

**ram:Gui4Cli**

**COLLABORATORS**

	<i>TITLE :</i> ram:Gui4Cli		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY		April 18, 2022	

**REVISION HISTORY**

<i>NUMBER</i>	<i>DATE</i>	<i>DESCRIPTION</i>	<i>NAME</i>

# Contents

<b>1</b>	<b>ram:Gui4Cli</b>	<b>1</b>
1.1	MAIN	1
1.2	Action	2
1.3	Advantages	3
1.4	All_the_Commands	5
1.5	Append	12
1.6	AppVar	13
1.7	Assign	13
1.8	BreakTask	13
1.9	CalcVar	14
1.10	CD	15
1.11	ChangeArg	15
1.12	ChangeGad	16
1.13	ChangeIcon	17
1.14	Command_Lines	17
1.15	Copy	18
1.16	Counter	19
1.17	CutVar	19
1.18	c_Gui	20
1.19	Delay	21
1.20	Delete	21
1.21	DelVar	21
1.22	DirList	22
1.23	DoCase	23
1.24	Extract	24
1.25	EZReq	24
1.26	FailAt	25
1.27	Fonts	25
1.28	Font_Sensing	26
1.29	GadFont	27

---

---

1.30	GadHelp . . . . .	28
1.31	GadID . . . . .	28
1.32	GadKey . . . . .	29
1.33	GadTitle . . . . .	29
1.34	GadTxt . . . . .	29
1.35	Gauge . . . . .	30
1.36	GoSub . . . . .	30
1.37	Graphics . . . . .	31
1.38	GuiEdit . . . . .	32
1.39	GuiLoad . . . . .	35
1.40	GuiRename . . . . .	36
1.41	GuiScreen . . . . .	36
1.42	GuiWindow . . . . .	36
1.43	If . . . . .	37
1.44	IfExists . . . . .	38
1.45	Images . . . . .	39
1.46	Important_Topics . . . . .	39
1.47	Info . . . . .	41
1.48	Introduction . . . . .	41
1.49	JoinFile . . . . .	44
1.50	KillScreen . . . . .	44
1.51	Launch . . . . .	45
1.52	Licence . . . . .	45
1.53	ListView . . . . .	47
1.54	LTWH . . . . .	49
1.55	LVAction . . . . .	50
1.56	LVDir . . . . .	50
1.57	LVHook . . . . .	51
1.58	LVMulti . . . . .	52
1.59	LV_Commands . . . . .	53
1.60	MakeDir . . . . .	55
1.61	MakeScreen . . . . .	55
1.62	Mark . . . . .	56
1.63	NewFile . . . . .	56
1.64	Operation . . . . .	57
1.65	Operator . . . . .	58
1.66	Options . . . . .	59
1.67	ParseVar . . . . .	61
1.68	PartRedraw . . . . .	61

---

---

1.69	Passing_Args	62
1.70	Paths	64
1.71	Programming	64
1.72	Quit	67
1.73	Redraw	68
1.74	Rename	68
1.75	ReqFile	68
1.76	ReSize	69
1.77	Run	70
1.78	Say	71
1.79	Screen	71
1.80	SendRexx	71
1.81	Set	72
1.82	SetColor	74
1.83	SetGad	74
1.84	SetGadValues	75
1.85	SetScreen	75
1.86	SetStack	76
1.87	SetVar	76
1.88	SetWintitle	76
1.89	Status	77
1.90	Stop	77
1.91	System	77
1.92	Text	78
1.93	Translation	79
1.94	TTGet	79
1.95	Update	80
1.96	UseTopaz	81
1.97	Variables	81
1.98	Variables_Env	82
1.99	Variables_Int	83
1.100	Variables_IntList	84
1.101	Variables_Normal	87
1.102	VarPath	91
1.103	Wait	92
1.104	While	92
1.105	WinBackGround	93
1.106	WinBig	94
1.107	WinFont	94

---

---

1.108WinOnMouse . . . . .	95
1.109WinOnWin . . . . .	95
1.110WinOut . . . . .	95
1.111WinShortcuts . . . . .	96
1.112WinSmall . . . . .	97
1.113WinType . . . . .	98
1.114xAppIcon . . . . .	98
1.115xAppMenu . . . . .	99
1.116xAppWindow . . . . .	99
1.117xArea . . . . .	99
1.118xButton . . . . .	100
1.119xCheckBox . . . . .	100
1.120xCycler . . . . .	101
1.121xIcon . . . . .	101
1.122xLVHook . . . . .	101
1.123xMenu . . . . .	102
1.124xOnJump . . . . .	103
1.125xOnKey . . . . .	103
1.126xOnLoad_etc . . . . .	104
1.127xOnReturn . . . . .	105
1.128xOn_Various . . . . .	105
1.129xPalette . . . . .	106
1.130xPipe . . . . .	106
1.131xRadio . . . . .	108
1.132xRoutine . . . . .	109
1.133xSlider . . . . .	109
1.134xTextIn . . . . .	110
1.135Index of ram:Gui4Cli.guide . . . . .	110

---

# Chapter 1

## ram:Gui4Cli

### 1.1 MAIN

```
*** Gui4Cli ***  
An interpretive GUI programming language  
(c) 1995-1997 by D.C.Keletsekis
```

Gui4Cli is an interpretive programming language for the easy creation of GUIs. It provides Gadgets, Menus, Appittems etc, through an easy script-like language, allowing you to construct and visually edit in real time resizable font-sensitive GUIs and through them, control other programs by issuing CLI and ARexx commands.

Introduction.

Important Topics!

Program Operation

Visual Editing

Programming GUIs

Advantages/Features

Author/Bugs/Licence

ToolTypes & Options

The GUI loader

The COMMANDS themselves

```
**** Gui4Cli WWW Page is now at http://users.hol.gr/~dck/gcmain. ↵  
htm ****
```

To run the demo, assign GUIs: to the directory where you have placed the Gui4Cli files and click on the Demo.gc icon, or open a shell and type :

```
> Run GUIs:Gui4Cli GUIs:Demo.gc
```

dck@prometheus.hol.gr

## 1.2 Action

```
ACTION <action> Source <destination>
```

This is a command which will recursively scan the directory tree and perform the given <action> to the Source.

LVAction will do the same things, but to all the selected files/dirs of a given dir listview.

The ESCAPE key - hit while a Gui4Cli window is active - will abort any operation taking place.

The "Source" can be a File or a Directory - *\*not\** a device.

The Actions that can be performed, are :

COPY Will copy the file or dir to the destination.  
ex : ACTION COPY dh0:MyDirectory ram:

COPYNEW Same as copy but will only copy files created at a later date than the destination files (if they exist).

MOVE Will move the Source to the destination (or just rename them if they are on the same device).

DELETE Will delete the source file or dir.  
You *\*must\** give a destination. This can be anything or "NOREQ".  
If you state "NOREQ" you will suppress a "Delete Files ?" kind of requester that will otherwise appear.

```
ex : Action delete Ram:t/MyDir NOREQ
- Will delete dir Ram:t/MyDir, without asking you first.
```

SIZE Here, <destination> must be the name of a variable where the total size of the File or Directory (or all dirs/files for lvaction) will be stored.

```
ex : Action SIZE ram:t/MyDir MyVariable
```

Starting with V3.2, the sizes will be shown in the listview for each directory.

PROTECT Protect requires that the <destination> argument be a MASK denoting the protection bits you want to set the file to. This MASK, is a 7-digit number, where each number can be a 1 or a 0, showing that you want the given bit set.

Each digit represents :



1st Delete -> 1 means file can NOT be deleted, 0 it can.  
 2nd Execute -> Used by shell commands - set to 0 (I think)  
 3rd Write -> 1 = File is writable  
 4th Read -> 1 = File is readable  
 5th Archive -> 1 = File has not changed  
 6th Pure -> 1 = File is program that can be made resident  
 7th Script -> 1 = File is a script

ex : Action Protect DH0:MyDir 1000000  
 - Protect all files in dh0:MyDir from deletion.

CLI This is new with V3.0 and it allows you to execute a command recursively through a directory and all it's subdirectories, for all files found therein. There is a special internal variable called \$\$FILE which will be valid whenever Gui4Cli is recursing through the files of a directory. Using it you can do this (for example) :

```
ACTION CLI MyDirectory 'search $$FILE string'
```

and all the files within the directory and it's subdirs, will be searched. This command is very important, because it can take simple CLI commands and make them recursive - saves on a lot of code repetition..

## 1.3 Advantages

### THE ADVANTAGES OF Gui4Cli

-----

Gui4Cli is not better, faster, sexier, or more complete than other GUI builders.

BUT it is a new way of programming, it's easy and it is INTERPRETIVE. (i.e. made from plain ascii files and not compiled)

This gives it many advantages, such as :

1.- The GUI is separate from the program, which means that :

(a) You can have many GUIs loaded, without the programs they are meant to control being also loaded. This means less memory is used.

(b) A programmer can write a program without bothering about the Wait() loops, intuition messages, resizable, font sensitive windows, workbench start-up code and all the other stuff he is supposed to do to make a GUI for his program. Just grab the command line arguments and run ! (like the good old days :)

(c) The program thus written will not only be smaller, but will also be more flexible since the user can use it as he sees fit.

---

2.- The user can change the GUI to fit his preferences.

With the new Visual Editing facility, the gui becomes completely plastic and changeable by the user at will, while it's running! This is very important because many programs fail on the functionality of their GUIs, not being as intuitive as the user would want them. A GUI is, just a way to communicate with the user, so let the user decide how he wants to be told what to do..

3.- The user can incorporate the GUI, with other GUIs.

There are many wonderful small utilities for the Amiga, which can be combined into common GUIs, as functions would in a program.

4.- With the (simple but sufficient) ARexx capability that Gui4Cli has, the user can make many different GUIs to take advantage of a program's ARexx interface (look at the CygnusEd and GoldEd demos).

5.- The user gets standard keyboard shortcuts and a standard working environment with every GUI. Much easier to remember.

Apart from the above, Gui4Cli is also simple and stand alone.

This means that anyone can use it, without programming knowledge compilers, libraries and/or patience.

Some of the other Features it has, are :

- Can open unlimited resizable, windows of various types.
- Can handle all Gadtools Gadgets & Menus, easily.
- Visual Editing of the gadgets makes it *\*very\** easy to construct guis.
- Keyboard shortcuts for gadgets & menus
- Can use icons as Button Gadgets, Window background or in Menus
- ARexx interface (for sending and receiving ARexx commands)
- Can use different fonts for gadgets, menus and graphics.
- Has Graphics commands for lines, boxes, circles etc.
- Standard keyboard shortcuts which apply to all windows.
- Much more..
- Give me a break.. isn't the above enough ?

Apart from the above, Gui4Cli programs, being 7 bit ASCII files, can be sent through the Internet. So what ? I hear you say.

Well.. ok, I just thought I'd mention it..

---

## 1.4 All\_the\_Commands

L T W H  
Values you can use.

Parser Commands :

NewFile  
NewFileName

----- GLOBAL COMMANDS -----

WinBig  
L T W H Title

WinSmall  
L T W H

WinType  
MASK (Close|Drag|Zoom|Depth|Borderless|Backdrop|RIGHT|BOTTOM)

WinOut  
ConsoleSpecification

WinOnWin  
GuiName LeftOffset TopOffset

WinOnMouse  
LeftOffset TopOffset

Screen  
PublicScreenName

WinFont  
FontName Size UL|BD|IT(Mask)

WinBackground  
SOLID|PATTERN|ICON|IMAGE APen|Name BPen

UseTopaz

NoFontSense

VarPath  
VariableSearchPath

----- GRAPHICS COMMANDS -----

Graphics  
The following commands :

---

```

BOX      L T W H  IN|OUT
CTEXT   L T Text FontName size FGpen BGpen
LINE    L T L T ColorNo
SQUARE  L T W H ColorNo FILL|NOFILL
CIRCLE  centerL centerT xradius yradius ColorNo FILL|NOFILL
ICON    L T IconName (no info)

```

#### Images

The following commands :

```

LOADIMAGE ImageFile Alias ScreenName|NoRemap
FREEIMAGE Alias
IMAGE     Left Top Alias
CHANGEIMAGE GuiFile GadID Left Top Alias

```

---

#### EVENTS

---

```

xButton
  L T W H Title

xCheckBox
  L T W H Title Variable OnText OffText ON|OFF

xVSlider
  L T W H Title Variable Min Max Current ShowStr

xHSlider
  L T W H Title Variable Min Max Current ShowStr

xTextIn
  L T W H Title Variable StartingText Bufflength

xCycler
  L T W H Title Variable

xRadio
  L T W H Variable Spacing

xArea
  L T W H COMP|BOX|NONE

xPalette
  L T W H

Gauge
  L T W H IN|OUT BUTTON|RIDGE|ICONDROP APEN BPEN PERCENT

xListView
  L T W H Title Variable File|Dir Offset NUM|TXT|MULTI|DIR
The
Directory ListView

LV Hooks
  LVDirHook HookID, LVHook HookID

```

---

```

xMenu
  Menu Item SubItem Shortcut

xIcon
  L T IconName (no .info)

xAppMenu
  AppMenuName Variable ON/OFF

xAppIcon
  L T IconName Title Variable ON/OFF

xAppWindow
  Variable

xOnKey
  Letter|#KeyValue

xRoutine
  RoutineName

xPipe
  PipeFileName ON/OFF

xOnReturn
  LaunchID

xOnJump
  Variable

Text
  L T W H Text Length BOX|NOBOX

System Events
  xOnLoad, xOnOpen, xOnClose, xOnQuit

Other Events
  xOnActive, xOnFail, xOnDiskIn etc..

