know that will test it with the appropriate archiver.

If you wanted to add a 'Test' menuitem to the popup menu for each type of archive, you would have to go into the Filetype Editor for each one and add it to the Icon popup menu.

Or, if you were going to do it as per below, you would need to change this line:

```
dopus command "Test" program "General" 'source' ext 'Test...' type a??? private
```

to this:

```
dopus command "Test" program "General" 'source' ext 'Test...' type LHA type LZX ↩
   type DMS private
```

adding in every archiver ID you have, as you can see you will end up with a very long line.

I, on the other hand, would just include a line like you can see below:

```
dopus command "Test" program "General" 'source' ext 'Test...' type a??? private
```

Using the wildcard feature for specifying a 'type', I have just added a 'Test' menuitem to every archive filetype.

Much easier ;-)

I also use the same system for pictures, (p????), and documents, (d????).

On with the example:

```
 1  /*
 2  $VER: General.dopus5 1.0 (10.07.96)
 3
 4  This module adds the following things:
 5
 6  Adds a 'Browse...' command to the popup menu for LhA and LZX archives.
 7  Adds a 'GetDir...' command to the popup menu for LhA and LZX archives.
 8  Adds an 'UnDMS...' command to the popup menu for DMS archives.
 9  Adds a 'Test...' command to the popup menu for ALL archives.
10  Adds a 'Container' command to the popup menu for left-out files/dirs.
11
12  Requirements:
13
14  Needs ArcDir.dopus5 and UnDMS.dopus5 installed, both by Edmund Vermeulen.
15  */
16  parse arg portname function srce dest filename .
17  address value portname
18  options results
19
20  /* Initialise */
21  if function = 'init' then
22    do
23      dopus command "Browse" program "General" 'source' ext 'Browse...' type  ↩
    aLHA type aLZX private
24      dopus command "GetDir" program "General" 'source' ext 'GetDir...' type  ↩
    aLHA type aLZX private
25      dopus command "UnDMS" program "General" 'source' ext 'UnDMS...' type aDMS  ↩
    private
26      dopus command "Test" program "General" 'source' ext 'Test...' type a???  ↩
    private
27      dopus command "Container" program "General" ext "'Open Container'" type  ↩
    leftout private
28      exit
29    end
30  if function = 'Browse' then dopus rx 'DOpus5:ARexx/ArcDir.dopus5 Browse'  ↩
    portname filename srce
31  if function = 'GetDir' then dopus rx 'DOpus5:ARexx/ArcDir.dopus5 GetDir'  ↩
    portname filename srce
32  if function = 'UnDMS' then dopus rx 'DOpus5:ARexx/UnDMS.dopus5' filename  ↩
    portname
33  if function = 'Test' then command User4 filename
34  if function = 'Container' then command scandir new container filename
35  exit
```

What it all means:

Lines:

1-15 Mandatory ARexx comment with version information, what it does and what it requires.

16 As shown in the previous module examples, Opus will pass along these five parameters to the module: Opus ARexx port, the function, the SOURCE handle, the DESTINATION handle and any arguments, (in this case the name of the file whose popup menu we used).

```
  17-18  We address the passed portname, (usually DOPUS.1), and enable
         results.
  21     As shown in the previous examples, when Opus first calls the module,
         it will call it with a function of 'init' to initialise the commands
         that you're adding.
  23     This 'dopus command' line adds the command 'Browse', the program
         that will be called is 'General.dopus5', (that is, the one above).
         We will require the SOURCE handle, the text that will appear in the
         menus is 'Browse...' and we only want it to appear for files with a
         filetype ID of 'aLHA' and 'aLZX', (yours are probably 'LHA' and 'LZX'
         unless you've changed them like I have).  The 'private' flag
         specifies that this command will not appear in the internal command
         list.
  24     Same as for line 23 except here we add the command 'GetDir'.
  25     In this line we add the command 'UnDMS' but only for a filetype ID
         of 'aDMS', (a DMS archive).
  26     This line is where we add the 'Test' command to all filetype IDs
         that match the pattern 'a???', on my system this is all archive
         filetypes.
  27     This adds the 'Container' command to all left-out files/dirs on the
         Desktop.
  28     We exit because the 'init' function is finished.

  The next time this module will be called, will be when the user has chosen
a menuitem from a file/dir popup menu that corresponds to one which we have
added, therefore:

  30     'Browse...' was selected from the menu over a LhA or LZX archive,
         this line will call Edmund's ArcDir.dopus5 script passing along the
         command 'BROWSE', the Opus port name, the filename and the handle of
         the lister.
  31     'GetDir...' was selected from the menu over a LhA or LZX archive,
         this line will call Edmund's ArcDir.dopus5 script passing along the
         command 'GETDIR', the Opus port name, the filename and the handle of
         the lister.
  32     'UnDMS...' was selected from the menu over a DMS archive, this line
         will call Edmund's UnDMS.dopus5 script passing along the filename
         and the Opus port name.
  33     'Test...' was selected from the menu over any archive, this line
         will call the 'User4' function for that filename, (the 'Test
         archive' function in my filetypes).
  34     'Open Container' was selected from a left-out popup menu, this will
         call the Opus ScanDir command.  The 'file new container' parameters
         specify that: a new lister will open in that files home directory
         with that file selected.
  35     We exit.
```

1.59 Magellan II ARexx Tutorial: A Simple Custom Handler for a Lister

To my mind the difference between a Custom Handler and a Module could be given as:

A Custom Handler allows you to 'capture events' from a lister or an AppIcon and act upon them.

A Module is a way of adding things to the system. As Helmut has said in his 'C' tutorial: 'A module is just a library of Opus routines'.

You can have a module that includes a custom handler, Leo Davidson's PrintAppIcon script is a demonstration of this, but each is a seperate entity within the script.

An event happens every time you select something from a toolbar, popup menu, do a Drag'n'Drop, and a few other things.

Custom handlers usually open their own lister or AppIcon, and then they wait for you to do something, or they can be attached to an already open lister.

Edmund Vermeulen's ArcDir script, (available from Aminet), is an excellent example of what can be done using custom handlers.

Below is a really simple example of a custom handler for a lister, in fact it doesn't do anything useful at all, you can just select files and use Opus commands, nothing will happen.
Every action you perform on the lister will be reported in a console window.

```
 1/*
 2 $VER: SLCH.dopus5 1.0 (26.9.98)
 3Simple Lister custom handler
 4*/
 5signal on error
 6signal on syntax
 7signal on halt
 8signal on break_c
 9
10address DOPUS.1
11options results
12options failat 11
13
14dopus front
15
16if ~show('l','rexxsupport.library') then
17  call addlib('rexxsupport.library',,-30)
18
19call open('window','CON:0/12/640/240/SLCH.dopus5')
20
21lister new
22handle = result
23lister read handle 'SYS:' force
24lister wait handle
25call openport('SLCH-handler')
26lister set handle handler 'SLCH-handler' subdrop quotes
27
28/* We are going to trap everything */
29dopus addtrap '*' 'SLCH-handler'
30
31do while event ~= 'inactive'
32  if waitpkt('SLCH-handler') then do
33    packet = getpkt('SLCH-handler')
```

```
34    if packet ~= '00000000'x then do
35      event = getarg(packet,0)
36      handle = getarg(packet,1)
37      name = getarg(packet,2)
38      user = getarg(packet,3)
39      pathstr = getarg(packet,4)
40      arguments = getarg(packet,5)
41      qualifier = getarg(packet,6)
42      deststr = getarg(packet,7)
43
44      call writeln('window','--------------------')
45      call writeln('window','Arg0 (event)       =' event)
46      call writeln('window','Arg1 (handle)      =' handle)
47      call writeln('window','Arg2 (name)        =' name)
48      call writeln('window','Arg3 (user)        =' user)
49      call writeln('window','Arg4 (path)        =' pathstr)
50      call writeln('window','Arg5 (arguments)   =' arguments)
51      call writeln('window','Arg6 (qualifier)   =' qualifier)
52      call writeln('window','Arg7 (destination) =' deststr)
53      end
54    call reply(packet,0)
55    end
56end
57
58/* remove all traps for my handler */
59error:
60syntax:
61halt:
62break_c:
63dopus remtrap '*' 'SLCH-handler'
64call closeport('SLCH-handler')
65if rc = 0 then call delay(100)
66call close('window')
67if rc ~= 0 then do
68  text = 'Error: 'rc', 'errortext(rc)' in line 'sigl'.'
69  dopus request '"'text'" OK'
70  end
71exit
```

Line:
 1 - 4 The obligatory ARexx comment.
 5 - 8 If anything goes wrong with the script we'll jump to the error
        handling routine down at line 59.
10 - 12 Address the Opus ARexx port, enable ARexx results and set the failat
        limit at 11.
14   Move the Opus screen/window to the front.
16 - 17 If the rexxsupport.library isn't loaded then we load it, we need the
        openport, closeport and delay commands it contains.
19   Open our output window.
21 - 24 Open the lister, assign handle to it's handle, read in the SYS:
        directory and wait until it's finished.
25   Call the openport command to open a system message port called
        SLCH-handler.

  NOTE: Message port names are case-sensitive, you will now need to refer to
        this message port exactly as typed.

```
26      Set the handler to the lister using the  lister set handler  command.
29      We are going to trap every event so we use the '*' to signify this.
        Otherwise we could trap specific events by specifying them, for
        example:  dopus addtrap 'Copy' 'SLCH-handler'
31      We enter a DO loop and will only exit when the event equals
        'inactive', which means the lister has been closed.
32      This is where the script normally 'parks' while waiting for something
        to happen, as soon as it recieves a message packet on our port this
        statement will become true and the script will continue.
33      We get the message packet in variable packet.
34 - 52 If it isn't a null packet we get the various arguments from it
        assigning them to variables.  We then write them to our output
        console.

54      We reply to the message packet,  THIS IS IMPORTANT YOU MUST DO IT!!

59 - 62 If we happen to have recieved some kind of error then we would have
        jumped to one of these labels, otherwise when the lister is closed
        we'll exit the DO loop and progress through here anyway.
63      We remove all the traps that we placed upon this handler.
64      Call the closeport command to close our message port.
65      If we exit normally then we'll delay for two seconds.
66      Close our output console.
67 - 70 If we didn't exit normally, (error condition), then we format the
        error information into a text string and display it in a requester.
71      Exit.
```

1.60 Magellan II ARexx Tutorial: A Simple Custom Handler for an AppIcon

The following example of an AppIcon custom handler was the simplest that I could think of, (Sorry :) that might be semi-useful. What it does is put an AppIcon on the Opus Desktop, any LhA archive that you drop onto this AppIcon will be extracted to a directory of your choice.

NOTE: This script relies on the fact that your filetype ID for LhA archives will be 'LHA'. If it isn't, then you will need to change the script to suit your ID.

```
 1/*
 2$VER$: LhAAppIcon.dopus5 1.0 (1.9.98)
 3Call as:  LhAAppIcon.dopus5 {Qp}
 4*/
 5options results
 6parse arg port .
 7
 8lf = '0a'x
 9LhAID = 'LHA'                                  /* Change this line to suit your  ↩
    LhA filetype ID */
10iconname = 'DOpus5:Icons/filetypes/Archive'  /* Change to use a different icon  ↩
   */
11output = 'RAM:'
12if port = '' then port = 'DOPUS.1'
13address value port
14
```

```
15If ~Show('L','rexxsupport.library') then call addlib('rexxsupport.library ↩
   ',0,-30,0)
16
17call openport('LhAAppIcon-handler')
18appmenu.count = 1
19appmenu.0 = 'Config'
20dopus addappicon 'LhAAppIcon-handler' "'LhAAppIcon'" 1 icon iconname info quotes ↩
    close local menu appmenu.
21iconhandle = result
22flag = 0
23do while flag = 0
24  if waitpkt('LhAAppIcon-handler') then do
25    packet = getpkt('LhAAppIcon-handler')
26    arg0 = getarg(packet,0)
27    arg1 = getarg(packet,1)
28    arg2 = getarg(packet,2)
29    arg3 = getarg(packet,3)
30    arg4 = getarg(packet,4)
31    call reply(packet,0)
32    if arg0 = 'menu' & arg2 = 0 then call Config
33    if arg0 = 'info' then do
34      text = 'LhAAppIcon 1.0 (1.9.98)'||lf||lf||'G''day :)'
35      dopus request '"'text'" OK'
36      end
37    if arg0 = 'dropfrom' then call WeGotOne
38    if arg0 = 'close' then flag = 1
39    if arg0 = 'removed' then flag = 2
40    end
41end
42if flag~=2 then dopus remappicon iconhandle
43call closeport('LhAAppIcon-handler')
44exit
45
46Config:
47valid = 0
48do while valid = 0
49  text = '"Please enter a path to extract files to:" 30 "'output||'" OK|Cancel'
50  dopus getstring text
51  if dopusrc = 0 | result = '' then result = 'RAM:'
52  if pos(right(strip(result,,'"'),1),':/') > 0 then
53    if exists(result) then do
54      valid = 1
55      output = result
56      end
57end
58return
59
60WeGotOne:
61if arg2 = '' then return
62lister query arg3 path
63path = result
64dopus setappicon iconhandle busy on
65files = arg2
66do while files ~= ''
67  parse var files'"'file'"' files
68  file = strip(path,,'"')||strip(file,,'"')
69  dopus getfiletype '"'file'"' ID
```

```
70  if result = LhAID then
71    address command 'LhA >NIL: x "'file'" 'output
72end
73dopus setappicon iconhandle busy off
74return
```

Line:

1 - 4 The obligatory ARexx comment containing such useful information as the script version and how to call it.

