

**awnp-docs**

<b>COLLABORATORS</b>
----------------------

	<i>TITLE :</i> awnp-docs		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY		July 31, 2024	

<b>REVISION HISTORY</b>
-------------------------

NUMBER	DATE	DESCRIPTION	NAME

# Contents

<b>1</b>	<b>awnp-docs</b>	<b>1</b>
1.1	AWNPipe-Docs	1
1.2	1 About	1
1.3	2 Read First	3
1.4	3 Pipe Functions	4
1.5	3.1 Overview	4
1.6	3.2 Conversion	5
1.7	3.3 Paths	6
1.8	3.4 Internal	7
1.9	3.5 Extensions	8
1.10	3.6 Tutorials	10
1.11	3.6.1 Simple	11
1.12	3.6.2 Advanced	12
1.13	3.6.3 Tricks	13
1.14	4 GUI Creation	14
1.15	4.1 Conversation	14
1.16	4.1.1 Basic	14
1.17	4.1.2 Step by Step	15
1.18	4.1.3 Details	16
1.19	4.2 Objects	16
1.20	4.2.1 Window	16
1.21	4.2.2 Simple Gadgets	19
1.22	4.2.2.1 Button	19
1.23	4.2.2.2 Integer	22
1.24	4.2.2.3 String	23
1.25	4.2.2.4 CheckBox	24
1.26	4.2.2.5 Chooser	26
1.27	4.2.2.6 RadioButton	27
1.28	4.2.3 Images	28
1.29	4.2.3.1 Label	28

1.30	4.2.3.2 Glyph . . . . .	29
1.31	4.2.3.3 DrawList . . . . .	30
1.32	4.2.3.4 PenMap . . . . .	30
1.33	4.2.3.5 Bitmap . . . . .	31
1.34	4.2.3.6 Image . . . . .	32
1.35	4.2.3.7 Space . . . . .	32
1.36	4.2.4 Special . . . . .	33
1.37	4.2.4.1 Layout(End) . . . . .	33
1.38	4.2.4.2 ClickTab . . . . .	35
1.39	4.2.4.3 TextAttr . . . . .	36
1.40	4.2.4.4 BrowserNode . . . . .	36
1.41	4.2.4.5 Menu . . . . .	37
1.42	4.2.4.6 Arexx . . . . .	38
1.43	4.2.5 Fancy . . . . .	38
1.44	4.2.5.1 GetFile . . . . .	38
1.45	4.2.5.2 GetFont . . . . .	42
1.46	4.2.5.3 TextEditor . . . . .	43
1.47	4.2.5.4 TextFeild . . . . .	46
1.48	4.2.5.5 ListBrowser . . . . .	47
1.49	4.2.5.6 Palette . . . . .	50
1.50	4.2.6 More . . . . .	51
1.51	4.2.6.1 Fuelgauge . . . . .	51
1.52	4.2.6.2 Scroller . . . . .	53
1.53	4.2.6.3 Slider . . . . .	54
1.54	4.2.6.4 WeightBar . . . . .	55
1.55	4.2.6.5 Commodity . . . . .	56
1.56	4.2.6.6 Sound . . . . .	57
1.57	4.3 Events . . . . .	58
1.58	4.4 Modify . . . . .	58
1.59	4.5 GUI Tutorial . . . . .	62
1.60	4.5.1 Design . . . . .	62
1.61	4.5.2 Operation . . . . .	63
1.62	4.5.3 Modification . . . . .	64
1.63	4.5.4 Extras . . . . .	67
1.64	4.5.5 Advanced . . . . .	67
1.65	4.5.6 Tips . . . . .	69
1.66	5 Demos . . . . .	69
1.67	AWNPipe Index . . . . .	71

---

# Chapter 1

## awnp-docs

### 1.1 AwnPipe-Docs

AWNPipe - Docs =====

**1** About **2** Read First **3** Pipe Functions **4** GUI Creation **5** Demos

### 1.2 1 About

1 About -----

AWNPipe: Multifunction Device Author: William H. M. Parker Version: 2.54 31 Jan 2001

AWNPipe is a new kind of ADOS device. It's functions can be accessed from almost any program including C, E, Assembler, Arexx, and simple ADOS scripts. AwnP has many uses but the most popular is the creation of Graphical User Interfaces. Other functions include clipboard support, tooltype access, pattern matching, Html conversion, data stream twinning, ... Programmers will find that GUI development is VERY fast using AwnP and testing various layouts is as easy as editing a text file.

Programs written using AwnP present the user with a GUI that is easy to use and consistent in form. The ease of adding menus, Arexx hosts, help information and images encourages authors to add more of the 'Nice' features to their programs.

Distribution

AWNPipe can be freely distributed in the form of this archive, complete and unmodified.

The operational parts (AWNPipe-handler and AwnPipe) may be included in the distribution of programs using AwnPipe if the following three conditions are met.

1. The distribution contains an install script and the AwnPipe files are handled as follows (NOTE the use of 'copylib' for awnpipes-handler).

```
(copylib (source "device/awnpipes-handler") (dest "I:") (prompt "Installing AwnPipe-handler") ) (copyfiles (dest "Devs:dosdrivers")
(source "device/awnpipes") (prompt "Installing AwnPipe") (infos) ) (working ("Activating AwnPipe:")) (run "C:assign AwnPipe:
dismount") (run "C:mount AwnPipe:")
```

2. The docs mention the use of AwnP and the version number of AwnPipe in the distribution.

3. The AwnPipe author is notified, bill@amitrix.com .

This software is CHILDWARE. I require whoever uses this program to make a donation to a beneficial organization working to help children. If you don't know of any, ask at your local post office and learn how to make a payment to UNICEF. The amount is up to you, but please do it!

Requirements

---

AWNPipe requires ADOS 3.x You must have Class Act installed or be using ADOS 3.5+ to use the pipes GUI building functions. Class Act can be found on Aminet, or use <http://www.amitrix.com/AWeb30ca.lha> (the latter has been tested for compatability with AWPipe).

## Support

Support is currently available from :

<http://www.yahogroups.com/community/awnpipes> <http://web.ukonline.co.uk/awnpipes/> or directly from the AWP Author, [bill@amitrix.com](mailto:bill@amitrix.com)

## History

- o AWPipe-handler Vers 2.53 -added hook type to string gadget.

- o AWPipe-handler Vers 2.53 -added 'target' option to 'drawme' modify command -added 'idme' keyword to allow tracking of labels,images and space gadgets. -added 'bounds' modify query to get object size and location. -added 'knob' and 'maxn' to scroller modify

- o AWPipe-handler Vers 2.52 -fixed handling of '&#???'; in unhtml function -added '&quote;' and '&copy;' to unhtml function -added '/xz' option to remove html tags <...> -added '/xp' option to get information on public screens -worked around listbrowser hit when presetting selected nodes -added 'xw' fast simple text wrapping -taps now wait for both ends of the pipe to close before closing. -added patterned backfill support to layout groups -added screen to front/back window modify command (GID 0) -added BD keyword to arexx host return line

- o AWPipe-handler Vers 2.51 -added fast release of images used in a button object.

- o AWPipe-handler Vers 2.50 -added sound object -fixed '/v' to handle clips with multiple char chunks. -fixed slider modify

- o AWPipe-handler Vers 2.49 - added label softstyle 'softstyle=' (ss=) - added hidechild modify to listbrowser - added hidechild modify to browser node - removed some duplicate events from attached getfile gadgets - fixed bitmap error response - removed some extraneous 'ok's - maximum predefined images increased to 50 - maximum defined master bitmaps increased to 10 - maximum layout group nesting increased to 50 - replacing a listbrowser no longer gives enforcer hits - PIPE NAMES ARE NO LONGER CASE SENSITIVE ('/' options should be lowercase)

- o AWPipe-handler Vers 2.48 - added work around for disable of clicktabs - added the ability to disable a single tab of a clicktab - fixed docs, disable keyword description was backwards in some places - '/m' multiple opens now work when reading data. - added '/m' explanation to pipe functions/tutorials/tricks - cleaned up docs in various places. - added plain text version of these docs - added Amiga Guide version of these docs - fixed bug in XO.rx demo

- o AWPipe-handler Vers 2.47 - added '/xs' (seticon) ability to write icon tooltypes, position, type, and defaulttool - expanded tutorial5 to cover writing tooltypes - GUI advanced doc has been expanded

- o AWPipe-handler Vers 2.46 - trailing '/' in no longer required in drawers only getfiles. - fixed asl getfile loss of initial title - fixed asl getfile missing space in event - fixed asl getfile retention of pos and neg text - worked around for missing refresh in multiselect listbrowser

- o AWPipe-handler Vers 2.45 - expanded drawlist commands (changed some directive values as well) - corrected Browsernode and listbrowser docs - added refresh ability to browsernodes - fixed multiselect listbrowser bug

- o AWPipe-handler Vers 2.44 - fixed bug ScreenTitle could be trashed when setting WindowTitle. - binary data for GUI is no longer limited to 10000 bytes - added '/xt' option to query icon tooltypes - fixed bug An extra 'ok' was returned when an addnode modify line included sc= - expanded readme file - tutorial 5 now checks tool types of its icon - Advanced GUI tutorial text updated - Added sanity check, you can't force gadget refresh while window is iconified - The dictionary demo is now a thesaurus as well (docs/demos/dict-thesar.rx)

- o AWPipe-handler Vers 2.43 - fixed return from commodity modify. - Added underscore keyword to label image

- o AWPipe-handler Vers 2.42 - Added Commodity Object, see GUI Creation/Objects/More. - Tutorial 5 is now a commodity. - Tutorial 5 is now an arexx host. - updated advanced gui tutorial docs.

- o AWPipe-handler Vers 2.41 - added order modify keyword to retrieve the order of nodes in a list browser - the CAList demo is now called GUIList and a short doc file added - the FontToy demo has been updated and a short doc file added

- o AWPipe-handler Vers 2.40 - fixed bug in ASL getfile title modify - added some important text to the getfile docs. - fixed bug, GUI Host could hang on selection of invalid browsernode - fixed bugs ASL getfile, height parameter, filename - added function, removenode now works with list detached - Listbrowsers can now be sorted by two fields at once. - added option to set positive

text is ASL getfile - added option to set negative text is ASL getfile - No more deadlocks when a read is pending on both ends of a pipe !!! - refined help window size and placement

o AWNPipe-handler Vers 2.39 - added '/Xea' option to generate keystrokes from a simple ANSI source. - fixed arexx host creation (could cause enforcer hits) - changed fonttoy demo

o AWNPipe-handler Vers 2.38 - oops 2.37 has new 'beep' modify command to flash screen(s). - added keystroke event generator. '/Xe' - added keystroke filter and notification '/Xk' - optimized A4 initialization for faster handler.

o AWNPipe-handler Vers 2.37 - added 'bufferpos' to read cursor position in string gadgets. - fixed bug, unmatched commands to an arexx host could cause software failure. - fixed cut and paste tutorial text in the docs. - expanded tutorial 5 teaching about ARexx hosts. - added sliders to gadgets 3 demo.

o AWNPipe-handler Vers 2.36 - started real history record - added 'askclose' to window definition to stop window close button from actually closing the window. - added 'bubble' modify command to open a help bubble. - added cursor position to help events. - added 'ASL' option to getfile to use an ASL rather than ClassAct/Reaction based requester. - added work around for the fact CA/Reaction GetFile and GetFont can trash the GUI when help events are enabled. - added tutorial 5 teaching bubble help. - added 'weightbar' gadget.. - 'label' and 'bitmap' now return the created image size.

o AWNPipe-handler Vers 2.33 and earlier - added 'draw' modify command to draw images manually into GUI. - added 'refresh' events to window to allow refreshing of manually drawn images. - added 'mouse' modify command to read mouse position. - added 'slider' gadget type. - added REAL menu bars. - made menu mutiselect. - extended appwindow support to allow dropping of a group of icons. - added some 'C' based tutorials, anyone using awnp from 'C' please contact the AWP author. - added unit number option to '/v' and '/c' reading and writing the clipboard. - fixed bug in modification of scroller gadgets. - added a bunch of new modify commands for list browsers. - added command to read the contents aof a list node. - added 'TextEditor' gadget type.

### Origin

AWNPipe was originally developed as a way to speed up the execution of my ARexx script AWebNews.awebx. As I continued writing scripts I added more and more functions to the pipe. Some of the functions have proven useful to others writing Amiga programs. I built these docs in the hope that AWPipe: will be of use to Amiga programmers in general.

### Acknowledgements

I would like to thank Gabriele Favrin (author of HTTX) for his input, testing and constant encouragement while writing AWPipe.

I also thank...

Bruce Steers for maintaining the AWP support page and finding many subtle bugs.

Nils Goers for his testing under ADOS 3.5 and developing his wonderful program T.H.E. using AWP, as well as his bug reports.

Bernd Gollesch for writing the script that creates the Amiga Guide and plain text versions of the docs automagically.

Everyone else who provided bug reports or feed back helping me make AWP better.

Bill

## 1.3 2 Read First

### 2 Read First -----

AWNPipe itself is stable and quite well tested. These docs are still a work in progress so please excuse typos and omissions.

At first glance AWPipe may appear complex. Actually each of the functions in the pipe are simple to use when taken individually. Don't try to learn everything at once, pick a single function and try it out. Each function can be used without knowing anything about the other functions.

Most of you will be primarily interested in building and operating GUI's. You can go directly to the GUI Creation section and start there. The important part to read about is the conversation, then try the tutorials in that section.

To learn about pipe functions go straight to Pipe Functions/Tutorials/Simple and work through the examples.

Right Amiga C can be used to cut and paste example text from the docs. This helps to avoid typos.

Feel free to contact me, <bill@amitrix.com>, with your questions and comments.

Have Fun !

## 1.4 3 Pipe Functions

3 Pipe Functions -----

[3.1 Overview](#) [3.2 Conversion](#) [3.3 Paths](#) [3.4 Internal](#) [3.5 Extensions](#) [3.6 Tutorials](#)

### 1.5 3.1 Overview

3.1 Overview -----

The pipe function sections are NOT required reading to learn how to build GUI's.

Using AWNPipe -----

All of AWNPipes functions are accessed through a pipe like interface as the name implies. Pipes are accessed like any other type of file using open, close, read, and write. From Amiga DOS a pipe can be used almost any place you would normally use a file name.

The name of each pipe starts with the device name 'awnpipe:'. This is followed by a unique name that is used to identify the pipe. The name MAY also be followed by a '/' and some option information to give the pipe special abilities. 'awnpipe:myfile' and 'awnpipe:test' are examples of pipe names.

The options after the '/' usually cause the data to be altered as it passes into the pipe. Then data you read out from the pipe will be different than the data you wrote into it. The '/h' function works this way, any '&' written into the pipe will be read back as '&amp;'.

Some other options connect the pipe to special hosts like the clipboard. Data written to a pipe ending in '/c0' goes into the clipboard unit 0. The data read from a pipe ending in '/v0' will be the contents of clipboard unit 0.

Here is a simple example showing data going in and out of a pipe.

```
echo "hello World" > AWNPipe:test type AWNPipe:test
```

This following information is intended as a reference for people already familiar with AWP. To learn how to use pipe functions work through the tutorials.

Pipe Types -----

/i immediate /u unHTML /e execute /l link /r read /p Postaweb /o <option> /v[unit#] readclip /w write /! do not wait /b backwards /c[unit#] writeclip /h HTML /t tap /f force /s status /g HTML2 /a abort /-vers# minimum version /@ seek enable /m Multiple Opens /l open first end

/x[option] extra function /xc open interactive Class Act pipe /xcr[filename] open interactive Class Act pipe, reading the window and gadget definitions from a separate file. /xcw[filename] open interactive Class Act window, writing the events to a separate file.(also sets immediate and force) /xm[c] string pattern match c=case sensitive /xm[c]w string pattern match c=case sensitive results to separate file /xm[c]r string pattern match c=case sensitive reading data from separate file. (also sets immediate and force) /x0 programmable replace loopback /x0r[filename] programmable replace from file /x0w[filename] programmable replace to file /xk remove keystrokes from input event stream and notify /xe write keystrokes to the input event stream from code & qual /xea write keystrokes to the input event stream from ANSI source /xt[filename] query icon tooltypes /xi[filename] read tooltype information for a file. /xs[filename] set tooltype information for a file. /xz remove html tags <...> loopback /xzc[filename] remove html tags <...> from file /xzw[filename] remove html tags <...> to file /xp get public screen names loopback /xpw[filename] get public screen names to file /xw[length] text wrap loopback /xw[length]r[filename] text wrap from file /xw[length]w[filename] text wrap to file

Details about AWNPipe -----

AWNPipe:test/h AWNPipe:TEST AWNPipe:test/h/f/rhelpme are all the same file. The unique name is NOT case sensitive, and terminates on the first '/' or the end of the name if no '/' is found. '%' may be substituted for '/' . Options after the '/' should be given in lower case.

A pipe name is usually only opened twice. After that calls to open will fail. After both ends of the pipe are closed the pipe name can be used again. A pipe can immediately be written after it is opened, even if the other end has not yet been opened. Some special options can override this behavior.



Some types of pipes are only opened ONCE. The second end of these pipes are automatically connected to something by the AWPipe internal code. Clipboard access is an example of this type of pipe.

All file handles can both read or write data. A check is made to see if pipes dead lock from pending reads on both ends. If this happens both of the pending reads are aborted to break the deadlock.

Data flow is never stopped by buffering.

If data is written into a pipe and the second end never opened, the pipe stores all written data.... forever (until you reboot).

## 1.6 3.2 Conversion

### 3.2 Conversion -----

#### Data Conversion -----

The data passing through the pipe may be modified by the pipe with the following options.

#### H option =====

html conversion - some special characters are translated into html tokens. & > < become &amp; &lt; &gt; This option must be specified on the file handle that is written.

```
copy ram:test AWPipe:test/h copy AWPipe:test ram:test2
```

#### G option =====

The same as the H option above, except that any special html character preceded by an '@' is not converted. The preceding '@' is removed during the translation process.

#### U option =====

UNhtml conversion - some html tokens are translated into special character. &amp; &lt; &gt; become & > < Tokens of the form &#NUM; are also translated. This option must be specified on the file handle that is written to..

```
copy ram:test AWPipe:test/u copy AWPipe:test ram:test2
```

#### P option =====

AWebPost conversion - some special characters are translated into standard ascii. This must be specified on the file handle that does the writing.

```
copy ram:test AWPipe:test/p copy AWPipe:test ram:test2
```

#### B option =====

backwards blocks - characters read from a pipe are returned in reverse order of the character blocks written. This must be specified on the file handle that does the reading.

if you write 5 blocks of characters .....

```
'hello' 'world' '1' '2' '3'
```

they are read back as

```
321worldhello
```

Funny things can happen if you read before all writes are completed.

#### O option =====

Adds the text '<option>' after every '0a'x . I use it in AWebNews.

```
copy ram:test AWPipe:test/o copy AWPipe:test ram:test2
```

## 1.7 3.3 Paths

### 3.3 Paths -----

Controlling data paths. -----

T option =====

TEE's or data taps. A tap is a extra read handle on a pipe connection. If the pipe is not yet created the tap will wait for it to be created. A tap does not see any data written to the pipe before the tap was created. Taps are useful to listen in on interactive (2way) pipes as they read the data written to both ends of the pipe.

Try this is 3 separate shells.

```
type AWPipe:test/t type AWPipe:test echo > AWPipe:test "test data"
```

R option =====

Read a file. The second end of a pipe can be connected to file automatically. The file name is given after the 'R'.

Create a file 'ram:test' containing some text including the characters '<>'. Now try this in a shell.

```
type AWPipe:test/h/ram:test
```

The data from ram:test is read through a modifying pipe.