```

----- GADGET MODIFIERS -----

Gadget Modifiers :

```

GadID
  IDNumber

GadHelp
  HelpText

GadFont
  FontName  FontSize  MASK(Underline|Bold|Italics)

```

```
GadTitle
ABOVE|BELOW|LEFT|RIGHT

GadKey
Letter (or #ASCII value)

GadTxt
LEFT|CENTER|RIGHT

LVDirHook
HookID
```

----- EVENT COMMANDS -----

Controlling Gadgets :

```
SetGad
GuiFile GadIDs ON|OFF|SHOW|HIDE (Arexx capable)

Update
GuiFile GadID Value (Arexx capable)

ChangeArg
GuiFile GadID ArgNumber NewValue

ChangeGad
GuiFile GadID L T W H Title

ReDraw
GuiFile

PartReDraw
GuiFile L T W H

ChangeIcon
GuiFile GadID L T NewIconName

SetGadValues
GuiFile
```

Control Statements :

```
If/ElseIf/Else/Endif/And..
Argument
Operator
Argument

IfExists/Else/EndIf..

SYSTEM
Name
```

---

```
While/EndWhile/And/Or
Argument
Operator
Argument

Mark/Goto
MarkName

Gosub/Return
GuiName RoutineName (ARexx capable)

DoCase/Case/Break/EndCase
(DoCase) Argument - Case
Operator
Argument

Stop
    --- All Commands below this line are ARexx capable ---

Quit
DOS Commands :

Run, CLI
CommandLine

SendRexx
PortName CommandLine

Wait

SYSTEM
Name TimeOut

MakeDir
DirName

Assign
Device: Path|REMOVE

Rename
OldFile NewFile

Launch
LaunchID CommandLine

FailAt
ErrorNumber

Recursive commands :

Copy
FileName (with wild characters) Destination
```

---

```

Delete
FileName (with wild characters)

Action
COPY|COPYNEW|MOVE|DELETE|SIZE|PROTECT|CLI File/Dir Destination

LVAction
COPY|COPYNEW|MOVE|DELETE|SIZE|PROTECT|CLI GuiFile GadID ↔
Destination

```

Note : DOS and Recursive Commands always set the \$\$RETCODE

Handling GUIs :

```

Load/Open...
GuiLoad GuiFullPathName - GuiOpen/GuiClose/GuiQuit GuiName

GuiRename
OldGuiName NewGuiName

Status

Info
GUI|GADGET|PALETTE|IMAGE Guiname|Guiname/GadID|ImageAlias

```

Handling Variables :

```

SetVar
Variable String

DelVar
Variable

AppVar
Variable Text

CutVar
SourceVar CUT|COPY CHAR|WORD|LINE Amount DestinationVar

Counter
Variable INC|DEC Amount

Append
File String

Extract
String Root|Path|File|Guipath|Clean|Unquote|Ext|Upper|Lower Var

JoinFile
Path File Variable

ParseVar
Variable

```

---



CalcVar  
ResultVar Argument operator Argument

ListView Commands :

LVUse  
GuiFile GadID

LVDel  
LineNumber

LVPut  
NewText

LVChange  
NewFromFile

LVSort  
ASC|DSC

LVFind  
String

LVAdd  
String

LVInsert  
(Before) LineNumber String

LVClear

LVSave  
FileName

LVMove  
+-Offset|#LineNumber

LVGo  
first|next|prev|last|#LineNumber

LVSearch  
string CI|CS First|Next

LVMode  
NUM|TXT|MULTI|DIR

LVMulti  
First|Next

LVDir  
Parent|Root|Disks|All|None|Refresh|NoRefresh|#DirName

Various Commands :

---

```
SetScreen
GuiFile ScreenName

GuiScreen
GuiFile FRONT|BACK

GuiWindow
GuiFile ON|BIG|SMALL|FRONT|BACK|WAIT|RESUME

SetWinTitle
GuiFile NewTitle

ReqFile
L T W H Title SAVE|LOAD|MULTI|DIR Variable DirName

CD
NewDirectoryName

Delay
Ticks

EZReq
Text Choices Variable

Say
Text

Set
[parameter] [value]

SetStack
StackSize

MakeScreen
ScreenName Depth Title

KillScreen
ScreenName

SetColor
GuiFile ColorNumber R G B

TTGet
FullPath/IconName (without ".info")

BreakTask
TaskName CDEF(signals)
```

## 1.5 Append

```
APPEND File String
```

This command attaches the specified String to the end of the File.

---

File - A File name (NOT an ENV: variable) - i.e. if you want to append something to an ENV: variable you have to say Env:MyVar, instead of .MyVar.

String - Any character, sentence, or whatever.

Example :

```
SetVar .MyVar "This is a " <-- Note the space
Append Env:.MyVar "Test"
```

Now .MyVar should contain "This is a Test"

You can use this command to append a String to any file.

## 1.6 AppVar

```
AppVar Variable Text
```

This command will append the Text you provide to the Variable.

If the variable does not exist it will be created.

Note that you can not have a variable larger than the size of Gui4Cli's buffer (which, by default, is about 1000 characters).

You can increase the buffer size if you want with the "BUFFERS" Tool Types option.

OR - use an env: variable and the APPEND command.

## 1.7 Assign

```
ASSIGN Device Path|REMOVE
```

This is a simple command that will assign a path to a device.

```
ex: Assign GUIs: dh0:Mydir/Gui4Cli
```

If, instead of a Path you give the keyword "REMOVE", the assign will be removed.

Multiple assigns are not implemented yet.

## 1.8 BreakTask

---

```
BreakTask TaskName CDEF(signals)
```

This command is usefull for stopping any programs you may have run/cli/launched. It does the same as the c:BREAK and c:STATUS commands combined.

Most programs will respond to one or more CONTROL signals. There are 4 possible control signals C, D, E and F. The most common is CONTROL-C - which will stop a program executing.

example :

```
if you have started a program with :
-> run 'c:search dh0:MyBankAccount OneBillionDollars'
```

```
you can stop it (if the taxman walks by), by issuing :
-> BreakTask c:search C
```

```
You can also send multiple signals like :
-> BreakTask c:search CF
which would send both Control-C and Control-F signals
```

\*\*\*\* IMPORTANT \*\*\*\*

You *\*MUST\** give the *\*exact\** name that the program was launched under, otherwise it will not be found.  
i.e. :  
"c:search" is *\*not\** the same as "dh0:c/search" or "search"

## 1.9 CalcVar

```
CALCVAR ResultVar Argument operator Argument
```

This command is a feeble attempt at giving Gui4Cli some mathematical abilities. Gui4Cli is not meant to be a programming language in the usual sense, so maths is usually not needed, still...

```
CalcVar Result 2 + 3
- will place the number 5 into variable Result.
```

```
You can also (like in the SetVar command) use :
Result == 2 + 3
- which is easier to the eye.
```

All calculations are *\*Integer\** only!

Operators are : + - \* / also % (modulus) and ^ (to the power of)

Note that if you need complex calculations, you can easily use AREXX to do them - look at the `guis:dir/calculator.gc` gui.

In fact, some users have seen this command as a really good

excuse to learn ARexx so that they can avoid using it.  
Whatever..

## 1.10 CD

```
CD NewDirectoryName
```

This command will change the current directory of the program.

This is useful when launching programs :

When you run a program, and unless specified otherwise by this command, the default directory of the program you launch will be the directory Gui4Cli resides in (i.e. c:).

With this command you can change the default directory before you launch the program.

## 1.11 ChangeArg

```
CHANGEARG GuiFile GadID ArgumentNumber NewValue
```

This command enables you to dynamically change a GUI, by changing the values of a gadget, while the program is running. If you just want to change the size position or title, use CHANGEARG instead.

The gadget you want to change is specified with :

GuiFile - which is the GUI program file the gadget resides in.  
GadID - which is the ID you have given to the gadget.

All gadgets have a number of arguments, which are counted from left to right, starting from 0.

A xBUTTON, for example, has the following arguments :

xBUTTON	Left	Top	Width	Height	Title
Argument Number	0	1	2	3	4

So, if you wanted to change the width of a button :

```
ChangeArg MyGui.gc 5 2 100
```

This would set the width of a button which has an ID of 5 and resides in file "MyGui.gc", to 100 pixels wide.

This command has no immediate effect, unless and until the

```
ReDraw
command is used. Then the window will be redrawn with this (and ←
any
```

other gadgets you may have changed) redrawn to their new dimensions and/or values.

\*\*\*\*\* IMPORTANT \*\*\*\*\*

- Be very careful with changing string arguments, such as the Title.

Gui4Cli (for reasons best left untold), will only store a string which is less than or equal to the length of the existing string.

So if you want to replace the title "My Button" with "My Other Button" declare the title as "My Button", so the new title fits.

GOLDEN RULE :

When replacing strings, always keep the string lengths same.

- Fonts & Font sizes can not be changed.

- Menus can NOT be changed.

- Icons must be changed with the  
ChangeIcon  
command.

- If the GuiFile or the GadID are not found, NOTHING will happen and the user will NOT be told (unless in DEBUG mode).

## 1.12 ChangeGad

CHANGEGAD GuiFile GadID Left Top Width Height Title

Similar to the CHANGEARG command, but this is easier and, if you want to change more than one of the gadgets arguments, faster.

Read the CHANGEARG command for more details.

All the restrictions specified there, apply here as well.

This command has no immediate effect, unless and until the

ReDraw  
command is used. Then the window will be redrawn with this (and ←  
any

other gadgets you may have changed) redrawn to their new dimensions and/or values.

New with V2.1 is that you can change the size of the WINDOW also, with this command, by giving 0 as the GadID. If you give -1 for either of the Left or Top arguments, these will remain unchanged.

Negative values for width & height have the same meaning as the WinBig command. Currently the Title argument is disregarded.

Example :

```
ChangeGad MyFile 2 10 20 100 20 "New Title"
```

- will set the size of the gadget bearing a gad-ID of 2, to 100 pixels wide by 20 pixels high, position it at 10,20 and give it the "New Title" - This will all happen after you have given the REDRAW command.

```
ChangeGad MyFile 0 -1 -1 100 200 ""
```

- Will resize MyFile's window to 100x200 (+ any resizing you may have done) and not change it's position.

## 1.13 Changelcon

```
CHANGEICON GuiFile GadID Left Top NewIcon (no .info extension)
```

This command will change the Icon of Gadget "GadID" in file "GuiFile" to the "NewIcon" and place it at Left, Top.

The effect will be immediate.

You can ONLY use this command on xICON or IMAGE gadgets. It will have no effect on others.

Icons in menus can NOT be changed.

NOTHING, will definitely happen, if the Icon is not found.

NOTE :

You can pass a negative value (-1) for either the Left or Top. In this case, these will remain unchanged.

## 1.14 Command\_Lines

```
CONSTRUCTING COMMAND LINES
```

```
=====
```

\*\*\* NOTE \*\*\* All special characters below, are preceded by a BackSlash character. Some versions of AmigaGuide will not show you the backslash unless it's preceded by another backslash, i.e. \  
 \

So when you see \  
 \n as just an "n", remember it should be \  
 \n

Special Characters :

```
-----
```

You can use the following special characters inside command lines :

```
\n = (BackSlash + n) NewLine
```

```

\r  = Carriage return
\t  = Tab

\#  = starts a character expressed as a decimal value
      ex : Say '\#7'
      will cause the screen to flash (7 is the decimal value
      of the "beep" or "bell" character). Another example :
      ex : Say "\#155\1;33;40;>0mText is now color 3"
      will print "Text is now color 3" in your console in
      color 3. All text will now be in color 3, until reset.

      The Amiga console device has *many* such commands, with
      which you can output to and control a shell as you need.

```

Also, any character preceded by a \ will be written literally.  
So, you can write :

```

\    = the BackSlash character
\"   = the " character
\'   = the ' character
\$   = the $ character
\A   = character A
..... etc

```

This is especially usefull when messing about with variable names,  
because the slash "\" character will also end a variable name.

So if you wanted for example to append ".info" to a filename which  
you have stored in a variable, you can do this :

```

filename = MyFile
say "$filename\."info"
- will print "MyFile.info"

```

It can also be done with the appvar command but this is faster..

Weird Stuff :

For some reason the console device will replace \$\$ with  
nothing. So if you want to, say, send a rexx command to Gui4Cli  
with GRX, containing an internal variable, you must do this:

```
grx Gui4Cli say \$\$G4C.dir
```

This will print out Gui4Cli's current dir.

## 1.15 Copy

```
COPY File/Dir(with wildcards) DestinationFile/Dir
```

This is a recursive command which will copy files or directories  
to the destination.



```
ex: Copy dh0:myfile#? ram:
or: Copy dh0:myfile ram:MyNewFile
```

If a directory is given as destination then the file(s) will be copied into that dir. Otherwise the destination file will be overwritten or created (if it did not exist).

## 1.16 Counter

```
COUNTER Variable INC|DEC Amount
```

This is a command which (INC)reases or (DEC)reases Variable by Amount.

Example :

```
SetVar MyVar 20
COUNTER MyVar INC 10
```

The variable MyVar will now contain "30"

MyVar must be a number for this to work.  
This command is useful in DoWhile loops.

## 1.17 CutVar

```
CutVar SourceVar CUT|COPY CHAR|WORD|LINE Amount DestinationVar
```

This command allows you a little control over what your variables contain.

It allows you to CUT or COPY a specific AMOUNT of CHARacters, WORDs or LINEs from SourceVar and put it into DestinationVar.

SourceVar - The source variable.

CUT|COPY - CUT will cut the CHARacters, WORDs or LINEs from SourceVar whereas COPY will just copy them into DestinationVar, without changing the SourceVar contents.

CHAR|WORD|LINE - Keywords denoting whether you want an Amount of CHARacters, WORDs or LINEs cut or copied.

Amount - How many CHARacters, WORDs or LINEs to cut or copy.  
A Positive amount means CUT or COPY from the beginning of the variable, whereas a negative amount means from the end.

DestinationVar - The destination variable to put the stuff cut or copied.

example :

---

```
SetVar MyVar "This is a variable"
CutVar MyVar CUT CHAR -2 MyOtherVar
```

After these 2 lines have executed, MyVar should contain "This is a variab" and MyOtherVar should contain "le"

## 1.18 c\_Gui

```
***** c:Gui                               *** NOTE : NEW VERSION WITH G4C ←→
      3.0 ***
```

Gui is a "LOADER" for Gui4Cli.