5 Enable ARexx results.

6 Parse the arguments for the Opus port name.

8 Set lf to the linefeed character.

9 Set LhAID to the filetype ID for LhA archives.

10 Set iconname to the path and filename of the icon that will appear on the Desktop.

11 Set output to the default output path of RAM:

12 If they forgot to give us a port then we set it to the default port 'DOPUS.1'.

13 Address the Opus ARexx port.

15 If the rexxsupport.library isn't loaded, load it.

17 We open the message port for our handler, it's name will be 'LhAAppIcon-handler'.

18 - 19 We set up the stem variable that is going to be used for the AppIcon popup menu. We set appmenu.count to the number of menuitems we will have, in this case 1. Stem variables start counting from 0, so the menuitem will be appmenu.0 which we have set to 'Config'.

20 The dopus addappicon command adds the AppIcon to the Desktop and assigns the handler to it. It breaks down as follows:

dopus addappicon	- The Opus ARexx command.
'LhAAppIcon-handler'	- The name of the message port that we opened earlier, any events that occur to the AppIcon will send messages to this port.
"'LhAAppIcon'"	- This is the label that will appear under the icon.
1	- This is an ID for the icon, it will be returned in messages sent to the handler.
icon iconname	- This is telling Opus to use the icon imagery that we set earlier in variable iconname, line 10.
info	- The popup menu Information item will be enabled.
quotes	- All file arguments will be enclosed in quotes.
close	- Display Close in the popup menu instead of the normal Open.
local	- The AppIcon will only appear on the Opus Desktop and not the normal Workbench screen.
menu appmenu.	- This tells Opus to add the menuitems found in the stem variable appmenu., which we specified earlier, to the AppIcon popup menu.

21 A handle will be returned, (much the same as opening a lister), which we assign iconhandle to.

22 - 23 We set flag to 0, and enter a DO loop in which we cycle around until flag doesn't equal 0.

24 This is where the script normally 'parks' while waiting for a message packet. When we recieve a packet, this if...then statement will

```
        become true and the script will continue.
25 - 30 We get the message packet in a variable called packet and then get
        the various arguments from it.

        Arg0 - <event>              string identifying the event
        Arg1 - <ID>                 ID specified in the addappicon command
        Arg2 - <data>               filenames/menu ID/other information
        Arg3 - <handle>             source lister handle - if applicable
        Arg4 - "icon"               string identifying this as an "icon" event

31      Reply to the message packet, YOU MUST DO THIS!!!
32      If the event was 'menu' and the menuitem was 0 then call the Config
        routine.
33 - 36 It was an 'info' event so we set text equal to some text and display
        it in a requester.
37      Someone dropped something on our icon so we call the WeGotOne
        routine.
38      The user picked 'Close' from the popup menu, so we set flag to 1.
        The next time around the DO loop we will exit.
39      The event 'removed' means that Opus has quit, so we set flag to 2,
        the next time around the DO loop we will exit.
40      End of the 'if waitpkt....'.
41      End of the 'do while....'.
42      We've exited the DO loop, either by quitting Opus or Closing our
        AppIcon.  If it was via the 'Close' menuitem then we issue a
         dopus remappicon  command to remove our AppIcon from the Desktop.
43 - 44 Close our message port and exit the script.
46      Sub-routine Config label.
47 - 48 Set valid to 0 and enter a DO loop until valid doesn't equal 0.
49 - 50 Set text to our display text and display it in a string requester.
51      If the user hit Cancel or entered no text then set it to the default
        directory of RAM:.
52      Strip the quotes off the front and back of the string and see if it's
        rightmost character is a ':' or a '/'.
53 - 56 If it was we check to see whether the path actually exists and if it
        does set valid to 1 and output to our new directory.
57      End of the 'do while valid....' loop, we'll keep going around until
        we get a valid directory from the user.
58      Return to the main handler loop.
60      Sub-routine WeGotOne label.
61      If Arg2 was empty, (no files), then we return immediately.
62 - 63 We ask for the path of the lister handle returned in Arg3, and set
        path to it.
64      Set the AppIcon's state to busy, so it will become ghosted on the
        Desktop and will ignore any more files dropped on it.
65      Set files to the list of files in Arg2.
66      Enter a DO loop while files is not empty.
67      Parse the first filename from files returning the rest of the string
        back to files, for example:

                files = "foo.lha" "bar.lha" "why.lha"
        After:  parse var files'"'file'"' files
                file = "foo.lha"
                files = "bar.lha" "why.lha"

68      Set file to the path and the filename without the leading and
        trailing quotes.
```

```
69      Get the filetype ID of the file.
70 - 71 If the ID equals the ID for LhA archives that we set back in line 9,
        then call the LhA archiver to extract the filename to the path set
        in variable output.
72      Loop around to the next file.
73      All files are done, set the AppIcon state back to idle.
74      Return to the main handler loop.
```

As you can see, it isn't very hard. You could expand on this by adding support for different archivers, saving the output directory in a config file, etc.

To add a different archiver is very easy, for example let's add LZX archives:

Find this line:

```
LhAID = 'LHA'
```

and add this after it:

```
LZXID = 'LZX'                /* or whatever your filetype ID for LZX is */
```

then find this line:

```
address command 'LhA >NIL: x "'file'" 'output
```

and add these two after it:

```
if result = LZXID then
  address command 'LZX >NIL: x "'file'" -d 'output
```

Done!

1.61 Magellan II ARexx Tutorial: Improving the inbuilt commands

One thing that may not become immediately obvious is the fact that all the internal Opus commands can be replaced by your own versions of them.

You can do this by writing your own module. Using the Opus SDK, (Software Developers Kit), you can write it in a language such as 'C', 'E', Assembler, or BASIC. You can also do it by using ARexx.

It is just the same as writing any normal ARexx module, you just specify your command name to be the same as one of the Opus internal ones.

Naming your commands the same as Opus ones, causes them to override the internal commands.

For example, you could improve the operation of the Copy command so that when it tried to copy over a file with the same name, it could prompt you with the option to rename the file, (instead of just the normal options of Skip and Replace), then proceed with the copy as normal.

The example below was initially a script in the DOpus5:ARexx/ directory, written by Sylvain Bourcier, <booze@videotron.ca>, I rewrote it so that it was a module replacing the internal versions of the Root and Parent command with slightly more functional ones.

If you want to use this module on your computer, simply click on this button and it will be copied to your DOpus5:Modules/ directory.

Since it replaces existing Opus commands, there's no need to make any special buttons. The next time you use Root or Parent in the root directory of a device, you will be presented with the device list.

```
/*
$VER: RootParent.dopus5 1.2 (12.8.98)
Replaces internal Root and Parent commands with versions that will display
the Device List if in the root of a device.

USAGE: Copy it to DOpus5:Modules/ It will replace the existing
       Root and Parent command functions.
 */
options results
parse arg port func srce dest args .
address value port

if func = 'init' then do
  dopus command "Root" program "RootParent" desc "Root/Devices" template "UNSELECT ↩
     /S"
  dopus command "Parent" program "RootParent" desc "Parent/Devices" template " ↩
     UNSELECT/S"
  exit
end

lister query srce path
path = result
if upper(args) = 'UNSELECT' then command source srce wait none
if right(path,1) = ':' then command source srce wait devicelist full
    else command source srce original func
exit
```

Here's another simple one for those people still using the Trashcan. An internal DeleteTC command will be added, using this instead of Delete will cause the selected entries to be moved to SYS:Trashcan/.
And we also add an EmptyTC command which you can put into the User Menus as 'Empty Trashcan', just like the good ol' days ;-)

```
/*
$VER: Delete.dopus5 1.0 (8.9.98)
When you hit Delete, will copy of selected entries to the Trashcan before
deleting them.
*/
options results
parse arg port func srce dest args .
address value port

if func = 'init' then do
```

```
  dopus command "DeleteTC" program "Delete" desc "Move selected to Trashcan"
  dopus command "EmptyTC" program "Delete" desc "Empty the trash"
  exit
  end

if func = 'DeleteTC' then command source srce original move 'TO SYS:Trashcan'
if func = 'EmptyTC' then command original delete 'SYS:Trashcan/#? QUIET'
exit
```

1.62 Magellan II ARexx Tutorial: Cloning source listers

Here's a script written by Gary Gagnon way back in 1995, (I can't even remember back that far). It clones any listers that are currently in state SOURCE.

So if you have a Source lister with path 'RAM:' and one with path 'HD0:', after running this script you'll end up with two listers open for each of those paths.

```
/*
 $VER:       CloneSRCE.dopus5 1.0 Jun-01-95
 Written by: Gary Gagnon (garyg@wimsey.com)
 Purpose:    ARexx script to clone SRCE lister(s)
 Syntax:     From function editor "ARexx DOpus5:ARexx/Same.dopus5 {Qp}"
 Note:       I keep my DOpus5 ARexx scripts in DOpus5:ARexx
 Note:       If no listers flagged as SRCE, nothing is opened
*/

options results
parse arg portname
address value portname

'lister query source'
srclist = result

if result = 'RESULT' then
  exit 0

do forever
  if srclist = '' then leave

  /* the following command line does a little ARexx magic */
  /* try this with any other language - it won't be as simple! */

  parse var srclist curlist srclist

  /*
    if you couldn't figure it out,  it extracts the first value from
    the string and moves the remaining values (if any) back to the
    same string.  This is what allows this to duplicate all SRCE
    listers within this simple loop (and also makes sure it won't be
    an endless loop). Languages without type checks can be nice!
  */

  'lister new'
  newlist = result
```

```
  'lister copy' curlist newlist
end
```

  Let's modify it a bit :)

```
/*
 $VER: Clone.dopus5 1.0 (23.9.98)

 Original by: Gary Gagnon (garyg@wimsey.com)
 Purpose:    ARexx script to clone SRCE/DEST lister(s)
 Syntax:     From function editor "ARexx DOpus5:ARexx/Same.dopus5 {Qp} <SOURCE| ↩
    DEST>
*/

options results
parse arg portname state .
address value portname
if state = '' then exit
if pos(upper(state),'SOURCE DEST') = 0 then exit

'lister query 'state
srclist = result

if result = 'RESULT' then
  exit 0

do forever
  if srclist = '' then leave

  /* the following command line does a little ARexx magic */
  /* try this with any other language - it won't be as simple! */

  parse var srclist curlist srclist

  /*
    if you couldn't figure it out,  it extracts the first value from
    the string and moves the remaining values (if any) back to the
    same string.  This is what allows this to duplicate all SRCE
    listers within this simple loop (and also makes sure it won't be
    an endless loop). Languages without type checks can be nice!
  */

  'lister new'
  newlist = result
  'lister copy' curlist newlist
end
```

  Now it will clone Source or Destiantion listers depending on whether you call it with the SOURCE or DEST parameter.

1.63 Magellan II ARexx Tutorial: Finding duplicated files in two listers

Here's another script of Edmund's, (workaholic isn't he), it compares the names of files/dirs in two listers and leaves the duplicated ones selected.

```
/*
  $VER: DupeCheck.dopus5 1.1 (7.5.96)
  Written by Edmund Vermeulen (edmundv@grafix.xs4all.nl).

  ARexx script for Directory Opus 5 to select all entries in the source
  lister that already exist in the destination lister (a.k.a. dupes).

  Function : ARexx      DOpus5:ARexx/DupeCheck.dopus5 {Qp}
*/

parse arg portname
if portname = '' then
  portname = 'DOPUS.1'
address value portname

options results
options failat 21

lister query source
if rc > 0 then do
  dopus request '"No source selected." OK'
  exit
  end
parse var result srchandle .

lister query dest
if rc > 0 then do
  dopus request '"No destination selected." OK'
  exit
  end
parse var result desthandle .

lister set srchandle busy on
lister set desthandle busy on

lister query desthandle entries stem entry.
do i = 0 to entry.count - 1
  lister select srchandle '"'entry.i'"' on
  end

lister refresh srchandle
lister set srchandle busy off
lister set desthandle busy off
exit
```

There's only four lines that do all the work, all the rest just make sure there's a source and destination lister and sets their state to busy while we're working.

```
lister query desthandle entries stem entry.
```

```
do i = 0 to entry.count - 1
  lister select srchandle '"'entry.i'"' on
  end
```

  All we do is get a list of all entries in a stem variable and then loop around selecting them in the source lister, if the name exists it will be selected. Simple!

1.64 Magellan II ARexx Tutorial: Adding a bit of Win95

  It's a sorry day when we copy a MicroSoft idea, but here we go.

```
/*
$VER: CutNPaste.dopus5 1.1 (30.10.97)
*/

/* Cut & Paste module - Adds "Cut" and "Copy" */
/* to file pop-up menus and "Paste" to lister */
/* pop-up menus.  Nothing is done until you   */
/* Paste when the appropriate Copy or Move    */
/* command is issued.  Andrew Dunbar Oct 1997 */

parse arg portname function source dest arguments
address value portname
options results

if function = 'init' then do
  cnptypes = 'type drawer type tool type project type trash'
  dopus command 'Clip-Copy' program 'CutNPaste' ext 'Copy' private cnptypes
  dopus command 'Clip-Cut' program 'CutNPaste' ext 'Cut' private cnptypes
  dopus command 'Clip-Paste' program 'CutNPaste' ext 'Paste' private type lister
end
else if function = 'Clip-Cut' then address command 'echo >T:clipcnp cut'arguments
else if function = 'Clip-Copy' then address command 'echo >T:clipcnp copy' ↩
   arguments
else if function = 'Clip-Paste' then do
  lister query source path
  pastepath = result
  if open('cnpfile','T:clipcnp','r') then do
    cnpinfo = readln('cnpfile')
    call close('cnpfile')
    parse var cnpinfo cmd srcfile
    srcfile = trim(srcfile)
    if srcfile ~= '' then do
      if cmd = 'copy' then command copy 'name='srcfile 'to='pastepath
        else if cmd = 'cut' then command move 'name='srcfile 'to='pastepath
      lister wait source
      lister read source pastepath force
    end
    else dopus request '''Paste: Clipboard invalid'''
  end
  else dopus request '''Paste: Nothing in clipboard'''
end
exit
```

After you put this into the DOpus5:Modules/ directory, the popup menus on files of the type drawer, tool, project and trach will have two menu items called 'Copy' and 'Cut'. Lister will inherit a popup menu item called 'Paste'.