W option =====

Write a file. The second end of a pipe can be connected to file automatically. The file name is given after the 'W'.

Create a file 'ram:test' containing some text including the characters '<>'. Now try this in a shell.

```
copy ram:test AWPipe:test/h/wram:test2
```

The data from ram:test is copied into a modifying pipe, the pipe automatically outputs the data into a second file ram:test2 .

L option =====

Read AND write to an interactive file such as con: . The interactive file name is given after the 'L'.

This is useful to be able to 'tap' into a two way conversation at a con: or other interactive file handle. Instead of opening con: directly, open AWPipe:test/Lcon:///mycon/ Now you can open AWPipe:test/t to tap the data exchanged with the con:.

V option =====

This option opens a pipe directly to the clipboard.

```
type AWPipe:test/v
```

You may also specify a clipboard unit, defaults to 0 (the primary clip).

To access clipboard unit 5

```
type AWPipe:test/v5
```

C option =====

This option allows you to set the clipboard

```
echo "set this as the clip" >AWPipe:test/c
```

You may also specify a clipboard unit, defaults to 0 (the primary clip)

To set clipboard unit 5

```
echo "set this as the clip" >AWPipe:test/c5
```

E option =====

execute a command . The file name is given after the 'E'. A command is executed with the pipe name as an argument. The pipe name replaces a '%' or is placed at the end. You CAN NOT use a '/' instead of a '%' !.

```
copy ram:test.doc AWPipe:test/h/eweb3:aweb-II
```

```
copy ram:test.doc "AWPipe:test/h/eweb3:aweb-II % config local"
```

Both of these load a file ram:test into AWeb through a modifying pipe and call AWeb to view it .

Into a file requester such as AWebs save requester... 'awnpipe:jpeg/eUtil:fjpeg\_ecs' will send the data directly to the viewer.

## 1.8 3.4 Internal

### 3.4 Internal -----

#### Internal controls. -----

These options control some of the behaviours of Awnp regarding opening pipes and reading data.

#### - option =====

Specify a minimum version of Awnp you need. The format is /-VVRRR where V is the version and R is the revision. Opening Awnp:name/-02009 will fail to open unless Awnp is version 2.9 or newer.

#### A option =====

Abort pending reads. If you try to open a pipe using the '/a' option any pending reads are aborted. This USUALLY causes the pipe to shut so you can reuse the pipename. It is seldom if ever needed now since Awnp catches most problems and shuts down 'hung' pipes automatically. The open itself ALWAYS fails, this is intended.

#### M option =====

Multiple opens. When a pipe is opened with this option it behaves normally. However when this file handle is closed that end of the pipe becomes available to be opened again. If you open the pipe name again with the '/1' (open first end) flag is set the open call only tries to open the first side of the pipe (the end that was opened when the pipe was created). If the donotwait flag is set the open call only tries to open the second side of the pipe (the end that was NOT opened first). When neither flag is set you get the first end of the pipe if available, else you get the second end.

#### I option =====

Immediate reads. Reads to the file handle will return if there is any data waiting to be read. If you try to read 10 characters from the pipe and only 5 are available, the read returns with the five characters immediately (rather than waiting for 5 more characters to be available). Therefore only a read of 0 length means end of file. This is useful with data originating interactively from con: or ser: .

#### F option =====

Force a pipe to be created. A new pipe is created even if the pipe name is already in use. The open will not fail, and it will not connect to any currently existing pipe end. After both ends of the forced pipe are opened any previous partial ( only opened once) pipe of that name becomes available again. (ONLY FORCE THE FIRST END OF THE PIPE)

#### ! or ~ option. =====

Do not wait. The open will fail unless the other end of the pipe is already open. A new pipe will not ever be created. Used on a tap it stops the tap from waiting for a matching pipe to be created.

#### @ option =====

Respond to seek packets. Normally seek is not supported by Awnp. If seek is turned on, the current position is reported as '0' if no data is available on the pipe, '1' if data is waiting to be read.

NOTE Do not use this function from ARExx as ARExx has a read ahead buffer that is flushed when you call seek. This can cause you to lose data. It may also happen in other languages if read data is buffered.

#### S or s option =====

Status of a pipe. This MUST always be a tap as well. Read only returns a single byte. This is the status of the real pipe the tap refers to. The bits/nibbles or byte meanings are as follows.

'FF' indicates file handle does not exist. (you can check just the high bit)

low nibble = First end that is opened. high nibble = second end opened.

in each nibble

bit 4 always zero when FH exists bit 3 FH opened. (nibble&4) bit 2 FH closed. (nibble&2) bit 1 FH data available. (nibble&1) sub for WaitForChar in AREXX.

Note 'FF' indicates no file handle active.

## 1.9 3.5 Extensions

### 3.5 Extensions -----

#### X or x option =====

These options let you open a pipe whose second end is connected to various special hosts. (you only open these pipes once)

The X option is always followed by additional character(s). The second character determines what type of pipe is opened.

#### Xw[length][(rlw)filename] (simple text wrap) -----

This pipe wraps text at 'length' characters. It DOES NOT rewrap (lines shorter than length chars are not joined into longer lines. Length defaults to 80 if it is not specified.

When using this function in loopback mode each 'chunk' of text to be wrapped MUST end in a linefeed. This allows the same pipe to be used to wrap several seperate 'chunks' of text. (loopback mode is when you do not use [(rlw)filename])

#### X0[(rlw)filename] (Programmable replacement) -----

Programmable replacement of a character. A single character is looked for in data written to the pipe and replaced by a multy character string.

First write two bytes to the pipe. the first byte is always '1' the second byte is the target character to be replaced. Then write a byte giving the length of the replacement string (<=255) followed by the replacement string. Do NOT include a terminating null .

Data written to the pipe after this point is echoed back to the pipe with any occurrences of the target character replaced by the specified string.

X0wfilename causes the data to be echoed to a file rather than back to the pipe.

X0rfilename causes the data to be read from a file rather than from the pipe. Note that the target character and replacement string is still read from the pipe not the external file. Only data to be parsed and replaced is read from the file.

#### Xz[mode][(rlw)filename] (process html) -----

The function used depends on 'mode'. Mode is a bitmapped decimal.

bit 1 (1)- Remove all characters between '<' '>' including the '<' and '>'. Condence bit 2 (2)- condence multiple spaces to a single space bit 3 (4)- do not condence at the start of a line. bit 4 (8)- replace '0a' characters with spaces bit 5 (16)- replace '0d' characters with spaces bit 6 (32)- replace tab characters with spaces bit 7 (64)- replace <br> and <p> with newline characters bit 8 (128)- format tables with newlines <td> <th> <table> <tr> bit 9 (256)- format lists with newlines <li> <ul> <ol>

The rest of the characters pass unaltered. Sum the numbers in brakets to select 'mode'. Mode defaults to 511 (all on).

EXAMPLE: Writing '1<2>3' to the pipe returns '13'.

Xzwfilename causes the data to be echoed to a file rather than back to the pipe.

Xzrfilename causes the data to be read from a file rather than from the pipe. Note that the target character and replacement string is still read from the pipe not the external file. Only data to be parsed and replaced is read from the file.

#### Xp[(w)filename] (get public screen information) -----

This pipe is only read, never written to.

This pipe returns the name of the front public screen on the first line. The default public screen name on the second line. Then information on ALL public screens follows, two lines each screen. The first line has the screen name, the second the following information 'Top Left Height Width Flags ScreenMode' All values are given in hexadecimal.

Example: 'type awnpipexp'

DM.1 (front screen) Workbench (default screen) Workbench (1st screens name) 0 0 320 4B0 4651 4002 (1st screens info) AWeb (2nd screens name) 0 0 320 258 4652 6002 (2nd screens info) DM.1 (3rd screens name) 0 0 320 258 4652 6002 (3rd screens info) DM.2 (4th screens name) 0 0 320 258 4652 6002 (4th screens info) TURBOTEXT (5th screens name) 0 FFFFFFFF 14 2CA 1D8 6212 F004 (5th screens info)

Xpwfilename causes the data to be echoed to a file rather than back to the pipe.

## ICON HANLDING =====

Xt[filename] (find Tooltypes) -----

This host is the best suited to quickly check for tooltypes and get there values.

The icon file for the specified file (you do NOT add '.icon') is queried.

Open the file awnpip:myname/Xt[filename]. Write a line containing the tooltype you wish to query to the pipe. Read back a response line.

If no icon file is found an eof is returned.

If the tooltype is NOT found you will read back a null line (only a <cr>).

If the tool type IS found you will read back 'ok ' followed by the tooltypes value (if it has one).

Xi[filename] (Icon information) -----

This host will retrieve fuller details about tooltypes and some other icon information. It is often used in combination with the following host.

The icon file for the specified file (you do NOT add '.icon') is read. The pipe returns the following lines...

icon\_type x y stack default tool tool window first tooltype second tooltype ...

If no icon file is found no data is returned. Note that for blank lines will be returned when the information does not apply to that icon type.

example: awnpip:/xidevs:dosdrivers/awnpip returns

4 3 139 4096 C:Mount

ACTIVATE=0

To get information on a volume (say dh1:) use 'awnpip:/xidh1:disk'.

Xs[filename] (Set icon information) -----

This host will alter or create the icon and tooltypes for the specified file. Data is written to the host in the same format as returned by the preceding host.

icon\_type x y stack default tool tool window first tooltype second tooltype third ...

Icon\_type, x, y, and stack are all integer values written on a single line.

Default tool and the tooltypes are ascii strings.

Tool window is ignored but MUST still be included, either use a blank line ('0A'x) or used the data returned by the Xi host.

The first three lines are always required. You may set as many tooltypes as you like including none.

some useful information:

setting x and y to -2147483648 unsnapshots the icon. icon\_type 1=disk 2=drawer 3=tool 4=project 5=trashcan

Xm[c][(rlw)filename] (pattern Matching) -----

Pattern match Conversation. This conversation supports full ADOS pattern matching. The trailing 'c' will make the match case sensitive.

Xmwfilename causes the match result to be sent to a file rather than back to the pipe.

Xmrfilename causes the data to be read from a file rather than from the pipe. The match results are read back from the pipe.

NOTE: the replies are ascii terminated with a newline.

Write the pattern to the pipe terminated with a newline. (<500 chars) The pipe returns 'ok 1' when you sent a valid pattern to match to. It returns 'ok 0' if you did not send a pattern ( the pipe will look only for exact matches).

write a string to the pipe terminated with a newline.<500 chars) read the reply '0' (no match) '1' (is a match)

write as many stings as you like . Close pipe to end. The terminating newlines are ignored while matching.

## KEYSTROKE HANDLING =====

**Xk (Keystroke filter) -----**

Filter keystrokes from the input device event stream and receive notifications of the keystroke.

First write a byte to the pipe setting the priority of the key parse handler. It is a signed byte -128 to 127. If you are not sure what you want use 51 (0x33).

Next write a byte to the pipe indicating how many keystrokes to filter. The maximum is 255 (0xff). 0 is NOT valid.

0x'0a' is 10 keys.

Then write two bytes to the pipe indicating how often a null match should be sent (to stop your task waiting forever for a match that never happens).

0x'0032' is 50/100s of a second.

Now write 4 bytes defining each key you wish to be notified of. The first two bytes are the key the second two bytes are the qualifier.

Writing 0x'00200001' to the pipe sets the 'a' key 'leftshifted' to be filtered from the event stream.

You MUST set exactly the same amount of keys as specified above.

Once the setup is complete you read notification from the pipe. Each notification is 1 byte giving a match number, the first keystroke you specified is 1, the second 2 ...

A byte of 0 is a null event, no match occurred.

Close the pipe to end key filtering. NOTE the keyparse handler actually does not get removed until the NEXT match or null event.

**Xe (key Event generator) -----**

Write keystrokes to the input device event stream.

Each keystroke is sent by writing 4 bytes to this pipe. The first two bytes are the key the second two bytes are the qualifier.

Writing 0x'00200001' to the pipe sends the 'a' key 'leftshifted'.

You may close the pipe at any time.

**Xea (key events form Ascii) -----**

Write keystrokes to the input device event stream from a string source.

Each keystroke is sent by writing a byte to this pipe. You do NOT need to write one character at a time. Simply write ascii text to the pipe and it will generate keystrokes as if the string had been typed at the keyboard.

This is easier than writing keycodes and qualifiers to the pipe but you can not generate 'special keystrokes' like Alt-F10 .

You may close the pipe at any time.

**Xc[(rlw)filename] (ClassAct/Reaction host) -----**

ClassAct Window Conversation. The Window and gadgets are defined by writing to the pipe:. Gadget hits and error/confirmation information is read from the pipe. A filename may also be given. If the file is specified as write 'w' the output of the pipe is directed to that file. If the file is specified as read 'r' the window and gadget definitions are read from the file.

See the GUI Creation section of the docs for details.

## 1.10 3.6 Tutorials

**3.6 Tutorials -----**

**3.6.1** Simple **3.6.2** Advanced **3.6.3** Tricks

## 1.11 3.6.1 Simple

### 3.6.1 Simple -----

HINT: Drag select the text then use right Amiga C to copy the examples out of these docs. This helps avoid typos. ---

The simplest use of awnpipes is as an ordinary pipe. That means put data into one end of a pipe and read it out the other.

In a shell type `'echo >awnpipes:mypipe hello'` Now type `'type awnpipes:mypipe'`

The word hello went into the pipe from the echo command, and came out with the type command.

`'mypipe'` was the name of the pipe. More than one pipe can exist at the same time.

In a shell type `'echo >awnpipes:mypipe1 hello'` In a shell type `'echo >awnpipes:mypipe2 goodbye'` In a shell type `'type awnpipes:mypipe1'`

In a shell type `'type awnpipes:mypipe2'`

Two pipes each with a different name.

---

Make sure you have two different shells available.

In the first shell type `'type awnpipes:mypipe'` In the second shell type `'echo >awnpipes:mypipe hello'`

You opened the read end of the pipe first. The type command waited until data was available from the pipe THEN returned.

A word of caution. Don't try to read from both ends of the pipe at the same time. If you `'type awnpipes:mypipe1'` in two separate shells both will hang waiting for data to come from the pipe. AWNPipes notices this and cancels both reads to break the deadlock, this causes both type commands to end without typing any data.

---

To use AWNPipes for other functions you add `'/'` followed by some parameters to the pipe name. `'/h'` will cause the pipe to translate a few special characters to their HTML equivalents. Note that the `'/h'` is only used when writing to the pipe.

In a shell type `'echo >awnpipes:mypipe/h "& > <"'` In a shell type `'type awnpipes:mypipe'`

The data is modified as it pass through the pipe.

See the advanced examples for more types of data conversion.

--- It is possible to have a pipe automatically connect it self to a file. In these cases you only open the pipe on one end, the other end is automatically connected to a file.

`'/rFILENAME'` reads the contents of FILENAME and places it in the pipe.

Create a file `'ram:test'` containing some text including the characters `'<>'`.

Now try this in a shell.

`type AWNPipes:test/h/ram:test`

(YES there are two colons `':'` in the pipename. Strange but don't let it bug you.)

The data from `ram:test` is read through a modifying pipe.

---

`'/wFILENAME'` writes the output of the pipe to FILENAME.

In a shell type `'echo >awnpipes:mypipe/h/wram:AWNPtest "& > <"'`

Take a look at the file `ram:AWNPtest` (with your favorite text editor).

---

`'/u'` reverses the translation done by `'/h'` and `'/rFILENAME'` reads data into a pipe.

In a shell type `'type awnpipes:mypipe/u/ram:AWNPtest'`

The contents of the file are read and translated.

---

It is possible to have a pipe automatically connect it self to the clipboard.

In a shell type `'echo >awnpipe:mypipe/c hello'`

The word hello was placed into the clipboard. Use your text editor to check it out. Then put different text into the clipboard using your text editor.

In a shell type `'type awnpipes:mypipe/v'`

The contents of the clipboard are typed out.

## 1.12 3.6.2 Advanced

### 3.6.2 Advanced -----

You should read the simple examples before trying these more advanced ones.

HINT: use right Amiga C the drag then clip the examples out of these docs to avoid typos.

---

It is possible to tap into or duplicate the data passed in a pipe. `'/t'` will add a readonly third end to a pipe.

Make sure you have two different shells available.

In the first shell type `'type awnpipes:mypipe/t'`

In the second shell type `'echo >awnpipe:mypipe hello'` In the second shell type `'type awnpipes:mypipe '`

The word hello came back from both the second end of the pipe and the 'Tap' third end.

---

In the first shell type `'copy con://///pipe-test/close awnpipes:mypipe'` In the second shell type `'type awnpipes:mypipe '`

Type a few lines of text into the test con:, nothing comes out in the second shell. Close the test con:, now the data comes out. This is because the type command wants to read large blocks of data.

`'/i'` will cause data to pass out of the pipe immediately (as soon as its available).

In the first shell type `'copy con://///pipe-test/close awnpipes:mypipe'` In the second shell type `'type awnpipes:mypipe/i '`

Now the text comes out of the pipe each time you hit return in the test con:. Close the test con: to end the example.

---

You can use AWNPipe to read the tooltypes of an icon. The second end of the pipe automatically reads the tooltypes of FILE-NAME when `'/xiFILENAME'` is used. NOTE that parameters that start with `'/x'` can not be mixed with other parameters!

In a shell type `'type awnpipes:mypipe/xiDevs:dosdrivers/awnpipe'`

YES the pipe name contains two colons `':'`, don't let it bug you.

The tooltype information from Devs:dosdrivers/awnpipe prints out. Note you do not include `'info'` in the file name.

The data is typed out in the following format..

icon\_type x y stack default tool tool window first tooltype second tooltype ...

If you get some weird number (-2147483648) for the X and Y values it means you did not snapshot the icon yet.

---

You can also use AWNPipe to write the tooltypes of an icon.

Copy a file with its icon into ram:., preferably an icon that has some tooltypes set in it.

In a shell `'copy awnpipes:/xiRam:filename ram:icondump'` , do not include `'info'` in the filename.

Now examine and edit the file ram:icondump change the default tool and tooltypes.



In a shell 'copy ram:icondump awnpipes:/xsRam:filename '.

Use the WB menu info function to verify the new setting in the icon.

---

AWNPipes can be used to pattern match. '/xm' is used for pattern matching. Data is usually written into the pipe, and the results of the pattern matching read back from THE SAME END OF THE PIPE. The second end of the pipe is automatically connected to a task that is doing the pattern matching. '/xmFILENAME' causes the data to be read from a file rather than the pipe, the match results are read from the pipe.

In a shell 'type awnpipes:mypipes/xmrcon:///pipetest/close'

In the pipetest con: type '#?a' . This sets a pattern to match any text ending in 'a'. 'ok 1' should come back from the pipe. ('ok 0' would mean you did not enter a pattern so an exact match will be tested for.)

In the pipetest con: enter some text. If the text ends with 'a' then the pipe responds with '1' meaning a match. If not the pipe responds with '0' meaning no match. Close the pipe test con: to end the example.

---

Several functions can be used when processing html documents.

'/xz' will remove html tags. This means a '<' and all following characters until a matching '>'.

Make a simple text file called 'file.html' as follows (or uses any html you already have).

'hello <tag>world <another tag> this is <tag>a test <another tag> 1234567890123456789012345678901234567890'

In a shell 'type awnpipes:/xzrpath:file.html'

'/xw[length]' allows you to wrap text. (Text parsed from html files is usually unwrapped.)

Use the same file as above.