\*ALWAYS\* use it to load your GUIs or to start up Gui4Cli, as it will make sure that Gui4Cli knows your default paths, even if started from the workbench. (The code to achieve this was written by Ralph Babel and can be found in Guis:Docs/WBPath).

You pass it the same options as you would pass to Gui4Cli, i.e. PORT, DEBUG, OUTPUT, BUFFERS etc - (see  
     ToolType&Options  
 )

It will then look if Gui4Cli is already running, and if it is, it will pass it the gui's filename to load, with the options specified.

Otherwise, it will start-up Gui4Cli with the filename & options given.

Gui4Cli will also do the same (i.e. look for itself) but this program is much smaller and faster.

GUI will look for Gui4Cli in the default path of your system and if not found there, it will look for it under GUIs:Gui4Cli.

This program is now pure and can be made resident (for faster loading) by putting this command in your user-startup file :

```
> Resident c:gui pure add
```

It is Workbench capable, so you can put it as the default tool in your Gui4Cli program icons.

If a gui script is started by clicking on it's icon, then the icon of the gui script will be read by GUI and any tool types found therein will be passed to Gui4Cli.

Most of the tooltypes will not do anything if Gui4Cli is already running, but some like the PORT=NewPortName tooltype \*will\*, i.e. Gui4Cli will be started again under a new port name.

## 1.19 Delay

```
DELAY ticks
```

This command will delay all activity performed by Gui4Cli for the number of ticks you gave. One tick = 1/50th of a second. Therefore...

```
eg : DELAY 100
```

will give a delay period of 2 seconds.

## 1.20 Delete

```
DELETE FileName/Dir
```

This deletes the file "FileName". Now delete is recursive and will also delete directories.

This argument can have wildcards in it :

```
Delete #?.bak
```

for example, will delete all files(or directories) ending with ".bak", in the current directory.

The Amiga is notorious at getting invalidated hard disks, if something goes wrong during writing or deleting a file. So be careful when deleting stuff.

The FileName is translated, before being parsed for wild characters.

## 1.21 DelVar

```
DelVar Variable
```

Delete a variable or variables. Can use wildcards for deleting Gui4Cli variables

```
ex : DelVar MyVar           - Delete MyVar
or: DelVar MyVar#?         - Delete all variables starting with
                           MyVar.
```

You can use this command for deleting both NORMAL or ENVIRONMENT variables (env: variables have a . (full stop) as the first character) If you want to use wildcards in deleting env: variables, use the DELETE command.

Nothing drastic happens if the variable does not exist.

Also, nothing happens if at end of the program you do not delete all the

---

ENV: variables that you have declared. They just remain in the ENV: directory of your ram: wasting space.

The NORMAL variables will be automatically be deleted when Gui4Cli quits.

## 1.22 DirList

### Directory ListViews

```
ex : xListView 10 10 200 100 ListView Var dh0:MyDirectory 10 DIR
```

A Directory ListView is a listing of the files devices etc on your computer. It's difference from an ASL requester is that it's easily Multi-Selected and provides much more control.

A Directory Listview will not "Happen" unless double clicked upon.

This is how it works :

Clicking on a file or directory (single click) :

-----  
A DIR type Listview will not "happen" when the user just clicks an item once - only when he double clicks on it.

If you need to be notified everytime the user single-clicks an item you have to declare a  
     xLVHook  
     event.

Double clicking on a directory :

-----  
In this case, the directory will be entered into and displayed automatically. If an  
     xLVDirHook  
     event has been  
 declared for this gadget, it will be executed.

This enables you to refresh the current LV directory name in some other part of your GUI.

Double clicking on a file :

-----  
In this case, the full path of the file will be stored into the Variable (quoted if needed), and the event commands attached to this particular ListView, executed.

Reading the MultiSelected files :

-----  
The files which you may have multiselected can be read with the  
     LVMulti  
     command, at any time.

General :

-----

In controlling DirLVs, you can also use :

The

```
LVDir
command.
```

The LV Internal Variables (see Internal Variables).

## 1.23 DoCase

```

                DOCASE  Argument
CASE  Operator Argument
      (commands....)
      BREAK
CASE  Operator Argument      <- you may have many case statements
CASE  Operator Argument
      (commands....)
      BREAK
ENDCASE
```

This command structure is like a multiple IF with OR ability (confused ?)  
It lets you choose easily between more than 1 choices.

example :

```
SetVar TestVar "TEST"
DoCase $TestVar
  Case = "EHH?"
  Case = "REAL"
    Say Real
    BREAK
  Case = "TEST"
    Say Test
    BREAK
EndCase
```

The above should print the word "Test"

You can have as many case statements as you want.

The

```

                OPERATOR
                argument has the same meaning as for IF and WHILE.
```

You can have DoCase within DoCase etc...

When docase finds a case which is true, it will start executing until  
it encounters an ENDCASE or a BREAK. - like in a C program

This means that you can have multiple case arguments, providing a  
rudimentary OR facility.

## 1.24 Extract

```
EXTRACT Variable ROOT|PATH|FILE|CLEAN|UNQUOTE|GUIPATH|EXT|UPPER| ↔
      LOWER ToVariable
```

Will extract from the given Variable the given item and place it into ToVariable.

Say you have a variable containing "Work:MyDir/This is my file "

These are the things that can be extracted and placed into Variable :

```
ROOT    => Work:          - It will not give the actual device (yet)
PATH    => Work:MyDir
FILE    => "This is my file  "
CLEAN   => Work:MyDir/This is my file    - delete the spaces also
UNQUOTE => Work:MyDir/This is my file    - will not delete the spaces
UPPER   => WORK:MYDIR/THIS IS MY FILE
LOWER   => work:mydir/this is my file
```

```
EXT     => This will extract the file's extension (including the dot)
        ex : Extract MyFile.gc EXT var
        -> var will now contain ".gc"
```

```
GUIPATH must be given a GUI name name as string, and it will place
        the full path (without the filename) of the GUI given, into
        the Variable.
```

This is useful in loading helper GUIs which reside in the same directory as the main GUI. See also  
Paths

## 1.25 EZReq

```
EZReq  Text  Choices  Variable
```

This command will put up a system requester (in the same screen as the GUI that called it) and wait for the user to click on a choice.

```
Text    - This is the text that will appear in the requester
```

```
Choices - These are the choices which will appear as buttons in the requester.
        the format for specifying them is to separate the choices with
        | characters i.e. : "Quit|Cancel" will supply you with 2 choices
        "Quit" and "Cancel"
```

```
Variable - The Variable variable will contain the number of the requester
```

button chosen.

The Right most button (which will usually be CANCEL) will put a 0 into Variable.

The other choices will put the number of the button, reading from left to right.

i.e. a Choices of "Abort|Quit|Exit|Cancel" will make a requester with 4 buttons, which will place the following values into Variable, if they are clicked :

Abort = 1, Quit = 2, Exit = 3, Cancel = 0.

## 1.26 FailAt

```
FAILAT  ErrorNumber
```

When you use CLI (without run) and other commands, you get a "return code" (\$\$RETCODE) when whatever you did with this command finishes executing.

This return code is stored in the internal variable \$\$RETCODE.

If this return code is equal or more than the ErrorNumber you have set with the FAILAT command, then Gui4Cli will stop execution immediately, and return to the waiting for something to happen.

The default value for ErrorNumber is 10, which most programs return if a pretty serious (but not fatal) error occurs. You can set this to 20 for example, if you do not want anything but a very serious error interrupting your CLI commands. (or 1000 for kamikaze pilots)

You can also declare an event type xONFAIL (see "Other Events"), whose commands will be executed if such an error occurs.

This command eliminates the need for constantly checking \$\$RETCODE to see if an error occurred, before proceeding to other commands.

\$\$RETCODE is reset to 0, after such an error.

## 1.27 Fonts

FONT COMMANDS :

In the Font Commands below, there are two very important keywords you can use, instead of actual font names :

```
#MONO      - specifies the user's preferred MONOSPACE font
#SCREEN    - specifies the user's preferred SCREEN font which will
             usually be PROPORTIONAL.
```

Use these instead of actual font names where ever possible, since they will change as per the user's preferences.

There are various FONT commands you can use :

Globals :

-----

WINFONT - will set a default font for all the gadgets which have no GADFONT declared. - it is the default window font.

USETOPAZ - This command will force Gui4Cli to disregard all fonts declared and use topaz 8 instead.

Event specific :

-----

GADFONT - This, attached to a Gadget will define the font for that gadget.

If none of the above commands are declared, then the SCREEN font will be used.

Gui4Cli comes also with a Graphic font, courtesy of G.Maddox. It has arrows, disk images, smileys etc.. and you can easily use it as text in buttons.

Also see

Font Sensitivity

## 1.28 Font\_Sensing

FONT SENSITIVITY :

Gui4Cli is Font Sensitive!

Well, ok.. not so much font-sensitive as font-aware..  
(font-respectful ?.. font-give-a-damn ?..)

You can turn this feature off with the NOFONTSENSE global command.

What it does, is the following :

- On opening the GUIs window, it will check to see if the text, using the default window font, will fit into the respective gadget, and if not, it will enlarge the window and resize the gadgets, so that the text does fit.

The following restrictions apply (for now) :

- Only xBUTTON, xCYCLER, TEXT and xTEXTIN gadgets will be checked and even then, only when no GADFONT command has been declared for them.

Other gadgets, such as sliders etc do not have text in them, and others, such as listviews, are self-adjusting, so no action is taken on their behalf.



This means that if you do not have any of these gadgets in your GUI, the GUI will not adjust itself.

- Resizing is done only to increase the GUI's size.
- Currently, width is not checked - (tried it and didn't like it..) The width is resized proportionally to the height.

#### NOTES ON DESIGNING GUIs :

The Left & Top sizes you declare will be offsets INSIDE the window. Do not account for the window border, it will be added automatically.

If you want to make a GUI which other people might use, it is best to make it so it looks good using TOPAZ 8 font, on a High-Res, Non-Interlaced screen. Then Gui4Cli can resize the window for a bigger font, if needed.

Use the #MONO and #SCREEN keywords instead of actual font names wherever possible, since they denote the users preferred monospace and screen (proportional) font.

Buttons, Text and Cycler type gadget heights will be resized to the font-height + 2 pixels, whereas xTEXTIN type gadgets will be resized to font-height + 5 pixels. - So make the xTEXTIN gads a little taller.

## 1.29 GadFont

```
GADFONT FontName FontSize MASK(Underline|Bold|Italics)
```

Gadget text is drawn in the default font, or the font declared by the WinFont command.

You can, however, have a different font for every gadget. This is the command that does it.

The MASK is a 3 number string (see the WinType command)

```
1st Number : Underline ?      - 1=Yes, 0=NO
2nd Number : Bold
3rd Number  : Italic
```

e.g. GadFont times.font 11 010

This defines Bold 11 point times font

For a font name, it is always best to use the #MONO or #SCREEN keywords. You can find more on them

Here

With Dir Listviews specially, you want to use the #MONO font, ←  
since

you want the filename etc lined up and a proportional font would look ugly..

## 1.30 GadHelp

```
GadHelp HelpText
```

Very nice gadget modifier added in V3.2, provides on line help.

```
xButton 10 10 60 12 MyButton  
  GadHelp "This is some help text"
```

Now, if the on-line help system is on, whenever the mouse pointer hovers over this button (could have been any gadget) the text declared will be shown in the TitleBar of the Screen the gui is open on.

To set HELP on/off you can either:

- Press the HELP key, or
- Use the command SET HELP ON|OFF

## 1.31 GadID

```
GADID IDNumber
```

This command assigns IDNumber to the gadget you attach it to.

Example :

```
GadID 10
```

The IDNumber is used by other commands to reference the gadget. You need not give a GadID to a gadget when you are not going to change it.

Other commands will usually reference the Gadget as follows :

```
Update GuiName GadID Value
```

In the above example, GuiName is the name of the file that the gadget resides in and the GadID is the ID you have given it.

In this way, you can SetGad, Update etc any gadget, in any loaded GUI.

---

## 1.32 GadKey

GADKEY Letter

This is to assign a short cut to a gadget.  
The effect of the short cut varies with different gadgets.

If you can not express the letter, you can assign it with its decimal ASCII value by putting a "#" character in front of it.

GadKey #13 for example is ENTER

IMPORTANT :

Note that you may also assign a short cut to a gadget by specifying it in it's Title. All you do is put an \_ (underscore) character before the letter in the Title that you want as a short cut.

The letter will be underscored when the gadget is drawn.

example :

```
xBUTTON 10 15 80 15 "Hit _Me" <- here the M is underscored and becomes  
the button's keyboard shortcut.
```

To avoid confusion, the underscore method takes precedence over the GadKey command, if both are used on the same gadget.

## 1.33 GadTitle

GADTITLE ABOVE|BELOW|LEFT|RIGHT

Usually the titles of the gadgets are placed to their left.

With this command you can change this :

GadTitle ABOVE

..for example will put the title above the gadget.

## 1.34 GadTxt

GadTXT LEFT|CENTER|RIGHT

This command will only work on a TEXT type gadget.

It will justify the text of the gadget on the LEFT, CENTER or the RIGHT side of the gadget box.

---

## 1.35 Gauge

```
GAUGE L T W H IN|OUT BUTTON|RIDGE|ICONDROP APEN BPEN PERCENT
```

It's one of those pretty little thingies which show you how far you have progressed in some action.

A Gauge is not actually an event, since it can not be selected with your mouse, so it will never "happen". If you attach commands to it, they will never be executed.

You can however, give it a GADID and UPDATE it at any time.

The arguments are the same as for a BOX except for

```
APEN      = the number of the foreground color,
BPEN      = background color
PERCENT  = the starting percentage.
```

## 1.36 GoSub

AREXX capable

```
GOSUB GuiName RoutineName
RETURN
```

A Routine is an Event declared with the  

```
xROUTINE
```

 command and is nothing more than a bunch of commands that can be called and executed from another event.