When you select either Cut or Copy from a file's popup menu it's name will be echoed to a file in T: along with the function, cut or copy.

When you select Paste from a lister's popup menu, this file will be opened if it exists, (an error requester iss displayed if it doesn't), and it's contents read. If the action was to Copy then the file is copied to the lister, if it was Cut then the file will be moved to the lister.

1.65 Magellan II ARexx Tutorial: Adding a directory tree function

Here's a short script I knocked up after being inspired by someone else, you know who you are ;-)

If any of you out there had Directory Opus V4, then you'd know you were able to display a 'tree' of all the directories for a particular drive. You could click on one to have that directory displayed.

While not as asthetically pleasing as the version Jon incorporated into DO4, this version does the same basic thing, that is, it displays a tree of directories for a particular device and allows you to go to one by double clicking on it.

```
/*
$VER: DirTree.dopus5 0.2 (7.10.98)

Call as:  ARexx  DOpus5:ARexx/DirTree.dopus5 [REGEN] <Device>

          Flags  Run Async

 where:  REGEN  = causes a new directory tree file to be created.
          Device = the device, eg. HD0:, SD0:, DF0:, etc

Example:  DirTree.dopus5 HD0:

*/
options results
parse arg regen device
if device = '' then device = regen
regen = regen ~= device

address 'DOPUS.1'
if ~show('l','rexxsupport.library') then
  call addlib('rexxsupport.library',0,-30)

device = strip(strip(device,,'"'))
oldcd = pragma('d',device)
device = pragma('d',oldcd)
if right(device,1) ~= ':' then do
  dopus request '"Specify DEVICE only, eg. ''HD0:''" OK'
  exit
  end
tfile = left(device,pos(':',device))||'.dirtree'
```

```
31if regen then call delete(tfile)
32
33if ~exists(tfile) then do
34  address command 'List 'device' DIRS ALL LFORMAT "%P%N" >T:dirtree.temp'
35  address command 'Sort T:dirtree.temp 'tfile
36  'command protect 'tfile' set H'
37  call delete('T:dirtree.temp')
38  end
39
40if ~open('dirtree',tfile,'r') then do
41  dopus request '"ERROR: Unable to open ''tree'' file." OK'
42  exit
43  end
44else do
45  totlines = 0
46  do while ~eof('dirtree')
47    call readln('dirtree')
48    totlines = totlines + 1
49  end
50  call seek('dirtree',0,'b')
51  nlength = length(totlines)
52  end
53
54lister new invisible mode name
55handle = result
56lister set handle field off
57lister set handle toolbar
58lister set handle busy on
59info.NAME = copies('0',nlength)
60info.DISPLAY = device
61lister addstem handle info.
62lister refresh handle full
63lister set handle value info.NAME device
64lister set handle visible on
65
66linenum = 1
67do while ~eof('dirtree')
68  dirline = readln('dirtree')
69  path = dirline
70  if pos(':',dirline) ~= 0 then parse var dirline dev ':' dirline
71  if dirline ~= '' then do
72    linenum = linenum + 1
73    if pos('/',dirline) ~= 0 then do
74      howmany = 0
75      do while pos('/',dirline) ~= 0
76        parse var dirline fore'/'dirline
77        howmany = howmany + 1
78      end
79      dirline = copies(' ',3 * howmany)||dirline
80      end
81    dirline = '   '||dirline
82    do i = 1 to length(dirline) by 3
83      if substr(dirline,i + 3,1) ~= ' ' | substr(dirline,i + 4,1) ~= ' ' then ↩
   do
84        dirline = overlay('+--',dirline,i)
85        leave
86        end
```

```
 87    end
 88    info.NAME = right(linenum,nlength,'0')
 89    info.DISPLAY = dirline
 90    lister addstem handle info.
 91    lister refresh handle
 92    lister set handle value info.NAME path
 93    end
 94end
 95call close('dirtree')
 96lister set handle busy off
 97
 98call openport('DirTree-handler')
 99lister set handle handler 'DirTree-handler'
100
101do until event = 'inactive'
102  if waitpkt('DirTree-handler') then do
103    packet = getpkt('DirTree-handler')
104    if packet ~= '00000000'x then do
105      event = getarg(packet,0)
106      name = getarg(packet,2)
107      if event = 'doubleclick' then do
108        lister query handle value name
109        path = result
110        lister set handle handler
111        lister set handle toolbar toolbar
112        lister wait handle quick
113        lister read handle path force
114        end
115    call reply(packet,0)
116    end
117end
118
119call closeport('DirTree-handler')
120exit
```

When run for the first time on a device, there will be a delay while every directory is listed and then sorted. This file is then save to the root of the device for quick access next time, it will be named as '<device>.dirtree', and will have the 'h' protection bit set so as to not be displayed if you have the 'Hidden' option set for your listers.

Some highlights of the script are:

Lines: 54 - 64 We open a new lister initially invisible while we set up it's toolbar and field titles to off. We set it's state to busy, add the first entry (the device), then refresh it and make it visible.

66 - 87 These lines just read the file a line at a time and format them ready for the lister.

88 - 92 We set the compound variable info.NAME to equal the line number in the file, (so each entry is individual), set the reformatted text to the compound variable info.DISPLAY and add them to the lister display using the lister addstem command. Because we have specified a DISPLAY field this will be what is displayed in the lister. Then we associate the NAME field with a value, in this case the path of the directory using the lister set value command.

```
       101 - 117 This is our custom handler routine which just waits for a
                 doubleclick event to happen.  When it does, we query the
                 value of the name/value pair we made previously, this gives
                 us back the path for that entry.  Set the handler to null,
                 this causes the handler for this lister to become inactive
                 and thus terminate.  Set the toolbar to the default toolbar,
                 usually called 'toolbar' in the DOpus5:Buttons directory.
                 Wait until the lister is idle, and then force the lister to
                 read in the requested path.

  All very simple :)

  Clicking this  button  will copy it to your DOpus5:ARexx/ directory.
```

1.66 Magellan II ARexx Tutorial: An Opus v4 CopyWin replacement

```
  Leo's script below, emulates the function of the 'CopyWin' function that
was available in Opus v4.  See the comments at the start of the script for
the usage of it.

  The script checks for existance of a SOURCE and DESTINATION lister,
if it can't find either then it exits with an error message.  If both exist, then
it finds the path of the SOURCE lister and tells the DESTINATION lister to
read it in.

  You could eliminate the steps where it finds the SOURCE path and reads
it into the DESTINATION lister by replacing them with the  lister copy
command.

  So this:

lister query source_handle.0 path
source_path = RESULT
lister read dest_handle.0 '"'||source_path||'"'

  could be replaced by this:

lister copy source_handle.0 dest_handle.0


/* WinCopy for Directory Opus 5.
   by Leo 'Nudel' Davidson for Gods'Gift Utilities
   email: leo.davidson@keble.oxford.ac.uk  www: http://users.ox.ac.uk/~kebl0364

$VER: WinCopy.dopus5 1.4 (26.12.95)

   NOTE: This script _requires_ DOpus v5.11 or above.
   NOTE: DOpusFuncs is an assembler version of this (and other) scripts.
   NOTE: This script does *NOT* do the same thing as Gary Gagnon's
         "CloneSRCE.dopus5" or "same.dopus5", read on...

   This script will copy the source lister to the destination lister,
   like the old CopyWin command in DOpus4.

   If you wish to open a _new_ lister which is a copy of the source
```

```
   one (as apposed to turning an existing lister into a copy of the
   source), you should use Gary Gagnon's "CloneSRCR.dopus5" script,
   which was included in the official DOpus v5.11 update as
   "DOpus5:ARexx/same.dopus5". Gary's script is also able to clone
   multiple source listers at once.

Call as:
------------------------------------------------------------------------------
ARexx   DOpus5:ARexx/WinCopy.dopus5 {Qp}
------------------------------------------------------------------------------
Turn off all switches.

*/

options results
options failat 99
signal on syntax;signal on ioerr                /* Error trapping */
parse arg DOpusPort
DOpusPort = Strip(DOpusPort,"B",'" ')

If DOpusPort="" THEN Do
        Say "Not correctly called from Directory Opus 5!"
        Say "Load this ARexx script into an editor for more info."
        EXIT
        END
If ~Show("P",DOpusPort) Then Do
        Say DOpusPort "is not a valid port."
        EXIT
        End
Address value DOpusPort

lister query source stem source_handle.

IF source_handle.count = 0 | source_handle.count = "SOURCE_HANDLE.COUNT" Then Do
        dopus request '"You must have a SOURCE lister!" OK'
        EXIT
        End

lister query dest stem dest_handle.

IF dest_handle.count = 0 | dest_handle.count = "DEST_HANDLE.COUNT" Then Do
        dopus request '"You must have a DESTINATION lister!" OK'
        EXIT
        End

lister query source_handle.0 path
source_path = RESULT
lister read dest_handle.0 '"'||source_path||'"'

syntax:;ioerr:                          /* In case of error, jump here */
EXIT
```

1.67 Magellan II ARexx Tutorial: An Opus v4 SwapWin replacement

Here are two different versions of the 'SwapWin' function, that was available in Opus v4, presented below. Each one uses a different approach to do the same thing.

Leo's, the top one, simply gets the path from each lister and tells the other lister to read it in. This retains the current lister format, unless the paths read have specific formats.

Edmund takes the approach of just getting each listers position and then sending that position to the other lister, so the listers effectively swap positions on the screen. The only thing is that if your destination lister was setup to show filetype in the format and the source wasn't, when you swap positions the opposite will now be true.

With Leo's script he asks for the handle of the SOURCE and DESTINATION lister, if either one of those don't exist then the script will exit, giving you an error message.

Edmund's, on the other hand, asks for the handles of all listers currently open, if there is less than two the script will end. It then checks to see if there are both a SOURCE and DESTINATION lister, if either or both are not available, then the first two from the list of handles we asked for are used as the SOURCE and DESTINATION respectively. So the only reason it should exit is because you have less than two listers open.

```
---------------------------------------8<---------------------------------------
/* WinSwap for Directory Opus 5.
   by Leo 'Nudel' Davidson for Gods'Gift Utilities
   email: leo.davidson@keble.oxford.ac.uk  www: http://users.ox.ac.uk/~kebl0364

$VER: WinSwap.dopus5 1.4 (26.12.95)

   NOTE: This script _requires_ DOpus v5.11 or above.
   NOTE: DOpusFuncs is an assembler version of this (and other) scripts.

   This script will swap around your source and destination listers,
   like the old swap command in DOpus4.

   Obviously, in DOpus5 it is possible to simply move the two lister
   windows, but somethimes this isn't desireable. For example, when they
   are locked in place via the option in the pull-off menu; when you're
   too lazy to move the windows; or when the two windows are positioned
   such that moving them would be difficult or would make a mess.

Call as:
-----------------------------------------------------------------------------
ARexx   DOpus5:ARexx/WinSwap.dopus5 {Qp}
-----------------------------------------------------------------------------
Turn off all switches.

*/

options results
options failat 99
signal on syntax;signal on ioerr                /* Error trapping */
parse arg DOpusPort
```

```
DOpusPort = Strip(DOpusPort,"B",'" ')

If DOpusPort="" THEN Do
        Say "Not correctly called from Directory Opus 5!"
        Say "Load this ARexx script into an editor for more info."
        EXIT
        END
If ~Show("P",DOpusPort) Then Do
        Say DOpusPort "is not a valid port."
        EXIT
        End
Address value DOpusPort

lister query source stem source_handle.

IF source_handle.count = 0 | source_handle.count = "SOURCE_HANDLE.COUNT" Then Do
        dopus request '"You must have a SOURCE lister!" OK'
        EXIT
        End

lister query dest stem dest_handle.

IF dest_handle.count = 0 | dest_handle.count = "DEST_HANDLE.COUNT" Then Do
        dopus request '"You must have a DESTINATION lister!" OK'
        EXIT
        End

lister query source_handle.0 path
source_path = RESULT
lister query dest_handle.0 path
dest_path = RESULT

lister read source_handle.0 '"'||dest_path||'"'
lister read dest_handle.0 '"'||source_path||'"'

syntax:;ioerr:                          /* In case of error, jump here */
EXIT

---------------------------------------8<---------------------------------------

/*
  $VER: SwapListers.dopus5 1.0 (30.1.96)
  Written by Edmund Vermeulen (edmundv@grafix.xs4all.nl).

  ARexx script for Directory Opus 5 to swap the source and destination
  listers around. If there is no source and/or destination lister it will
  take the next other lister.

  It takes a different (better :-) approach then Leo's WinSwap script. It
  simply swaps the lister positions around, instead of the directory paths.

  Function : ARexx      DOpus5:ARexx/SwapListers.dopus5 {Qp}
*/

parse arg portname .
if portname = '' then  /* in case they forgot */
  portname = 'DOPUS.1'
```

```
address value portname
options results

lister query all stem handle.
if rc > 0 | handle.count < 2 then
  exit

lister query source
if rc = 0 then do
  parse var result srchandle .
  do i = 0 to handle.count - 1
    if handle.i = srchandle then do
      handle.i = handle.0
      handle.0 = srchandle  /* make 0 the source handle */
      leave
      end
    end
  end

lister query dest
if rc = 0 then do
   parse var result desthandle .
   if handle.0 = desthandle then
      handle.0 = handle.1
   handle.1 = desthandle  /* make 1 the destination handle */
   end

do i = 0 to 1
  lister query handle.i position
  pos.i = result
  end

lister set handle.0 position pos.1
lister set handle.1 position pos.0
exit

----------------------------------------8<----------------------------------------
```

1.68 Magellan II ARexx Tutorial: TroubleShooting

OK, you've written some scripts but they don't seem to work, there are a few methods or tools you can use for finding out what's wrong.

The simple things
ARexx error codes
ARexx tracing
The Opus CLI

1.69 Magellan II ARexx Tutorial: TroubleShooting - The simple things

ARexx server

The very first thing you should check is: Is the Arexx server running?

That is, have you at least started RexxMast in some fashion, either through your User-Startup, WBStartup or double-clicking on it's icon?