In a shell 'type awnpipes:/xw5rpath:file.html' In a shell 'type awnpipes:/xw20rpath:file.html' In a shell 'type awnpipes:/xwrrpath:file.html'

## 1.13 3.6.3 Tricks

### 3.6.3 Tricks -----

Now for some strange looking but powerful ideas. You don't have to understand them all just give them a try. Some interesting things can be done by setting a few simple aliases. Some of these aliases take advantage of AWPipes GUI building feature covered in a different section of these docs.

alias see echo >awnpipes:/xc "defg\*nbitmap fn []\*nimage\*nopen"

Now 'see FILENAME' will display pictures using your datatypes.

alias readtext echo >awnpipes:/xc "defg a cs \*ntextfield a minw 200 minh 200 gt 0 bd ro datain []\*nopen"

Now 'readtext FILENAME' will display a textfile.

alias toolt type awnpipes:/xi[]

Now 'toolt FILENAME' will display the tooltypes of FILENAME. (do not include .info in FILENAME).

alias text2html copy awnpipes:t2h/f/h/r[]

Now 'text2html infile outfile' will create special html sequences for certain characters in a text file and save it in a new file.

alias html2text copy awnpipes:t2h/f/u/r[]

Now 'html2text infile outfile' will convert special HTML sequences in a html file and save it in a new file.

---

This next example is rather complex and is best studied after you have a working knowledge of AWP.

It might be useful to be able to check an icons tooltype from an ADOS script, as earlier examples have shown the `/Xt` option can do this. There is a problem however. You need to write data to the pipe (the tooltype you wish to check), and read data from the pipe (the response to your query).

It is simple to write data to the pipe with echo.

```
'echo "filetype" >awnpipetest/Xtpath:file'
```

A pipe connected to path:file.icon is opened and the data `'filetype'` is written to it. When the echo command completes it closes the pipe. This breaks the connection to the tooltype host and the host exits. You do not get a chance to read the response from the host.

The `/m` option (multiple opens) solves this by allowing you to open the pipe connection to the tooltype host more than once.

```
'echo "filetype" >awnpipetest/m/Xtpath:file'
```

Again a pipe is connected to the tooltype host is created and sent data. This time however when echo closes its pipe the connection to the host is not lost. AWPipe knows you want to connect to the same pipe again later.

```
'type awnpipetest'
```

This reads the response back from the tooltype host. Note you only specified the pipe name, you do NOT include `/Xtpath:file` again. When the type command completes the connection to the host is broken and the host exits.

This idea can be taken a step further.

```
'echo "filetype" >awnpipetest/m/Xtpath:file' 'type awnpipetest/m' 'echo "filesize" >awnpipetest/m' 'type awnpipetest/m'
'echo "filedate" >awnpipetest/m' 'type awnpipetest'
```

The connection to the host is not broken until you finally open the pipe a last time without the `/m` option.

## 1.14 4 GUI Creation

4 GUI Creation -----

[4.1 Conversation](#) [4.2 Objects](#) [4.3 Events](#) [4.4 Modify](#) [4.5 GUI Tutorial](#)

## 1.15 4.1 Conversation

4.1 Conversation -----

[4.1.1 Basic](#) [4.1.2 Step by Step](#) [4.1.3 Details](#)

## 1.16 4.1.1 Basic

4.1.1 Basic -----

The sections on the conversation are important but do not try to memorize it all. It is enough to understand the general process.

GUIs are defined and operated by means of a conversation with a special host built into the AWP device. The host is accessed by opening a file . You write lines of text to the file and read the file to see the response.

The GUI is defined by writing text lines to the file. You first write a line that defines the window title and other attributes of the window you want to create. Now you read back a response line from the pipe to make sure the window definition was ok. Next you write a line of text defining a gadget to have in the window and read another response line. This line will contain the Gadget ID assigned to the gadget you created. You can create as many gadgets as you like. Once you finish defining gadgets you open the window by writing a line containing the single word `'open'`. You read back a response line confirm the window opened ok.

Now that the GUI is created and open you simply continue to read lines from the file. When the user selects a gadget in the GUI you receive an event line containing the GID of the gadget selected and other data such the text entered in a string gadget. When the GUI window is closed you receive an EndOfFile.

In some cases you will want to modify a gui after it has already been opened. This can be done by writing a lines of text to the file after it has been opened or after you read an event. You read a response to the modify line.

For many GUIs you can tell the host to not send any response lines making the conversation even simpler. You can just write the GUI definition lines then read the event lines.

## 1.17 4.1.2 Step by Step

### 4.1.2 Step by Step -----

Understanding the conversation is the key to building GUIs with AWP. Here is a walk through the creation and use of a simple example.

Open the file AWPipe:myGUI/xc. myGUI can be any name you want. The /xc must be used as it tells AWPipe this file is for GUI creation. From now on I will refer to this file as the 'pipe'

Write a Window definition line to the pipe. 'title "My First gui" defaultgadgets' The line read from the pipe should be 'ok window' Anything else is an error response.

Write a Gadget definition line to the pipe. 'button gadgettext "testbutton" The line read from the pipe should be 'ok 1' The Gadget ID of the button is 1. If the response line does not start with 'ok' then an error has occurred.

Write a Gadget definition line to the pipe. 'checkbox gadgettext "tryme" The line read from the pipe should be 'ok 2' The Gadget ID of the checkbox is 2. If the response line does not start with 'ok' then an error has occurred.

Write open command line to the pipe. 'open' The line read from the pipe should be 'ok window' Anything else is an error response.

Now you read an event from the pipe. It will come when the user selects a gadget.

If the checkbox is selected you get the line 'gadget 2 1' the first word 'gadget' tells you the event was from a gadget being hit. 2 is the gadgets GID, the 1 at the end tells you the checkbox is now selected.

If the checkbox is selected a second time you get the line 'gadget 2 1' the first word 'gadget' tells you the event was from a gadget being hit. 2 is the gadgets GID, the 0 at the end tells you the checkbox is now unselected.

If the button is selected you get the line 'gadget 1 0' the first word 'gadget' tells you the event was from a gadget being hit. 1 is the gadgets GID, the 0 at the end tells you the button is not highlighted. In this example you can ignore the 0 (button state).

when the GUI window is closed you get the line 'close 0' the first word 'close' tells you the event was from the window being closed. 0 (close source) tells you it closed when the close gadget was hit. If the user entered <CTRL\> the close source would be -1. After the close event any further attempts to read the pipe will return EndOfFile.

You keep reading events until the end of file is received. You could also close the pipe after the close event is read.

This is a second conversation in which the GUI is modified after it is opened.

Open the file AWPipe:myGUI/xc. Write a Window definition line to the pipe. 'title "My First gui" defaultgadgets modify' The modify keyword tells the pipe you want to be able to modify this GUI later. The line read from the pipe should be 'ok window'

Write a Gadget definition line to the pipe. 'button gadgettext "testbutton" The line read from the pipe should be 'ok 1'

Write a Gadget definition line to the pipe. 'checkbox gadgettext "tryme" The line read from the pipe should be 'ok 2'

Write open command line to the pipe. 'open' The line read from the pipe should be 'ok window'

After the window is opened you get your first chance to modify it. We want to disable the button so we send the following line. 'ID 1 disable 1 refresh' The button is specified with 'ID 1' and 'disable 1' disables it. The refresh at the end tells the pipe to refresh the button so the changes show. The line read from the pipe should be 'ok' If the response does not start with 'ok' an error occurred.

We also want the check mark to show as selected so we write the line 'ID 2 selected 1' The button is specified with 'ID 1' and 'selected 1' turns causes it to be checked (the checkmark showing in the box). The refresh at the end tells the pipe to refresh the button so the changes show. The line read from the pipe should be 'ok'

WE have finished modifying the GUI for now so we send a line to tell the pipe we are finished sending modify instruction and want to read an event from the pipe. 'continue' The line read from the pipe should be 'ok'

Now you read an events from the pipe.

If the checkbox is selected you get the line 'gadget 2 0' The user selected the checkbox gadget, it is now unchecked. We want enable the button gadget. AFTER SENDING AN EVENT THE PIPE STARTS READING MODIFY INSTRUCTIONS AGAIN. We send the following modify line to the pipe. 'ID 1 disable 0 refresh' The button is specified with 'ID 1' and 'disable 0' enables it. The refresh at the end tells the pipe to refresh the button so the changes show. The line read from the pipe should be 'ok' WE have finished modifying the GUI for now so we send a line to tell the pipe we are finished sending modify instruction and want to read another event. 'continue' The line read from the pipe should be 'ok'

If the checkbox is selected a second time you get the line 'gadget 2 1' The 1 at the end tells you the checkbox is now selected. We want to disable the button gadget so we send the line 'ID 1 disable 1 refresh' The line read from the pipe should be 'ok' WE have finished modifying the GUI for now so we send a line to tell the pipe we are finished sending modify instruction and want to read another event. 'continue' The line read from the pipe should be 'ok'

If the button is selected you get the line 'gadget 1 0' WE do not want to modify anything at this time but we still need to send the continue line so the pipe will stop looking for modify lines and send us another event. 'continue' The line read from the pipe should be 'ok'

when the GUI window is closed you get the line 'close 0' The window is closed but we can still send modify commands. For example we could read the final state of the gadgets. In this case we will do nothing but send the continue line. 'continue' The line read from the pipe should be 'ok' Any further reads from the pipe return EOF.

You keep reading an event then sending modify commands until the end of file is received. You could also close the pipe after the close event is read.

## 1.18 4.1.3 Details

### 4.1.3 Details -----

When building a complex GUI you will use objects other than simple gadgets. These objects are created the same way as gadgets, by writing a line to the pipe and then reading a response. Some of these objects return a GID and some simply return an ok.

You will also handle more types of events than just gadget and close events. You can receive menu events, keystrokes, window activation and other types of events. These events are handled the same way gadget events are handled.

A wide range of modify commands can be used. As well as modifying gadgets you can iconify the window, read gadget contents, set the windows busy pointer and other things. The range of modify commands you use will be determined by the type program you are writing.

Some types of gadget events (textfields, texteditors), and ARexx events require special handling since they can be more than one line. The tutorials will show you how to handle these events.

You should now work through the examples. They will not only help you learn to use AWNP, they will provide example code you can use for the basis of your own GUIs.

## 1.19 4.2 Objects

### 4.2 Objects -----

[4.2.1 Window](#) [4.2.2 Simple Gadgets](#) [4.2.3 Images](#) [4.2.4 Special](#) [4.2.5 Fancy](#) [4.2.6 More](#)

## 1.20 4.2.1 Window

### 4.2.1 Window -----

The parameters defining the CA window must all be on one line terminated by a newline. The pipe will reply with 'window ok'.  
screentitle="screen title text" (st=)

---

Set the text for the screen title when CA window is active.

title="window title text"

Set the title for the CA window.

pubscreen="Screen name" (ps=)

Open the CA window on a public screen

IconifyIcon (ii=)

Set the name of the icon to use when you iconify the window.

IconTitle="icon title text" (it=)

Set the title for the CA window when iconified.

backfill="filename" (bf=)

Image file to use as backfill for window. If backfill is not specified you get the default window backfill. Setting backfill="" can be used to have no backfill at all. Use this carefully as it overrides the user CA prefs.

NoBorder (NB)

Make the window borderless.

Quiet (q)

Tell the pipe not to reply to the window, gadget definitions, or modify commands. This is actually a toggle switch turning replies off and on. SOME modify commands are ALWAYS replied to. (addnode and getfile selected 011)

NoWindow

switch to causes no window to be opened. This will not work unless modify is set and an arexx object is defined.

NOTE: At times you may want to modify a GUI after it is created, but BEFORE the window is opened. Use the 'NoWindow' keyword in the window definition, create the GUI as usual. When you send 'open' the window does NOT open but you do go into modify mode. Modify the GUI then send 'id 0 s 64' to open the window after it has been modified.

app

Make this window a application window.

activate (a)

Switch to activate theCA window when it opens.

depthgadget (dg)

Switch to include a depth gadget on the CA window.

dragbar (db)

Switch to include a drag bar on the CA window.

closegadget (cg)

Switch to include a close gadget on the CA window.

askclose

Switch to stop the close gadget from actually closing the window. The close gadget will generate an 'askclose' event instead. This switch is ignored unless modify is also set.

sizegadget (sg)

Switch to include a window size gadget on the CA window.

iconifygadget (ig)

Switch to include an iconify gadget on the CA window.

fullscreen (fs) centerscreen (cs) topleft (tl)

---

Relative position to open the CA window at. Defaults to centermouse.

top=number left=number width=umber height=number

Position and size to open the CA window. Using the left parameter over rides the relative positioning above.

vertical (v)

Switch to display gadgets vertically, defaults to horizontal.

even (e)

Switch to make gadgets/groups all the same size.

defaultgadgets (defg)

Switch to include close, depth, size, and drag gadgets on the CA window.

sendkeys (sk)

Switch to have the CA window return keystrokes.

sendqual (sq)

Switch to have the CA window send qualifier events. (see events.doc)

modify (m)

Switch to allow the CA window to be modified AFTER it has been opened. (See modify parameters below)

help (h)

Switch to have the CA window return help events.

state

Switch to have the CA window return active/inactive events.

refresh

Switch to have the CA window return refresh events. This is only useful if you are doing your rendering directly into the window.

defer

Switch to defer window layout from the input device to the pipe task. This will make input.device more responsive, and drop the (possibly quite heavy)process of recalculating the display to normal application priority instead of the priority 20 of input.device.

It also means window refresh is blocked if you are sending modify commands to the pipe. The window refreshes AFTER you send 'continue'.

SpaceOuter (so)

Leave a blank space outside the root layout of the CA window.

SpaceInner (si)

Leave a blank space around elements in the root layout of the CA window.

fixwidth (fw)

Do not allow the width of the window to be adjusted.

fixheight (fh)

Do not allow the width of the window to be adjusted.

shrinkwrap (sw)

Keep all gadgets in the window as close together as possible.

specialchar="character" (sc=)

'Character' is a single character that replaces '|', in use as a separator inside parameters like chooserlabels, tags, penmapdata, ...

tags="tags|data[|tag|data...]|0

A list of tag and value pairs in HEX. Do not forget the trailing null !

info

Include the drawinfo pen array for the screen in the reply to the window definition line.

If successful the reply is

'ok window NUMBERofPENS PEN0 PEN1 ...'

see include:intuition/screen.h for more about drawinfo pens.

Modify GID 0 -----

GID 0 is used to specify the GUI window itself. It can be used in modify lines to control certain aspects of the window.

selected=function\_bitmap (s=) This should really have been in HEX but its not for backward compatability.

bit# decimal function

0 1 activate window 1 2 window to front 2 4 window to back 3 8 set window title 4 16 set screen title 5 32 iconify window 6 64 uniconify (or open) window 7 128 Close window (but do not dispose) 8 256 Set busy pointer and disable window 9 512 Clear busy pointer and enable window 10 1024 window to front 11 2048 window to back

You can use more than one function at the same time. 66=uniconify and windowtofront.

gadgettext= (gt=)

Text for setting screen or window title.

disable=BOOL (dis=)

0 enables all gadgets in window. !1 disables all gadgets

wide=num high=num

Attempt to resize the window to the given width and/or height. Set wide=1 high=1 to get the minimum size possible.

top=num left=num

Move the window to the given position(s).

'ID 0 read' returns the windows 'left top width height' settings. (width and height report as 0 until you get at least 1 event back from the pipe)

refresh (ref)

Refresh all gadgets in the window.

## 1.21 4.2.2 Simple Gadgets

4.2.2 Simple Gadgets -----

4.2.2.1 Button 4.2.2.2 Integer 4.2.2.3 String 4.2.2.4 CheckBox 4.2.2.5 Chooser 4.2.2.6 RadioButton

### 1.22 4.2.2.1 Button

4.2.2.1 Button -----

Button gadget event -----

'gadget GID selected\_state'

selected\_state =0 not selected, !=0 selected

Button gadget parameters. -----

font=GID

This sets the font for the gadget. The GID must point to an ALREADY defined text attribute.

close (c)

This gadget will close the window when selected by the user.

readonly (ro)

This gadget is read only.

disable=number (dis=)

The gadget is enabled if number = 0 , disabled if number !=0. Defaults to enabled.

selected=number (s=)

The gadget is selected if number != 0 , unselected if number = 0. Defaults to unselected.

autobutton=number (ab)

This button uses a scaled glyph image. The number selects the glyph.

0 POPFILE 1 POPDRAWER 2 POPFONT 3 CHECKBOX 4 CANCELBOX 5 UPARROW 6 DNARROW 7 RTARROW 8 LFARROW 9 POPTIME 10 POPSCREEN 11 POPUP

pushbutton (pb)

This button is a pushbutton.

gadgettext="text" (gt=)

Set the text of the gadget.

bevel=TYPE (b=)

Set the bevel type for this gadget. 0=NONE 1=THIN 2=BUTTON 3=GROUP 4=FIELD 5=DROPPBOX 6=SBAR\_HORIZ 7=SBAR\_VERT 8=BOX 9=STANDARD

leftjustify (lj) rightjustify (rj)

Set the justification of the text for the gadget. Defaults to centerjustify.

minwidth=number (minw=)

Set the minimum width for this gadget

minheight=number (minh=)

Set the minimum height for this gadget

weightedwidth=number (weiw=)

Set the weighted width for this gadget

weightedheight=number (weih=)

Set the weighted height for this gadget

nominalsize (noms)

Set this gadget to its nominal size.

useimage (ui)

Use the last defined image (bitmap/drawlist/penmap) as the gadgets image.

Anim="x|y|width|height|offsetx|offsety|count"

Use portions of a bitmap for this gadgets image(s). x,y the left and top pixel of the first image. width,height the size of the images. offsetx,offsety the offset to the next image. count must be 0 for a standard two image button (normal and selected image)

The first image is the gadget graphic, the offset is to the selected image. offsets can be 0 to use a single image.

If count!=0 an array of images is produced to allow animation of the button. If count >0 an array of count images is built. The button image is set to the first image in the array. If count <0 an array of absolute value(count)\*2 images is built. The button image is set to the first image in the array. The selected image is set to image absolute value(count)+1.



See the 'animimage' modify parameter for details about animating the button.

FileName='path:file'

A image file to use as the source bitmap for the anim data. If no file name is given an anim is given the bitmap from the previous anim will be reused. This bitmap is held in a buffer shared by bitmap images. See the image docs-bitmap for more info.

trans

The animation background should be transparent if the image type allows it.

tags="tags\data[|tag\data...]|0

A list of tag and value pairs in HEX. Do not forget the trailing null !

childlabel (chl) childlabelr (chlR)

Use the previously defined image as a childlabel for this gadget. Display the childlabel to the right or left of the gadget.

replace=GID

Use this gadget to replace an existing gadget specified by GID.

Definition Reply -----

When the gadget creation is successful the pipe replies with

'ok GID'

Button gadget modify parameters. -----

readonly=number (ro=)

The gadget is readonly if number = 1 , functional if number= 0. Note that this is different handling then in gadget definitions.

disable=number (dis=)

The gadget is enabled if number = 0 , disabled if number != 0.

selected=number (s=)

The gadget is selected if number != 0 , unselected if number = 0.

gadgettext="text" (gt=)

Set the text of the gadget.

newimage=number (ni=)

Replace the gadget image with a previously defined image. If number =0 replace normal image, else replace selected image.

animimage=number (ai=)

When used with the 'newimage' keyword the button image is replaced with an image from the buttons image array. The image to be used is specified by number.

get=TAG

Read the value of a specific gadget attribute. Returns a blank line if attribute is not readable. This should not be combined with any other modify command on the same line.

read

returns the state of the button

refresh (ref)

Redraw the gadget.

## 1.23 4.2.2.2 Integer

### 4.2.2.2 Integer -----

Integer gadget event -----

'gadget GID integer\_value'

Integer gadget parameters. -----

font=GID

This sets the font for the gadget. The GID points to an ALREADY defined text attribute.