GOSUB, is the command that does this. You can call GOSUB for routines in other GUI files. This enables you to have GUI files which contain only routines, which you can call on from other GUIs.

GuiName : The name of the GUI in which the xROUTINE exists.  
 Note that if the GUI is not loaded, the routine will NOT be executed, and the User will NOT be told.

RoutineName : The name of the routine to execute

GOSUB can also take up to 6  

```
Arguments
```

 and pass them to the routine it calls, as internal variables \$\$ARG.0 to 5

RETURN, returns from the routine which GOSUB went to and continues execution with the command following the relevant GOSUB.

RETURN can also be used in xONLOAD etc type events and can \*also\* take up to 6 arguments, which it returns to the calling GoSub, Guiload etc. as internal variables \$\$RET.0 to 5

## 1.37 Graphics

### GRAPHICS COMMANDS

=====

These are the graphics commands that you can use to tart-up your GUI. No commands can be attached to them and they can appear anywhere in the program, outside the body of an event.

For clarity sake though, I usually put them at the beginning of the program, after the Global Commands.

Note that you can give a GadID to these commands, as if they were Events, and then can setgad/changegad them and redraw them to new positions.

You can not, however, attach any commands to them, as a graphic does not "happen" so they will never be executed..

**BOX**    L T W H    IN|OUT    BUTTON|RIDGE|ICONDROP

Draw a beveled box of size L T W H.  
 IN means that the box will be recessed, OUT means the opposite.  
 You can have 3 types of boxes : BUTTON, RIDGE or ICONDROP. These have different border types - try them out.

**CTEXT**   L T Text FontName size FGpen BGpen UL|BD|IT|EMBOSS|SIZE(Mask)

Draw colored text, starting at (L)eft, (T)op.  
 (New with V3.2 is that you can add a "Shading" or 3D effect)

Text        = The text to be drawn  
 FontName = the font to use (ex: topaz.font or #MONO or #SCREEN)  
 Size        = The size of the font  
 FGpen      = Foreground pen number  
 BGpen      = Background pen number (-1 means leave the background alone)  
 Mask        = This is a mask (see the WinType command for explanation)  
 Digit 1 = Undelined (1=yes, 0=no)  
 Digit 2 = Bold  
 Digit 3 = Italic  
 Digit 4 = The color number you want to use for "shading"  
 Digit 5 = The offset of the shade (defaults to 1)

Example : CTEXT 10 30 "Test text" times.font 11 2 -1 01032

**LINE**    X Y x y ColorNo

Draw a line from X,Y to x,y, of ColorNo color.

---

SQUARE X Y x y ColorNo FILL|NOFILL

Draw a square from X,Y to x,y, of color ColorNo.  
FILL means filled - NOFILL means that only an outline is drawn.

CIRCLE centerL centerT xradius yradius ColorNo FILL|NOFILL

Draw a circle at centreL, centerT, with the respective radii.  
FILL it or NOFILL it with ColorNo color.

ICON Left Top IconName

Display icon as image at Left, Top. The only difference from  
the xIcon event, is that this is not clickable.

## 1.38 GuiEdit

VISUAL EDITING

Gui4Cli GUIs can be edited in real-time, i.e. while they are  
running. This is done by using the CONTROL key together with  
the mouse :

Moving Gadgets :

Press CONTROL-MouseClick on a Gadget or Graphic to Select it.  
You will see that an outline of the gadget is drawn. You can  
now let go of the CONTROL button and move the gadget outline  
around in your window.

When you are satisfied with the new position, just click the  
mouse and the gadget/graphic will be redrawn to this new position.

You can also use CONTROL-H instead of CONTROL-MouseClick to  
select a gadget, and CONTROL-H again to place it where you  
want.

This comes in handy if :

- (a) you are running programs like CycleToMenu etc which may  
interfere with your mouse clicks, or
- (b) you are trying to paste some gadget over another gadget  
in which case GadTools may eat-up the mouse clicks.

There is also a GRID available, which makes lining up the gadgets much  
easier. By default the grid size is 1 which means "no grid".  
You can set the grid size to any size you want, with "SET GRID 5"  
(5 is a good size..) - or through the Prefs gui.

---

### Resizing Gadgets :

You can resize a gadget by clicking on it's bottom right corner.

Note that some gadgets (such as ICONs, Images, xICONs and CTEXT) can not be resized.

Also note that to resize a listview you have to click on the listview's bottom right corner - *\*not\** on the arrow buttons (unless you use control-h).

Since LVs adjust their size automatically to show as many lines as possible, they may be a little difficult to select correctly for resizing..

### Resizing the window :

You can enlarge or reduce the window size by resizing the window while holding down the CONTROL key. In this case, the window is resized, while the gadget sizes/positions remain the same.

### Cloning gadgets :

After selecting a gadget you can "clone" it (i.e. make a copy) by using the CONTROL-J shortcut. In this case a copy of the gadget will be created and drawn where the mouse is at.

Note that *\*only\** the gadget information is copied. The gadgets modifiers and it's commands are *\*not\** copied.

### Inter-GUI Cloning :

After selecting a gadget you can also place it in another Gui4Cli window. In this case a copy of the gadget will be created and drawn into the window where you clicked.

This enables you to make new guis by copying gadgets from other guis - in effect, a gui editor.

Here also, *\*only\** the gadget information is copied.

### Deleting gadgets :

You can delete a gadget or graphic by selecting it and pressing the DELETE key.

### Saving your GUIs :

Once you have made the GUI of your dreams, you can save it by pressing CONTROL-G. A simple requester will ask you if you want to save the gui.

---

IMPORTANT notes on saving :

- NEVER - load a gui, then change it's file by manually editing it and then edit & save the gui. To keep all your file notes etc in tact, Gui4Cli remembers the line numbers of the gadgets when the file is loaded, and then, when saving, it goes and changes only these lines, leaving everything else untouched.

So if you have meanwhile changed the gui manually the gadget line numbers will have changed and ... well you're looking for trouble, that's what!

Same will happen if you visually edited the file before and added 1 or more xCYCLER or xRADIO type gadgets. Since these need extra lines to describe the fields they'll have, it throws the whole numbering scheme off if you try to save the gui again without having reloaded it first.

Hitting CONTROL-R to reload the gui (if you have changed it manually or have added cycler/radio gads) \*before\* starting to visually edit it, will reload and thereby refresh the correct line numbers etc.

Gui4Cli will check and tell you if it doesn't find the edited gadget where it should be in the file.

- BE VERY CAREFUL when saving multi-gui files, if you are trying to edit more than one of the file's guis at the same time. Reloading the gui will \*not\* refresh the gadget line numbers of the other guis, since only the active gui is reloaded..

If you must edit many guis of a multi-gui file at the same time, then reload \*all\* the file's guis everytime you save any one of them.

- If you have deleted existing gadgets, then the lines of the original GUI describing the gadget and all it's attached commands will be commented (i.e. a ; will be added in front)

This will be done for all the lines following the gadget, until the next gadget is declared, or the end of file is reached.

- If you have created new gadgets, these will be added at the bottom of the GUI file. If it's a multi-gui file, then at the end of the given gui within the file.
- Any RESIZE\_BIG/RESIZE\_SMALL commands that the gui file may contain will be commented out and the gui will be saved at it's current size.

Quirks :

- Circles are selected by clicking on their lower right quarter.
  - Boxes (even filled ones) are selected by clicking on their border.
-



You have to click *\*exactly\** on a pixel on their border.

- TextIn gads (and maybe some others) will give you a little trouble if you try to paste them over themselves (i.e. move them a tiny bit. Try to pick them up from their edges..

Final word :

Visual editing is very helpful for creating and setting up a gui just the way you want it. However, a gui will do nothing unless you tell it what to do - this can only be achieved by actually programming them.

So Press CONTROL-E and get on with it..

## 1.39 GuiLoad

	GuiLoad	GuiFullPathName
GuiOpen	GuiName	
GuiClose	GuiName	
GuiQuit	GuiName	

These commands load,open,close and unload(quit) GUIs.

When loading a GUI, you have to give the GUIs full path, i.e. :

```
GuiLoad  GUIs:MyGUIs/ThisGui.gc
```

Thereafter, the GUIs are referred to by the file name only, i.e. :

```
GuiOpen  ThisGui.gc
```

When opening/closing etc the GUIs, the respective events :

```
xOnLoad, xOnOpen, xOnClose and xOnQuit
```

are executed, if defined.

Starting with V3.2, these commands can also take

Arguments

This ability makes them very powerfull, allowing for function-type guis which can be called at will. (see guis:tools/rtn)

\*\*\*\*\* IMPORTANT note about GuiQuit :

When an GuiQuit is encountered, the file which has been instructed to quit is marked as such and the execution of the commands continues uninterrupted.

When all the commands have executed, then Gui4Cli looks around to see if there are any files marked for deletion, and deletes (unloads) them.

So, if you want a gui to quit immediately, you have to issue a STOP command after the GuiQuit command :

```
Guiquit mygui.gc
Stop
; more commands here..
```

otherwise the commands following GuiQuit *\*will\** be executed, and the file will be unloaded afterwards..

## 1.40 GuiRename

```
GUIRENAME OldGuiName NewGuiName
```

This renames a GUI.

It is useful in many instances, for example if you want to load many copies of a certain GUI, or reloading a GUI in the middle of a command sequence, like so :

```
GuiRename MyGui MyGui2      ; Reload a GUI
GuiLoad   MyGui
GuiQuit   MyGui2
```

Be careful though..

A renamed GUI has no file, so you can't edit it, or save the `resize BIG|SMALL` commands into it.

## 1.41 GuiScreen

```
GUISCREEN GuiFile FRONT|BACK
```

This will bring the screen that the GuiFile is open on, to the FRONT or move it to the BACK.

No action will be taken if the GUI is not open.

## 1.42 GuiWindow

```
GUIWINDOW GuiFile ON|BIG|SMALL|FRONT|BACK|WAIT|RESUME
```

With this command you can do various things to a GUI.

ON - will activate the GUI

---

```

BIG      - will zoom the GUI to its WinBig size (if it was zoomed small)
SMALL    - will zoom the GUI to its small size

FRONT    - bring the GUI to the front
          (also done by double-clicking on the window)
BACK     - Move GUI to the back of other windows.

WAIT     - Will change the pointer to a wait pointer and disable the GUI
RESUME   - will break the WAIT above.

```

WAIT and RESUME are usefull for when you launch a command, or do something which will take some time. You can set the gui to wait so that you don't have to worry about the user clicking buttons while he's waiting.

## 1.43 If

```

          IF      Argument
          OPERATOR
          Argument
ELSEIF, ELSE, ENDIF
AND, OR

```

This is the standard If/Else/EndIf stuff, common in any computer language. The way to use it, is to think logically.

```

var = 1
IF      $var = 2
OR      $var = 3
    say 'var = 2 or 3\n'
ELSEIF $var = 5
AND     $var = 1
    say 'var = 1 and 5\n'
ELSEIF $var = 1
    say 'var = 1 - correct\n'
ELSE
    say 'don't know..\n'
ENDIF

```

The above should print out 'var = 1 - correct'

Rules of the game :

- In any if construct, you MUST start with IF and end with ENDIF.
- In between you may use only one ELSE
- You can have as many ELSEIF's as you want - only the commands attached to the 1st ELSEIF which satisfies the criteria will be executed.
- You can have many AND or OR statements - these MUST follow exactly after an IF or ELSEIF or ELSE.
- You can have if within if within if constructs, up to 126 levels.

If you don't understand how it works, run gui4cli in debug mode. This will show you what it's doing.

---

The Arguments can be strings, or numbers or variables.  
Note that if you want to compare a variable you MUST place a \$ character in front of it (to indicate that you mean the contents of the variable).

See the OPERATOR explanation to see what operators you can use.

IF (as well as WHILE etc) is smart enough to understand the comparison of Numbers. So if you compare :

```
IF 2 > 10
```

...the answer will be FALSE, since 2 is less than 10, although if they were compared strictly as strings 2 would be more than 10.

IF can also be replaced by the IFEXISTS command. This has different arguments (see its explanation), but otherwise replaces IF in every way.

IMPORTANT :

The most common mistake I make when writing GUIs is this :

```
If variable = 10      <- This compares 10 to the name of the variable
                        instead of :
```

```
If $variable = 10    <- Correct. Compares 10 to the contents of "variable"
```

## 1.44 IfExists

```
IFEXISTS
SYSTEM
NAME
ElseIfExists, Else, EndIf
AndIfExists, OrIfExists
```

This command works exactly the same as the "IF" command, but with different arguments. Look at the if command to understand it.

It lets you see if various things such as PORT, FILE, DEVICE etc.. exist in your system.

Example :

```
TestVar = "MyWindow"
IfExists WINDOW $TestVar
    Say "My Window is open."
Else
    Say "My Window is closed"
EndIf
```

## 1.45 Images

IMAGES (pictures, brushes, JPGs etc)

Images can be used, via the `datatypes.library` (OS V39+ only)

Images are handled differently from icons, so that memory is saved. You first load an image and give it a "Alias" (i.e. any name you want, to which you can thereafter refer). Then you can use it in many guises, as a background or as an Image, without having to load another copy of it. When you're done, you unload it..

The Image specific commands, are :

- o `LoadImage ImageFile Alias ScreenName|NoRemap`  
will load the `ImageFile`, call it the `Alias` you gave and if you give a screen name, the image will be remapped to that screen's colors. If "NoRemap" is given, the image will not be remapped.
- o `FreeImage Alias`  
will unload a previously loaded image
- o `IMAGE Left Top Alias`  
will place the (already loaded) picture at the position specified. L T of -1 means center in window.
- o `ChangeImage GuiFile GadID Left Top Alias`  
will change the specified picture, to the "Alias" named picture which must have already been loaded.  
L T of -1 means position unchanged.