I know it seems obvious, but you wouldn't believe the problems some people have, (right Ash? :)

If you're not sure, just run RexxMast, if it has already been run it will tell you the server is already running.

Script Format

Possibly the reason is something as simple as you haven't made the first line of the script a comment. If the first line isn't a comment ARexx won't even give you the benefit of a error code, as far as it is concerned, it isn't a script.

No results

The script seems to run, but you don't get the right results, you might have forgotten to put options results near the start of the script to enable the RESULT variable.

Opus ARexx commands fail but look OK

I've found that the Opus ARexx interface is very sensitive to the presence or lack of quotes around commands. Jon would say that this is normal for ARexx, but I don't believe him :)
So if you are having problems with some Opus ARexx commands try putting quotes around all or part of the line in question.

No output, wasn't even executed

Sometimes a script won't execute and you get no indication, it's usually because the interpreter (ARexx server) found a syntax fault with it when it parsed it before execution.

Try executing it from a shell, you'll get all error messages then.

1.70 Magellan II ARexx Tutorial: TroubleShooting - ARexx error codes

ARexx errors

The ARexx interpreter itself is pretty informative with it's error messages when a script fails, usually you will get an error number and the reason.

For example, copy the following to a seperate file called 'Fail.rexx' and then execute it from a shell with rx Fail.rexx.

```
/* Fail.rexx */
say "Hello world!
```

```
exit
```

If you did that, you'll probably get an error message as below:

```
+++ Error 5 in line 2: Unmatched quote
```

This is pretty self-explanatory, there is a quote missing or extra in line 2, in this case line 2 needs an extra quote on the end of the line:

```
say "Hello world!"
```

That's enough to fix this script, and allow it to run.

Sometimes you only get an error number, (although I can't remember the last time I saw just a number), in that case you can just look up what it means in the ARexx documentation that was included with OS2.04+ or ARexx 1.15 if you bought it seperately.

If the error has occured because of an Opus ARexx command, you will get an error code generated for it but no message indicating what went wrong.

```
/* Fail2.rexx */
address 'DOPUS.1'
lister query 123456 path
path = result
exit
```

The above example, if you copy it to a file and execute it, will generate an error code such as the one below:

```
  3 *-* lister query 123456 path;
+++ Command returned 10
```

Not very informative is it? All you know is that it failed with error code 10 at line 3. However, you can look this up in the Opus manual or the Opus ARexx AmigaGuide to find out what it really means, and you would get:

```
invalid lister handle
```

Opus also provides the dopus error ARexx command that you can use within your scripts to provide more information. If you open a shell and type the following:

```
rx "address 'DOPUS.1';options results;dopus error 10;say result"
```

You'll get the same message text as above.

So if we also use the ARexx special variable sigl , (sigl is the line which caused a transfer of control, in this case the line that caused the script to jump to the error routine), in the example above, so that it now looks like:

```
/* Fail2.rexx */
options results
signal on error
address 'DOPUS.1'
lister query 123456 path
```

```
path = result
exit
error:
dopus error rc
say result' in line 'sigl':' sourceline(sigl)
```

  Now when you run it, instead of just the error code you got above, you'll get the message:

```
invalid lister handle in line 5: lister query 123456 path
```

  More informative to you and to any users of your scripts.

  A list of Opus ARexx error codes can be found here .

1.71 Magellan II ARexx Tutorial: TroubleShooting - ARexx tracing

 ARexx tracing

  Tracing is the ARexx version of SnoopDOS. You don't need any special programs to do it, if you have ARexx installed you already have the capability.

  What follows is a very simplified description of tracing, for more information please refer to your ARexx documentation.

  Your ARexx installation included four programs called TCO, TCC, TE and TS, the names stand for:

```
Tracing Console Open
Tracing Console Close
Tracing End
Tracing Start
```

  In a shell type TCO, you'll notice a window open with no text and the title ARexx. This is the Tracing Console output window, any ARexx script that executes will be displayed in here line by line with any results.
  The console isn't enabled until you enter the command TS in your shell.

  Now when you execute an ARexx script, you will see each line as it executes, it's line number and any results. Tracing forces causes the script to stop after each result and present you with a prompt in the console. To have the next line execute, simply press Enter at the prompt.

  For the original Fail2.rexx script in TroubleShooting - ARexx error codes , the console output will look like this:

```
  1 *-* ;
  2 *-* options results;
>+>                                    <-- This is the prompt.
  3 *-* address 'DOPUS.1';
>+>
  4 *-* lister query 123456 path;
      >>> "LISTER QUERY 123456 PATH"
>+>
```

```
  5 *-* path = result;
      >>> "RESULT"
>+>
  6 *-* exit;
```

As you can see, no error is generated for this script, even though you know it isn't a valid lister handle. However, knowing what the script is supposed to be doing, (getting the path of a lister), you know that the path should not equal "RESULT". This should indicate to you that something went wrong when you asked for the path in the lister query statement.

If any major problem occured that caused the script to terminate, then an error code with text will be displayed as well, but only for the normal ARexx commands.

To end tracing simply enter the TE command in a shell, then the TCC command to close the console. The console will not be closed until you have acknowledged any outstanding prompts.

Tracing can also be implemented from within your ARexx script with many different options, please refer to the ARexx documentation.

1.72 Magellan II ARexx Tutorial: TroubleShooting - The OpusCLI

Opus provides an internal command called CLI .

This command provides a shell which generally behaves as does a normal AmigaShell, with a couple of exceptions:

1) You can enter any internal Opus command and it will work providing you've used the correct format, etc.
2) You can enter any Opus ARexx command by prefixing it with a +.

Create a menuitem/button/etc with the following function:

```
Function : Command    CLI
```

That's it! Now use the button or menuitem, a shell will open with the following text in it:

```
Directory Opus 5 Command Line Interpreter v0.04
© Copyright 1997 by Jonathan Potter

Type 'help' for help.
>
```

Let's try an ARexx command, type the following at the prompt:

```
> +dopus error 10
--> Invalid lister handle
```

As you can see, you get an immediate response, try a few more:

```
> +dopus query background desktop
> +lister new
> +lister close all
```

So if you're having trouble with an Opus ARexx command in a script, you can try it out in the Opus CLI.

1.73 Magellan II ARexx Tutorial: Credits

This is where we thank all those people that made it all (im)possible:

First off, I guess, would have to be Jon Potter for creating the enormously, bloate...eeerrr, useful, (yeah, that's what I meant to say :) Directory Opus v5+. While it's not the only file management program available, it is the only one that has the power to do what YOU want, and not what the programmer thinks you want, (although I believe there were times when he thought he knew better than the beta-testers, boy was he wrong).

It IS unrivalled on ANY platform.

Dr Greg Perry, who with firm hand and iron will has guided the unruly band of beta-testers, (they really provided no input into the final product, according to Greg and Jon :) towards the unattainable goal of a bug-free product, (remember kids, inside every Directory Opus program there's a free, that's right, FREE undocumented 'feature' just for you). I believe he also provided some source code.

Who else...oh yeah, how about Andrew Dunbar, (ex. of GPSoftware, now hiding out in Mexico apparently, I've heard there's no extradition from there to Australia). He, who created such monoliths as the 'OpusFTP' and 'Andrew's Filetype Creator', (although it has been said that it really means 'Automatic Filetype Creator', we know better ;) I think he also provided Greg with coffee. (BTW Andrew, if you're watching, I'll expect the first shipment next month ;-)

How about Martin Steigerwald, who first planted the seed of a Opus-Plus CD firmly in everybodies mind, (in fact it was more deeply rooted than Windows95, and just as hard to get rid of), and then when it seemed to be cancelled due to lack of interest, (like we all wish Windows was), was there to witness the rebirth.
And what a birth it was, it came out kicking and screaming, trying desperately to avoid the light of day. Yes that's right, it's Martin's fault that what little spare time I get to myself was subverted into a non-profit venture.

Now that I have that off my chest...:^)

All the users out there, who in support of this great product, should buy two copies. Give one to your mum, she'll love it. (It'll be the most expensive drinks coaster she'll ever recieve.)

I really should mention all those people whose ARexx scripts I pilfered and butchered to my own ends in the composing of this tutorial. My thanks to the following: Edmund Vermeulen, Sylvain Bourcier, Leo Davidson and all those others whose scripts I found somewhere long ago and can't remember.

Last but not least, those poor, unfortunate beta-testers. They worked for a pittance, braved ferocious HD-eating bugs, were beaten unmercifully,

tirelessly worked through the night, submitted idea after idea after idea.... only to have them all thwarted at the very last instant but those nasty, overbearing programmers.
  Nonetheless, they have managed to win minor battles, skirmishes, all in the name of those many users who would have otherwise had no collective voice with which to speak. A few of us fell along the wayside, unable to keep up with the demands being placed upon their pitiful resources, we remember those with fond...er...memories.
  People the likes of Leo, Edmund, Peter (a couple of them), Dave (couple of them too, I think, could be three by the time I finish this), and quite a few others, who now only exist as a name in the Directory Opus About credits, (by the way chaps, there's a rumour going around that if you don't send in your money this month you'll be removed from the credits. Leo, because you've also got a picture in there, the price has doubled :)
  I never realised until I sat and watched the scrollies in the About just how long I've been testing this program, geezz I can remember most of these guys, looks like you and me, Trevor, are the 'old guys'.
  Hats off to you chaps, you deserve everything you get.

  Oh yeah, I guess I should mention Commodore in here somewhere....I think that will do.

  I think that about sums it up...'Sorry, what was that Greg?' 'Oh right, I've just been told there's a vacancy for a beta-tester...'

1.74 Magellan II ARexx Tutorial: Dopus User Position

 Gullible aren't you :^)

1.75 Magellan II ARexx: Results from commands.

Results from commands.

If a command returns a value or information, the data will generally be returned in the RESULT variable. The only exceptions to this are the dopus getdesktop , dopus getstring and lister getstring commands which return information in the special DOPUSRC variable.

 Error codes are returned in the RC variable.

You must include the line "OPTIONS RESULTS" near the top of your script to enable the RESULT variable. See an ARexx manual for more information.

1.76 Magellan II ARexx Tutorial: Error Codes

ARexx Error Codes

Lister handles are the actual address in memory of the lister structure. Opus 5 will reject any non-valid handles with an RC of 10.

All commands that return data return it in RESULT (with the exception of dopus getstring , dopus getdesktop and lister getstring) or a specified (stem) variable; if an error occurs, the error code is returned in RC .

An RC of 0 generally indicates that everything is ok.

Error codes are:-

1 RXERR_FILE_REJECTED
The file you tried to add was rejected by the current lister filters.
Note that this is not an error, just a warning. The file is still added, it will just not be visible until the filters are changed.

5 RXERR_INVALID_QUERY
RXERR_INVALID_SET
The query/set item you specified was invalid.

6 RXERR_INVALID_NAME
RXERR_INVALID_KEYWORD
The filename or keyword you specified was invalid.

8 RXERR_INVALID_TRAP
The trap you tried to remove didn't exist.

10 RXERR_INVALID_HANDLE
The lister handle you gave was invalid.

12 RXERR_NO_TOOLBAR
The lister has no valid toolbar.

15 RXERR_NO_MEMORY
There wasn't enough memory to do what you wanted.

20 RXERR_NO_LISTER
A lister failed to open (usually because of low-memory).

You can convert the error codes returned in RC into meaningful error messages (for error reports and so on) with the dopus error command.

1.77 Magellan II ARexx Tutorial: Author

You might be wondering who wrote this tutorial, what he does for a living, how old he is, how long he's been playing with his Amiga, how he got conned into writing it, whether in fact he is a he.

Well, I can safely say I know all the answers to those questions, but for the more investigative among you, my name is hidden in the guide somewhere.

A Clue:

"Let him who hath understanding reckon the number of the Beast, for it is a human number. It's number is six hundred and sixty-six."
- Iron Maiden, The Number of the Beast

BTW, you would have found it if you'd read the guide as it was meant to be read, see Format .

1.78 ArcDir.dopus5: Intro

```
/*
  $VER: ArcDir.dopus5 1.0 (31.7.96)
  Written by Edmund Vermeulen (edmundv@grafix.xs4all.nl).

  ARexx script for Directory Opus 5 to show the contents of an LhA or LZX
  archive in an Opus lister and operate on the files and directories inside
  the archive as if it is a normal directory.

  Function : ARexx      DOpus5:ARexx/ArcDir.dopus5 Browse {Qp} {f} {Ql}
  Flags : Run asynchronously
*/
```

  Welcome to Edmund Vermeulen's ArcDir script for Directory Opus v5+.

  I have split it up into sections in the hope of trying to describe how it works in a relatively easy manner, (might not happen though, 'best laid plans of mice and men...', etc :)

  At the top is Edmund's introduction at the start of his script, I'm not sure if the email address is still correct, we at the Opus Beta-testing Clinic have not heard from Edmund for almost a year, (come back Edmund, Leo misses you...Leo? LEO? Uh Oh, we've lost another one).

  I've seperated the script into what I thought was the most logical parts, the links to which are below. It probably won't make much sense if you jump from one spot to another, as they are shown in exactly the same order as you encounter the statements in the script.

  I'll generally skip over the standard ARexx commands, only giving comment when it is needed to clarify something else.