TabCycle (tc)

This gadget can be activated using the tab key.

close (c)

This gadget will close the window when selected by the user.

readonly (ro)

This gadget is read only.

disable=number (dis=)

The gadget is enabled if number = 0 , disabled if number != 0. Defaults to enabled.

arrows (a)

Switch to display up/down arrows for this gadget.

minchars=number (minc=)

Minimum number of character/digits in this gadget. Defaults to 5.

maxchars=number (maxc=)

Maximum number of character/digits in this gadget. Defaults to 10.

minnumber=number (minn=)

Minimum value for this gadget. Defaults to 0.

maxnumber=number (maxn=)

Maximum value for this gadget. Defaults to 32768.

defnumber=number (defn=)

Value for this gadget when window is opened. Defaults to 1.

leftjustify (lj) centerjustify (cj)

Set the justification of the text for the gadget. Defaults to rightjustify.

minwidth=number (minw=)

Set the minimum width for this gadget

minheight=number (minh=)

Set the minimum height for this gadget

weightedwidth=number (weiw=)

Set the weighted width for this gadget

weightedheight=number (weih=)

Set the weighted height for this gadget

nominalsize (noms)

Set this gadget to its nominal size.

tags="tags\data[|tag\data...]|0

A list of tag and value pairs in HEX. Do not forget the trailing null !

childlabel (chl) childlabelr (chlr)

Use the previously defined image as a childlabel for this gadget. Display the childlabel to the right or left of the gadget.

replace=GID

Use this gadget to replace an existing gadget specified by GID.

Definition Reply -----

When the gadget creation is successful the pipe replies with

'ok GID'

Integer gadget modify parameters. -----

disable=number (dis=)

The gadget is enabled if number = 0 , disabled if number != 0.

defnumber=number (defn=)

Set the value for this gadget.

## 1.24 4.2.2.3 String

4.2.2.3 String -----

String gadget event -----

'gadget GID string'

String gadget parameters. -----

Hook=number

Select a special string gadget type where number =

PASSWORD 1 IPADDRESS 2 FLOAT 3 HEXIDECIMAL 4 TELEPHONE 5 POSTALCODE 6 AMOUNT 7 UPPERCASE 8

NOTE: some types do not seem to be implemented in string gadget yet. 1,4,5,8 seem to work under OS 3.9.

font=GID

This sets the font for the gadget. The GID points to an ALREADY defined text attribute.

TabCycle (tc)

This gadget can be activated using the tab key.

close (c)

This gadget will close the window when selected by the user.

readonly (ro)

This gadget is read only.

disable=number (dis=)

The gadget is enabled if number = 0 , disabled if number != 0. Defaults to enabled.

gadgettext="text" (gt=)

Set the text in the gadget.

minchars=number (minc=)

Minimum number of character/digits in this gadget. Defaults to 10.

maxchars=number (maxc=)

Maximum number of character/digits in this gadget. Defaults to 100.

leftjustify (lj) centerjustify (cj)

Set the justification of the text for the gadget. Defaults to rightjustify.

minwidth=number (minw=)

Set the minimum width for this gadget

minheight=number (minh=)

Set the minimum height for this gadget

weightedwidth=number (weiw=)

Set the weighted width for this gadget

weightedheight=number (weih=)

Set the weighted height for this gadget

nominalsize (noms)

Set this gadget to its nominal size.

replace=GID

Use this gadget to replace an existing gadget specified by GID.

Definition Reply -----

When the gadget creation is successful the pipe replies with

'ok GID'

String gadget modify parameters. -----

disable=number (dis=)

The gadget is enabled if number = 0 , disabled if number != 0.

gadgettext="text" (gt=)

Set the text in the gadget.

selected=num (s=)

Set the cursor position in a string gadget, and activate the gadget.

bufferpos

Read the current cursor position in the string gadget. This keyword should be used by itself. ( 'id GID getposition')

It is replied to with 'POSITION ok'.

## 1.25 4.2.2.4 CheckBox

4.2.2.4 CheckBox -----

CheckBox gadget event -----

'gadget GID selected\_state'

CheckBox gadget parameters. -----

font=GID

This sets the font for the gadget. The GID points to an ALREADY defined text attribute.

close (c)

This gadget will close the window when selected by the user.

readonly (ro)

This gadget is read only.

disable=number (dis=)

The gadget is enabled if number = 0 , disabled if number != 0. Defaults to enabled.

selected=number (s=)

The gadget is selected if number != 0 , unselected if number = 0. Defaults to unselected.

gadgettext="text" (gt=)

Set the text of the gadget.

leftjustify (lj)

Set the placement of the text for the gadget. Defaults to Rightjustify.

minwidth=number (minw=)

Set the minimum width for this gadget

minheight=number (minh=)

Set the minimum height for this gadget

weightedwidth=number (weiw=)

Set the weighted width for this gadget

weightedheight=number (weih=)

Set the weighted height for this gadget

nominalsize (noms)

Set this gadget to its nominal size.

tags="tags\data[|tag\data...]|0

A list of tag and value pairs in HEX. Do not forget the trailing null !

childlabel (chl) childlabelr (chlr)

Use the previously defined image as a childlabel for this gadget. Display the childlabel to the right or left of the gadget.

replace=GID

Use this gadget to replace an existing gadget specified by GID.

Definition Reply -----

When the gadget creation is successful the pipe replies with

'ok GID'

CheckBox gadget modify parameters. -----

disable=number (dis=)

The gadget is enabled if number = 0 , disabled if number != 0.

selected=number (s=)

The gadget is selected if number != 0 , unselected if number = 0.

gadgettext="text" (gt=)

Set the text of the gadget.

## 1.26 4.2.2.5 Chooser

### 4.2.2.5 Chooser -----

Chooser gadget event -----

'gadget GID selected'

selected = the selected choice in the chooser.

Chooser gadget parameters. -----

font=GID

This sets the font for the gadget. The GID points to an ALREADY defined text attribute.

close (c)

This gadget will close the window when selected by the user.

readonly (ro)

This gadget is read only.

disable=number (dis=)

The gadget is enabled if number = 0 , disabled if number !=0. Defaults to enabled.

gadgettext="text" (gt=)

Set the text for the gadget.

selected=number (s=)

Set which choice is selected when the window opens. Selections start at 0 . This defaults to choice 0.

ChooserLabels="choice0|choice1|choice..." (cl)

Set the choices for the gadget. Maximum 50 choices, do not exceed this !

Maxnumber=count (maxn=)

The maximum number of selection to display. Defaults to 12.

Popup (pu)

This is a popup chooser. Defaults to dropdown.

minwidth=number (minw=)

Set the minimum width for this gadget

minheight=number (minh=)

Set the minimum height for this gadget

weightedwidth=number (weiw=)

Set the weighted width for this gadget

weightedheight=number (weih=)

Set the weighted height for this gadget

nominalsize (noms)

Set this gadget to its nominal size.

tags="tags|data[|tag|data...]|0

A list of tag and value pairs in HEX. Do not forget the trailing null !

childlabel (chl) childlabelr (chlr)

Use the previously defined image as a childlabel for this gadget. Display the childlabel to the right or left of the gadget.

### TabCycle (tc)

This gadget can be selected by cycling with the tab key.

replace=GID

Use this gadget to replace an existing gadget specified by GID.

Definition Reply -----

When the gadget creation is successful the pipe replies with

'ok GID'

Chooser gadget modify parameters. -----

disable=number (dis=)

The gadget is enabled if number = 0 , disabled if number !=0.

selected=number (s=)

Set which choice is selected . Selections start at 0 .

chooserlabels="choice0|choice1|choice..." (cl=)

Change the choices available in the chooser.

defnumber=count (defn=)

The maximum number of selection to display.

## 1.27 4.2.2.6 RadioButton

4.2.2.6 RadioButton -----

Radio Button gadget event -----

'gadget GID selected\_state'

Radio Button gadget parameters. -----

font=GID

This sets the font for the gadget. The GID points to an ALREADY defined text attribute.

close (c)

This gadget will close the window when selected by the user.

readonly (ro)

This gadget is read only.

disable=number (dis=)

The gadget is enabled if number = 0 , disabled if number != 0. Defaults to enabled.

selected=number (s=)

Set which choice is selected when the window opens. Selections start at 0 . This defaults to choice 0. to unselected.

RadioLabels="choice0|choice1|choice..." (rl=)

Set the choices for the gadget. This parameter MUST be given.

minwidth=number (minw=)

Set the minimum width for this gadget

minheight=number (minh=)

Set the minimum height for this gadget

weightedwidth=number (weiw=)

Set the weighted width for this gadget

weightedheight=number (weih=)

Set the weighted height for this gadget

nominalsize (noms)

Set this gadget to its nominal size.

replace=GID

Use this gadget to replace an existing gadget specified by GID.

Definition Reply -----

When the gadget creation is successful the pipe replies with

'ok GID'

Radio Button gadget modify parameters. -----

disable=number (dis=)

The gadget is enabled if number = 0 , disabled if number != 0.

selected=number (s=)

Set the current selection of the radio button.

## 1.28 4.2.3 Images

4.2.3 Images -----

[4.2.3.1 Label](#) [4.2.3.2 Glyph](#) [4.2.3.3 DrawList](#) [4.2.3.4 PenMap](#) [4.2.3.5 Bitmap](#) [4.2.3.6 Image](#) [4.2.3.7 Space](#)

### 1.29 4.2.3.1 Label

4.2.3.1 Label -----

Label gadget parameters. -----

IDME

Force this object to be assigned a GID. Normally no GID is assigned to labels. The definition reply becomes 'ok GID WIDTH HEIGHT'

The label produces an image and usually puts it into your GUI right away. To use the label as a child label of a gadget You MUST use the unattached keyword.

unattached (ua)

The unattached keyword does not produce a gadget. It creates an image to be used. Up to 50 images can be predefined, the last one defined is the first one used. Last In First Out. No GID is assigned EVEN WITH 'IDME'.

font=GID

This sets the font for the label. The GID points to an ALREADY defined text attribute.

Selected=num (s=)

Set the pen number used to render the text.

SoftStyle=num (ss=)

Set the soft style for the label. This is a bitmapped decimal number. example 3=bold + underline decimal bit NORMAL 0 - UNDERLINED 1 0 BOLD 2 1 ITALIC 4 2 EXTENDED 8 3



gadgettext="Label text" (gt=)

Set the text of the Label.

underscore="CHAR"

Use an alternate character to flag the hotkey for the label. CHAR is a single character. This defaults to underscore '\_', changing it will allow an underscore to be used as a plain character in a label.

leftjustify (lj) rightjustify (rj)

Set the justification of the text for the gadget. Defaults to centerjustify.

minwidth=number (minw=)

Set the minimum width for this gadget

minheight=number (minh=)

Set the minimum height for this gadget

weightedwidth=number (weiw=)

Set the weighted width for this gadget

weightedheight=number (weih=)

Set the weighted height for this gadget

nominalsize (noms)

Set this gadget to its nominal size.

useimage (ui)

Use the last defined image (bitmap/drawlist/penmap) as the gadgets image.

Even

Correct a possible layout problem in the image position. You will probably never use this parameter. Technically it moves the image up one pixel, which is useful when mixing different fonts in the same line.

Definition Reply -----

When the image creation is successful the pipe replies with

'ok WIDTH HEIGHT'

## 1.30 4.2.3.2 Glyph

4.2.3.2 Glyph -----

Glyph parameters -----

The glyph keyword does not produce a gadget. It creates an image to be used later. Up to 50 images can be predefined, the last one defined is the first one used. Last In First Out.

defnumber=number (defnum)

Choose the image this glyph displays. the possible values number are

0 NONE 1 DOWNARROW 2 UPARROW 3 LEFTARROW 4 RIGHTARROW 5 DROPDOWN 6 POPUP 7 CHECKMARK 8 POPFONT 9 POPFILE 10 POPDRAWER 11 POPSCREENMODE 12 POPTIME 13 RADIOBUTTON 14 RETURNARROW

Definition Reply -----

When the image creation is successful the pipe replies with

'ok '

## 1.31 4.2.3.3 DrawList

### 4.2.3.3 DrawList -----

#### DrawList parameters -----

The drawlist keyword does not produce a gadget. It creates an image to be used later. Up to 50 images can be predefined, the last one defined is the first one used. Last In First Out.

bindata (bd)

The drawlistdata will be passed to this pipe in binary after the command line. The drawlistdata parameter now gives the length of the data. Setting datain means the data is read from an external file rather than the pipe.

dataout="filename" (do=)

Send the data for this gadget to a file. This is a development tool to allow you to create binary data to pass to the pipe. It is ignored if the Bindata keyword is also present.

datain="filename" (di=)

Read the data for this gadget from a file. It is ignored if the Bindata keyword is not present.

minheight=number (minh)

Set the height of the virtual raster the drawlist data relates to.

minwidth=number (minw)

Set the width of the virtual raster the drawlist data relates to.

drawlistdata="value1|value2|..." (dld=)

The draw list data defines a set of drawing instruction to create an image. Each drawing instruction has 6 values. directive|X1|Y1|X2|Y2|pen

The possible directives are

0 END 1 LINE 2 RECT 3 FILL 4 ELLIPSE 5 CIRCLE 6 LINEPAT 7 FILLPAT 8 AMOVE 9 ADRAW 10 AFILL 11 BEVELBOX 12 ARC 13 START = BOUNDS 14 LINESIZE

Register as a Class Act developer and read the docs for more info on drawlists.

When sending binary data you send an array of structure DrawLists.

struct DrawList { WORD dl\_Directive; WORD dl\_X1, dl\_Y1; WORD dl\_X2, dl\_Y2; WORD dl\_Pen; }; The last structure MUST be END (dl\_Directive=0).

Definition Reply -----

When the image creation is successful the pipe replies with

'ok'

## 1.32 4.2.3.4 PenMap

### 4.2.3.4 PenMap -----

#### PenMap parameters -----

The penmap keyword does not produce a gadget. It creates an image to be used later. Up to 50 images can be predefined, the last one defined is the first one used. Last In First Out.

bindata (bd)

The PenMapPalette, penmapdata, and selectedimage will be passed to this pipe in binary after the command line, AND IN THAT ORDER. The penmapdata, selectedimage, and PenMapPalette parameters now give the length of each data segment. Do NOT specify a data length of 0 for selected image, omit the keyword instead.

dataout="filename" (do)

Send the data for this gadget to a file. This is a development tool to allow you to create binary data to pass to the pipe. It is ignored if the Bindata keyword is also present.

penmapdata="data" (pmd)

The image data. NOTE all values are ascii HEX. The data is in the following format.

"width\_h\_bytelwidth\_l\_bytelheight\_h\_bytelheight\_l\_byteldata1|data2..."

If the bindata keyword is used the image data is the following format

word width, height; char data[];

Note the size specified in penmapdata INCLUDES the 4 bytes for width and height.

selectedimage="data" (si)

The image data for a selected image. NOTE all values are ascii HEX. The data is in the following format.

"width\_h\_bytelwidth\_l\_bytelheight\_h\_bytelheight\_l\_byteldata1|data2..."

If the bindata keyword is used the image data is the same format as for penmapdata.

PenMapPalette="data" (pmp)

The palate data. NOTE all values are ascii HEX. The data is in the following format.

"triplet\_count|red1|green1|blue1|red2|..."

If the bindata keyword is used the palate data is the format

ULONG palette[] = { number\_triplets, red1, green1, blue1, red2, green2, blue2, ... };

Trans

Make pen 0 transparent.

## 1.33 4.2.3.5 Bitmap

### 4.2.3.5 Bitmap -----

BitMap parameters -----

The bitmap keyword does not produce a gadget. It creates an image to be used later. Up to 50 images can be predefined, the last one defined is the first one used. Last In First Out.

FileName='path:file' (fn)

The file to read and produce an image from using datatypes.

selectedimage='path:file' (si)

The file to read and produce the selected image from using datatypes.

trans

Make the image background transparent (if the associated image and datatype allows this).

Part="x|y|width|height|offsetx|offsety|buffer"

x,y the left and top pixel of the first image. width,height the size of the images. offsetx,offsety the offset to the selected state image. buffer the buffer used to store the root bitmap. (0-9) ( buffer 0 is also used for button animations )

Use portions of a bitmap for this image.

The image is only part of the root bitmap specified by x|y|width|height, the offsets can be set to 0 to make the normal and selected state the same.

filename="" will cause this image to use the next available predefined bitmap for the root bitmap. (only with 'part')

Not including a file name at all will reuse an existing root bitmap. This allows you to only load one bitmap with a images needed. You can then reuse parts mutiple times. Also you only need to have a single image file for most applications rather than several small ones.(only with part)

It is possible to redefine an image buffer. When this is done the previous contents of the image buffer are saved before the new bitmap is created. See the freeimage modify command for more information on handling image buffers.

Definition Reply -----

When the image creation is successful the pipe replies with

'ok BITMAP\_width BITMAP\_height'

## 1.34 4.2.3.6 Image

4.2.3.6 Image -----

Image gadget parameters. -----

You must first define an image before you can produce an image gadget. The last defined image is placed into the GUI.

IDME

Force this object to be assigned a GID. Normaly no GID is assigned to Images. The definition reply becomes 'ok GID'

minwidth=number (minw=)

Set the minimum width for this gadget

minheight=number (minh=)

Set the minimum height for this gadget

weightedwidth=number (weiw=)

Set the weighted width for this gadget

weightedheight=number (weih=)

Set the weighted height for this gadget

nominalsize (noms)

Set this gadget to its nominal size.

Definition Reply -----

When the gadget creation is successful the pipe replies with

'ok'

## 1.35 4.2.3.7 Space

4.2.3.7 Space -----

Space gadget parameters. -----

IDME

Force this object to be assigned a GID. Normaly no GID is assigned to spaces. The definition reply becomes 'ok GID'

trans

Turn transparency on for this space.

minwidth=number (minw=)

Set the minimum width for this gadget

minheight=number (minh=)

Set the minimum height for this gadget

bevel=TYPE (b=)

Set the bevel type for this gadget. Defaults to none. 0=NONE 1=THIN 2=BUTTON 3=GROUP 4=FIELD 5=DROPBOX 6=SBAR\_HORIZ 7=SBAR\_VERT 8=BOX 9=STANDARD

Definition Reply -----

When the gadget creation is successful the pipe replies with

'ok'

## 1.36 4.2.4 Special

4.2.4 Special -----

[4.2.4.1 Layout\(End\)](#) [4.2.4.2 ClickTab](#) [4.2.4.3 TextAttr](#) [4.2.4.4 BrowserNode](#) [4.2.4.5 Menu](#) [4.2.4.6 Arexx](#)

### 1.37 4.2.4.1 Layout(End)

4.2.4.1 Layout(End) -----

LayOutEnd parameters -----

LayOutEnd=le=pg

(pg is for historical reasons)

This ends the current layout group. It takes no parameters.

layout parameters. -----

Layouts can be nested up to 50 deep.

page=GID

Use this layout group as a page in the Clicktab gadget specified by GID.

selected=number (s=)

The clicktab page to be displayed on startup, defaults to 0. Ignored unless page is also specified. It is only useful when adding the final page to a clicktab.

font=GID

This sets the font for the whole group. The GID points to an ALREADY defined text attribute.

bevel=TYPE (b=)

Set the bevel type for this gadget. 0=NONE 1=THIN 2=BUTTON 3=GROUP 4=FIELD 5=DROPBOX 6=SBAR\_HORIZ 7=SBAR\_VERT 8=BOX 9=STANDARD

gadgettext="group label" (gt=)

set the group label for this group. Ignored if nobevel is set.

labr labl labc

Set the position of the layout label (topleft/topright/topcenter). Defaults to the users prefs setting, so don't set/force this unless you really need to..

defn=PEN

The pen number to use when backfilling this layout.

`fpat="HEX"`

The fill pattern to backfill this layout. A 24 bit hexadecimal number, interpreted as a 16 x 2 pattern. `fpat` is ignored unless `defn` is also specified.