Sometime in the future I will try to make the image loading a background process, so that `Gui4Cli` doesn't have to stop and wait till the datatype has loaded the image.. Till then, grin and bear it :)

## 1.46 Important\_Topics

Things you should know :

=====

Variables :  
Important - MUST Read!

Normal Variables

Env: Variables

Internal Variables

Internal Variables List

Passing Arguments

Important - MUST Read!

Translation

Important - MUST Read!

Making Commands Lines

Important - MUST Read!

Finding Paths

Make GUIs run from anywhere.

Resizable Windows

How to make them.

Window Shortcuts

Very usefull shortcuts.

Fonts

Fonts & Font commands

Font Sensitivity

Font Sensing

Other useful notes :

-----

When writing a GUI, apart from the DEBUG option, there are 2 very useful keys you should know.

Control E - will run "c:ed" loading the GUI for editing.  
(can be changed to any name with the SET command)

Control R - will Re-Load the active GUI.

These will make editing much easier, especially for seeing the positions of the gadgets and graphics.

writing Commands :

Gui4Cli is "line oriented", i.e. the commands can not span lines.  
A line can be 1000 characters long (you can increase this with BUFFERS)

A ";" character ends the line (unless it is inside a quoted string), and you may place comments or anything else after it.

A "parser" is computer-jargon for that part of the program that reads your program file and tries to make sense out of it.

The parser, in this case, is pretty simple-minded.  
It's designed to be as fast as possible - not as intuitive as possible.

Do not try to confuse it because you probably will.

When you write commands, separate each argument with a space or a comma.

---

```
IF $Variable= 10      <- is NOT correct.
IF $Variable=10      <- is NOT correct.
IF $Variable = 10    <- IS correct.
```

A string which starts with " ends with a " and if it starts with a ' it ends with another '

If you load files and put them in a variable, and if these files contain a space they will be quoted e.g. "Ram Disk:MyFile"

So, if your arguments are likely to contain such quoted strings, it is vital that you use single quotes ' to surround each argument.

## 1.47 Info

- INFO GUI|GADGET|PALETTE|IMAGE GuiName|GuiName/GadID|ImageAlias

When a window or a gadget is clicked on by the user, it becomes the CURRENT one. All internal variables will reflect it's values.

You may want though to know the values of some gadget or window at other times, when the user has not specifically selected them.

This is the command that will do it for you. After this command is issued, the specified item becomes CURRENT and all the internal variables will reflect \*it's\* values.

The arguments you can use are :

GUI GuiName

- will select another gui. The WIN & SCREEN int vars will show you that gui's values.

GADGET GuiName/GadID

- will select another gadget. Note - for Listviews use the LVUse command - not INFO

PALETTE GuiName/GadID

- select another palette gadget (for the palette specific internal variables).

IMAGE Alias

- select another image (for the image specific intvars).

## 1.48 Introduction

INTRODUCTION & BASICS

-----

A Graphical User Interface (GUI), is the Buttons, Cyclers, Sliders, Menus and all the other gadgets you play with when using a program.

Gui4Cli is a program which enables anyone to write such a GUI and from it, issue CLI or ARexx commands to other programs.

What you do, is write a script in the Gui4Cli programming language, (which is very easy) and just run it!

How it works :

-----

Gui4Cli is and Event-Driven language. This means that it reacts to various events, such as the clicking of a gadget by the user, or a command to close the window etc.

Events can be Gadgets, Menus, AppMenuItems, User Actions etc.

A GUI program is a series of such events, defined by the programmer, and attached to these events are commands that are executed each time the event "happens"

Gui4Cli relies heavily on the use of

#### Variables

Variables are temporary storage places where you can place any sentence or sentences (strings). You can think of them as little files.

These Variables are referenced as follows :

MyVar - is the name of a variable called MyVar (case sensitive)  
\$MyVar - is the contents of a variable called MyVar

Example :

```
> SetVar MyVar "DPaint" ; This stores "DPaint" into variable MyVar  
> Say $MyVar
```

Gui4Cli will actually execute :

```
> Say DPaint
```

We will call this process

Translation

(because I don't know what else to call it)

This translation is done on ALL the arguments of ALL the Event Commands, making Gui4Cli very flexible.

Using this Translation method, Gui4Cli is able to provide a GUI that will form any CLI or ARexx command and run it.

For example, a Cycler (that's the gadget which looks like a button with a bent arrow at one end and when you click on it it changes the text displayed on the button), is declared in the following way:



```
xCYCLER 100 12 100 15 "This is a cycler"  CyclerFile
CSTR "DPaint" "DH0:Dpaint/Dpaint"
CSTR "PlaySound" "DH0:Music/Playsound"
CSTR "LightWave" "DH0:MyDir/MyOtherDir/LightWave"
```

This declares a CYCLER type gadget (or EVENT), with it's top left corner at position 100,12 (pixels) of your window, 100 pixels long, and 15 pixels high. The title next to the CYCLER will be "This is a cycler", and the name of the Variable that it will put it's value in will be "CyclerFile".

This CYCLER will have 3 choices : Dpaint, PlaySound and LightWave. Every time you click on the CYCLER, the text shown above next to each of the choices will be placed in the Variable "CycleFile".

So, if you click on the CYCLER several times and at the end the choice displayed is "PlaySound", then the Variable CyclerFile will contain the words "DH0:Music/Playsound".

Now, say you declare a BUTTON gadget as follows :

```
xBUTTON 10 30 100 15 "RUN"
CLI 'Run >NIL: $CyclerFile'
```

This declares a BUTTON at position 10,30, 100 pixels wide and 15 pixels high, with the text <RUN> on it.

Attached to this Gadget, is the Gui4Cli command "CLI", which means run whatever is declared after it as if it were typed at a shell.

If you hit this button, the command that will actually be performed will be :

```
Run >NIL: DH0:Music/Playsound
```

Using other gadgets, you can declare other variables, such as the File name you want loaded (with file requesters), the program's Cli options (with checkboxes or cyclers), the volume/rate etc (with sliders), and have a command line like :

```
Run >NIL: DH0:Music/Playsound MyFile Volume=64 Option1 Option2 etc...
```

This, in effect, provides a GUI for Cli based programs.

This, is the simplest way in which Gui4Cli can be used. There are many other uses it can be put to, as you will see in the Demos.

In the above example for the BUTTON gadget, we "attached" the command <CLI 'run NIL: \$CyclerFile'> to the BUTTON gadget or EVENT.

There are many such commands which can be attached to all types of Events, forming small programs which are executed every time the user makes the Event happen (i.e. clicks on the gadget, chooses the menu, closes the window, deactivates the window... etc).

These are called "EVENT COMMANDS" and provide control of what you want to achieve (with commands such as if/endif, gosub/return,

while/endwhile etc), the ability to perform Cli commands, send ARexx messages, tell other Gui4Cli windows to open or close and more.

Apart from the EVENTS and their EVENT COMMANDS there are also GLOBAL COMMANDS, which control the size of your window, which screen it opens on etc. and GRAPHICS COMMANDS which allow you to draw lines, boxes, circles, color text etc, in order to give your GUI a nicer appearance.

The best way to learn how to make a GUI yourself, is to read the Demo files (which are plain ASCII files with Gui4Cli commands in them) and which are commented. Feel free to change them and try them out to see what happens.

If you make a mistake, Gui4Cli will tell you what line and what type the mistake was.

There is also a DEBUG option that can be used, which shows you what commands Gui4Cli performs every time you click on a Gadget or choose a Menu, making it easier to work out the logic of your program.

## 1.49 JoinFile

JOINFILE Path File Variable

Will join the path to the file and place it into Variable.  
It's there to make life easier when dealing with paths or filename which contain spaces in them.

Path and File are STRINGS not Variables.

ex:

```
setvar path "Ram Disk:"  
JoinFile $path MyFile Var  
Say '$Var\n'
```

will print out "Ram Disk:MyFile"

## 1.50 KillScreen

KILLSCREEN ScreenName

This command will delete the screen "ScreenName".  
ScreenName IS case-sensitive.

IMPORTANT :

---

- 1.- All windows must be closed before the screen can close. If they are not, you will be notified with an EasyRequester.
- 2.- Gui4Cli will try to close the Public Screen you tell it.  
It will NOT check if this screen has been opened by Gui4Cli.  
Becarefull, therefore, not to close a screen of another program !!

You can check if the screen closed, with the IFEXISTS command.

## 1.51 Launch

### LAUNCH LaunchID CommandLine

This is a V2.1 command that allows some multitasking to take place.  
Thanks to Michael Van Elst for the help.

Some programs (like c:search for example) take a long time to finish their work. You need however to know when they are finished, so that you can display their results, or go on to another command.

In this case you can not use RUN (as you wouldn't know when it's finished).

If you use the CLI event command, then Gui4Cli would start the command and then sit around waiting for it to finish, thereby blocking all other operations.

In these cases, you can use LAUNCH.  
LAUNCH will create a new process which will run the command, thereby freeing Gui4Cli to do something else..

When the command finishes executing, it will return (unless the ESCAPE key has meanwhile been pressed), to a

```
xONRETURN
event
```

with the same LaunchID and do the commands declared there.

example :

```
xButton 10 10 100 15 "Click Me!"
Launch 1 'c:search dh0:PlanetEarth IntelligentLife'
; now Gui4Cli is free to continue with other commands etc

xOnReturn 1
Say 'The Search has finished\nNo Intelligent Life found..'
; just kidding...
```

## 1.52 Licence

## BUGS :

-----

Many mosquitoes, some cockroaches, but hopefully no tarantulas.

Gui4Cli is (in my experience) a stable program and it is difficult to crash it (I earnestly hope). However, since I mostly program the binary instead of making GUIs for it, there are and will forever continue to be minor errors, unless and until someone tells me about them.

If something doesn't seem to work correctly, use the DEBUG option to see what the program is doing and if you see a fault, please tell me.

I hope to be able to improve it, but since I have to work for a living it will be a slow and painful process.

Don't let all the above put you off though. Gui4Cli is very usable as it is, and a lot can be done with a little effort.

## Licence to use Gui4Cli :

-----

The program Gui4Cli, the concept, the language and the logic it uses as well as any and all other aspects of Gui4Cli are the copyright (c) of Dimitris C. Keletsekis. All rights are reserved. Commercial usage is forbidden, without the prior written consent of the Author. Use of the program is at your own risk. No warranty is made or implied. Permission to use this program in Magazine cover disks, Aminet CDs or Fish Cds, is hereby granted. A small charge for copying the program, is allowed.

Personal use of this program is allowed under the concept of SHAREWARE. If you use it regularly you are expected to register by paying USD \$10 (more or less) in any major currency, at the address bellow.

There are no rules for the GUIs you make and may want to distribute. Do with them as you like. You may include Gui4Cli in your distribution, but make sure that you also include c:gui and you state clearly in your documentation where the latest version of Gui4Cli can be found.

Latest versions of this package can always be found :

- on AmiNet, under Dev/Gui/Gui4Cli.lha, or
- at my web page, <http://users.hol.gr/~dck/gcmain.htm>

Gui4Cli is NOT disabled, in any way.

## Contacting the Author

-----

Gui4Cli is my first attempt at writing a computer language. I hope it doesn't show too much. Writing the manual was absolutely the worst part and I hope it shows.

In general, Gui4Cli is just a matter of reading in arguments given by you, the user, and passing them to the Amiga OS, which has all these commands and many more, built into it's amazing ROM. The windows, gadgets,

menus and all the other stuff, the Amiga does itself, as it has been doing since before mud was invented, so get down on your knees and thank the Lord for the fact that such a computer, made for people who love and not just use computers, still exists.

You can contact me by mail or Email at the following address :

AUTHOR : D. Keletsekis  
 14 King George Str.  
 Athens 10674  
 Greece.

EEmail : dck@prometheus.hol.gr <- Preferred.

THANKS TO :

- Michael Van Elst, for writing most of the code for the window backfill routines, the Launch command and for generally helping me understand the Amiga better.
- Graham Maddox for lots of great GUIs, a beautiful new graphic font and bug reports. He has released CLImaster (a very nice Gui4Cli File Manager) on aminet - it was also featured on a cover disk.
- John Collett, for writing the tutorials, various guis, and for the bug reports and encouragement.
- Brian Jones for the dir.xpack gui plus the total overhaul of the FileTypes checking by adding a hell of a lot of file types.

## 1.53 ListView

```
xLISTVIEW L T W H Title Variable FromFile|Dir Offset NUM|TXT| ←
          SHOW|DIR|MULTI
```

Listviews come in 3 kinds according to the last keyword you use :

Normal : (NUM or TXT or SHOW)

-----

The normal listview is a single-select simple listing of a file.

```
xListView 10 10 200 100 ListView Var dh0:MyFile 10 TXT
```

This is a listview which will show the file "dh0:myfile" and when clicked upon, the Text (TXT or SHOW) of the line you clicked on will be placed into the "Var" variable. If you had given the NUM keyword (instead of TXT or SHOW) then the line number of the line would have been stored into the variable.

The difference between TXT and SHOW, is that SHOW will show you the selected item i.e. the line clicked on will remain highlighted, until an other line line is clicked.

## Multi-Listviews : (MULTI)

-----

This is a multi-select listview, and all you do to get it, is :

```
xListView 10 10 200 100 ListView Var dh0:MyFile 10 MULTI
```

(i.e. change the last keyword) - This kind allows you to make multiple selections and read them afterwards. This listView will only "Happen" if the user double-clicks on an item - then this will be stored into the variable.

You can "read" the selected items at any time.

See the Directory ListView explanation - they both work in the same way.

## Directory Listviews : (DIR)

-----

These are File/Dir listers and this is how to get them :

```
xListView 10 10 200 100 ListView Var dh0:MyDirectory 10 DIR
```

These types have more dedicated commands and need more explanation which you can read

HERE!