   Setting up
   Attaching the handler
   Waiting for an Event
   Simple Events
   Event - DoubleClick
   Event - Drop
   Event - DropFrom
   Event - Copy
   Event - ViewCommands
   Event - Not Supported
   Event - Unknown
   Cleaning Up
   Parent/Root Action
   Path Action
   Delete Action
   Make a Directory
   Create Directories
   Listing the Archive
   Extracting Entries

```
   Adding Entries
   View Single Files
   Getting Selected Entries
   Patching Filenames
   Getting Catalog String
   Check for Valid Handler
   Syntax Error
   User Halts Script
   Displaying Errors
   Displaying a Requester
```

1.79 ArcDir.dopus5: Setup

Here we encounter the general checks and variables required before we get into the script proper.

```
 1parse arg cmd ' ' portname ' "' arcfile '" ' handle ' ' arcsubdir
 2
 3address value portname
 4options results
 5options failat 21
 6signal on syntax
 7signal on halt
 8signal on break_c
 9lf = '0a'x
10
11dopus getfiletype '"'arcfile'"' id
12arctype = result
13if arctype ~= 'LHA' & arctype ~= 'LZX' then
14  exit
15
16if ~show('l','rexxsupport.library') then
17  call addlib('rexxsupport.library',0,-30)
18
19if exists('LIBS:locale.library') then do
20  if ~show(l,'locale.library') then
21    call addlib('locale.library',0,-30)
22  catalog = opencatalog('ArcDir.catalog','english',0)
23  end
24else
25  catalog = 0
26
27dopus version
28newopus = result ~= 'RESULT' & translate(result,'.',' ') >= 5.1215
29
30if upper(cmd) = 'BROWSE' | handle = 0 then do
31  lister new
32  handle = result
33  lister set handle source
34  end
35else
36  lister empty handle
37
38call arclist
```

```
Line:
 1      We parse in the arguments it was called with, the command (BROWSE or
        GETDIR), the Opus portname (DOPUS.1 assumed), name of the file, the
        lister handle and the sub-directory in the archive.  The last is
        used when ArcDir is called from within an ArcDir lister to open a new
        lister with a sub-directory from the first.  Make sense?  It's when
        you shift-doubleclick on a directory within an ArcDir lister.
 3      Address the port, DOPUS.1
 4      Enable results, the  RESULT  variable.
 5      Set the fail limit.
 6 - 8  Jump to syntax, halt and break_c when any of those errors happen.
 9      Set lf to equal ASCII code 10, a linefeed.
11 - 14 First Opus command, we get the Filetype ID of the file the script was
        called with, and assign the variable arctype to it.  If the ID wasn't
        LHA or LZX, (the two types of archives ArcDir deals with), then we
        exit.
16 - 25 Load the rexxsupport.library if not already, load locale.library if
        it exists and not loaded already.  Then try to open the correct
        ArcDir catalog file, set catalog to true if successful, false if not.
27 - 28 What version of Opus are we dealing with?  If it's 5.1215 or later,
        then newopus is set to true.
30 - 36 What command did we call ArcDir with?  If it was BROWSE then open a
        new lister, store it's handle and set it to SOURCE.  If not,
        (GETDIR), then display an empty cache in the current lister, if there
        aren't any, we'll just clear the current one.
38      Call the routine which lists the archive in the lister.
```

1.80 ArcDir.dopus5: Handler

We attach a unique handler name to the lister and set which commands we want to trap.

```
 1handlername = 'ArcDir'handle
 2lister set handle handler handlername quotes
 3call openport(handlername)
 4
 5viewcommands = 'Read HexRead Show Play'  /* you may add other Opus commands if  ↩
    you wish */
 6notsupported = 'CopyAs Move MoveAs Rename Comment Protect'
 7traps = 'Copy Delete MakeDir Parent Root ScanDir' viewcommands notsupported
 8do while traps ~= ''
 9  parse var traps trapcommand traps
10  dopus addtrap trapcommand handlername
11  end
12
13thishandle = handle
14lister set handle busy off
```

```
Line:
 1      We make the handler name equal to 'ArcDir' plus the handle of the
        lister, since there can only be one lister with that handle it gives
        a unique name.
 2      Assign that handler to that lister, telling Opus we want quotes (")
        surrounding every filename argument from the handler.
 3      Enable the message port for that handler.
```

```
 5 - 7  These are the events we'll be trapping, supported and unsupported.
 8 - 11 Loop around, picking up one command at a time until all are done,
        and adding the trap to the handler.
13 - 14 Assign thishandle to the handle of our lister and make sure it's not
        busy.
```

1.81 ArcDir.dopus5: Capturing an event

```
  OK, now we start waiting for the user to do something, when they do a
message packet will arrive.

  1do until event = 'inactive'
  2  if waitpkt(handlername) then do
  3
  4    packet = getpkt(handlername)
  5    if packet ~= '00000000'x then do
  6
  7      event = getarg(packet,0)
  8      handle = getarg(packet,1)
  9      namestr = getarg(packet,2)
 10      user = getarg(packet,3)
 11      pathstr = getarg(packet,4)
 12      qualifier = getarg(packet,6)
 13      deststr = getarg(packet,7)
 14
 15      if newopus then
 16        lister wait thishandle quick
 17      else do
 18        lister query thishandle busy
 19        if result = 1 then call delay(10)
 20        end

Line:
 1      An event of inactive means that the lister has been closed, so we
        loop around waiting for it to happen.
 2      This is where there script usually 'parks' waiting for a message
        packet, when a packet arrives this statement becomes true and the
        script continues.
 4      We get the message packet in variable packet.
 5      If it isn't a null packet we continue, otherwise we would end up
        waiting for a real packet at line 2 again.
 7 - 13 We get the arguments from the packet, assigning them to variables.
15 - 20 We see if the lister is busy, if it's a late version of Opus we can
        use  lister wait , if it isn't then we use  lister query busy
        command.
```

1.82 ArcDir.dopus5: Event - Miscellaneous

```
  These are just the events that call sub-routines later in the script.

  1      select
  2        when upper(event) = 'PARENT' | upper(event) = 'ROOT' then
```

```
 3          call doparentroot
 4
 5        when event = 'Delete' then
 6          call dodelete
 7
 8        when event = 'MakeDir' then
 9          call domakedir
10
11        when event = 'reread' | event = 'ScanDir' then do
12          call delete('T:ArcDir.list'handle)
13          call arclist
14          end
15
16        when event = 'path' then
17          call dopath
```

```
Line:
 1      Start of the Select...When conditional block.
 2 - 3  Event is PARENT or ROOT, jump to  doparentroot .
 5 - 6  Event is Delete, jump to  dodelete .
 8 - 9  Event is MakeDir, jump to  domakedir .
11 - 14 Event is reread or ScanDir, we delete the file containing the
        contents of the archive, (for that handler), and call the  arclist
        routine to generate another.
16 - 17 User has typed a path in to the path string gadget, so we jump to
         dopath .
```

1.83 ArcDir.dopus5: Event - doubleclick

```
  The user has double-clicked on an entry, we work out what it was and what
to do.
```

```
 1        when event = 'doubleclick' then do
 2          if left(namestr,1) = '"' then
 3            parse var namestr '"' namestr '"'
 4          if namestr = '' then
 5            fileinfo.type = 1
 6          else
 7            lister query handle entry '"'namestr'"' stem fileinfo.
 8          if fileinfo.type > 0 then do  /* it's a dir */
 9            if qualifier = 'shift' then do
10              lister new
11              newhandle = result
12              address command 'Copy >NIL: T:ArcDir.list'handle 'T:ArcDir.list' ↩
   newhandle
13              lister set newhandle source
14              address command 'Run >NIL: <NIL: RX DOpus5:ARexx/ArcDir.dopus5  ↩
   GETDIR' portname '"'arcfile'"' newhandle arcsubdir||namestr'/'
15              end
16            else do
17              arcsubdir = arcsubdir||namestr'/'
18              call arclist
19              end
20            end
21          else
```

```
 22            call viewsingle
 23          end

Line:
 1      Event is a doubleclick of LMB over an entry.
 2 - 3  If the left character of the entry is a " then we parse the name of
        the entry out from between the quotes.
 4 - 7  If namestr is empty, then we assume it's a directory otherwise we do
        a  lister query entry  on it and check it's TYPE, (this is
        different from it's filetype).
 8      If the TYPE is greater than 0 then it's a directory otherwise it's a
        file.
 9 - 15 It was a directory, we check to see whether a shift key was held
        down.  If it was: open a new lister, store it's handle, copy the
        archive contents list to another file for the new handler, set it to
        SOURCE then call ArcDir.dopus5 on it with the command GETDIR, Opus'
        ARexx port name, archive filename, handle of the new lister AND the
        name of the directory on which we doubleclicked.
16 - 20 Shift wasn't held down, so we add the name of the directory to the
        path of the directory we are currently in and call the routine
         arclist .
21 - 23 We doubleclicked on a file, so we call the  viewsingle  routine.
```

1.84 ArcDir.dopus5: Event - drop

Someone dropped something on our ArcDir lister, they're going to pay for that!

```
 1        when event = 'drop' then do
 2          parse var namestr '"' droppath '"'
 3          if pos(right(droppath,1),'/:') > 0 then
 4            lister read handle '"'droppath'"' force
 5          else do
 6            parse var namestr '"' dropfile '"'
 7            if pos(':',dropfile) = 0 then do
 8              lister query user path
 9              dropfile = result||dropfile
10              end
11            dopus getfiletype '"'dropfile'"' id
12            if result = 'LHA' | result = 'LZX' then do
13              arctype = result
14              arcfile = dropfile
15              arcsubdir = ''
16              call delete('T:ArcDir.list'handle)
17              call arclist
18              end
19            else do
20              allents = namestr
21              call getall
22              otherhandle = user
23              call arcadd
24              end
25            end
26          end
```

```
Line:
 1      The event was drop which means something was 'dropped' onto our
        lister, if they had dragged from our lister and dropped onto another
        one then it would be a dropfrom event.
 2      We parse what was dropped on us in namestr from between the quotes,
        (I'll call it thing just to give it a name :)
 3 - 4  If the right character of thing is a ':' or a '/' then they dropped
        either a device on the lister or a directory.  So we'll just read
        that path into the lister, this will kill our handler.
 5 - 10 It wasn't a device/directory, what a relief, so now we parse it again
        giving it the name dropfile.  If it doesn't contain a ':' character
        then we need to find out where it came from by querying the path of
        the lister whose handle was given in variable user.  We then let
        dropfile equal the path and the filename.
11      We find out what filetype ID the file has.
12 - 18 If it was an LZX or LhA archive then we set arctype to the ID,
        arcfile to the filename, the initial archive directory as root,
        delete ArcDir's current working archive contents list and call the
         arclist  routine to generate a new one on the new archive.
19 - 25 OK, it's not a device, directory or another archive, must be a file
        or a directory!  We make allents equal the namestr argument then
        call  getall  to get all the names of the entries that were dropped
        on the lister.  We then set otherhandle to equal the handle of the
        lister where the files/dirs came from and call the  arcadd  routine
        to add them to the archive.
```

1.85 ArcDir.dopus5: Event - dropfrom

A 'dropfrom' event, someone took something out of our lister and dropped it on another one, thieves!

```
 1        when event = 'dropfrom' then
 2          if qualifier = 'shift' then do
 3            parse var namestr '"' namestr '"'
 4            lister query handle entry '"'namestr'"' stem fileinfo.
 5            if fileinfo.type > 0 then do
 6              address command 'Copy >NIL: T:ArcDir.list'handle 'T:ArcDir.list' ↩
    user
 7              address command 'Run >NIL: <NIL: RX DOpus5:ARexx/ArcDir.dopus5  ↩
    GETDIR' portname '"'arcfile'"' user arcsubdir||namestr'/'
 8              end
 9            end
10          else do
11            allents = namestr
12            call getall
13            otherhandle = user
14            call arcextract
15            end

Line:
 1      Event is dropfrom, dragged from the ArcDir lister and dropped onto
        another one.
 2      Were they holding down a shift key when they dragged and dropped?
 3 - 4  They were!  Get the name of the entry from between the quotes.
 5      Find out what type of entry it was, if it's a file the type will be
```

```
        less than 0, if it's a directory it will greater than 0.
 6 - 9  It was a directory, so we copy across the ArcDir archive contents
        list to a new file using the handle of the 'dropped' on lister.
        Then we run the ArcDir script on the other lister with the GETDIR
        command, Opus' ARexx port name, the name of the archive, the handle
        of the other lister and the archive sub-directory that we dropped on
        that lister.
10 - 15 It was files/dirs that we dropped on to the other lister, we make
        allents equal the string of filenames that was sent in the packet,
        call the  getall  routine to seperate it into indiviual file/dir
        names, set the otherhandle to the handle of the other lister
        specified in the packet and then call the  arcextract  routine to
        extract the files/dirs.
```

1.86 ArcDir.dopus5: Event - Copy

```
  The user selected some entries in a lister and then hit the copy button.

  1         when event = 'Copy' then do
  2           lister query handle selentries
  3           allents = result
  4           call getall
  5           if handle = thishandle then do
  6             otherhandle = user
  7             call arcextract
  8             end
  9           else do
 10             otherhandle = handle
 11             handle = user
 12             call arcadd
 13             end
 14           end

Line:
 1      It's a Copy event so let's do it.
 2 - 4  We ask for a list of all selected entries storing the result in
        allents, then call the  getall  routine to seperate the list into
        seperate names and types.
 5 - 8  If the handle given in the packet matches the handle for our ArcDir
        lister, then we are copying from the ArcDir lister to the other one,
        (extracting entries from the archive), so we set otherhandle to the
        handle for the other lister and call the  arcextract  routine.
 9 - 13 Otherwise, we're copying to the ArcDir lister, (adding entries to the
        archive), so we make otherhandle equal to the other lister's
        handle, handle equal to the ArcDir handle and call the  arcadd
        routine.
14      End of the When loop.
```

1.87 ArcDir.dopus5: Event - View commands

```
  You had a bunch of files selected and then caused a Read, HexRead, Show or
Play event.  ArcDir will work upon the first selected.
```

```
  1        when pos(event,viewcommands) > 0 then do
  2          lister query handle firstsel
  3          parse var result '"' namestr '"'
  4          lister select handle '"'namestr'"' off
  5          lister refresh handle
  6          call viewsingle
  7          end

Line:
 1      The event matched one in our viewcommands string set back  here .
 2      We get the name of the FIRST selected entry.
 3      Parse the result out from between the quotes.
 4      Turn the selection state for the entry to off.
 5      Refresh the lister display so you see the selection state change.
 6      Call the  viewsingle  routine on that one entry.
 7      End of the When loop.
```