`minwidth=number (minw=)`

Set the minimum width for this gadget

`minheight=number (minh=)`

Set the minimum height for this gadget

`weightedwidth=number (weiw=)`

Set the weighted width for this gadget

`weightedheight=number (weih=)`

Set the weighted height for this gadget

`nominalsize (noms)`

Set this gadget to its nominal size.

`SpaceOuter (so)`

Leave a blank space outside the layout group.

`SpaceInner (si)`

Leave a blank space around elements in the layout group.

`disable=number (dis=)`

The gadgets in this layout are enabled if `number = 0` , disabled if `number != 0`. Defaults to enabled.

`centerjustify (cj) rightjustify (rj) bottomjustify (bj)`

Set the justification of the gadgets in this layout. Defaults to leftjustify or topjustify.

`vertical (v)`

Switch to display gadgets vertically, defaults to horizontal.

`even (e)`

Switch to make gadgets/groups all the same size.

`shrinkwrap (sw)`

Switch to cause gadgets in this group to have no space between them.

`tags="tags|data[|tag|data...]|0`

A list of tag and value pairs in HEX. Do not forget the trailing null !

`childlabel (chl) childlabelr (chlr)`

Use the previously defined image as a `childlabel` for this gadget. Display the `childlabel` to the right or left of the gadget.

`replace=GID`

Use this gadget to replace an existing gadget specified by `GID`.

Definition Reply -----

When the gadget creation is successful the pipe replies with

`'ok GID'`

`LayoutEnd` replies with

`'ok'`

layout modify parameters. -----

`disable=number (dis=)`

The gadgets in this layout are enabled if `number = 0` , disabled if `number != 0`. Defaults to enabled.

`refresh (ref)`

## 1.38 4.2.4.2 ClickTab

### 4.2.4.2 ClickTab -----

#### ClickTab gadget event -----

'gadget GID selected\_page'

#### ClickTab gadget parameters. -----

This actually creates a pair of connected gadgets, a clicktab->pagegadget. Each tab labels will display a different page in the page gadget. See parent group for how to attach a page to a clicktab gadget.

font=GID

This sets the font for the gadget. The GID points to an ALREADY defined text attribute.

readonly (ro)

This gadget is read only.

disable=number (dis=)

The gadget is enabled if number = 0 , disabled if number != 0. Defaults to enabled.

ClickTabLabels="choice0|choice1|choice..." (ctl)

Set the choices for the gadget. This parameter MUST be given.

minwidth=number (minw=)

Set the minimum width for this gadget

minheight=number (minh=)

Set the minimum height for this gadget

weightedwidth=number (weiw=)

Set the weighted width for this gadget

weightedheight=number (weih=)

Set the weighted height for this gadget

nominalsize (noms)

Set this gadget to its nominal size.

replace=GID

Use this gadget to replace an existing gadget specified by GID.

#### Definition Reply -----

When the gadget creation is successful the pipe replies with

'ok GID'

#### ClickTab gadget modify parameters. -----

disable=number (dis=)

The gadget is enabled if number = 0 , disabled if number != 0.

NOTE: when you disable a layout, Clicktab gadgets in the layout do not properly disable themselves (CA/Reaction bug?). You must manually disable the clicktabs your self.

target=number (tar=)

When used with disable only the targeted tab is enabled or disabled. If target is not specified all tabs will be disabled or enabled.

selected=number (s=)

Set the page displayed by the clicktab gadget.

refresh (ref)

Redraw the gadget.

### 1.39 4.2.4.3 TextAttr

#### 4.2.4.3 TextAttr -----

TextAttr parameters. -----

gadgettext="fontname" (gt=)

This text defines the font. Remember the '.font' like 'gt=topaz.font'

defnumber=number (defn=)

The font size attribute.

selected="number" s=

The fonts style attributes. Sorry it a decimal number...

Definition Reply -----

When the gadget creation is successful the pipe replies with

'ok GID HEXADDRESS'

The address allows you to use the text attribute with the 'set' modify command.

The same text attribute can be used to specify a font for many gadgets.

### 1.40 4.2.4.4 BrowserNode

#### 4.2.4.4 BrowserNode -----

BrowserNode gadget parameters. -----

Browsernodes always belong to last defined listbrowser. NEVER use the BrowserNode keyword after you GUI window has opened. Use the AddNode modify command instead.

selected=number (s=)

The node is selected if number != 0 , unselected if number = 0. Defaults to unselected.

Set which choice is selected when the window opens. This is only for multiple select list browsers. Set selected in the Listbrowser definition for single select lists.

GadgetText="[^\[]text0|^\[]text1|^\[]text..." (gt=)

Set the text(s). This parameter MUST be given. YOU MUST NOT GIVE MORE TEXTS THAN THE AMOUNT OF COLUMNS IN THE LISTBROWSER. A leading '^' makes the text editable. A leading 'p' (ALT p) uses the last defined image for the column rather than text. Note that images used in browsernodes can not be more than 255 pixels high. (hint: split larger images and use more than one node)

NOTE: the following parameters may only be used if the ListBrowser containing the node allows parenting.

defnumber=number (defn=)

Set the Generation of this browser node.

browsernodeparent (bnp)

This browser node has children.

Definition Reply -----

When the node creation is successful the pipe replies with

'ok GID'

BrowserNode gadget modify parameters. -----

GadgetText="[^\[]text0|^\[]text1|^\[]text..." (gt=)



Set the text(s). YOU MUST NOT GIVE MORE TEXTS THAN THE AMOUNT OF COLUMNS IN THE LISTBROWSER. A leading `` makes the text editable. A leading 'P' (ALT p) uses the last defined image for the column rather than text. Note that images used in browsernodes can not be more than 255 pixels high. (hint: split larger images and use more than one node)

removenode (remn)

remove this browsernode from its ListBrowser

selected=number (s=)

The node is selected if number != 0 , unselected if number = 0. This is only for nodes in multi select listbrowsers, for single select browsers see the listbrowser modify docs.

read sort=COLUMN#

To read a browser node you must also specify a column. The text contents of that column are returned.

refresh (ref)

Refresh this listnodes listbrowser (not just the node itself). When setting multiple nodes in a multiselect listbrowser only refresh when you set the last node. This avoids multiple refreshes.

hidechild=num (hc=)

If num =0 show all this nodes children . If num >0 then hide all this nodes children.

## 1.41 4.2.4.5 Menu

### 4.2.4.5 Menu -----

Menu gadget event -----

'menu menu\_number item\_number sub\_item\_number checked'

checked=0 Item is not checked checked=1 Item is checked

Menu gadget parameters. -----

gadgettext="menudata" (gt=)

This text defines a menu for the CA window. The data is the following format.

menu\_title|menu\_item0|menu\_item1 ...

Each menu\_item is the following format

[@hotkey][\$][-][^][%][&][!][`][#HEX\_number#]menu\_item\_title

@ = hotkey for menu, following character is hotkey. \$ = sub Item (of previous menu\_item) - = Display a separator bar (ignore text and other options for this item) ^ = menu toggle % = menu checkable (unchecked at startup) & = menu checkable (checked at startup) ! = highlight none ` = disabled # = mutual exclude. Note the trailing '#' MUST be used. If multiple flags are set they MUST be in the order shown.

Note: `` is the only flag that may be set for the menu title. Note: Separator bars are considered a menu item. Example

gt="dothis|-ldothat" dothis is item 0, the bar is item 1, dothat is item 2.

replace=GID

Use this menu to replace an existing menu specified by GID.

Menu gadget modify parameters. -----

disable=number (dis=)

The menu or menu item is enabled if number = 0 , disabled if number != 0.

selected=number (s=)

The menu item is checked if number != 0 , unchecked if number = 0. Ignored if a target is not specified.

target=number (tar=)

If a target is specified you are modifying a menu item. If a target is not specified you are modifying the menu itself.

## 1.42 4.2.4.6 Arexx

### 4.2.4.6 Arexx -----

#### Arexx event stream -----

Arexx events are handled differently than other types of events, be careful.

The events are as follows.

'arexx FID textlength <newline> text'

FID is the function ID number

textlength is the length of the text after the newline character.

If modify is turned on, after each event you MUST write a single line to the pipe in the following format.

'rc=number result="(result textlength)" [bd]'

If rc > 0 result is ignored, if RC is omitted it defaults to 0.

If the bd (binary data) keyword is given result MUST be the length of the data to be returned, and you then MUST write length chars of data to the pipe.

The pipe replies with 'ok' to your result line (unless quiet is set).

If modify is not turned on all arexx events are given a rc of 0 (success) and a null result.

#### Arexx object parameters. -----

gadgettext="HOSTNAME|Function0|Function1|..." (gt=)

The gadget text defines the arexx host name and the functions available. Hostname should be all capitals, function names should not contain spaces. The first function is FID 0, the second FID 1, ...

slot

This switch causes a slot number to appended to the host name. TESTHOST would become TESTHOST.1 or TESTHOST.2 etc.

#### Arexx definition Reply -----

When the ARExx object creation is successful the pipe replies with

'ok HOSTNAME'

NOTE only one arexx host can be defined for each GUI window.

#### Additional info -----

To have a arexx host with no GUI window, set the NoWindow switch in the window definition. See the window docs for more information.

## 1.43 4.2.5 Fancy

### 4.2.5 Fancy -----

[4.2.5.1 GetFile](#) [4.2.5.2 GetFont](#) [4.2.5.3 TextEditor](#) [4.2.5.4 TextFeild](#) [4.2.5.5 ListBrowser](#) [4.2.5.6 Palette](#)

## 1.44 4.2.5.1 GetFile

### 4.2.5.1 GetFile -----

There are actually two file requesters available in AwnPipe. The ClassAct/Reaction getfile.gadget, and the ASL file requester. Only the getfile.gadget can be displayed in a gui. Both can be opened under your control to present the user with a file requester. The details for the ASL requester are at the end of this file.

\*\*\* GETFILE GADGET CLASS \*\*\*

GetFile gadget event. -----

'gadget GID success ["filename" ["filename" ...]]'

Success=0 user selected no file or canceled. Success!=0 user selected a file. Filenames are fully qualified with path.

GetFile gadget parameters. -----

font=GID

This sets the font for the gadget. The GID points to an ALREADY defined text attribute.

readonly (ro)

This gadget is read only.

disable=number (dis=)

The gadget is enabled if number = 0 , disabled if number != 0. Defaults to enabled.

gadgettext="file requester title" (gt=)

Set the title of the file requester.

pattern="pattern" (pat=)

Set and Enable pattern matching in the file.

multi (m)

Allow multi selection in the file requester. (does not seem to work)

save

This is a save requester.

drawers (dr)

Only show and allow selections of drawers.

icons (i)

Do NOT show .info files.

filename="path:filename" (fn=)

Set the filename and directory for the file requester. It must end in a '/' for drawer only file requesters

Wide=num High=num

Set the size of the requester when opened.

unattached (ua)

This file requester is not attached to the CA window. It may still be activated in the modify conversation.

minwidth=number (minw=)

Set the minimum width for this gadget

minheight=number (minh=)

Set the minimum height for this gadget

weightedwidth=number (weiw=)

Set the weighted width for this gadget

weightedheight=number (weih=)

Set the weighted height for this gadget

nominalsize (noms)

Set this gadget to its nominal size.

---

tags="tags\data[|tag\data...]|0

A list of tag and value pairs in HEX. Do not forget the trailing null !

childlabel (chl) childlabelr (chlr)

Use the previously defined image as a childlabel for this gadget. Display the childlabel to the right or left of the gadget.

replace=GID

Use this gadget to replace an existing gadget specified by GID.

Definition Reply -----

When the gadget creation is successful the pipe replies with

'ok GID'

GetFile gadget modify parameters. -----

selected=number (sel=)

This modify command is only valid for UNATTACHED getfile gadgets. Number is ignored but must be given. This opens the file requester and the selected files are returned rather than a simple ok. You should not use other modify instructions on the same line as 'selected'.

NOTE: You can not force an attached getfile to open.

disable=number (dis=)

The gadget is enabled if number = 0 , disabled if number != 0.

save=number

If number=0 this is NOT a save requester. If number~=0 this is a save requester.

filename="path:filename" (fn=)

Set the filename and directory for the file requester.

gadgettext="title" (gt=)

Set the file requesters title.

\*\*\* GETFILE ASL \*\*\*

The ASL requester does not send events.

GetFile ASL parameters. -----

ASL

This tells AWP you want to define an ASL file requester.

font=GID

This sets the font for the gadget. The GID points to an ALREADY defined text attribute.

gadgettext="file requester title" (gt=)

Set the title of the file requester.

pattern="pattern" (pat=)

Set and Enable pattern matching in the file.

multi (m)

Allow multi selection in the file requester. (does not seem to work)

save

This is a save requester.

drawers (dr)

Only show and allow selections of drawers.

icons (i)

Do NOT show .info files.

filename="path:filename" (fn=)

Set the filename and directory for the file requester. It must end in a '/' for drawer only file requesters

Wide=num High=num

Set the size of the requester when opened.

top=num left=num

Set the position of the requester when it opens.

postx="positive text"

Set the text for the positive choice in the file requester. Defaults to OK.

negtx="negative text"

Set the text for the positive choice in the file requester. Defaults to Cancel

Definition Reply -----

When the gadget creation is successful the pipe replies with

'ok GID'

GetFile ASL modify parameters. -----

You can only modify the ASL getfile requester when opening it. Unless the selected keyword is present the modify line is ignored.

selected=number (sel=)

Number is ignored but must be given. This opens the file requester and you get returned the following line rather than a simple ok.

SUCCESS "path:drawer/filename" "path2:drawer2/filename2" ...

SUCCESS=0 if the user aborted the file requester without making a selection. SUCCESS !=0 if the user selected a file, each file name is quoted and all are on a single line.

save=number

If number=0 this is NOT a save requester. If number~=0 this is a save requester.

filename="path:filename" (fn=)

Set the filename and directory for the file requester.

gadgettext="title" (gt=)

Set the file requesters title.

Wide=num High=num

Set the size of the requester when opened.

top=num left=num

Set the position of the requester when it opens.

postx="positive text"

Set the text for the positive choice in the file requester. Defaults to OK.

negtx="negative text"

Set the text for the positive choice in the file requester. Defaults to Cancel

## 1.45 4.2.5.2 GetFont

### 4.2.5.2 GetFont -----

GetFont gadget event. -----

'gadget GID success fontname fontsize fontstyle fontflags'

Success=0 user selected no font or canceled. Success!=0 user selected a font.

GetFont gadget parameters. -----

font=GID

This sets the font for the gadgets text. The GID points to an ALREADY defined text attribute.

close (c)

This gadget will close the window when selected by the user.

readonly (ro)

This gadget is read only.

disable=number (dis=)

The gadget is enabled if number = 0 , disabled if number != 0. Defaults to enabled.

gadgettext="font requester title" (gt=)

Set the title of the file requester.

style

This requester includes style settings.

fixed

This requester includes fixed width fonts only.

Minn=num Maxn=num

The minimum and maximum font size allowed. Defaults to 6-20

Wide=num High=num

Set the size of the requester when opened.

unattached (ua)

This file requester is not attached to the CA window. It may still be activated in the modify conversation.

minwidth=number (minw=)

Set the minimum width for this gadget

minheight=number (minh=)

Set the minimum height for this gadget

weightedwidth=number (weiw=)

Set the weighted width for this gadget

weightedheight=number (weih=)

Set the weighted height for this gadget

nominalsize (noms)

Set this gadget to its nominal size.

tags="tags|data[|tag|data...]|0

A list of tag and value pairs in HEX. Do not forget the trailing null !

childlabel (chl) childlabelr (chlR)

Use the previously defined image as a childlabel for this gadget. Display the childlabel to the right or left of the gadget.

replace=GID

Use this gadget to replace an existing gadget specified by GID.

Definition Reply -----

When the gadget creation is successful the pipe replies with

'ok GID'

GetFont gadget modify parameters. -----

selected=number (sel=)

Number is ignored but must be given. This opens the font requester and the selected font is returned rather than a simple ok.

disable=number (dis=)

The gadget is enabled if number = 0 , disabled if number != 0.

## 1.46 4.2.5.3 TextEditor

4.2.5.3 TextEditor -----

The TextEditor.gadget is only available under ADOS 3.5 . At this time only beta versions were available but the TextEditor gadget is so great it had to be included in AWNPipe. These functions should all work once it is released.

TextEditor gadget event. ----- An event is only generated when the user double clicks the gadget.

'gadget GID DoubleClickPositionX DoubleClickPositionY'

DoubleClickPositionX/Y is the position in the current line the user double clicked.

NOTE: You must the TextEditor directly at appropriate times in your program. This means you should have modify turned on if the TextEditor gadget is not read only.

TextEditor gadget parameters. -----

font=GID

This sets the font for the gadget. The GID points to an ALREADY defined text attribute.

arrows (a)

Give the TextEditor a scroll gadget on the right.

readonly (ro)

This gadget is read only.

disable=number (dis=)

The gadget is enabled if number = 0 , disabled if number != 0. Defaults to enabled.

export="num"

Set the export hook for the TextEditor.

num=0 Plain text hook num=1 EMail hook

import="num"

Set the import hook for the TextEditor.

num=0 Plain text hook num=1 EMail hook num=2 Mime hook num=3 Mime quoted hook

block

The texteditor should mark a block of text as well as generate an event when it is double clicked.

gadgettext="text" (gt=)

Set the text in the gadget. given.

bindata (bd)

The text will be passed to this pipe in binary after the command line. The gadgettext parameter now gives the length of the data. Setting datain means the data is read from an external file rather than the pipe.

datain="filename" (di=)

Read the data for this gadget from a file. It is ignored if the Bindata keyword is not present. A length of 0 means to use the complete file.

minwidth=number (minw=)

Set the minimum width for this gadget

minheight=number (minh=)

Set the minimum height for this gadget

weightedwidth=number (weiw=)

Set the weighted width for this gadget

weightedheight=number (weih=)

Set the weighted height for this gadget

nominalsize (noms)

Set this gadget to its nominal size.

replace=GID

Use this gadget to replace an existing gadget specified by GID.

Fixed

Use a fixed font for this TextEditor.

Definition Reply -----

When the gadget creation is successful the pipe replies with

'ok GID'

TextEditor gadget modify parameters. ----- read

This should not be used with any other parameters. It is ALWAYS responded to with 'SIZE<newline>CHARACTERS'.

SIZE is the number of characters after the newline. The characters are the current contents of the TextEditor gadget.

info

This should not be used with any other parameters. It is ALWAYS responded to with 'CURSORX CUSORY BLOCKMARKED [BSTARTX BSTARTY BENDX BENDY]'.

CURSORX and CUSORY are the current cursor location, BLOCKMARKED=0 if no block is marked, BLOCKMARKED~=0 means a block is currently marked. BSTARTX/Y and BENDX/Y (the start and end of the marked block) are only given if BLOCKMARKED!=0.

update=number

If number=0 then turn off updating of the Texteditor gadget. If number!=0 then turn updating of the gadget back on.

disable=number (dis=)

The gadget is enabled if number = 0 , disabled if number != 0.

readonly=num (ro=)



if num != 0 then this gadget is read only. If num =0 then the gadget is NOT readonly.

clear

Remove all text from the TextEditor gadget.

CursorY="num" (cy=)

Set the cursor to Y position to num.

Cursorx="num" (cx=)

Set the cursor to Y position to num.

command="string" (cmd=)

Have the TextEditor gadget execute the Arexx command given in string. Command should not be used with other modify parameters.