Understanding ListViews & their Commands :

-----

All Listviews have the following

dedicated commands

Listviews are governed by the notion of CURRENT LISTVIEW and CURRENT LINE (of the current listview).

The CURRENT LISTVIEW is the one on which all listview commands will act on.

When the user clicks on a listview, that listview becomes the CURRENT LISTVIEW and all listview commands executed thereafter will refer to that Listview.

You can also specify the current listview you want, with :

```
> LVUSE GuiFile GadID
```

Now, all commands will use the specified listview.

Each listview is made up of a number of lines (or "records"). Commands that act on individual lines (like LVPUT) will act on the CURRENT LINE of the listview.

The current line is the line that the user last clicked upon, but that can also be changed to any line you want with the LVGO command.

More over, commands like LVSEARCH, LVADD will change the CURRENT LINE to the line they return or add etc.

-----

The internal variables `$$LV.LINE`, `$$LV.REC` and `$$LV.TYPE` will always show you what the CURRENT LINE of the CURRENT LISTVIEW is. Each listview will keep track of it's own CURRENT LINE, and remember it throughout it's life.

Listview movement

-----

LVs can be shifted right/left (if there is something to show). The "Offset" argument is the number of characters it shifts. You can set a 0 value, to disable this feature.

LVs can also be moved with the arrow keys. For this to happen you must NOT have declared any `xOnKey #R` (i.e. Raw Key commands) in the gui they reside. Gui4Cli will move the CURRENT LV :

- Up/Down/Left/Right arrows - normal movement
- Control key + arrows - Fast movement
- Shift + arrows - go to top,bottom,left,right of document.

COLORED TEXT IN LISTIVIEWS :

-----

Starting with V3.2, listviews will interpret ANSI sequences such as `"\#27[32mText is now white"`, and render text in colors or bold italics etc.

And it was a real bitch to get it right, too...

## 1.54 LTWH

L T W H -> (L)eftEdge, (T)opEdge, (W)idth, (H)eight

These 4 values are the sizes you declare for windows, gadgets & ASL requesters. Apart from normal numbers, you can also use the following numbers, which have special meaning :

L = LeftEdge - if L is a negative value :

- If used with a window or requester, then these will be centered horizontally on the screen.
- If used with a gadget, then the gadget will be centered horizontally in it's window.

T = TopEdge - if T is a negative value :

- If used with a window or requester, then these will be centered vertically on the screen.
- If used with a gadget, then the gadget will be centered vertically in it's window.

NOTE : The Left & Top sizes you declare will be offsets INSIDE the window. Do not account for the window border, it will be added automatically.

W = Width      - if W is less than or equal to 0, then :  
                   - If used with a window or requester, their width will be the SCREEN WIDTH minus this number  
                   i.e. -1 will make a window 1 pixel smaller than the screen size, and if 0 it will be screen sized.  
                   - If used with a gadget, the width of the gadget will be the width of it's window minus this number.

H = Height     - if H is less than or equal to 0, then :  
                   - If used with a window or requester, their height will be the SCREEN Height minus this number  
                   - If used with a gadget, the height of the gadget will be the height of it's window minus this number.

The only case where the above does not apply, is to the WinSmall command - look at it to see the meanings.

## 1.55 LVAction

```
LVACTION <action> GuiFile GadID <Destination>
```

This command is practically the same as the  
                   ACTION  
                   command.

See it for more information.

The difference is that ALL SELECTED Files and/or Directories of the Dir ListView which resides in GuiFile and has the given GadID will be operated on.

All needed updating of the listview will be done automatically. With the SIZE action, the dir sizes will be displayed next to their names as they are being processed.

Makes life a lot easier in dealing with dir listviews..

## 1.56 LVDir

```
LVDir Parent|Root|Disks|All|None|Refresh|NoRefresh#DirName
```

This will do one of the following things to the CURRENT Dir Listview:

Parent      Will change the directory to the parent directory

Root        Will change to the Root dir.



Disks Will change to the Device List

All Will select all the files or dirs

None Will clear all selections

NoRefresh - This takes some explanation :

The DirListView, loads the files it finds on your computer, and then places them as strings in the listview. Thereafter it acts like a normal multiselect listview. Now, if you were to e.g. delete some files and not refresh the listview accordingly, the files shown would be wrong, so any further actions on them would result in errors...

Gui4Cli will automatically check and if there is any change in the directory, it will ABORT all commands, and instead re-read the current directory. (Don't I think of everything...)

If though you have correctly deleted all entries in the listview according to the files you actually deleted, then you can give this command, so as to stop this automatic refreshing.

Refresh Will reload the current directory.

#DirName Give this, to change to a specific directory.  
ex : LVDir #DF0:MyDir

## 1.57 LVHook

LVDIRHOOK HookID - only for directory & multi listviews

----- YOU NO LONGER HAVE TO DECLARE THIS MODIFIER. ----

If you do not declare it, the HookID will be the same as the GadID if this gadget.

The HookID is a number of a  
XLVDirHook  
event which will be  
executed every time you change directories, or in an xLVHOOK

The Directory ListView will show you the file list, but it will not show you what directory you are in.. You have to do this yourself, using this modifier and the event it points to.

## 1.58 LVMulti

```
LVMulti First|Next|On|Off      (First_xxx|Next_xxx)
```

This allows you to read the selections made on multiple and directory listviews. It will act on the CURRENT LISTVIEW.

You MUST start with :

```
- LVMULTI FIRST
(or First_xxx)
```

This will store into the given ListView's variable, the text of the first selection in the list. (or "" i.e. nothing, if there is none).

Thereafter you read selections in some kind of loop, usually using the WHILE/ENDWHILE commands :

```
- LVMULTI NEXT
(or Next_xxx)
```

This will store the next selection into the listview's variable. You will call this many times, until it stores a "" (i.e. nothing) into the variable, whereby you know that the selections have finished.

Example : (Filename = MyGui)

```
-----
G4C
WinBig  -1 -1 320 100 "My Directory"

xListView 10 10 200 100 "" LVar dh0:MyDir 10 DIR
GadID 1

xButton  210 10 100 15 "List Selections"
LVUse    MyGui 1        ; use the above LV
LVMulti  First        ; goto the 1st selected record
while    $$lv.line > "" ; while there are records
        Say '$LVar\n'  ; print it
        LVMulti Next   ; goto next selected record
EndWhile ; close while loop

----- end of file
```

You can also use LVMulti to set lines ON(select them) or OFF.

```
ex: LVMulti ON
will select the current record of the current LV.
```

So, what the hell are <First\_xxx> and <Next\_xxx>, I hear you scream..

\*\*\* Not Implemented yet \*\*\*

Well, many programs can take myltiple files as arguments, so there is no need to feed each file one by one to them.

By using these two, the DirListView's Variable will be filled with all selections made (i.e. the file names you've selected, listed one after the other), until the first xxx bytes (ex: First\_910) of your "buffer" are full.

By default, the "Buffer" in Gui4Cli is 1024 bytes. This means your variables can not have more than 1024 characters (spaces inclusive). The buffer size can be changed with the "ToolTypes and Options".

So when executing a command line you would give a command :

```
CLI 'MyProgram $MyFiles $Option1 $Option2 etc etc'
```

Here, you must calculate approximately the size that the program name and all the other options will take up in the command line, and ask for First\_xxx Next\_xxx number of bytes in the \$MyFiles variable to be filled, so that the complete resulting command line fits into the 1024 bytes (or whatever) buffer available.

You got all that ?

## 1.59 LV\_Commands

### LISTVIEW COMMANDS

Listviews have the following dedicated Event Commands that you can use. All the commands (except LVUse) will act on the CURRENT LISTVIEW, or the CURRENT RECORD of the current listview. (See the LV explanation)

LVUSE        GuiFile GadID

Select another listview and make it the CURRENT LISTVIEW.

When the user selects an item from an LV, then that LV becomes the current one. However, you can change the CURRENT LV at any time, using this command.

This is an important command, as all other LV commands act on the CURRENT LV, and this is the command which can change it.

LVCHANGE    NewFromFile

Load another file into the current LV.

The current record will be reset to 0 - i.e. the 1st line.

LVFIND        String

This command is for sorted LVs and it will find the line (record) containing the String and visually move the ListView to it.

If the line is not found, it will stop at the closest alphabetic match to the line.

The internal variables \$\$lv.line, \$\$lv.rec etc will tell you the line found.

LVSORT        ASC|DSC

This command will sort the listview alphabetically, in ascending

order. The ASC|DSC argument is (currently) disregarded. Just state "LVSort ASC".

The current record will be reset to 0.

LVADD String

Add a new line (or record) to the current listview.

The current record will be the line added.

LVINSERT RecordNumber String

Insert a new line (or record) to the current listview, *\*before\**

RecordNumber (-1 means before current record)

The line added will become the new "current" record.

ex: LVInsert 0 "New Top Line" - will insert a new first line.

LVSAVE SaveFileName

Save the listview as "SaveFileName"

LVCLEAR

Clear the listview. All records will be deleted and the current line will be "" i.e. nothing.

LVPUT string

will change the text of the current record the LV is at, to what ever you give in "String".

LVMOVE +-offset (do also #rec)

Visually move a LV left or right (+-offset) or visually move it to a given record (LVMove #42 - move to rec No.42)

This command will move the LV *\*without\** changing the current record the LV is at.

LVGO first|next|prev|last|#LineNumber

This command will change the current record of the LV *\*without\** visually moving it.

For example - the following code will print out the LV's contents:

```
LVUse MyGui 1          ; use LV Nol in MyGui
LVGo first            ; goto the 1st record
while $$lv.line > ''  ; while there are records (or lines)
    say '$$lv.rec\n'  ; print the line
    LVGo next        ; go to next record
endwhile              ; end of while loop
say 'Boy.. that was easy!\n' ; superfluous comment :)
```

LVSEARCH string cs|ci first|next

Search the LV for a given text ("string").

CS|CI means = Case Sensitive (CS) or Case Insensitive (CI)

First|next are for control.

LVSearch "MyString" cs first

- will return with the first line containing "MyString". You can get the line number, text etc of the line found, by looking at the internal variables. You can thereafter call

LVSearch "MyString" CS next

- again and again, to get the rest of the lines.

If there are no more more lines containing the given text, then

the \$\$lv.line internal variable will be "" (i.e. nothing -

note "" is *\*not\** the same as 0 which is the 1st line in LVs).

LVMODE NUM|TXT|MULTI|DIR

This command will change the mode of the LV on the fly, to the mode you give. Be careful with this command as it can get confusing..

LVDEL LineNumber (or -1 for current line)

Will delete the line with the line number given, or, if -1 is given, the current line the LV is at.

Important : if the current line is deleted then the previous line becomes the current one, unless it's the first line in the listview, whereby the new top line becomes the current line.

LVCOLORS GuiFile GadID ForeGround BackGround Selected Dirs

- will set the colors of the listview :

ForeGround is the color the normal text is drawn in

Background is the.. emm.. background..

Selected is the background color of the selected items

Dirs is the color of the directories in DirListViews

The defaults are : LVColors MyGui GadID 1 0 3 2

Commands for DIRECTORY LISTVIEWS \*only\*

(These are listed here in short - there are detailed explanations of them in the command list).

LVDIR All|None|Parent|Root|Disks|#dirname(string)

LVMULTI First|Next|ON|OFF - will change the current line

LVACTION COPY|COPYNEW|MOVE|DELETE|SIZE|PROTECT|CLI Dest|REQ|NOREQ|Var|Mask

## 1.60 MakeDir

MAKEDIR NewDirName

Create a new directory, named NewDirName.

That's all really...

## 1.61 MakeScreen

MAKESCREEN ScreenName Depth Title

This command will make a public screen, on which you can open windows.

ScreenName The name of the screen you want to make.

The name is case sensitive, so when referring to it, you have to have the same case letters.

Depth A number, signifying the bitplanes of the screen. It can be from 1-8. If it isn't, the depth will be set to 2.

Title            A string which will be the default title of the screen.

Ex : MakeScreen MyScreen 4 "Hey! - This is my Screen!"

Apart from the name, depth and title, the new screen will be the same as the Workbench screen, and at a TEXT sized overscan.

You can check if the screen opened with the IFEXISTS command.

IMPORTANT :

The Screen will NOT be automatically closed when Gui4Cli quits. You MUST (if you want) specifically close it, after closing all the windows that you have opened on it.

If you do not close it, it will just stay there.

## 1.62 Mark

```
MARK    MarkName
GOTO    MarkName
```

This is a simple way to jump around inside an Event's commands. You don't really need this command, since it may be confusing.

When you declare MARK, you are just marking a point in the program. No action is taken when the command is encountered.

GOTO goes to the MARK which bears the same MarkName as itself.

You can not jump to a Mark of a different Event.

## 1.63 NewFile

```
NEWFILE    NewFileName
```

This is a "parser" command, i.e. it is executed while G4C is reading your file. It enables you to have many GUIs in one file.

When G4C encounters this command in a file, it marks the beginning of a new GUI. No new "G4C" marker is needed !!

example :

```
----- File named test1.gc
G4C
```

```
WinBig -1 -1 400 100 "test 1"
```

---

```
NEWFILE test2.gc
```

```
WinBig 10 10 300 150 "test 2"
```

```
----- end of file
```

This file, when loaded, will result in 2 GUIs, one named test1.gc and another named test2.gc.

IMPORTANT :

Such multiple GUI files have the following limitations :

- Only the xOnload commands of the 1st file will be executed. xOnLoad events of other files, if present, will be ignored.
- On Ctrl-R (reload), all GUIs contained in the file will be reloaded but only the GUI you asked to reload, will have quit first, so you will get reports that the other files exist - This is normal.
- On Ctrl-E (edit), the "mother" file will be loaded for editing.
- RESIZE\_BIG & RESIZE\_SMALL commands now have filename. But the old versions will also work.

## 1.64 Operation

### PROGRAM OPERATION

-----

Gui4Cli can be started from the Workbench by clicking on it's icon or by having it as the default tool of your script, or from the CLI.