1.88 ArcDir.dopus5: Event - Unsupported

```
  If you end up here it means you specifically used an event that was in the
unsupported list...NAUGHTY!

  1        when pos(event,notsupported) > 0 then do
  2          lister set handle busy on
  3          call displayerror(getcatstr(23,'Command not supported in ArcDir.'))
  4          lister set handle busy off
  5          end

Line:
 1      Compare the event against the list of not supported commands set back
         here .
 2      Set the lister state to busy.
 3      Get the  displayerror  routine to...eerr...display an error message,
        translating if necessary through  getcatstr .
 4      Set the lister back to idle state.
 5      End of the When loop.
```

1.89 ArcDir.dopus5: Event - Anything else

```
  If you got to this bit, it means that we recieved an event we don't know
about.  Time for you to brush up on your ARexx skills and add more :)

  1        otherwise
  2          nop
  3        end
  4
  5      lister set handle busy off
  6      call reply(packet,0)
  7      end
  8    end
  9  end
```

```
Line:
 1 - 3  The Otherwise statement in the Select...When conditional block, we
        recieved an event we didn't recognise, so we do nothing.
 5      Turn off the busy state of the lister.
 6      Reply to the message packet we recieved, YOU MUST DO THIS!!!!
 7 - 9  End of 'if packet...'.
        End of 'if waitpkt...'.
        End of 'do until...'.
```

1.90 ArcDir.dopus5: Cleaning Up

We recieved an event of 'inactive', (the user closed the lister), so we start cleaning up.

```
 1call delete('T:ArcDir.list'handle)
 2call closeport(handlername)
 3if catalog ~= 0 then
 4  call closecatalog(catalog)
 5exit

Line:
 1      We delete the archive list that ArcDir was using to list the archive.
 2      Close the message port.
 3 - 4  If we opened a catalog, then we close it.
 5      Bye Bye
```

1.91 ArcDir.dopus5: Parent/Root action

The user has clicked on the Parent or Root button, or used the keyboard shortcuts.

```
 1doparentroot:
 2  if arcsubdir = '' then do
 3    cuthere = lastpos('/',arcfile)
 4    if cuthere = 0 | upper(event) = 'ROOT' then
 5      cuthere = pos(':',arcfile)
 6    normaldir = left(arcfile,cuthere)
 7    if qualifier = 'shift' then do
 8      lister new normaldir
 9      newhandle = result
10      lister wait newhandle
11      lister set newhandle source
12      end
13    else do
14      lister set handle title
15      lister read handle normaldir
16      end
17    end
18  else do
19    if upper(event) = 'ROOT' then
20      newsubdir = ''
```

```
 21    else do
 22      cuthere = lastpos('/',left(arcsubdir,length(arcsubdir) - 1))
 23      newsubdir = left(arcsubdir,cuthere)
 24      end
 25    if qualifier = 'shift' then do
 26      lister new
 27      newhandle = result
 28      address command 'Copy >NIL: T:ArcDir.list'handle 'T:ArcDir.list'newhandle
 29      lister set newhandle source
 30      address command 'Run >NIL: <NIL: RX DOpus5:ARexx/ArcDir.dopus5 GETDIR'  ↩
    portname '"'arcfile'"' newhandle newsubdir
 31      end
 32    else do
 33      arcsubdir = newsubdir
 34      call arclist
 35      end
 36    end
 37  return

Line:
 1      Sub-routine label.
 2 - 17 If arcsubdir is empty, it means we're in the root directory of
        the archive.  We then check to see if there is a '/' character
        starting from the end of arcfile, (which is the name of our
        archive with full path, for example - RAM:Foo/bar.lha), if there is
        no '/' then we must be in the root of the device and we get the
        position of the ':'.  normaldir is then set to the path where our
        archive is, so if it was RAM:Foo/bar.lha normaldir will be RAM:Foo/.
        If the shift key was held down we open a new lister and read the
        path specified in normaldir, we get it's handle in
        newhandle, wait until it's idle and set it's state to source.
        If the shift key wasn't held down, we set the titlebar to the default
        and read the directory into the lister which kills our handler.
18      If arcsubdir wasn't empty we start here.
19 - 20 If the event was ROOT then we set newsubdir to an empty string.
21 - 24 Otherwise we find the second last ocurrance of '/' and set
        newsubdir to the path to the left of it, for example if
        arcsubdir is RAM:foo/bar/who/ then newsubdir will
        be RAM:foo/bar/.
25 - 31 If the shift key was held down, we open a new lister, get it's
        handle, copy the archive contents list to another file for the new
        handler, set it to SOURCE then call ArcDir.dopus5 on it with the
        command GETDIR, Opus' ARexx port name, archive filename, handle of
        the new lister AND the path in newsubdir.
32 - 35 The shift key wasn't held down, so we set arcsubdir to newsubdir and
        call  arclist  to update the entry display in the lister.
37      Return.
```

1.92 ArcDir.dopus5: Path gadget action

The user typed a string into the path gadget, so we got to go and get the path, bother.

```
 1dopath:
 2  if pos(right(namestr,1),'/:') = 0 then
```

```
 3    namestr = namestr'/'
 4  if left(namestr,length(arcfile)) = arcfile then do
 5    if namestr = arcfile'/'arcsubdir then
 6      call delete('T:ArcDir.list'handle)
 7    else
 8      arcsubdir = substr(namestr,length(arcfile) + 2)
 9    call arclist
10    end
11  else do
12    cuthere = pos('.LHA/',upper(namestr))
13    if cuthere = 0 then
14      cuthere = pos('.LZH/',upper(namestr))
15    if cuthere > 0 then
16      arctype = 'LHA'
17    else do
18      cuthere = pos('.LZX/',upper(namestr))
19      if cuthere > 0 then
20        arctype = 'LZX'
21      end
22    if cuthere > 0 then do
23      call delete('T:ArcDir.list'handle)
24      arcfile = left(namestr,cuthere + 3)
25      arcsubdir = substr(namestr,cuthere + 5)
26      call arclist
27      end
28    else do
29      lister read handle '"'namestr'"' force
30      end
31    end
32  return
```

```
Line:
 1      Sub-routine label.
 2 - 3  Check to see if the rightmost character of the path is a ':' or a
        '/', if it wasn't then add a '/'.
 4      We check to see if the leftmost part of the path matches our current
        archive name.
 5 - 6  The path equalled our current path, so we do the equivalent of a
        ReRead.  Delete the current archive contents list...
 7 - 8  Otherwise we set our current archive sub-directory to what was typed
        in...
 9 - 10 and call  arclist  to generate a lister display.
11 - 21 These lines are checking to see if the path typed in was actually
        another archive name and setting cuthere to greater than 0
        if it was.
22 - 27 If the path was another archive we delete the current archive
        contents list, set arcfile to the name of the archive, set arcsubdir
        to the sub-directory in the archive and call  arclist  to set up the
        lister display for the new archive.
28 - 30 It wasn't a path in the current archive, it wasn't a new archive, it
        must be a real path - so we read it into the lister, this will kill
        our handler and it will now be a normal lister.
31      End of 'if left(namestr...'.
32      Return to where we came from.
```

1.93 ArcDir.dopus5: Delete action

The user has selected some entries in the archive and decided to delete them, so let's get rid of them.

```
 1dodelete:
 2  lister set handle busy on
 3  lister query handle selentries
 4  allents = result
 5  call getall
 6  if entries = 0 then
 7    return
 8
 9  lister query handle numselfiles
10  nfiles = result
11  lister query handle numseldirs
12  ndirs = result
13  call dorequest('"'getcatstr(5,'Warning: you cannot get back'lf||,
14    'what you delete! OK to delete:'lf||lf'%s file(s) and'lf||,
15    '%s drawer(s) (and their contents)?',nfiles,ndirs)'"',
16    getcatstr(6,'Proceed|Cancel'))
17  if ~rc then
18    return
19
20  lister set handle title getcatstr(7,'Deleting from archive...')
21  lister refresh handle full
22
23  select
24    when arctype = 'LHA' then do
25      call open('actionfile','T:actionfile'handle,'w')
26      do i = 1 to entries
27        if type.i > 0 then
28          wild = '/#?'
29        else
30          wild = ''
31        call writeln('actionfile','"'patch(arcsubdir||name.i,1)||wild'"')
32        end
33      call close('actionfile')
34      address command 'LhA d -q -X -Qp -Qo "'patch(arcfile,0)'" @T:actionfile' ↩
   handle
35      problem = rc > 0
36      address command 'Delete >NIL: T:LhA_ArcWork.#? QUIET'
37      problem = problem | rc = 0
38      call delete('T:actionfile'handle)
39      end
40    when arctype = 'LZX' then do
41      lzxcmd = 'LZX d -q -X0 --' lzxkludge(patch(arcfile,0))
42      linelen = 0
43      n = 0
44      do i = 1 to entries
45        if type.i > 0 then
46          wild = '/#?'
47        else
48          wild = ''
49        dothis = lzxkludge(patch(arcsubdir||name.i,0)||wild)
50        linelen = linelen + length(dothis) + 1
```

```
 51        if i = 1 | linelen > 255 then do
 52          n = n + 1
 53          dothese.n = dothis
 54          linelen = length(lzxcmd) + length(dothis) + 1
 55          end
 56        else
 57          dothese.n = dothese.n dothis
 58        end
 59      do i = 1 to n
 60        address command lzxcmd dothese.i
 61        problem = rc > 0
 62        if problem then
 63          leave
 64        end
 65      end
 66    end
 67
 68  if problem then
 69    call displayerror(getcatstr(8,'Error while deleting from archive.'))
 70  else do
 71    call delete('T:ArcDir.list'handle)
 72    do i = 1 to entries
 73      if name.i = '' then do
 74        lister query handle separate
 75        if result = 'filesfirst' then do
 76          lister query handle numfiles
 77          entryno = result
 78          end
 79        else
 80          entryno = 0
 81        lister remove handle '#'entryno
 82        end
 83      else
 84        lister remove handle '"'name.i'"'
 85      end
 86    end
 87
 88  lister set handle title 'ArcDir:' arcname
 89  lister refresh handle full
 90  return
```

```
Line:
 1      Sub-routine label.
 2      Set the lister state to busy so the user can't play around with it.
 3 - 7  Get the list of selected entries and assign allents to it, then call
        the  getall  routine to seperate them into individual file/dir names.
        If there weren't any entries selected then return.
 9 - 18 Get the number of selected files and the number of selected
        directories then tell the user he's about to delete so many of each
        via the  dorequest  routine translating via the  getcatstr  routine
        if necessary.  If he chose Cancel then return.
20 - 21 Change the lister titlebar to inform the user what we're about to do,
        and refresh the lister display so that he sees it.
23 - 66 The type of archive will select which block of statements in the
        Select...When conditional block we execute.  For this particular
        example I'll assume it's a LhA archive because it's easier.
25      We open a file in T: for writing to, it will contain the entries that
```

```
        we want to delete.
26 - 32 A DO loop in which we check the TYPE of the entry, (as detected in
        the  getall  routine), if it's a directory we set wild to '#?',
        if not then we set it to an empty string.  We then write the entry to
        the temporary file in T:,  patching  the string for strange
        characters and adding wild.  Keep doing it until there are no more
        entries to add.
33      Close the temporary file.
34      Call the archiver command to delete the entries from the archive,
        telling it to use the list in the temporary file.
35      If it wasn't successful then problem will be set to 1.
36      Delete temporary output file.
37      If there was a problem with the archiver command or the delete
        command, then problem will equal 1.
38 - 39 Delete the temporary action file in T: and exit the Select...When
        block.
68 - 69 If there was a problem then tell the user using  displayerror .
70 - 71 Otherwise, delete the archive contents list in T:, we'll need a new
        one.
72 - 85 We enter a DO loop to remove the entries from the lister display.
        If the entry has an empty string for a name then we check the lister
        seperation method, (files first, mixed or directories first).  If
        it's files first then we ask for the number of files in the display
        and make the entry number equal to the total number, if not we make
        it equal to 0.
          Remember, an entry with an empty string for a name is assumed to be
        a directory, so if the seperation method is files first the first
        directory entry will equal the number of files in the display (zero
        numbering for entries).  If the seperation method is directories
        first or mixed, then the entry with no name is going to be the very
        first, hence entry number will equal 0.
        We then use  lister remove  to remove that entry number.

        If the entry had a name then we just use  lister remove  to remove
        the entry with that name.
88 - 89 Set the lister titlebar back to it's normal display and refresh it so
        that the user sees that the entries have gone.
90      Return.
```

1.94 ArcDir.dopus5: Making new directories

OK, you've decided to make a new directory in the archive, this is where you end up.

```
 1domakedir:
 2  lister set handle busy on
 3  dopus getstring '"'getcatstr(15,'Enter directory name')'" 31 "" getcatstr ↩
    (16,'OK|Cancel')
 4  dirtomake = result
 5  if dirtomake == '' | dirtomake = 'RESULT' then
 6    return
 7
 8  now = date('i') * 86400 + time('s')
 9  call createdirs(dirtomake'/')
10
```

```
 11  select
 12    when arctype = 'LHA' then
 13      address command 'LhA a -q -e -r -X -Qo "'patch(arcfile,0)'" T:ArcDir' ↩
    handle'/' '"'patch(arcsubdir||dirtomake,1)'"'
 14    when arctype = 'LZX' then do
 15      oldcurrent = pragma('d')
 16      call pragma('d','T:ArcDir'handle)
 17      address command 'LZX a -q -e -r -X0 --' lzxkludge(patch(arcfile,0))  ↩
    lzxkludge(patch(arcsubdir||dirtomake,0))
 18      call pragma('d',oldcurrent)
 19      end
 20    end
 21
 22  if rc > 0 then
 23    call displayerror(getcatstr(13,'Error while adding to archive.'))
 24  else do
 25    lister add handle '"'dirtomake'" -1 1' now '----rwed'
 26    lister refresh handle
 27    end
 28
 29  address command 'Delete >NIL: T:ArcDir'handle 'ALL QUIET'
 30  call delete('T:ArcDir.list'handle)
 31  return
```

```
Line:
 1      Sub-routine label.
 2      Set the lister to busy state, stop the user playing with it.
 3      Put up a requester asking for the directory name, translating the
        requester text if required with the  getcatstr  routine.
 4      Get the result of the requester.
 5 - 6  If it was an empty string or equalled 'RESULT' then return.  This
        means you couldn't make a directory called RESULT, this can be fixed
        very easily by changing line 5 to read:

        if dopusrc = 0 | dirtomake = '' then

 8      We get the time since 1-Jan-1978 in seconds to use as the creation
        date.
 9      Call the  createdirs  routine with the name of our directory.
10 - 20 Depending upon the archive type, we now just add the directory
        structure just created by  createdirs  to the archive, which of
        course adds our new empty directory.
22 - 23 If the archiving wasn't successful we tell you about it.
24 - 27 Otherwise, we add an entry to the lister using  lister add  using the
        time worked out in line 8.
29      Delete the directory structure created by  createdirs .
30      Delete the archive contents list that ArcDir works from since we now
        need a new one.
31      Return from whence we came.
```