Command always replies with 'ok SUCCESS [SIZE] <newline>[CHARACTERS]' if SUCCESS!=0 then the command was handled by the TextEditor, if SUCCESS=0 then the command was not recognised. If the command succeeds you MUST check to see if SIZE is given. If size is specified you then read SIZE the number of characters after the newline. This is data returned by the command to the TextEditor.

replace

Replace the text highlighted with the text given in gadgettext.

Replace ALWAYS responds with 'RESULT' where result=0 if a marked string is not found, and RESULT !=0 if the replace is successful.

search

Search for the text given in gadgettext.

Search ALWAYS responds with 'RESULT' where result=0 if the search string is not found, and RESULT !=0 if the search string is found.

gadgettext="text" (gt=)

Insert the text in the gadget. OR text to search for if search parameter is specified, OR text to replace highlighted text with if replace is specified.

target="num" (tar=)

If num=0 insert text at start of current contents. If num!=0 then insert text at the cursor position. If target is not specified the text is inserted at the end of the current contents. If target is not given insert text at cursor position (default).

OR

With search, num is a bit field for search modifiers.

bit 0 (1) search from top bit 1 (2) search next (default) bit 2 (4) search case sensitive bit 3 (8) search dospattern bit 4 (16) search backwards

Unfortunately this is implemented as a decimal number. See the 'Hints' section for easy ways to handle this.

bindata (bd)

The text will be passed to this pipe in binary after the command line. The gadgettext parameter now gives the length of the data. Note that you can NOT use datain during modify.

Search responds with 'ok RESULT' where result=0 if the search string is not found, and RESULT !=0 if the search string is found.

## 1.47 4.2.5.4 TextFeild

### 4.2.5.4 TextFeild -----

TextField gadget event. ----- NOTE: The textfield gadget has two event forms. For more information see the shorthevent parameter below

Short Event 'gadget GID'

or Default event. 'gadget GID size <newline> characters[size]'

The next 'size' number of characters after the newline are the contents of the TextField gadget.

Textfield gadget parameters. -----

font=GID

This sets the font for the gadget. The GID points to an ALREADY defined text attribute.

arrows (a)

Give the textfield a scroll gadget on the right.

readonly (ro)

This gadget is read only.

disable=number (dis=)

The gadget is enabled if number = 0 , disabled if number != 0. Defaults to enabled.

TabCycle (tc)

This gadget can be activated using the tab key.

gadgettext="text" (gt=)

Set the text in the gadget.

bindata (bd)

The text will be passed to this pipe in binary after the command line. The gadgettext parameter now gives the length of the data. Setting datain means the data is read from an external file rather than the pipe.

datain="filename" (di=)

Read the data for this gadget from a file. It is ignored if the Bindata keyword is not present. A length of 0 means to use the complete file.

shorthevent (se)

The events sent by the textfield gadget do not include the contents of the textfield. This parameter only works if modify is turned on. Also see events above.

centerjustify (cj) rightjustify (rj)

Set the justification of the text for the gadget. Defaults to leftjustify.

bevel=TYPE (b=)

Set the bevel type for this gadget.

0=NONE 1=THIN 2=BUTTON 3=GROUP 4=FIELD 5=DROPPBOX 6=SBAR\_HORIZ 7=SBAR\_VERT 8=BOX 9=STANDARD

minwidth=number (minw=)

Set the minimum width for this gadget

minheight=number (minh=)

Set the minimum height for this gadget

weightedwidth=number (weiw=)

Set the weighted width for this gadget

weightedheight=number (weih=)

Set the weighted height for this gadget

nominalsize (noms)

Set this gadget to its nominal size.

replace=GID

Use this gadget to replace an existing gadget specified by GID.

Definition Reply -----

When the gadget creation is successful the pipe replies with

'ok GID'

Textfield gadget modify parameters. ----- disable=number (dis=)

The gadget is enabled if number = 0 , disabled if number != 0.

gadgettext="text" (gt=)

Set the text in the gadget to text if scroll keyword is not used. Else insert the text at the position set by scroll.

scroll="num"

Set the cursor to position num.

delete=num

Delete num of characters from the cursor forward.

bindata (bd)

The text will be passed to this pipe in binary after the command line. The gadgettext parameter now gives the length of the data. Note that you can NOT use datain during modify.

All modifications of a textfield (except a lone disable) are responded to by 'ok POSITION' where POSITION is the final cursor position.

## 1.48 4.2.5.5 ListBrowser

4.2.5.5 ListBrowser -----

ListBrowser gadget event -----

'gadget GID event\_source event\_column NodeGID [NodeGID [NodeGID ...]]'

event\_source&1 means normal selection of node.

event\_source&2 means this nodes children have been hidden.

event\_source&4 means this nodes children are now shown.

event\_source&8 means node has been edited.

event\_source&16 means double click on node.

event\_column is the column the mouse was over when the node was clicked.

ListBrowser gadget parameters. -----

font=GID

This sets the font for the gadget. The GID points to an ALREADY defined text attribute.

readonly (ro)

This list browser entries can not be edited.

disable=number (dis=)

The gadget is enabled if number = 0 , disabled if number != 0. Defaults to enabled.

selected=number (s=)

Set which choice is selected when the window opens. Selections start at 0 . This defaults to none. CAREFUL modify uses the listnode GID, this parameter uses the ordinal position of the node.

ListbrowserLabels="text0|text1|text..." (lbl=)

Set the column labels for the list. Defaults to 1 unlabeled column. NO MORE THAN 20 COLUMNS can be defined !

bnparent (bnp)

Allow parenting in this listbrowser.

showtitles (st)

Show the column tiles of this listbrowser.

vertical (v)

Use vertical separators in the listbrowser.

hori (h)

Use horizontal separators in the listbrowser.

arrows (a)

Show arrows for horizontal scrolling and make this a virtual width list.

multi (m)

Switch to allow multi selection in the list. Multi select by holding the shift key.

minwidth=number (minw=)

Set the minimum width for this gadget

minheight=number (minh=)

Set the minimum height for this gadget

weightedwidth=number (weiw=)

Set the weighted width for this gadget

weightedheight=number (weih=)

Set the weighted height for this gadget

nominalsize (noms)

Set this gadget to its nominal size.

SPECIAL FUNCTION ----- /\* do not use this function it may be removed !\*/ /\* instead see the note at the end of this doc \*/ Using sort on a line BY ITSELF during gadget definitions will cause the last defined listbrowser to be sorted. This is to allow a list browser to be sorted before the gui window is opened. (the modify sort command sorts the listbrowser after the window is opened.)

sort=COLUMN#

(optional) sort=COLUMN# defn=COLUMN#

Sort list browser based on the column number, if the sort entries are the save a secondary sort based on the defaultnumber column is done. Specifying a secondary column to sort by is optional. Remember columns start at 0. You MUST NOT specify an invalid column number.

Definition Reply -----

When the gadget creation is successful the pipe replies with

'ok GID'

ListBrowser gadget modify parameters. ----- down

Select the next node in the list.

up

Select the previous node in the list.

addnode (addn)

Add a new node to this ListBrowser.

You should keep this modify command as simple as possible, only use keywords that specifically deal with the creation of the node. (tar gt bnp defn sc )

Put additional changes like select= or disable = in a separate modify line. Some keywords like refresh, tick, disable, sort, and others could cause extra 'oks' to be in the response line.

Keep it simple and this modify command is responded to with 'ok GID' where GID is the ID of the browser node.

GadgetText="[^\[]text0|^\[]text1|^\[]text..." (gt=)

Set the text(s) for the added node. This parameter MUST be given if you are using 'addnode' . YOU MUST NOT GIVE MORE TEXTS THAN THE AMOUNT OF COLUMNS IN THE LISTBROWSER. A leading '^' makes the text editable. A leading '[' (ALT p) uses the last defined image for the column rather than text. Note that images used in browser nodes can not be more than 255 pixels high. (hint: split larger images and use more than one node)

defnumber=number (defn=)

Set the Generation of this browser node.

target=GID (tar=)

If GID=0 add the new node at head of list. If GID=-1 add the new node at tail of list. Else add the new node immediately after the node specified by GID.

browsernodeparent (bnp)

The new browser node has children.

List=number

If number=0 detach the node list from the listbrowser. If number!=0 then reattaches the node list to the listbrowser.

removenode (remn)

Remove ALL browser nodes from this ListBrowser.

readonly (ro)

This list browser entries can not be edited.

disable=number (dis=)

The gadget is enabled if number = 0 , disabled if number != 0.

scroll=GID (scr=)

Scroll the listbrowser so the node specified by GID is at the top.

selected=num (s=)

For multi select lists

Setting selected=0 sets all nodes as unselected. Setting selected=-1 sets all nodes as selected. You may set the nodes individually instead of using this command. See browsernode modify commands.

selected=node\_GID (s=)

For single select lists

Set the selected node. Selected=0 sets all nodes as unselected, else the specified node is selected. CAREFUL modify uses the listnode GID not the ordinal position. This is different then in the listbrowser definition.

order

Read the order of nodes in the listbrowser. This does not return the normal 'ok' instead it returns a sequence of GID's in the same order as the nodes in the listbrowser. THIS COMMAND SHOULD NOT BE USED WITH ANY OTHER MODIFY COMMANDS ON THE SAME LINE.

sort=COLUMN#

(optional) sort=COLUMN# defn=COLUMN#

Sort list browser based on the column number, if the sort entries are the save a secondary sort based on the defaultnumber column is done. Specifying a secondary column to sort by is optional. Remember columns start at 0. You MUST NOT specify an invalid column number.

autofit

This keyword forces the listbrowser to evaluate its size. It is only needed when a listbrowser uses a font other than the screen default font AND no new browser nodes are added after the GUI window is opened.

hidechild=num (hc=)

If num =0 show all the children in this listbrowser. If num !=0 then hide all the children in this listbrowser.

NOTE: At times you may want to modify a listbrowser after is is created, but BEFORE the window is opened. Use the 'NoWindow' keyword in the window definition, create the GUI as usual. When you send 'open' the window does NOT open but you do go into modify mode. Modify the GUI then send 'id 0 s 64' to open the window after it has been modified.

## 1.49 4.2.5.6 Palette

4.2.5.6 Palette -----

palette gadget event -----

'gadget GID selected\_color'

Palette gadget parameters. -----

font=GID

This sets the font for the gadget. The GID points to an ALREADY defined text attribute.

close (c)

This gadget will close the window when selected by the user.

readonly (ro)

This gadget is read only.

disable=number (dis=)

The gadget is enabled if number = 0 , disabled if number != 0. Defaults to enabled.

minnumber=number (minn=)

The first color to display in this gadget. Defaults to 0.

maxnumber=number (maxn=)

How many colors to display in this gadget. Defaults to 8.

selected=number (s=)

Color selected when window is opened. Defaults to 0.

minwidth=number (minw=)

Set the minimum width for this gadget

minheight=number (minh=)

Set the minimum height for this gadget

weightedwidth=number (weiw=)

Set the weighted width for this gadget

weightedheight=number (weih=)

Set the weighted height for this gadget

nominalsize (noms)

Set this gadget to its nominal size.

replace=GID

Use this gadget to replace an existing gadget specified by GID.

Definition Reply -----

When the gadget creation is successful the pipe replies with

'ok GID'

Palette gadget modify parameters. -----

disable=number (dis=)

The gadget is enabled if number = 0 , disabled if number != 0.

selected=number (s=)

Set the color selected in the gadget

## 1.50 4.2.6 More

4.2.6 More -----

[4.2.6.1 Fuelgauge](#) [4.2.6.2 Scroller](#) [4.2.6.3 Slider](#) [4.2.6.4 WeightBar](#) [4.2.6.5 Commodity](#) [4.2.6.6 Sound](#)

### 1.51 4.2.6.1 Fuelgauge

4.2.6.1 Fuelgauge -----

The Fuel gauge do not produce events.

FuelGauge gadget parameters. -----

font=GID

This sets the font for the gadget. The GID points to an ALREADY defined text attribute.

close (c)

This gadget will close the window when selected by the user.

readonly (ro)

This gadget is read only.

disable=number (dis=)

The gadget is enabled if number = 0 , disabled if number !=0. Defaults to enabled.

gadgettext="text" (gt=)

Set the text of the gadget.

centerjustify (cj)

Set the justification of the text for the gadget. Defaults to leftjustify.

vertical (v)

This fuelgauge is vertical. Defaults to horizontal.

ticks=number (t=)

Set the 'number' of ticks in the fuel gauge. Defaults to 11.

percent (per)

Show the percentage full as the fuel gauge text.

minnumber=number (minn=)

Empty value for this gadget. Defaults to 0.

maxnumber=number (maxn=)

Full value for this gadget. Defaults to 100.

defnumber=number (defn=)

Value for this gadget when window is opened. Defaults to 50.

minwidth=number (minw=)

Set the minimum width for this gadget

minheight=number (minh=)

Set the minimum height for this gadget

weightedwidth=number (weiw=)

Set the weighted width for this gadget

weightedheight=number (weih=)

Set the weighted height for this gadget

nominalsize (noms)

Set this gadget to its nominal size.

tags="tags\data[|tag\data...]|0

A list of tag and value pairs in HEX. Do not forget the trailing null !

childlabel (chl) childlabelr (chlr)

Use the previously defined image as a childlabel for this gadget. Display the childlabel to the right or left of the gadget.

replace=GID

Use this gadget to replace an existing gadget specified by GID.

Definition Reply -----

When the gadget creation is successful the pipe replies with

'ok GID'

FuelGauge gadget modify parameters. -----

disable=number (dis=)

The gadget is enabled if number = 0 , disabled if number != 0.

gadgettext="text" (gt=)

Set the text of the gadget.

---



defnumber=number (defn=)

Value for this gadget .

get=TAG

Read the value of a specific gadget attribute. Returns a blank line if attribute is not readable. This should not be combined with any other modify command on the same line.

refresh (ref)

Redraw the gadget.

## 1.52 4.2.6.2 Scroller

4.2.6.2 Scroller -----

Scroller gadget event. -----

'gadget GID scroller\_value'

Scroller gadget parameters. -----

font=GID

This sets the font for the gadget. The GID points to an ALREADY defined text attribute.

readonly (ro)

This gadget is read only.

disable=number (dis=)

The gadget is enabled if number = 0 , disabled if number != 0. Defaults to enabled.

vertical (v)

This gadget is free vertically, defaults to free horizontally.

arrows (a)

Switch to display arrows for this gadget.

delta=number

The change in the gadget when an arrow is clicked. Defaults to 1.

knob=number

The size of the knob in the scroller.

maxnumber=number (maxn=)

Maximum value for this gadget. Defaults to 100.

defnumber=number (defn=)

Value for this gadget when window is opened. Defaults to 0.

minwidth=number (minw=)

Set the minimum width for this gadget

minheight=number (minh=)

Set the minimum height for this gadget

weightedwidth=number (weiw=)

Set the weighted width for this gadget

weightedheight=number (weih=)

Set the weighted height for this gadget

nominalsize (noms)

Set this gadget to its nominal size.

replace=GID

Use this gadget to replace an existing gadget specified by GID.

Definition Reply -----

When the gadget creation is successful the pipe replies with

'ok GID'

Scroller gadget modify parameters. -----

disable=number (dis=)

The gadget is enabled if number = 0 , disabled if number != 0.

defnumber=number (defn=)

Value for this gadget.

knob=number

The size of the knob in the scroller.

maxnumber=number (maxn=)

Maximum value for this gadget. Defaults to 100.

## 1.53 4.2.6.3 Slider

4.2.6.3 Slider -----

Slider gadget event. -----

'gadget GID slider\_value'

slider gadget parameters. -----

readonly (ro)

This gadget is read only.

disable=number (dis=)

The gadget is enabled if number = 0 , disabled if number != 0. Defaults to enabled.

ticks=number

Set how many ticks to display along the slider.

Sticks

Show every second tick as a short tick.

Flip

Reverse the min and max positions on the slider.

vertical (v)

This gadget is free vertically, defaults to free horizontally.

maxnumber=number (maxn=)

Maximum value for this gadget. Defaults to 100.

minnumber=number (minn=)

Maximum value for this gadget. Defaults to 100.

selected=number (s=)

Value for this gadget when window is opened. Defaults to 0.

useimage (ui)

Use the last defined image as the knob for this slider.

bodyimage (bi)

Use the last defined image as the body of this slider.

NOTE: if both useimage and bodyimage are given, the knob gets the last defined image , the body gets the send last defined image.

minwidth=number (minw=)

Set the minimum width for this gadget

minheight=number (minh=)

Set the minimum height for this gadget

weightedwidth=number (weiw=)

Set the weighted width for this gadget

weightedheight=number (weih=)

Set the weighted height for this gadget

nominalsize (noms)

Set this gadget to its nominal size.

replace=GID

Use this gadget to replace an existing gadget specified by GID.

Definition Reply -----

When the gadget creation is successful the pipe replies with

'ok GID'

slider gadget modify parameters. -----

disable=number (dis=)

The gadget is enabled if number = 0 , disabled if number != 0.

selected=number (s=)

Value for this gadget.

## 1.54 4.2.6.4 WeightBar

4.2.6.4 WeightBar -----

WeightBar parameters -----

WeightBar

Place a user controllable weightbar in the GUI. WeightBar has no parameters.

Definition Reply -----

When the gadget creation is successful the pipe replies with

'ok'

## 1.55 4.2.6.5 Commodity

### 4.2.6.5 Commodity -----

#### Commodity event -----

Commodity "cx TYPE [SUBTYPE]"

TYPE=hotkey The commodities hotkey has been detected. TYPE=kill The commodity has been requested to quit. TYPE=unique Another commodity with the same name tried to start (but always fails) TYPE=show The commodity has been requested to show its GUI. TYPE=hide The commodity has been requested to hide its GUI. TYPE=disable The commodity has been disabled. TYPE=enable The commodity has been enabled.

With dis/enable your commodity is automatically enabled or disabled. You MUST respect this in your program. With other commodity events you SHOULD TRY to react in some proper way.

The next three types are included to allow for future enhancements of the Amiga Commodity ability.

TYPE=command SUBTYPE The commodity has received an unrecognized command. The subtype is the messageID. TYPE=event SUBTYPE The commodity has received an unrecognized event. The subtype is the messageID. TYPE= NUMBER SUBTYPE The commodity has received an unrecognized message type. Number is the message type, subtype is the messageID.

#### Commodity parameters -----

You may only create one commodity object, per GUI.

CxName="name"

Name Commodities uses to identify this commodity.(REQUIRED)

CxPri=number

The priority of your commodity, defaults to 0.

CxTitle="title"

Title of commodity that appears in CXExchange.

CxDesc="description"

Description of the commodity.

CxHotkey=" hotkey description"

example CxHotKey="ctrl alt f1"

CxNoGui

Do not allow CXExchange to request GUI open and closing.

#### Definition Reply -----

When the commodity creation is successful the pipe replies with

'ok'

#### Modify parameters -----

Normally you should not need to disable or enable your commodity as it is handled automatically. This gives you the ability to override the automatic control, use it carefully.

Commodity=number

If number =0 then disable the commodity, else enable the commodity. Replied to with ok.

## 1.56 4.2.6.6 Sound

### 4.2.6.6 Sound -----

Sound objects do not generate events -----

Sound object parameters. -----

FileName="path:file"

Specify the file file containing the audio data. This parameter MUST be given. Datatypes are used to generate the sound object.

Volume=NUM

The volume to play the sound at 0-64. Defaults to 64.

Period=NUM

The period for playback of the sound. Defaults to natural period for the sample.

Cycles=NUM

The amount of cycles of the sound to play. Defaults to 1.

Definition Reply -----

When the object creation is successful the pipe replies with

'ok GID'

Sound object modify parameters. -----

Period=NUM

The period for playback of the sound.

Volume=NUM

The volume to play the sound at 0-64. Defaults to 64.