One of the best ways to start-up Gui4Cli is to write a small script, declaring your favourite editor, output etc and save it in your WBStartup drawer. Give the script an icon with the default tool set to GUI.

When Gui4Cli starts up, it may or may not be given a program (script file) or programs to execute. In any case, it will load itself and install an APPMENU Item, by the name of "Gui4Cli...".

This will appear in the Tools menu of the Workbench.

This is your anchor to the world. When you choose this menu item, an easy-requester will appear, allowing you to Quit, Load, Open, or Unload Guis.

If you choose Quit, all windows will be closed, all GUIs unloaded and the program will quit.

Load will take you to the default directory where your GUIs are, so you can choose one and load it. - If no directory has as yet been given, you will be asked for Volume GUIs: (unless you have already made such an assign to the directory where you keep your GUIs - Good idea! ).

---

Open and Unload will take you to a directory called ENV:Gui4Cli, where the names of the loaded GUIs are listed, and you can choose one.

A GUI can be loaded into memory and be or not be open. You can have as many GUIs as you want loaded into memory, and as many of these as you want, open at the same time.

Thereafter, every time you run Gui4Cli, it will look for a running instance of itself and if it finds one, it will send it an ARexx command to load the program you have specified, or clicked on.

This means that you will only have one running Gui4Cli, with many GUIs loaded and some or all of them open.

If you need another instance of Gui4Cli, you have to specify another PORT name in the tool types, or as a CLI option. In this case, another APPMENU item will be created with the name of the PORT you have specified and through this other GUIs can be run separately.

In effect, you can have unlimited, easily constructed GUIs running on your Workbench or on other Public screens.

```

      |---> A GUI                                |--> Command 1
      |                                         |--> Event 1 -->|--> Command 2
      |                                         |                               |--> etc
      |                                         |
      |---> Another GUI ---->|--> Event 2
Gui4Cli ----> |                               |
      |                                         |--> Event (..well you
      |---> and another .                       get the point).
      |                                         .
      |                                         .
      |---> etc
      .
      .

```

This is what it looks like in concept.

## 1.65 Operator

The following are the operators you can use with the IF, WHILE and DO CASE commands (all comparison are case-sensitive unless stated otherwise) :

EQUAL (Insensitive)	==	(Comparison is CASE-INSENSITIVE)
EQUAL (Sensitive)	=	(Comparison is CASE-SENSITIVE)
NOT EQUAL	!=	
MORE	>	(Left side more than right side)
MORE than or EQUAL to	>=	



```

LESS < (Left side less than right side)
LESS than or EQUAL to <=

FILE HEADER COMPARE H= This operator MUST have a file name
on the left of it and a comparison
string on the right :

IF df0:MyFile H= "FORM????ILBM"

the comparison string contains the
starting bytes of the file type you
are looking for. - Question marks
are wild (-see below-)

STRING COMPARISON S= This MUST have a string on either
side of it and will compare them.
it is case-insensitive and wild
characters can be used (see below).

ex : IF "This" S= "THIS"

This is meant to be used along with
the ReadVar command and provides a
faster method of checking file headers
than the above H= operator.

DEVICE COMPARISON D=
FILE/DIR COMPARISON F= Both of these need a file, dir or
device on either side.
D= will yield TRUE if the files/dirs
are on the same device, and
F= will be TRUE if they are the same
file or directory.

```

Wild Characters that can be used with H= and S=

```

? Means any character
# Means any Printable character
% Means any UnPrintable character

```

Operators MUST have a space to each side of them :

```

if This= That <- is not correct
if This=That <- is not correct
if This = That <- is correct

```

## 1.66 Options

TOOLTYPES & OPTIONS

---

The CLI or TOOLTYPE options are identical. The only difference is that the TOOLTYPES are in the icons.

PORT=PortName Usually the Public message port name is Gui4Cli. This way all GUIs are run from the same one program. You can, if you want, though, launch as many Gui4Cli's as you want, by specifying different PORT names for them. This keyword is the way to do it.

DEBUG If this word appears in the commandline, then Gui4Cli is started in the "debug" mode, which means that it will print out in the default shell all commands that it executes, in the order it executes them. Use this command to see what the program is doing, if it behaves illogically. The STACK statements that are printed out, you can forget.

DEBUG (in a simplified form) can also be started/stopped at any time, with the SET DEBUG ON|OFF command.

BUFFERS=Bytes When Gui4Cli starts up it grabs 12 buffers (pieces of memory), in which it does all the translating and other command processing. These are 1024 bytes long, by default. This means, for example, that your translated command line in a CLI or SENDREXX command, can not be longer than 1024 characters. It also means that your variables can not be bigger than this size. You may, if you want, change this with this option.

This command can ONLY be used when Gui4Cli is loading for the 1st time under the port name given.

OUTPUT=Console By default, the console specification for any output from CLI commands, or Gui4Cli error reports etc, is :

- If started from a shell -> the shell itself
- If started from WB -> "con:0/12/450/80/Gui4Cli Output/AUTO/CLOSE/WAIT"

This is the console that is passed to all CLI commands. (Note that RUN, doesn't get a console).

You may want to change this (to KCon:, or NIL: for example) so this is the way to do it.

Note also that you may specify a different console for each GUI (with the WinOut command), but this is wasteful, as each new console you specify uses memory.

EDITOR=Editor (NOT CLI) When you press CONTROL-E, the default editor will be started up, in order to edit the current GUI. The default editor is c:ed. With this option you can change the name of the editor to whatever you want (ex : EDITOR=C:CygnusEd)

NOREQUEST If you specify NOREQUEST as an option, you will disable the file requester which appears at startup. If no filename is given - allows you to put a GUI with an icon

into the WBStartup drawer without getting a requester at startup time.

The OUTPUT and EDITOR can also be changed with the SET command.

## 1.67 ParseVar

PARSEVAR VariableName

This command will take a variable and split it up into arguments (words or quoted strings), returning them as internal variables.

The internal variables are :

\$\$PARSE.0, \$\$PARSE.1, \$\$PARSE.2 etc - up to PARSE.29  
which will hold the various arguments (30 maximum)  
or words resulting from the splitting of the variable  
The 1st word will be in \$\$PARSE.0

\$\$PARSE.TOTAL will contain the number of arguments that have been parsed.

Note - the ; character ends the line being parsed (like in G4C)

The values of the PARSE internal variables will remain unchanged until another ParseVar command is issued.

## 1.68 PartRedraw

PARTREDRAW GuiName L T W H

This command was added to make redrawing of windows easier to the eye. It will redraw only part of the window, thereby reducing the flashing.

Use this if you're just changing some gadgets in one area of the window and do not want a full redraw.

L T W H denote the area which you want redrawn.

It's exactly the same as the  
Redraw  
command, only that  
a part of the window is erased instead of all the window as is done in the Redraw command. Otherwise, the gadgets, graphics etc are drawn from scratch, same as with Redraw.

## 1.69 Passing\_Args

PASSING ARGUMENTS :

Arguments can be passed by the following commands :

```
GUILOAD   to xONLOAD or xONRELOAD
GUIOPEN   to xONOPEN
GUICLOSE  to xONCLOSE
GUIQUIT   to xONQUIT
```

```
GOSUB     to xROUTINE
```

The way this is done is by declaring the arguments in the command's command line, like so

```
GUILOAD  guis:MyGui Arg0 Arg1... Arg5
or
GOSUB    MyGui   MyRoutine   My_Arg   My_Other_Arg
```

Up to 6 arguments may be passed.

Inside the event they call, these arguments can be accessed in either of two ways :

1. As Arguments in the Event's definition.

If declared in the event's definition, they'll be automatically converted to variables. For example :

```
> xONLOAD  Var1 Var2
```

If you were to call this event with :

```
> GUILOAD  MyGui.gc SomeText "Some more text"
```

then when MyGui.gc loads, there would be 2 variables created :

- "Var1" which would contain the word "SomeText" and
- "Var2" which would contain "Some more text"

You can also pass variables:

```
> gui/myvar = "Some text"
> GOSUB MyGui.gc RoutineName $gui/myvar
```

If you pass variables you should use the variable's *\*full\** path (as is done above) to avoid confusion.

2. As internal variables \$\$ARG.0 to \$\$ARG.5

Note that even if you define variables in the event's definition as above, they'll still be available as \$\$ARG.xx also.

\$\$ARG.TOT will (always) contain the actual number of arguments passed.

Furthermore, the above events (xONLOAD, xROUTINE etc) can now RETURN arguments via the RETURN command.

```
RETURN Arg0 Arg1... Arg5
```

The arguments can be accessed (when they return to the routine that called the event) as internal variables \$\$RET.0 to \$\$RET.5. \$\$RET.TOT will contain the actual number of arguments returned.

Example :

```
xBUTTON 10 10 60 12 Test
  var = "This is a test"
  GoSub MyGui routinel $var
  ; We only pass "var" as an argument here. We could have had up to 6 args.
  ; the routine will return one argument to us (also could have been 6)
  ; which we just print..
  say 'Return returned $$RET.0\n'
```

```
xROUTINE routinel MyVar
  ; here variable "MyVar" will contain $var (i.e. "This is a test")
  ; Note that $$ARG.0 will also contain the same thing.
  say '$MyVar is the same as $$ARG.0\n'
  ; now we return something...
  Return "Thank you.."
```

\*\* WARNING :

- If you want to pass a variable you should give the variable's full path (i.e. RETURN MyGui/MyVariable), since it may be in another gui which may not have you in it's varpath.

\*\* WARNING 2:

- You can not pass an \$\$ARG.0 type of internal variable you receive as an argument to an other routine/xonload etc. :

```
> GOSUB mygui myroutine $$ARG.0          <-* is WRONG
```

That's because when Gui4Cli attempts to translate the variable when it gets to the routine, it will find variable \$\$ARG.0 containing \$\$ARG.0 and it will go into a wild loop trying to translate it..

This loop will time out after 100 rounds (i.e. the default depth of variables within variables that you can have) and the error will be reported.

There are no such problems if you use the definition in the event's command line, so use that instead.

## 1.70 Paths

FINDING other Guis and their paths:

Whenever you make a GUI which other people might also use, you will need to know various paths where you can load any helper GUIs or icons that your GUI needs, since you don't know how other people's systems are set up.

There are 2 ways of doing this :

1st is to use the

Extract  
command, as follows :

```
Extract MyGui.gc GuiPath MyPath
```

Now variable MyPath will contain the full path of MyGui.gc GUI. You can now assign this path, or use it to AppVar various other GUI names and load them.

There is a much easier way though, and that's by putting a ":" character in front of the filename or icon name, like so :

```
GuiLoad :MyGui.gc
```

Gui4Cli will automatically convert the ":" character to the full path of the GUI in which this command exists. This method also works with icons, so you can do :

```
xIcon 10 10 :MyIcon
```

and the icon will be loaded from the current file's path.

## 1.71 Programming

HOW TO MAKE A Gui4Cli PROGRAM

-----

The structure of a Gui4Cli program may, in the beginning, be a little confusing to understand since it is not like other languages. Once you understand it though, it is very easy.

Start with G4C

First of all, every Gui4Cli program MUST start with the letters G4C. These must be the first letters of the first line of every GUI, so that Gui4Cli knows it is a loadable file.

---

## GLOBAL &amp; GRAPHICS COMMANDS

Then come the GLOBAL COMMANDS and the GRAPHICS COMMANDS (if any).

These can appear in any place in the file, but it is best to put them at the beginning of the program, for clarity's sake.

The Global Commands specify the window size and type as well as other general characteristics of the GUI, such as the screen it should open on etc.

The most important of the Global Commands is "WinBig", which describes the size of the window. If this command is not declared, no window will open.

The GRAPHICS COMMANDS are simple drawing routines, enabling you to tart-up your GUI. They provide colored text, lines, boxes, circles etc.

The rest of the program consists of EVENTS and their EVENT COMMANDS

## EVENTS :

All events (except for TEXT) start with an "x" e.g. xBUTTON, xLISTVIEW, xONOPEN, xONCLOSE etc.

Events can be gadgets, menus, routines, AppItems or User actions.

For example :

```
xBUTTON is a Gadget
xMENU   is a Menu
xONCLOSE is a user action (i.e. if the window is closed)
xROUTINE is a routine
xAPPMENU is an AppMenu Item (WB Tools menu)
```

All events, of whatever type are treated in the same way.

Whenever an event "happens", the commands attached to it will be executed in the order they appear.

The only exceptions to this are :

- a) The xROUTINE event which is a subroutine, called on by other events
- b) The TEXT event which is just a text display and never "Happens"
- c) The xOnReturn event which is a routine linked with the LAUNCH command

The Gadget and Menu type events, also specify the type and specifics of the gadget or menu.

A button type gadget (EVENT), for example, can be declared as follows :

```
xBUTTON 10 20 100 15 "Click me!"
```

This declares a button positioned 10 pixels from the left side of the window and 20 pixels from the top. The size of the button is 100 pixels wide and 15 pixels high, and it will bear the title "Click me!".

Most gadgets, will have a variable where they will put their value.

---

## EVENT COMMANDS :

Event Commands are "attached" to Events.

You can attach any number of event commands to any event, making small sub-programs (like routines or functions in other languages), which will be executed every time the event they are attached to "happens".

These commands do different things.

Some, such as IF, WHILE, GOTO etc, provide a means to control your program. Please note that Gui4Cli is not the most powerful of languages (to say the least) in this field. The control commands provided are not very powerful or intuitive, but they are quite sufficient for Gui4Cli, since it is not meant to be a "programming" language so much as it is a "control" language.

Others such as SETVAR, RUN, SENDREXX etc perform some action.

Finally, some Event commands, such as GADFONT, GADID etc, modify the specifications of the event (usually a gadget) to which they are attached.

Event commands can be ARexx capable or not. The detailed command descriptions say whether each command is ARexx capable or not.

If it is, it means that they can be passed to Gui4Cli from other programs as ARexx commands and will be executed as if they were attached to an Event.