1.95 ArcDir.dopus5: Create Directories

```
  1createdirs:
  2  parse arg subdir
  3  dirstocreate = 'T:ArcDir'handle'/'arcsubdir||subdir
```

```
  4  here = 0
  5  do until here = 0
  6    here = pos('/',dirstocreate,here + 1)
  7    if here > 0 then
  8      call makedir(left(dirstocreate,here - 1))
  9    end
 10  return

Line:
 1      Sub-routine label.
 2      We get the directory that was passed to the routine and assign
        subdir to it.
 3      We set dirstocreate to the complete path in T: of the directory we
        are going to create, for example if we wanted to create a directory
        called 'help' under a sub-directory of 'wanted' then 'dirstocreate'
        will be 'T:ArcDir12345678/wanted/help/'.  The ArcDir12345678 is the
        unique name of the handler associated with this lister from the
         here .
 4      Set here to 0.
 5 - 9  This loop cycles around creating the directories a sub-level at a
        time.  With the example in line 3 above, it would first create
        T:ArcDir12345678/, then T:ArcDir12345678/wanted/, then
        T:ArcDir12345678/wanted/help/.
10      We return.
```

1.96 ArcDir.dopus5: Listing the archive

This routine is the heart of the script, without it the handler wouldn't work very well at all.

What it does is create a list of the archive contents using the list command of the various archivers, a seperate program called ArcDirList then reformats this output into something a bit easier and quicker for the script to understand.

The script works from this list when you change directories, when you add or delete files from the archive a new list has to be generated to keep the lister display up to date.

```
  1arclist:
  2  lister set handle busy on
  3  lister clear handle
  4  lister set handle title getcatstr(1,'Listing archive...')
  5  lister set handle path arcfile'/'arcsubdir
  6  lister refresh handle full
  7
  8  if ~exists(arcfile) then do
  9    call displayerror(getcatstr(22,'Error: archive not found'))
 10    return
 11    end
 12
 13  if ~exists('T:ArcDir.list'handle) then do
 14    select
 15      when arctype = 'LHA' then
 16        address command 'LhA >T:ArcDir.list'handle 'vv -N -X -Qw -Qo "'arcfile ↩
    '"'
```

```
 17      when arctype = 'LZX' then
 18        address command 'LZX >T:ArcDir.list'handle 'v -X0 --' lzxkludge(patch( ↩
    arcfile,0))
 19      end
 20    if rc > 0 then
 21      call displayerror(getcatstr(2,'Error while listing archive.'))
 22    end
 23
 24  oldcurrent = pragma('d')
 25  call pragma('d','DOpus5:C')
 26  address command 'ArcDirList >T:ArcDir.list'handle'@ T:ArcDir.list'handle '"' ↩
    patchstar(arcsubdir)'"'
 27  call pragma('d',oldcurrent)
 28
 29  if ~open('tempfile','T:ArcDir.list'handle'@','r') then do
 30    call displayerror(getcatstr(24,'ArcDirList not found!'))
 31    return
 32    end
 33  thisline = readln('tempfile')
 34  do while thisline ~= ''
 35    lister add handle thisline
 36    thisline = readln('tempfile')
 37    end
 38  call close('tempfile')
 39  call delete('T:ArcDir.list'handle'@')
 40
 41  cuthere = lastpos('/',arcfile)
 42  if cuthere = 0 then
 43    cuthere = lastpos(':',arcfile)
 44  arcname = substr(arcfile,cuthere + 1)
 45  lister set handle title 'ArcDir:' arcname
 46  lister refresh handle full
 47  return
```

```
Line:
 1      Sub-routine label.
 2 - 6  Set the state of the lister to busy, clear the lister display, change
        the titlebar to say 'Listing archive...', set the path gadget to the
        path within the archive and refresh the lister display so we see all
        these changes.
 8 - 11 Check to see if the archive exists, if it doesn't return a nasty
        message translating through  getcatstr  if necessary.
13 - 19 If an archive contents list doesn't currently exist, we use the
        appropriate archiver to generate another one.
20 - 22 If there was an error listing the archive, tell the user.
24 - 27 Get the current directory, change directory to DOpus5:C so we can run
        ArcDirList to format the output from the archivers in lines 13-19.
        Then change back to the directory we were originally in.
29 - 32 We try to open the reformatted archive contents list, if we can't we
        display an error message and return.
33 - 37 We read a line of the file then enter a DO loop until there is
        nothing more to be read from the file.  We add the line we read
        initially to the lister display, then read another, add it, read
        another, and so on until all entries have been read and added.
38 - 39 Close the file and delete the temporary file.
41 - 45 Here we look for a '/' or a ':' in the arcfile variable, when we find
        it we set arcname to everything to the left of it and then set the
```

```
        titlebar to 'ArcDir:' plus the archive name.
46      Refresh the lister display so we see all the entries and titlebar
        changes.
47      Return to the calling function.
```

1.97 ArcDir.dopus5: Extracting from the archive

```
  The user has copied something out of the lister, or dragged something from
the lister and dropped it on another one.  Either way we need to extract it.

  1arcextract:
  2  lister set handle busy on
  3  if otherhandle = 0 then
  4    if newopus then
  5      winpath = deststr
  6    else do
  7      call displayerror(getcatstr(9,'No destination selected!'))
  8      return
  9      end
 10  else do
 11    if checkhandler() then
 12      return
 13    lister set otherhandle busy on
 14    lister query otherhandle path
 15    winpath = result
 16    end
 17
 18  lister query handle numdirs
 19  anydirs = result > 0
 20  mustmove = anydirs & arcsubdir ~== ''
 21  if mustmove then do
 22    destpath = winpath'ArcDir'handle
 23    call makedir(destpath)
 24    destpath = destpath'/'
 25    end
 26  else
 27    destpath = winpath
 28
 29  lister set handle title getcatstr(10,'Extracting from archive...')
 30  lister refresh handle full
 31
 32  select
 33    when arctype = 'LHA' then do
 34      call open('actionfile','T:actionfile'handle,'w')
 35      do i = 1 to entries
 36        if type.i > 0 then
 37          wild = '/#?'
 38        else
 39          wild = ''
 40        call writeln('actionfile','"'patch(arcsubdir||name.i,1)||wild'"')
 41        end
 42      call close('actionfile')
 43
 44      if anydirs then
 45        cmd = 'x'
```

```
 46      else
 47        cmd = 'e -x2'
 48      address command 'LhA' cmd '-q -a -C0 -X -Qo "'patch(arcfile,0)'" "' ↩
    destpath'" @T:actionfile'handle
 49      problem = rc > 0
 50      call delete('T:actionfile'handle)
 51      end
 52    when arctype = 'LZX' then do
 53      if anydirs then
 54        cmd = 'x'
 55      else
 56        cmd = 'e'
 57      lzxcmd = 'LZX' cmd '-q -a -C0 -X0 --' lzxkludge(patch(arcfile,0))
 58
 59      linelen = 0
 60      n = 0
 61      do i = 1 to entries
 62        if type.i > 0 then
 63          wild = '/#?'
 64        else
 65          wild = ''
 66        dothis = lzxkludge(patch(arcsubdir||name.i,0)||wild)
 67        linelen = linelen + length(dothis) + 1
 68        if i = 1 | linelen > 255 then do
 69          n = n + 1
 70          dothese.n = dothis
 71          linelen = length(lzxcmd) + length(dothis) + 1
 72          end
 73        else
 74          dothese.n = dothese.n dothis
 75        end
 76
 77      oldcurrent = pragma('d')
 78      call pragma('d',destpath)
 79      do i = 1 to n
 80        address command lzxcmd dothese.i
 81        problem = rc > 0
 82        if problem > 0 then
 83          leave
 84        end
 85      call pragma('d',oldcurrent)
 86      end
 87    end
 88
 89  if problem then
 90    call displayerror(getcatstr(11,'Error while extracting from archive.'))
 91  else
 92    do i = 1 to entries
 93      lister select handle '"'name.i'"' off
 94      end
 95
 96  lister set handle title 'ArcDir:' arcname
 97  lister refresh handle full
 98
 99  if mustmove then do
100    address command 'DOpus5:C/Move >NIL: "'destpath||arcsubdir'#?" "'winpath'"'
101    address command 'Delete >NIL: "'winpath'ArcDir'handle'" ALL QUIET'
```

```
102    end
103
104  if otherhandle ~= 0 then do
105    lister set otherhandle busy off
106    lister read otherhandle '"'winpath'"' force
107    end
108  return
```

Line:

1 Sub-routine label.

2 Change the ArcDir lister to state busy.

3 If the handle given to us for the other lister equals 0 then...

4 - 5 If it's an Opus version later than 5.1215, (newopus was set back here), then we set winpath to the string returned in argument 7 of the message packet.

6 - 9 If it's an earlier version of Opus, then we display an error message and return immediately, (upgrade :)

10 - 16 If the handle didn't equal 0 then we check to see if the other lister is a custom handler, if it is then we return immediately. If it isn't we set the state of the other lister to busy and get it's path in winpath.

18 - 19 We see if there are any directories in our ArcDir lister display, if there are we set anydirs to 1.

20 We set mustmove to 1 if anydirs was set to 1 and we are not in the root directory of the archive.

21 - 25 If mustmove is true (1), then we will have to extract to a directory structure in T:, so we call the makedir routine to create it and set destpath to the created directory.

26 - 27 Otherwise we set destpath to equal winpath.

29 - 30 Set the ArcDir lister titlebar to tell the user what we're doing and refresh the display.

32 - 87 Now depending what type of archive it is, we do the appropriate part of the Select...When conditional block, I'll assume LhA because it's easier :)

34 We open a file in T: for writing to, it will contain the entries that we want to extract.

35 - 41 A DO loop in which we check the TYPE of the entry, (as detected in the getall routine), if it's a directory we set wild to '#?', if not then we set it to an empty string. We then write the entry to the temporary file in T:, patching the string for strange characters and adding wild. Keep doing it until there are no more entries to add.

42 Close the temporary file.

44 - 47 If anydirs was set to true (1), then we make the extraction method 'x' (full paths) otherwise we set it to 'e -x2' (ignore paths), see the LhA docs.

48 Run the archiver with the name of our archive, patching for any strange characters, and the list of files to extract.

49 If there was a problem set problem to true (1).

50 Delete the temporary file in T:.

51 End of the When block for LhA archives.

89 - 90 If there was a problem extracting entries, tell the user via the displayerror routine, translating if necessary through the getcatstr routine.

91 - 97 If there wasn't any problem then we unselect all our selected entries, set the ArcDir titlebar back to it's normal display and do a full refresh to update the lister display.

```
99 -102 If mustmove was true, then we need to move the extracted entries from
        the directory structure in T: to the destination path, destpath.
        Then we delete the directory structure in T:.
104-107 If otherhandle doesn't equal 0, then there is a destination lister,
        we set it's state back to idle and force it to reread the directory.
108     We return.
```