NOTE: To work around a pre-ADOS 3.5 sound.datatype bug you can change the volume or frequency at the same time playback is triggered.

topipe('ID soundGID Volume new\_volume Period new\_period S 2')

Under OS3.5 volume and frequency can be changed any time, but period may not work as expected.

Selected=TriggerMethod (s=)

Cause the sound object to react. Most sound objects only support the trigger method Play (2) and method Stop (14).

Some other trigger methods for future compatability.

PAUSE 1 PLAY 2 REWIND 12 FASTFORWARD 13 STOP 14 RESUME 15

Select is replied to with 'ok RESULT' where result =1 when the trigger method was supported and result=0 when it is not supported.

NOTE: Trigger method 14 (stop) appears to be broken under ADOS 3.x and can hang the system. It should not use it at present.

FreeSound

FreeSound will dispose of the sound object. Under ADOS 3.1 the command may wait until the sound finishes playing. Under ADOS 3.5 the sound is stopped immediatly and the object freed.

Sound objects are disposed of automaticaly when the GUI host is closed. Uses this function when it is better to free the sound object sooner.

## 1.57 4.3 Events

### 4.3 Events -----

EVENT TYPES ----- Gadget hit "gadget GID [gadget specific data]" Read gadget information for specifics.

Help "help GID MOUSEX MOUSEY<cr>" The mouse is over the gadget specified. The mouse location is useful in determining where to open a help window.

Keystroke "key KEYCODE QUALIFIER<cr>" The user hit a key. Raw keycodes are returned along with the current qualifier.

Qualifier "qual QUALIFIER<cr>"

A change in the current qualifier is detected. Not all qualifier changes will cause events. HOWEVER the last qualifier event will ALWAYS be correct for the event(s) that follow it.

Close "close CLOSESOURCE<cr>" The window has been closed. CLOSESOURCE=0 WindowClose Gadget CLOSESOURCE=-1 Closed by CTRL-BackSlash else CLOSESOURCE=GID Closed by GID

Commodity "cx TYPE [SUBTYPE]"

The Commodity object received a message.

AskClose "askclose"

User tried to close the GUI window. (not always used see window docs)

Refresh "refresh"

The window has been sized or altered and you should refresh any components you manually draw into the window. (You only ask for and watch these events if you use the 'Draw' modify command.)

Menu "menu MENU# ITEM# SUBITEM# CHECKED<cr>"

Menu selected. checked=0 Item is not checked checked=1 Item is checked

Be careful of false (undefined) menu hits.

Window Active "active 0|1<cr>" 0= window inactive 1= window active

Application "app FILENAME <cr>"

An icon has been dropped on the window.

Iconify "iconify MODE<cr>"

mode 0 = uniconify mode 1 = iconify

Iconify events are only sent if modify is turned on. You must manually iconify or uniconify the window in this case. See window.doc.

If modify is not turned on the iconification is automatic and no iconify events are sent.

ARexx command "arexx FID len <cr> command"

You must first read the arexx event, the read len additional characters.

See the docs for each gadget type for more information.

## 1.58 4.4 Modify

### 4.4 Modify -----

Modifying the GUI Window and its gadgets =====

See the documentation for each gadget type for information on modifying that type of gadget type. This document explains many 'special commands' that can be used to modify the GUI in general. It also has some commands that can be used on any gadget type.

Most modify commands are replied to with either 'ok' or 'error'. If more than one gadget parameter is changed by the modify command the reply can be 'ok ok ...'. A reply of a blank line means the modify command was not understood and was ignored.

Modify commands that change gadget parameters begin with 'id GID' Gadgets are modified with the following line format

'id GID parameter1[=value] parameter2[=value] parameter3[=value] ...

example 'id 2 gadgettext="new text" disable=1'

The parameters allowed depend on the type of gadget that GID refers to.

NOTE: When modifying gadgets on a clicktab page, you should add the page parameter to the modify line.

NOTE: At times you may want to modify a GUI after it is created, but BEFORE the window is opened. Use the 'NoWindow' keyword in the window definition, create the GUI as usual. When you send 'open' the window does NOT open but you do go into modify mode. Modify the GUI then send 'id 0 s 64' to open the window after it has been modified.

Modify Parameters that can be used with all gadgets (except menu, arexxhost, textattr, commodity...) -----

page="GID"

Specify the clicktab gadget that controls the page containing the gadget to be modified. Most gadgets seem to modify fine without this being set, but it must be used for string gadgets. It should probably be used when modifying all gadgets on clicktab pages to ensure future compatability. NOTE: you supply the Gadget ID of the clicktab, NOT the number of the page inside the clicktab.

set

'Set' sets gadget parameters from the supplied tag list. Both 'set' and 'tags' must be used to do this.

tags="tags\data[|tag\data...]|0

A list of tag and value pairs in HEX. Do not forget the trailing null !

Refresh (ref)

Refresh the entire GUI when used alone, or refresh a specific object when used with a GID.

Informational Gadget modify instructions -----

read

Read must not be used with other parameters besides GID. 'id GID read' Reads the gadget specified by GID. The pipe returns the same information as when a user hits a gadget but without the leading 'gadget gid'. In the case of GID=0 read returns the windows 'left top width height' settings.

bounds

Read the size and location of object associated with the GID. This modify command MUST be 'id GID bounds' and returns 'LEFT TOP WIDTH HEIGHT'

address

Read the address of object associated with the GID and information about the gadget type. This modify command MUST be 'id GID address' and returns 'ADDRESS GADGETTYPE'

An address of 0 means no object is associated with the GID.

mouse

Get the position of the pointer. This returns the both the position relative to the window and relative to the screen.

'windowX windowY screenX screenY'

Get="hexvalue"

This special modify command lets you read a value associated with a gadget The pipe replies with 'ok value' when successful, a blank line if it fails.

example 'id 3 get xxxxxxxx'

This reads the selected color in a palette gadget (whose GID is 3).

Gadget interconnection modify -----

This function may only be useful to programmers already familiar with interconnection who have registered CA documentation.

Target="gadget\_ID" (tar=)

The gadget to be interconnected with.

tags="tagsourceltagdest[|tagsourceltagdest..]|0

The list of tags for interconnection in HEX. Tags must be in pairs, and do not forget the trailing null!

example. 'id 3 target 2 tags="50010051500100510" '

Selections made on chooser gadget 3 are connected to chooser gadget 2.

Modify Control instructions -----

Some modify commands do not actually modify any gadgets.

pointer

Set the pointer for the window, if pointerdata is not specified set the pointer to default pointer. I pointer data is given see below.

pointerdata="height|width|hotx|hoty|data1|data2|..." (pd=)

Pointer data is the definition for a pointer. For more information see the intuition call SetPointer(). Pointerdata="" is a special case setting the pointer to a null (invisible) pointer.

specialchar="character" (sc=)

'Character' is a single character that replaces '|', in use as a separator inside parameters like chooserlabels, tags, penmapdata, ...

You should usually use this alone on a modify line, used inside modify commands like add node it will cause an extra 'ok' in the response line.

Quiet (q)

Tell the pipe not to reply to the modify commands. This is actually a toggle switch turning replies off and on. SOME modify commands are ALWAYS replied to. (addnode and getfile selected 011)

refresh (ref)

Refresh the gadget referenced by GID. The refresh command can be combined with any other modify command and is always executed last. GID=0 will refresh the windows root layout, this is the best way to refresh all gadgets.

Refresh without a GID specified will refresh the entire window, usually it is better to refresh the root class as described above.

close

Close the CA window. Dispose of window and end the pipe connection to the GUI host.

tick=number

End the modify conversation, Reply with a tick event after number/100 seconds.

Continue (con)

End the modify conversation.

Modifyoff (m)

End the modify conversation AND turn modify off. (do not start any new modify conversations). Note there will be no way to turn modify back on.

Beep=num

Beep the display. If num =0 beep all screens , else beep the GUI's screen.

Bubble Help -----

Help="num"

If num =0 turn help events off, if num!=0 then turn help events on.

Each GUI can only show one bubble help window at a time.



```
bubble top=numx left=numy gt="Help Text"
```

This opens a bubble showing the help text. Top and left are mouse location for the help (as sent to you in the help event you are responding to). Any previous help window will be closed.

'bubble' sent with no parameters will close the bubble with out opening a new one.

Image Functions -----

```
FreeImage="num" fi=
```

if num=-1 Dispose of a previously defined image without using it. Else free the root bitmap image in buffer num. Be careful not to do this while images still exist that reference this root bitmap.

```
Draw="num" top="num" left="num" [target=GID][nozz][noclip]
```

Draw an image into the gui. Num is the number of the predefined image to be drawn. You MUST set 'top' and 'left' and can also set

```
target=GID
```

Reference (clip & offset) drawing to inside the object specified by GID. If target is not given drawing is referenced to inside the borders of the GUI.

```
nozz
```

Make top and left relative to the whole window rather than the inner panel or specified target.

```
noclip
```

Allow images to overwrite the entire window, borders and all.

The image number relates to images you have created but not used in your GUI. For example...

You create an image 'bitmap fn="cat.gif"' it is image 0. You create another image 'bitmap fn="dog.gif"' it is image 1. You create another image 'bitmap fn="bird.gif"' it is image 2.

You create a BUTTON using the last image created 'button useimage', the button used the bird image so it is no longer in the image list.

You create another image 'bitmap fn="frog.gif"' it is image 2.

You now have 3 images available to draw

```
0 cat 1 dog 2 frog
```

Drawing images DOES NOT remove images from the image list. Incorporating images directly in your GUI DOES remove images from the image list. The image number is NOT related to the buffer number in the bitmap image parameters.

```
GID 0 -----
```

GID 0 is used to specify the GUI window itself. It can be used to control certain aspects of the window. (see window docs)

Defining new gadgets during modify -----

```
define
```

This causes the current line to be used as a Gadget definition statement rather than modify. You can add or replace gadgets in the already opened window. You may need to send a refresh modify command to make the changes visible.

```
replace="GID"
```

The newly defined gadget replaces the existing gadget specified by GID. The pipe tries to free the resources used by the previous gadget right away, but some memory is not freed until the pipe is closed.

WHEN USING REPLACE DO NOT ASSUME WHAT THE RESULTING GID OF THE NEW GADGET WILL BE. The usual sequence of allocated gadget ID's is not followed. Replacing a layout group frees all gadgets in the group as well as the layout itself. Replacing a listbrowser frees its nodes.

You MUST keep track and not try to access gadgets that no longer exist.

It may be better to close the window and open a new one rather than using replace or define. On the other hand some pretty neat thing can be done using define and replace. ;-D

## 1.59 4.5 GUI Tutorial

4.5 GUI Tutorial -----

[4.5.1 Design](#) [4.5.2 Operation](#) [4.5.3 Modification](#) [4.5.4 Extras](#) [4.5.5 Advanced](#) [4.5.6 Tips](#)

### 1.60 4.5.1 Design

4.5.1 Design -----

Working through these tutorials in order is the best way to learn to build GUI's with AWNPipe. A few minutes here will save you hours of development time later.

TUTORIAL 1 -----

Building GUIs can be VERY easy, This first tutorial will develop a simple GUI.

Create a simple 4 line text file called first.gui. (You can use drag selection and Right-Amiga-C to copy text from these docs.)

```
title "My first GUI" defaultgadgets button gadgettext "YES" button gt "NO" open
```

Now, in a shell, type 'copy first.gui awnpipes:/xc'.

That was easy. You already guessed that gt was a short form for gadgettext.

Now we want to see what events the GUI sends. We can direct the events to a con by adding 'wcon:' to the end of the pipe name. Try

```
'copy first.gui awnpipes:/xcwcon:'
```

Look at the output in the con:. Notice that the pipe replies to each line of the definition file, as well as sending events.

Now add a little text to the gui. We will also provide keyboard shortcuts for the gadgets.

```
title "My first GUI" defaultgadgets label gt " Select YES or NO " button gadgettext "_YES" button gadgettext "_NO" open
```

That worked, but it looks a little UGLY. Let's try a vertical layout instead of the default horizontal layout.

```
title "My first GUI" defaultgadgets vertical label gt " Select YES or NO " button gadgettext "_YES" button gadgettext "_NO"
open
```

The GUI is starting to look better, but we really want the two gadgets side-by-side. To do that, we will use a layout group. ("le" is short for "layout end".)

```
title "My first GUI" defaultgadgets vertical label gt " Select YES or NO " layout button gadgettext "_YES" button gadgettext
"_NO" le open
```

That is much better! The GUI looks good, but it would be nicer if the window closed after the user selected yes or no. (You should also notice that the layout received a GID number, so yes and no are now gadget 2 and 3.)

```
title "My first GUI" defaultgadgets vertical label gt " Select YES or NO " layout button gadgettext "_YES" close button gadgettext
"_NO" c le open
```

It's done and it might even be useful.

Since we have a working GUI, let's use it to see a few other types of events the GUI can generate.

```
title "My first GUI" defaultgadgets vertical sendkeys sendqual help state label gt " Select YES or NO " layout button gadgettext
"_YES" close button gadgettext "_NO" c le open
```

Activate and deactivate the window. Hold the mouse over each gadget. Type a few keys. Try the shift, alt etc.

## 1.61 4.5.2 Operation

### 4.5.2 Operation -----

This tutorial has two versions, Arexx and 'C'. The 'C' tutorial is after the Arexx one.

Before reading further, take a moment to run the tutorial so you understand what the program does.

### TUTORIAL 2 ARexx -----

Now that we know how to designed a GUI we want to be able to operate one. To follow this example you need a basic knowledge of ARexx. Have a look at the file Tutorial2.rx in the demos drawer.

The program flow is straight forward. 3 simple steps.

1. Default values are set. These are the values of the gadgets when the GUI opens.
2. A routine (buildgui) is called to write the GUI definition to the pipe. (More on this routine later)
3. Events are read from the pipe and processed until the stream of events ends.

### BUILDGUI

The 'buildgui' routine contains the information defining the GUI window and its gadgets. The data is sent line by line to the 'topipe' routine which sends the data to the pipe, checks for errors, and returns the GID returned by the pipe. (more on 'topipe' later.)

For most lines sent with 'topipe' the response is ignored. For gadget definitions however the return value is stored so we can identify gadget events later on.

### TOPIPE

This routine takes a line of text and writes it to the pipe. The pipes response is read. If everything is ok the second parameter of the response is returned. This is the GID if a gadget is defined and a null string in most other cases. If an error occurs the problem line is output and the script exits.

You should use this routine or one like it when writing your own script for AWPipe. It will help avoid common errors and make debugging much easier.

### EVENT HANDLING

The main loop reads an event from the pipe and parses it into its parts. The first word of an event always tells the event type. This information is used to call a routine for that type of event.

### CLOSE EVENT

If a close event is received write a little text and then exit the script.

### GADGET EVENT

We check the second word of the event to see which gadget sent the event. For most gadgets we store event information and return. If the cancel gadget is the cause of the event we output some text and exit. If it was the done gadget we output the information for each gadget and exit. If it was the reset gadget we close the pipe, then create the gui again to restore the default values.

### MENU EVENT

The second word of a menu event is the menu number, the third word id the item number, and the forth the subitem number. This information is used to determine what action should be taken in response to an event.

Almost any GUI can be operated using this approach demonstrated in this tutorial. Use these functions as building blocks in your own scripts.

### TUTORIAL 2 in C -----

The files tutorial2.c and tutorial2 can be found in in the demos drawer.

Now that we know how to designed a GUI we want to be able to operate one. To follow this example you need a basic knowledge of 'C'. Have a look at the file Tutorial2.c in the demos drawer. We will examine each of the functions in detail since they will be useful building blocks for making other GUIs. There is only 1 interesting structure.

```
struct GUIpipe myGP
```

This structure is used to manage the GUI. It contains the GUI's file handle, error status, and read buffer. It also has storage for some useful data parsed from replies or events.

```
int main( int argc, char *argv[]);
```

The program flow is straight forward. Default values are set. We call a routine to build the GUI and operate the gui if we built it successfully. To operate the GUI we simply read events and react to them. Before we exit we make sure the GUI's filehandle is closed.

```
int getline(struct GUIpipe * GP);
```

We need to handle text one line at a time when operating the GUI. This routine reads the pipe making a sure a full line is available before returning. It will remove the previous line from the buffer first if there was one. We delimit the line with a null byte so we can use it as a string in other functions.

```
__sdargs int topipe(struct GUIpipe * GP, UBYTE * data,...)
```

Again and Again you will need to write a line to the pipe and read the response. This routine does that for you. It writes data to the pipe and then reads a reply from the pipe. The reply line is parsed, and an error is flagged if the first part of the reply is not 'ok'. We return the value of the second reply part if all is ok, return 0 if an error occurred.

topipe allows Printf like formatting of data written to the pipe (VFPrintf is used). The regular Printf formats are supported. This is not needed in this example but it will be very useful as we expand the program in the other tutorials.

```
int buildgui(struct GUIpipe * GP);
```

All the information that defines the GUI is contained in this function. It opens a filehandle on the AWNPipe device and returns an error if this fails. The window and gadget definitions are written to the pipe and the returned gadget ID's are stored. If all gadgets are created ok the window is opened. The final error status of the GUI is returned.

```
int getevent(struct GUIpipe * GP);
```

This routine reads a line from the pipe and parses it storing information about the event. It returns the error status of the pipe.

```
int gadgets(struct GUIpipe * GP); int menu(struct GUIpipe * GP);
```

These routines look at the information parsed from the last event and take appropriate action depending which gadget or menu item was selected. Some times the gadget state information from the event is stored. Other times information is output to the user.

```
int gperror(struct GUIpipe * GP,int error);
```

When an error is detected while reading or writing to the pipe this routine is called. It sets the error status and alerts the user an error has occurred.

```
UBYTE * eventstr(struct GUIpipe * GP, int num);
```

Some fields in events , like string gadget contents, can contain spaces. This functions returns a pointer to the 'num' word of the event. Since the line is delimited by a null you can treat this as a string pointer. If the 'num' word is not found NULL is returned. The pointer is only valid until the next call to getline(), getevent(), or topipe().

Almost any GUI can be operated using this approach demonstrated in this tutorial.

## 1.62 4.5.3 Modification

### 4.5.3 Modification -----

Before reading this section take a moment and run tutorial 3, knowing what the GUI does will help in understanding the code behind it.

This tutorial has two versions, Arexx and 'C'. The 'C' tutorial is after the Arexx one.

### TUTORIAL 3 AREXX -----

This tutorial deals with modifying a GUI after it has been opened. You should make sure you understand tutorial 2 before reading further.

Three new gadgets have added to the gui. The load and save gadgets on the lower left, and a get file gadget that you do not see. The getfile gadget is unattached (ua) so it is not displayed in the gui. More on the getfile gadget later.

The window definition line (in buildGUI:) has the modify option added to it. This means the pipe will look for modify commands after you first open the window, and after each event is sent. Any number of modify commands can be written to the pipe, including 0 (none). After writing the modify commands 'continue' must be sent to the pipe to tell it to start sending events. After you get an event the pipe starts waiting for modify commands again.

Only two changes are made in the program flow.

1. After BUILDGUI is called to build and open the window we send a modify command to the pipe.

```
topipe('id 'savegad' dis 1 ref')
```

The id specifies the save gadget, 'dis 1' disables it, and 'ref' refreshes it so the changes will show. (we want the save gadget disabled because the name gadget is empty at start up, and we need a name before we can save).

2. In the mail loop a 'continue' is written to the pipe BEFORE we wait for an event. This tells to pipe to stop looking for modify commands and send an event.

After an event is read from the pipe it automatically starts looking for more modify events. This means that modify commands can be used immediately after receiving an event from the pipe.

## RESETFORM

In tutorial 2 we had to close then open the GUI to reset its contents. With modify commands we can do this with the gui remaining open. The function resetform is near the end of tutorial3.rx.