1.98 ArcDir.dopus5: Adding to the archive

Well, the user has decided to Copy something to our lister or dropped something on us, either way we've now got the hard work of adding it to the archive.

```
 1arcadd:
 2  if checkhandler() then
 3    return
 4  lister set handle busy on
 5  lister set otherhandle busy on
 6  lister query otherhandle path
 7  frompath = result
 8
 9  mustcopy = upper(right(src,length(arcsubdir))) ~== upper(arcsubdir)
10  if mustcopy then do
11    homedir = 'T:ArcDir'handle''
12    call createdirs
13    end
14  else
15    homedir = left(frompath,length(frompath) - length(arcsubdir))
16
17  if mustcopy then
18    do i = 1 to entries
19      lister query otherhandle entry '"'name.i'"' stem fileinfo.
20      if fileinfo.type > 0 then
21        address command 'Copy "'frompath||name.i'" "T:ArcDir'handle''arcsubdir ↩
    ||name.i'" ALL CLONE QUIET'
22      else
23        address command 'Copy "'frompath||name.i'" "T:ArcDir'handle''arcsubdir ↩
    '" CLONE QUIET'
24      end
25
26  lister set handle title getcatstr(12,'Adding to archive...')
27  lister refresh handle full
28
29  select
30    when arctype = 'LHA' then do
31      call open('actionfile','T:actionfile'handle,'w')
32      call writeln('actionfile','"'patch(homedir,0)'"')
33      do i = 1 to entries
34        call writeln('actionfile','"'patch(arcsubdir||name.i,0)'"')
35        end
36      call close('actionfile')
37
38      if pos('.LZH/',test) > 0 then
39        method = '-0'
40      else
```

```
 41        method = ''
 42      address command 'LhA r' method '-q -e -r -X -Qo "'patch(arcfile,0)'" @T: ↩
    actionfile'handle
 43      problem = rc > 0
 44      call delete('T:actionfile'handle)
 45      end
 46    when arctype = 'LZX' then do
 47      lzxcmd = 'LZX u -q -a -e -r -X0 --' lzxkludge(patch(arcfile,0))
 48      linelen = 0
 49      n = 0
 50      do i = 1 to entries
 51        dothis = lzxkludge(patch(arcsubdir||name.i,0))
 52        linelen = linelen + length(dothis) + 1
 53        if i = 1 | linelen > 255 then do
 54          n = n + 1
 55          dothese.n = dothis
 56          linelen = length(lzxcmd) + length(dothis) + 1
 57          end
 58        else
 59          dothese.n = dothese.n dothis
 60        end
 61
 62      oldcurrent = pragma('d')
 63      call pragma('d',homedir)
 64      do i = 1 to n
 65        address command lzxcmd dothese.i
 66        problem = rc > 0
 67        if problem then
 68          leave
 69        end
 70      call pragma('d',oldcurrent)
 71      end
 72    end
 73
 74  if mustcopy then
 75    address command 'Delete >NIL: T:ArcDir'handle 'ALL QUIET'
 76
 77  if problem then
 78    call displayerror(getcatstr(13,'Error while adding to archive.'))
 79  else do
 80    do i = 1 to entries
 81      lister select otherhandle '"'name.i'"' off
 82      end
 83    lister refresh otherhandle
 84    call delete('T:ArcDir.list'handle)
 85    call arclist
 86    end
 87
 88  lister set otherhandle busy off
 89  return
```

```
Line:
 1      Sub-routine label.
 2 - 3  We call the  checkhandler  routine to make sure that the other lister
        wasn't a custom handler, if it was we return immediately.
 4 - 5  Turn the state of both listers to busy so the user can't play with
        them.
```

```
 6 - 7  We get the path of the other lister so we know where the files are
        coming from and assign it to frompath.
 8      This seems rather a complex way of doing things to me, but we set
        mustcopy to true (1) if we're not in the root directory of the
        archive.
 9 - 12 If mustcopy is true then we set homedir to a temporary directory in
        T:, and call the  createdirs  routine to create it.
13 - 14 mustcopy wasn't true so we set homedir to the path in the other
        lister.
17 - 24 If mustcopy did get set in line 9, then we need to copy the files or
        dirs to the directory structure created in T:.  We use
         lister query entry  to get each entries TYPE, if it's a file, (<0),
        we copy it straight to the directory in T:, if it's a directory,
        (>0), we copy it's directory structure to the directory in T:.
26 - 27 Change the lister's titlebar to show what we're doing, translating if
        necessary through  getcatstr .
29      Depending on what type of archive it was, LhA or LZX, the appropriate
        block of statements in the Select...When conditional block gets
        executed.  I'll just describe the LhA side, you can work out the
        other :)
31 - 32 We open a file in T: for writing to, it will contain the entries that
        we want to add.  The first line we write is the variable homedir.
33 - 35 A DO loop in which we write the entry to the temporary file in T:,
         patching  the string for strange characters.  Keep doing
        it until there are no more entries to add.
36      Close the temporary file.
38 - 41 If the archive ends in .LZH we set the archiver method to '-0' to
        retain compatibility when we add the entries, (see LhA docs).
42      Call the archiver to add the entries  patching  for any strange
        characters in the archive name and passing the list of entries to add
        in the temporary file.
43      If there was a problem we set problem to 1.
44      Delete the temporary file.
45      End of the When block for LhA archives.
74 - 75 If mustcopy was set, then we delete the directory structure that was
        created in T:.
77 - 78 There was a problem adding the files and RC got set to something
        other than 0, so we display an error message via the  displayerror
        routine.
79 - 86 There wasn't a problem, so we unselect the files in the other lister
        and refresh it's display.  We delete the archive contents list in T:
        because it's now out of date and call  arclist  to generate a new
        one.
88 - 89 We set the state of the other lister to idle and return.
```

1.99 ArcDir.dopus5: Viewing a file

If you've doubleclicked on a file or selected a file and used one of the view commands, (read, show, play, etc), then you'll end up in this routine.

Basically, it extracts the file to T: and then performs the specified function on it.

```
1viewsingle:
2  lister set handle busy on
3  lister set handle title getcatstr(10,'Extracting from archive...')
```

```
 4  lister refresh handle full
 5
 6  select
 7    when arctype = 'LHA' then
 8      address command 'LhA e -q -x2 -X -Qo "'patch(arcfile,0)'" T: "'patch( ↩
    arcsubdir||namestr,1)'"'
 9    when arctype = 'LZX' then
10      address command 'LZX e -q -X0 --' lzxkludge(patch(arcfile,0)) 'T:'  ↩
   lzxkludge(patch(arcsubdir||namestr,0))
11    end
12
13  if rc > 0 then
14    call displayerror(getcatstr(11,'Error while extracting from archive.'))
15
16  /* some creative ARexx programming :-) */
17  address command 'Run >NIL: <NIL: RX',
18    '"address' portname';',
19    'thisfile = ''T:'namestr''';',
20    'command' event '''"'''thisfile''"'''';',
21    'command wait protect name ''"'''thisfile''"''' set RWED;',
22    'do until ~exists(thisfile) | delete(thisfile);',
23      'call delay(200);',
24      'end"'
25
26  lister set handle title 'ArcDir:' arcname
27  lister refresh handle full
28  return
```

```
Line:
 1      Sub-routine label.
 2      Make the lister busy so the user can't disturb us.
 3      Call  getcatstr  for a message translation if required,
        and put it in the lister titlebar.
 4      Refresh the lister display using FULL so that the titlebar is also
        updated.
 6 - 11 Using a Select...When conditional block, we extract the file to T:
        depending on whether it is an LZX or LhA archive.
13 -14  If there was an error we call the  displayerror  routine to display
        it passing along a translated message text if necessary.
16 - 24 As Edmund says: 'Some creative ARexx programming' :)  But all this
        block of statements really does is create the following script,
        (assuming the file's name is 'foo.bar' and the action was 'Read'):

        address 'DOPUS.1'
        thisfile = 'T:foo.bar'
        command Read "T:foo.bar"
        command wait protect name "T:foo.bar" set RWED
        do until ~exists("T:foo.bar") | delete("T:foo.bar")
          call delay(200)
        end

  And what it does is call the action upon the file in T:, set it's
protection bits to RWED, then loop around until either the file no longer
exists or the file was successfully deleted.

26 - 27 Set the lister titlebar back to it's usual display of the archive
        name and refresh it so that the lister titlebar is updated.
```

```
28      Back to where we came from.
```

1.100 ArcDir.dopus5: Getting all the files

```
  Any selected or dropped files have been passed in one variable called
allents, each filename is surrounded by quotes and seperated by a space.
  This routine seperates allents into individual entries and gets it's type,
for example:

allents = "foo" "bar" "why"
          file  file   dir

  will become:

name.1 = "foo"    type.1 = -1
name.2 = "bar"    type.2 = -1
name.3 = "why"    type.3 = 1

  1getall:
  2  entries = 0
  3  do while allents ~= ''
  4    entries = entries + 1
  5    parse var allents '"' name.entries '"' allents
  6    if name.entries = '' then
  7      type.entries = 1
  8    else do
  9      lister query handle entry '"'name.entries'"' stem fileinfo.
 10      type.entries = fileinfo.type
 11      end
 12    end
 13  return

Line:
 1      Sub-routine label.
 2      Set variable entries to 0.
 3      Loop while allents isn't an empty string.
 4      Increment variable entries.
 5      One of the many wonders of ARexx, here we parse variable allents,
        extracting the first filename into name.entries and returning the
        rest of the string back to variable allents.

Example:         allents = "foo.zip" "bar.zip" "now.zip"
         After:  parse var allents '"' name.entries '"' allents
                 allents = "bar.zip" "now.zip"
                 name.entries = "foo.zip"

 6 - 7  We had an entry with an empty string for a name in which case we
        assume it's a directory and set it's type to 1.
 8 - 11 Otherwise we do a  lister query entry  on it and get it's entry type.
12      Loop around.
13      We return to the calling function.
```

1.101 ArcDir.dopus5: Patching filenames

These routines just deal with the fixing of filenames with strange characters in them. I won't go into them as they don't really have anything to do with Opus, just with the idiosyncracies of AmigaDOS, LZX and LhA.

```
1patch:  /* patch filenames containing strange characters */
2  parse arg patched,doapo
3  if arctype = 'LZX' then
4    strange = '*#?|%()[~'
5  else
6    strange = '*#?|%()[]~'
7  if doapo then
8    strange = strange"'"
9  pos = 1
10 do until here = 0
11   here = verify(substr(patched,pos),strange,'m')
12   if here > 0 then do
13     pos = pos + here + 1
14     patched = insert("'",patched,pos - 3)
15     end
16   end
17 if arctype = 'LHA' & left(patched,1) = '@' then
18   patched = '%'patched
19 if arctype = 'LZX' then
20   if length(patched) - lastpos('/',patched) > 30 then
21     patched = patched'#?'
22 return patched

1patchstar:
2  parse arg remain
3  patched = ''
4  do until remain = ''
5    parse var remain before '*' remain
6    patched = patched||before
7    if remain ~== '' then
8      patched = patched'**'
9    end
10 return patched

1lzxkludge:
2  parse arg string
3  if pos(' ',string) > 0 then
4    do while pos("'*",string) > 0
5      parse var string fore "'*" aft
6      string = fore'?'aft
7      end
8  if pos('*',string) = 0 then
9    string = '"'string'"'
10 return string
```

1.102 ArcDir.dopus5: Getting catalog string

This routine checks to see if a catalog was opened back in Setting Up ,

if there was then the translation will be fetched and returned to the calling function.

```
  1getcatstr:
  2  parse arg msgno,msgstring
  3  if catalog ~= 0 then
  4    msgstring = getcatalogstr(catalog,msgno,msgstring)
  5  do i = 3 to arg()
  6    parse var msgstring fore '%s' aft
  7    msgstring = fore||arg(i)||aft
  8    end
  9  return msgstring
```

```
Line:
 1      Sub-routine label.
 2      From the arguments passed to us, we get the message number used in
        the catalog and the message text.
 3 - 4  If a catalog was opened then we get the translated text using a
        locale.library call.
 5 - 8  Routine  dodelete  is the only routine that calls getcatstr with
        more than three arguments any others will just fall through this bit.
        We parse the string getting the text before and after the %s, then
        format it substituting in arg(3) and arg(4) that were passed, (number
        of files and directories).
 9      Return the translated message text.
```

1.103 ArcDir.dopus5: Checking for valid handler

What's happening here is that we are checking to make sure that the other lister we're using has not got a custom handler and returning the result to whoever called us.

```
  1checkhandler:
  2  lister query otherhandle handler
  3  return ~(result = 'RESULT' | result = '')
```

```
Line:
 1      The sub-routine label.
 2      We ask for the handler associated with the other lister.
 3      OK, if result = 'RESULT' or an empty string then the statement inside
        the brackets is true (1), but we are returning the opposite, hence
        the ~ negation sign outside the brackets.  So if there is no handler
        associated with the lister we will return false (0).
        If result did equal something other than RESULT and an empty string,
        then the statement inside would be false (0) but we would return
        true (1) because of the ~ outside.
```

1.104 ArcDir.dopus5: Syntax error

If a syntax error happens in the ArcDir script, it will jump to here, we'll pass the result code, the error message and the line number to the routine that will call the translate and display routines.

```
  It's routine stuff.

  1syntax:
  2  call displayerror('Syntax Error' rc',' errortext(rc) 'in line' sigl'.')
  3  exit

Line:
 1      Sub-routine label, in this case, for syntax errors.
 2      Call the  displayerror  routine with RC, error text and line number.
 3      It was a terminal error, so we exit.
```

1.105 ArcDir.dopus5: User halts script

```
  Looks like the user used a control-C, or the command HI to terminate the
script, so we'll try to exit cleanly.

  1halt:
  2break_c:
  3  lister set thishandle handler
  4  lister clear thishandle
  5  lister set thishandle path
  6  lister set thishandle title 'ArcDir.dopus5 halted.'
  7  lister refresh thishandle full
  8  lister set thishandle title
  9  exit

Line:
 1 - 2  Sub-routine labels, in this case since we want to do the same things
        if either of these events occur, it's OK to have them both together.
 3      We clear the custom handler for this lister by setting it to nothing.
 4      We clear the contents of the lister.
 5      Set the path to nothing.
 6      Set the title to say this lister is halted.
 7      Do a full refresh so we get to see the changes above.
 8      Set the lister title to the default.
 9      Exit the script.
```

1.106 ArcDir.dopus5: Displaying errors

```
  This routine basically just flashes the screen, changes the lister title
to the message as well as calling the routine to translate the message into
the language set in Locale using the appropriate catalog.

  1displayerror:
  2  parse arg message
  3  lister set handle title message
  4  lister refresh handle full
  5  command flash
  6  call dorequest('"'message'"' getcatstr(4,'OK'))
  7  lister set handle title 'ArcDir:' arcname
  8  return
```

```
Line:
 1      Sub-routine label.
 2      Parse the arguments for the message text.
 3      Set the lister title to the message.
 4      Refresh the lister display, we're using the FULL keyword to update
        the title as well.
 5      Call the Opus internal command Flash, to flash the display and get
        our attention.
 6      Pass the message text to the  dorequest  routine via the  getcatstr
        routine which will do the Locale translation.
 7      After the message has been acknowledged in the  dorequest  routine
        and returned, we change the lister title back to what it normally
        displays.
 8      Return from whence we came.
```

1.107 ArcDir.dopus5: Displaying a requester

This routine displays a requester on the screen with any passed message text, (including buttons).

```
  1dorequest:
  2  parse arg reqargs
  3  if newopus then
  4    lister request handle reqargs
  5  else
  6    dopus request reqargs
  7  return

Line:
 1      Sub-routine label.
 2      Parse the arguments for the message text.
 3 - 6  If we are using an Opus later than 5.1215 then we can use the
         lister request  command, otherwise we'll use the  dopus request
        command to display the message text.
 7      Return from whence we came.
```