This function sends a modify command for each of the gadgets resetting them to the default values. It also disables the save gadget.

## SAVEFORM

This function is called when the user hits the save gadget. It writes the forms contents to a file in 't:'. The files name is the text from the name gadget with '.formdemo' appended to it. The save gadget is disabled unless the name gadget contains text. See the handling of namegad in the GADGET: routine.

## LOADFORM

This function makes use of the unattached GETFILE gadget. A modify command is used to set the file name in the file requester, and to open the requester itself. The pattern used in the file requester was set when the gadget was created in the BUILDGUI routine.

```
call writeln(ca,'id 'getfilegad' fn "t:" s 1')
```

'fn "t:"' sets the file requester to the drawer T:.. 's 1' opens the requester.

The return from the pipe to this modify command is read and parsed to get the file selected and to test if the user selected a file or aborted the requester. The reply from the pipe is ...

```
SUCCESS ["filename"]
```

Success is 0 if the user aborts the file requester, non zero if a file is selected. The file name is always returned in quotes since multiple files can be selected in some situations. (but not this situation).

If the user selected a file, it is opened and the contents placed in the GUI. No error checking is done on the file in order to keep the tutorial clear and easy to understand.

## TUTORIAL 3 in C -----

This tutorial deals with modifying a GUI after it has been opened. You should make sure you understand tutorial 2 before reading further.

```
int buildgui(struct GUIpipe * GP);
```

The window definition line (in buildgui()) has the modify option added to it. This means the pipe will look for modify commands after you first open the window, and after each event is sent. Any number of modify commands can be written to the pipe, including 0 (none). After writing the modify commands 'continue' must be sent to the pipe to tell it to send an event. After you get an event the pipe starts waiting for modify commands again.

Three new gadgets have added to the gui. The load and save gadgets on the lower left, and a get file gadget that you do not see. The getfile gadget is unattached (ua) so it is not displayed in the gui. More on the getfile gadget later.

```
int main( int argc, char *argv[]);
```

Only two changes are made in the program flow.

1. After buildgui() is called to build and open the window we send a modify command to the pipe.

```
topipe(GP,"id %ld dis 1 ref\n",savegad);
```

The id specifies the save gadget, 'dis 1' disables it, and 'ref' refreshes it so the changes will show. (we want the save gadget disabled because the name gadget is empty at start up, and we need a name before we can save).

2. In the main loop a 'continue' is written to the pipe BEFORE we wait for an event. This tells to pipe to stop looking for modify commands and send an event.

After an event is read from the pipe it automatically starts looking for more modify events. This means that modify commands can be used immediately after receiving an event from the pipe.

```
int gadgets(struct GUIpipe * GP);
```

The 'loadgad' and 'savegad' gadgets are recognised, and the handling of 'namegad' and 'resetgad' have changed.

In tutorial 2 we had to close then open the GUI to reset its contents. With modify commands we can do this with the gui remaining open. We set our default form information and call updategui().

Namegad enables or disables the savegad. If no name is set we can not generate a filename to save the information to.

```
int updategui(struct GUIpipe * GP);
```

This routine updates the GUI to show the values we have stored internally. A modify line is sent for each gadget. The function updateform is near the end of tutorial3.rx.

```
int saveform(VOID);
```

This function is called when the user hits the save gadget. It writes the forms contents to a file in 't:'. The files name is the text from the name gadget with '.formdemo' appended to it. The save gadget is disabled unless the name gadget contains text. See the handling of namegad in the gadget() routine.

```
int loadform(struct GUIpipe * GP);
```

This function makes use of the unattached GETFILE gadget. A modify command is used to set the file name in the file requester, and to open the requester itself. The pattern used in the file requester was set when the gadget was created in the BUILDGUI routine.

```
FPrintf(GP->file,"id %ld fn \"t:\" s 1\n",getfilegad); if (getevent(GP)) return(1);
```

'fn "t:"' sets the file requester to the drawer T:.. 's 1' opens the requester.

The return from the pipe to this modify command is read and parsed as an event since it does not return the usual 'ok'. The reply from the pipe is ...

```
SUCCESS ["filename"]
```

Success is 0 if the user aborts the file requester, non zero if a file is selected. The file name is always returned in quotes since multiple files can be selected in some situations. (but not this situation).

If the user selected a file, it is opened and the contents parsed into out internal form information. Then updategui() is called.. No error checking is done on the file in order to keep the tutorial clear and easy to follow.

```
__stdargs int topipe(struct GUIpipe * GP, UBYTE * data,...); int getline(struct GUIpipe * GP); int gperror(struct GUIpipe * GP,int error); int getevent(struct GUIpipe * GP); int menu(struct GUIpipe * GP); UBYTE * eventstr(struct GUIpipe * GP, int num); VOID setdefaults(VOID);
```

These routines have not been changed from Tutorial 2.

## 1.63 4.5.4 Extras

### 4.5.4 Extras -----

If you have read the first three tutorials, but not tried building a GUI of your own, you should try writing a simple gui of your own before reading further. Tutorial 4 will be simpler to understand once you have gained a little experience with AWP.

Before reading this section take a moment and run tutorial 4, knowing what the GUI does will help in understanding the code behind it.

Although Tutorial 4 is available in 'C' or Arexx the discussion is kept general and it applies to both.

### TUTORIAL 4 -----

This tutorial deals with adding some of the finishing touches to AWP GUI's. You should make sure you understand tutorial 3 before reading further.

A few simple changes are made in the GUI. It has an iconify gadget and can be resized. This is done by adding 'ig' iconifygadget, 'sg' sizedgadget and 'ii' iconimage. 'ii' specifies the file (without the .info) whose icon is used when iconifying the GUI, it can be omitted and default icon will be displayed. The 'fh' fixheight limits the sizing gadget to changing the GUI's width.

Another change has been made in the GUI, it is set as an app window 'app'. Now when an icon is dropped on the gui (or its iconified image) an event is generated passing the file name of the icon that was dropped. The event is 'app path:filename'.

We respond to the app event by uniconifying the GUI (if necessary) and loading the file that was dropped. Try saving some info then dropping the icon of the saved info on the GUI.

The iconify gadget also produces a special event. 'iconify MODE' where MODE=1 means the user wants to iconify the GUI, and 0 means it should be uniconified. A simple modify line does this.

'id 0 s 32' to iconify 'id 0 s 64' to uniconify

The menu now has options to snapshot the window. The windows size and location are read with

'id 0 read' the pipe responds with 'TOP LEFT WIDTH HEIGHT' not with 'ok'

The information is stored in ENV(ARC): . Before the GUI is opened we check for the information and use it in the window definition line if its available.

The about item in the menu now opens a separate GUI. It is a simple window with no gadgets just a label used to display some text. Although the routine used to build the about GUI is trivial it has an interesting ability. It can be made to close automatically after a specific delay.

A gui window will close automatically if it tries to read a modify command and you have already closed your end of the pipe. We open the GUI then send a 'tick' modify command to the pipe. This works like 'continue' except the pipe will send back our tick as an event whether the user hits a gadget or not. After sending us the tick event back AWP tries to read a modify command and THEN discovers we have closed the pipe. If the delay is set as 0 we simply turn modify off and the GUI waits until the close gadget is hit.

The data menu items now display information the same way the about requester does.

## 1.64 4.5.5 Advanced

### 4.5.5 Advanced -----

Tutorial 5 adds new functions, Bubble Help, an arexx host, tooltypes and commodity. Although they are advanced features by now it should be easy for you to follow the logic. They work in very similar ways to the other objects you have learned to deal with. Adding these sort of features to your own GUI's will help make them Quality programs.

Before reading this section take a moment and run tutorial 5, knowing what the GUI does will help in understanding the code behind it.

Tutorial 5 is only available in Arexx for now, the discussion is kept general and it applies to both 'C' and ARExx.

### TUTORIAL 5 -----

Most users only scan documentation at best. To allow users to quickly understand your programs Bubble help is very useful.

A few simple changes are made in the window definition. It now tells the GUI to send help and window activation events. This is done by adding 'help' and 'state' keywords.

A menu option has been added to turn the help bubbles on or off.

A change has been made in the main loop, we send 'tick 50' rather than continue when we want to get an event from the pipe. Like continue this ends the modify command part of the conversation and tells the GUI to send us an event. It also forces the GUI to send us a 'tick' event after 50/100 of a second. We use these tick events to delay the displaying of help bubbles.

Tick events have other uses, like returning control back to your code after a set delay so it does not wait forever for an event. For now we will just look at the help delay.

Each time an event is read in the main loop we check a delay counter. If the counter counts down to 0 we open a help bubble (using some previously stored information). The HELPX and HELPY are used to determine a position for the bubble and MUST be given.

```
'bubble top HELPX left HELPY gt "HELP TEXT"'
```

We also must handle new events, help, arexx, cx and active.

We respond to the help event by calling the bubble routine. This routine checks to see if the help event is the same as the last one. If yes we do nothing. If it changed we close the old help window, if we had one, by sending a bubble modify command with no parameters.

```
'bubble'
```

If the event was gadget 0 or -1 we have nothing more to do since they represent the mouse not being over a valid gadget. If the gadget number is valid we store the mouse position that is returned in the help event and the gadget number. We start the delay counter and return.

If we receive any activation event, or iconify the gui, we call bubble(0) to close any open bubble help. This needed in case our window becomes inactive due to another window being opened.

An arexx host has been added to the GUI. The ARExx host name is 'TUTORIAL5'. This allows a few things.

```
topipe(' arexx gt "TUTORIAL5lagelsexknowledgefrontquit"')
```

If upon startup the host already exists we know tutorial 5 is already running so the already running gui and brought to front rather than a second occurrence of tutorial 5 being opened.

The following arexx commands are supported.

front -bring the window to front. quit -quit tutorial5 age [#] -read or set the age sex [#] -read or set the sex knowledge [#] -read or set knowledge (See the rxhst: routine)

Tutorial 5 is also a commodity.

```
topipe('commodity cxname Demo5 cxtile="AWNPipe demo 5" cxdesc="Disable Enable ignored..." cxhotkey="ctrl alt 5"')
```

It supports a hotkey (ctrl alt 5), show or hide the gui, and remove. Enable/disable are ignored as they make no sense in the context of Tutorial 5 ... (See the commodity: routine)

## TOOLTYPES

Tooltype reading is actually not part of the AWPipe GUI host, its a completely different host inside AWPipe. Since tooltypes are desirable for almost all programs it has been added to this tutorial. You find the function simple to use.

### Reading tooltypes

The tooltypes are read by a function called tooltypes:. First the path to our program is found using 'parse source'. Then a special pipe is opened 'awnpipe:name/xtPATH:PROGRAM'. A line containing the tooltype we wish to check is written to this pipe. If the tool type is found we will read back from the pipe 'ok VALUE' where VALUE is the value associated with the tool type we asked for. If the tooltype is not found we read back a <cr> (empty line).

When all the tooltypes have been checked we simply close the pipe connection to the tooltype host.

In tutorial5 the windows size/position and bubble help setting are now kept as a tooltype.

There are also tooltypes to preset the age, knowledge, sex and name. You can force the window to iconify after opening with the iconify tooltype.

### Writing tooltypes



## 1.65 4.5.6 Tips

### 4.5.6 Tips -----

A few quick hints about building and debugging Awnp applications.

Before writing an Awnp GUI application prototype the GUI with a simple text file (mygui). Use 'copy mygui awnp:pipe:pipeName/xcwcon'.

This speeds up testing of layout changes! Also note you can comment out lines in your file by starting them with ';'. Any line with a semicolon in the first position is ignored.

--

### NAME YOUR PIPES !

Rather than opening 'awnp:xc' provide a unique name. Use 'awnp:PipeName/xc'.

A common problem when developing an Awnp application is to find the pipe conversation hangs. Usually this means your program is trying to read data from the pipe, and the pipe is trying to read data from your application. (a read is pending on both ends of the pipe).

This can be cleared by issuing a simple shell command.

```
'echo >awnp:PipeName/a'
```

Both reads are aborted and Awnp usually notices a problem and exits cleanly.

--

Some times it will be useful to trace the conversation your program has with Awnp. Again this can be done with a simple shell command.

```
'type awnp:PipeName/t/i'
```

All data passing either way through the pipe will be output. When your program exits the type command will return. ( this is not limited to to GUI applications. It works for ALL awnp connections!)

You can also use

```
'copy awnp:PipeName/t/i mylogfile'
```

This saves the conversation to the logfile of course.

You can use both at the same time, in fact you can 'twin' the data stream as many times as you like.

Some times during testing you might create a GUI with no close gadget, you can usually close ANY GUI with 'control backslash' like you can with a con:.

--

In general you can not assume that you can modify all the parameters you can define for objects. Also in some cases the parameter names or usage may be different in the modify command when compared to the definition line.

## 1.66 5 Demos

### 5 Demos -----

Several GUIs of example gadgets can be viewed by running the Gadgets# in the demos drawer inside Awnp-Docs. To generate these GUIs a simple text file is being copied to a GUI generating pipe. The 'wcon:300//200/200/PipeEvents' at the end of the pipe name simply sends the output of the pipe to a con so you can see what is going on.

Some example scripts using pipe generated GUIs are also in the demo drawer. These demos are mostly ARexx or ADOS scripts. Load them in your text editor and take a look at them. I think you will find they are very simple script considering the powerful GUIs they create. Reading the comments in the scripts should help you when learning to build and operate your own Awnp GUIs.

---

CAList is an ADOS script. It is graphical wrapper for the standard list command. It is a VERY simple script. Put it in 'C:' with its script bit set if you like it enough to make regular use of it. Use 'calist ?' to get a usage display.

This script gives a good example of how to build a simple gui. It shows how much can be accomplished in a short script.

---

Form is an ADOS script that displays a form. Use 'execute Form' or set the script bit for the file and simply type 'form'.

This gui is truly interactive and requires the name to be set before the form is accepted. It allows resting of the form and knows when the user aborts. Its rather extreme for an ADOS script but is included to show you just how much can be accomplished with ADOS and AwnPipe working together. ADOS wizards may find the handling of the event file and environment variables interesting.

---

Fonttoy is an ADOS script to play with fonts. Use 'execute FontToy' or set the script bit for the file and simply type 'fonttoy'.

This script creates a very simple gui. However it uses gadget interconnection to do some magic that only experienced GUI developers are likely to understand. It also uses numerical tag values, you will likely never do this. Note that numerical tags CAN be used with defining AwnPipe GUIs and this means you will be able to access new CA/Reaction features as they come out without changes to AwnPipe.

---

Dict-thesar.rx is an arexx script. It is run from its icon or shell. words are search in the Merriam Webster site so you must be online. The menu will start and control miami if you have it installed. Bubble help is provided and it is simple to use so no real docs are needed.

You should find this tool useful enough to keep installed on your system.

---

xo.rx is an arexx script. Use 'rx xo.rx' to run it. You must 'cd' into the demo drawer first so it can find its image file (xo.gif). It does not play against you so you must play both sides of the game. (click in the squares).

It changes the button images in a operating GUI. It shows how to create multiple images from a single source image. It also lets you see how to modify a gui after it has been opened. An additional gadget is even added to the GUI after it has been opened. Take a look at xo.gif in an image viewer, you may be surprised to see how small the image is.

---

DropBox.rx can be run from its icon or by typing 'rx dropbox.rx' in a shell. Looking at the tooltypes of Drop box should help you understand what is going on. You set tooltypes to tell the program what to do with files or drawers that are dropped on it. I hope it is something you will keep around and be able to make use of. With a little modification it could be very powerful indeed.

The gui is so simple it almost unbelievable, however it is an app window. That makes it very special. The script also makes heavy use of AwnPipes ability to do pattern matching. Finally the script will also teach you how to read tooltypes using AwnPipe.

Possible tool types.

pat (pattern)=YourCommandText

This tells dropbox to execute your command for any files that match the pattern you have set. '%f' in the command is replaced by the name of the file that was dropped on dropbox.

pat (pattern)=drawer

Tells DropBox to use its default drawer handling for that pattern.

pat (pattern)=image

Tells DropBox to use its default image handling for that pattern.

pat (pattern)=text

Tells DropBox to use its default text handling for that pattern.

bin BinaryPattern=YourCommandText

This tells dropbox to execute your command for any files whose first 12 bytes match the binarypattern you have set. '?' is the only wildcard allowed in the BinaryPattern. '%f' in the command is replaced by the name of the file that was dropped on dropbox. iconify

A tooltip of 'iconify' tells DropBox to iconify itself as soon as it is run. You can still drop files on the iconified window to have dropbox process them.

multi

A tooltip of 'multi' tells DropBox to allow files dropped on it to have more than one command run for them. The default is for dropbox only to only respond once to each file dropped.

---

The first real projects to use AWPipe GUIs were some aweb utilities.

CacheControl.awebrx allows you to micro manage AWebs caching of websites.

Put the script and CacheControl\_doc.html in AWeb3:plugins, it may be called directly from the ARexx menu or from an AWeb button.

GUI Button name CCtrl command run awebpath:plugins/cachecontrol.awebrx

ARexx Menu title CacheControl macro awebpath:plugins/cachecontrol.awebrx

---

AWebModes.awebrx allows you to easily control many of AWebs settings. Gabriele Favrin has kindly allowed a version of it to be included here. Put the script and AWebModes\_Doc.html in AWeb3:plugins, it may be called directly from the ARexx menu or from an AWeb button.

GUI Button name Modes command run awebpath:plugins/AWebModes.awebrx

ARexx Menu title AWebModes macro awebpath:plugins/AWebModes.awebrx

---

DropZone.awebrx opens an appwindow on WB so you can drag and drop files to be displayed in AWeb, even while aweb is on its own screen. Put the script and DropZone\_Doc.html in AWeb3:plugins. It may be called directly from the ARexx menu or from an AWeb button.

GUI Button name DZone command run awebpath:plugins/DropZone.awebrx

ARexx Menu title ChangeModes macro awebpath:plugins/DropZone.awebrx

## 1.67 AWPipe Index

AWNPipe - Index =====

1 About 2 Read First 3 Pipe Functions 3.1 Overview 3.2 Conversion 3.3 Paths 3.4 Internal 3.5 Extensions 3.6 Tutorials 3.6.1 Simple 3.6.2 Advanced 3.6.3 Tricks 4 GUI Creation 4.1 Conversation 4.1.1 Basic 4.1.2 Step by Step 4.1.3 Details 4.2 Objects 4.2.1 Window 4.2.2 Simple Gadgets 4.2.2.1 Button 4.2.2.2 Integer 4.2.2.3 String 4.2.2.4 CheckBox 4.2.2.5 Chooser 4.2.2.6 RadioButton 4.2.3 Images 4.2.3.1 Label 4.2.3.2 Glyph 4.2.3.3 DrawList 4.2.3.4 PenMap 4.2.3.5 Bitmap 4.2.3.6 Image 4.2.3.7 Space 4.2.4 Special 4.2.4.1 Layout(End) 4.2.4.2 ClickTab 4.2.4.3 TextAttr 4.2.4.4 BrowserNode 4.2.4.5 Menu 4.2.4.6 ARexx 4.2.5 Fancy 4.2.5.1 GetFile 4.2.5.2 GetFont 4.2.5.3 TextEditor 4.2.5.4 TextFeild 4.2.5.5 ListBrowser 4.2.5.6 Palette 4.2.6 More 4.2.6.1 Fuelgauge 4.2.6.2 Scroller 4.2.6.3 Slider 4.2.6.4 WeightBar 4.2.6.5 Commodity 4.2.6.6 Sound 4.3 Events 4.4 Modify 4.5 GUI Tutorial 4.5.1 Design 4.5.2 Operation 4.5.3 Modification 4.5.4 Extras 4.5.5 Advanced 4.5.6 Tips 5 Demos