There is no special statement that ends an Event. Each event routine ends when another Event is declared.

## EXAMPLE PROGRAM :

For example, this is a little "Hello World" program. We'll name it "Hello.gc"

```
-----Start of file
G4C                ; You must put this at the beginning.

WINBIG 100 20 100 50 TEST    ; Specify a window

xONLOAD            ; To be done on loading the GUI
GuiOpen Hello.gc        ; Open the GUI's window.

xBUTTON 10 12 80 15 HERE!   ; Specify a button
CLI 'echo "Hello World!'"   ; Attach a command to it.

-----End of file.
```

When you run this program, a window will open with a button in it. When the button is clicked on, a standard con: window will appear (since you have not specified otherwise) and the words Hello World! will be echoed.

If you click on the window's close gadget, the window will close but



the GUI will remain in memory, since we have not told it to quit.

If we wanted the GUI unloaded from memory as soon as the window closed, we would add the following Event :

```
xONCLOSE
GuiQuit Hello.gc
```

Note that the commands can be in capitals or small letters.

We did not use "translation" in this program, since we did not have to. Let's make a simple program using translation. (call it Hello2.gc)

```
-----
G4C

WINBIG 0 0 100 50 "2nd Test"

xONLOAD                                ; upon loading of the gui
Var1 = "Hello World"                   ; declare variable Var1
GUIOPEN Hello2.gc                       ; and open the gui

xBUTTON 10 15 80 15 "Hit Me"
CLI 'echo "$Var1"'                       ; echo the variable
IF $Var1 = "Goodbye World.."           ; see explanation..
    GUIQUIT Hello2.gc
ELSE
    Var1 = "Goodbye World.."
ENDIF
-----
```

Here, in the beginning, we put the words "Hello World" into Var1, and open our window.

When the button is clicked on, we echo the value of the variable, and afterwards we set it to "Goodbye World..", so that the second time the user clicks on the button, after "Goodbye World.." is echoed, the GUI will quit.

Ok.. enough talking - you can see how easy it is.

The best way to learn to write GUIs, is to look at the tutorials, so go look at them. You can change them in any way you want and see what happens.

You will be impressed at what can be achieved (I hope :)

## 1.72 Quit

```
QUIT
```

QUIT will quit Gui4Cli.

It will close all open windows, executing any xOnClose events (if any)

---

and will unload all guis, executing any xOnQuit events.

QUIT can also be chosen from the STATUS requester.

## 1.73 Redraw

```
REDRAW GuiFile
```

This command will Re-Draw the GuiFile's window, taking into account any changes you may have made to the GUI with the ChangeGad command or with the SetGad command (with the SHOW|HIDE keywords).

Use it after using ChangeGad or SetGad on all the gadgets you want to change.

example :

```
setgad mygui.gc 1 HIDE
setgad mygui.gc 2 SHOW
redraw mygui.gc
```

A cooler redrawing can be achieved with the  
PartRedraw  
command.

## 1.74 Rename

```
RENAME OldName NewName
```

Will rename the given file or dir.

## 1.75 ReqFile

```
ReqFile L T W H Title SAVE|LOAD|MULTI|DIR Variable DirName
```

This is an ASL file requester. You must have the asl.library in your LIBS: directory.

(L)eft, (T)op, (W)idth and (H)eight are the starting position and size of the requester. If you resize it thereafter, the new sizes will be used.

Title is the title in the requester window title bar.

SAVE, LOAD, MULTI, DIR are the types of requesters you can use.

---

Variable is the name of the variable, where the names of any file(s) that were chosen will be placed. If its a multiple selection, the files will be listed one after the other. If the file names contain spaces, they will be quoted.

DirName is the starting directory that the requester will open on. Thereafter, the directories will be remembered.

Note : You can give "" as the DirName.

This will use the last directory used by other ReqFile commands, or sys: in case no other ReqFiles have been executed yet.

## 1.76 ReSize

### RESIZABLE GUIs :

-----

How to make them :

All you have to do in order to make a GUI resizable is declare the border gadgets with the

```
WinType
command.
```

Notes on resizing :

Gui4Cli is (or tries its best to be)

```
font-sensitive
```

This means that it will automatically resize a GUI to fit the font ←  
you

use.

However, the best method of getting and keeping the GUI size you want on your system, is to use the CONTROL-W short cut :

When you press CONTROL-W on a window, a requester will appear which will tell you the current window size & position and have the following choices :

BIG : Save this window size as the default size it opens with

Small : Save this window size as the minimum size

Adjust : Resets the current minimum window size and allows you to resize the window to any size.

CANCEL : You know...

Saving the window sizes, instructs Gui4Cli to append one of 2 commands :

```
RESIZE_SMALL Left Top Width Height - or
RESIZE_BIG Left Top Width Height
```

to the specified GUI file.

This command will instruct Gui4Cli to resize the window to the specified size, upon loading it. Note that when Gui4Cli loads the file, it can encounter many such RESIZE\_xx commands, since you may have saved the window sizes many times. It will only use the last ones it encounters, so you may as well delete any others..

NOTE :

When you resize a GUI, the gadgets are redrawn to their new sizes positions etc. and with the values they currently have.

The xLISTVIEW's current value, is the line number of the last line you clicked on it (if any, otherwise 0), and so when redrawn it will show this line as the 1st one. This may cause some confusion....

It's the fault of the GadTools library, so leave me out of it..

## 1.77 Run

Two commands give you the means to run CLI commands :

```
CLI   CommandLine
RUN   CommandLine
```

Both are AREXX capable.

The CommandLine is anything that can be typed at a shell command prompt. Note that the command line is a single argument, so, if it has spaces in it (which it probably will), you MUST enclose it in quotes (single quotes preferably).

Example :

```
RUN 'Dpaint $Myfile'
```

The difference between the 2 commands is :

```
RUN - will run the command line asynchronously
CLI - will run the command synchronously.
```

BE CAREFUL with the CLI command, as it will stop all processing of Gui4Cli until the command you have given it has finished executing (unless you have given something like : CLI 'run >NIL: Dpaint').

Both commands will set the \$\$RETCODE internal variable which you can check to see if the command you issued worked or failed. 0 means OK. This, usually, is only useful when using the CLI command, and then, only if you don't "run" it.

Use IFEXISTS, WAIT and other methods if you want to make sure that you have

---

run the program successfully.

IMPORTANT : Some programs require a large amount of stack to run. The default stack passed with the above CLI/RUN commands is 4000 bytes. You can change this with the SetStack command, to any number you want.

## 1.78 Say

SAY Text

This command writes the Text you give it to your default console (if any)

ex :

```
SetVar Roses "red"
SetVar Violets "blue"
SAY "Roses are $Roses, violets are $Violets \n"
```

This should print out "Rose are red, violets are blue"

The \n is a

Special character

## 1.79 Screen

Screen PublicScreenName

Specifies the name of the Public screen that the window will open on.

If you do not declare this command, the gui will open on the FRONT most screen. This is the most intuitive setting as the guis will follow the user to whichever screen he is currently at.

In all cases, the front screen can be denoted with '\*' (asterisk)

If the screen is not found, the window will open on WorkBench, if available.

Example :

```
Screen CygnusEdScreen1
```

Note : \*\*\*\*\* SCREEN NAMES ARE CASE-SENSITIVE ! \*\*\*\*\*

## 1.80 SendRexx

---

```
SENDREXX REXXPortName CommandLine
```

This is the command used to send ARExx messages to other programs.

RExxPortName is the name of the REXx port to which you want to send the command. Port names are always CASE SENSITIVE.

CommandLine is the REXx command you want to send.

example :

```
SendRExx rexx_ced 'open $Myfile'
```

- will send the command to load a specified file to the rexx port of the CygnusEd editor. "rexx\_ced" is CygnusEd's rexx port name.

This command sets the following internal variables :

\$\$RETCODE contains the value returned by ARExx - 0 means OK

\$\$REXXRET contains the value returned in RESULT2, if any.

If you don't know anything about ARExx, you're not alone - neither do I. You don't have to know anything though to use this command. Most programs provide ARExx commands. All you have to know, is what they are and use them as you would use CLI commands.

You can do a lot with this command, as you will see.

## 1.81 Set

```
SET [parameter] [value]
```

With this command you can set the following GLOBAL parameters :

```
DEBUG ON|OFF
```

This will enable or disable the internal debugger of Gui4Cli.

You can get debug information in two ways :

One is by starting Gui4Cli in DEBUG mode. This will show you exactly which line is executed, as it is written in your file.

Or you can use this new form of DEBUG which can be started/stopped at any time, without having to start up G4C in debug mode. The new one will show you cryptic statements instead of KeyWords and Operators, but is otherwise same.

Also, if you press any key while in any debug mode you'll get it's Code & Qualifier values.

```
HELP ON|OFF
```

Will start/stop the on-line help system. See also

GadHelp  
GRID GridSize

Will set the Grid size for  
Visual Editing  
ex : Set grid 5

LOAD\_DIR DirName

This will set the directory that the Status Requester LOAD  
button will open on to DirName.

TRANSLATION ON|OFF

Usually all arguments will be translated by Gui4Cli.  
In some cases though you may not want something translated.  
With this command, you can set the translation OFF.  
This will last for one round of command executions.  
Translate will automatically be set on again after all commands  
have executed.

DEEPTRANS ON|OFF

Setting DEEPTRANS OFF will stop Gui4Cli from translating variables  
within variables. As with TRANSLATION, this will only last for one  
execution cycle.

This comes useful in the filetype checking routines which are now  
much faster.

OUTPUT ConsoleSpecification

All output from CLI commands or Gui4Cli error reports, will  
go to the default console. (see explanation in the ToolTypes)

If you want to change this console you can do it with this.  
You can specify KCon: for example, or NIL:

FILEPATTERN FilePattern

The default file pattern for all file requesters, is :  
~(#?.(bak|info|auts))"  
You can change this with this command.

EDITOR EditorName

The default editor which will be called upon to edit the window  
when the user presses CONTROL-E, is "c:ed"  
You can change this here.

GUIPATTERN NewPattern

The default GUI file name extension for Gui4Cli is #?.gc  
(i.e. anything ending in ".gc" - subfiles end with ".g", ".g1" etc)

You can change this (but it is not a good idea)

---

## 1.82 SetColor

```
SETCOLOR GuiFile ColorNumber R G B
```

This command allows you to change the color of a screen.

GuiFile            Is the Gui4Cli file who is, or will, open on the screen whose color you want to change.

ColorNumber        The number of the color you want to change.  
The first color (the background) is 0.

R G B              3 numbers, ranging from 0 to 15, specifying the values for Red, Green and Blue respectively.

Ex :

```
SetColor MyGui 0 10 8 8
```

- will set the background of the screen that MyGui is (or will) open on, to a yukky brownish color.

## 1.83 SetGad

```
SETGAD    GuiFile   GadID/range   ON|OFF|SHOW|HIDE
```

This command will set the specified gadget ON or OFF or make it appear (SHOW) or disappear (HIDE)

GuiFile - The name of the GUI file the gadgets reside in.

GadID    - The ID number you have given to the gadget (see the GadID command)  
\*or\* a range of gadgets denoted as First/LastGadget

Example :

```
SetGad MyGui.gc   10 OFF    - will set Gad No.10 off
```

```
SetGad MyGui.gc 5/15 HIDE - will hide gadgets Nos. 5 to 15 inclusive
```

The SHOW and HIDE keywords will have no immediate visible effect unless and until the

```
ReDraw
command is used.
```



## 1.84 SetGadValues

- SETGADVALUES GuiName

Will store the values of all the gadgets of a gui into their respective variables.

When you declare a gadget which has a value, say :

```
xHSlider  L T W H Title Variable Min Max Current ShowStr
          (here the value is the Current level)
or...
```

```
xTextIn   L T W H Title Variable StartingText Bufflength
          (here the value is the StartingText)
```

Gui4Cli does *not* automatically place the starting value of the gadget in it's Variable. You must do this manually - usually in the xONLOAD event. This provides better control (but can also be a pain..)

With this command (which you would also issue, most probably, in the xONLOAD event) you can set the starting values for all the gadgets in a gui, in one go.

## 1.85 SetScreen

```
SETSCREEN  GuiFile ScreenName
```

This will reset the name of the screen the GUI should open on.

If the window is already open, you will have to close it and reopen it in order for the new screen to take effect.

If the screen is not found, the GUI will open on the Workbench (if available)

To state the "Front most Screen" you can give an '\*' (asterisk).

See also the

```
Screen
global command.
```

## 1.86 SetStack

```
SETSTACK StackSize
```

You may need this command when using RUN or CLI commands.

Gui4Cli provides each program you run with 4000 bytes of stack. Some programs need more than that though. This command is how you do it.

example :

```
SetStack 10000
RUN      "ABigProgram"
SetStack 4000
```

Note that we change the stack size before we call the RUN or CLI command, and we change it back to the default size after RUNNING, so that the next RUN/CLI does not inherit the big stack size.

## 1.87 SetVar

```
SetVar Variable String
```

With this command you can put a value (String) into a variable. If the variable does not already exist, it will be created.

Variable - The name of the variable

String - Any character, sentence, text, number - whatever.

Example :

```
SetVar MyVariable "This is my variable and nobody touch it!"
```

\*\*\* NEW with V2.2 is that you can do this :

```
MyVariable = "This is my variable and nobody touch it!"
```

instead of using SetVar. It is much more logical.  
SetVar still works though.

## 1.88 SetWintitle

```
SETWINTITLE GuiFile NewTitle
```

Will set the window title of GuiFile's window to "NewTitle"

NewTitle is translated (as is GuiFile), so if the resulting text

---

is bigger than the original untranslated text, it will be clipped.

Do this :

```
SetVar MyVar "This is some text"  
SetWinTitle MyGuiFile "$MyVar" ;note we leave extra space.
```

Here, the buffer provided by you for the wintitle, is big enough to fit the translated version.

## 1.89 Status

STATUS

This is a command that opens the "Status" requester, from which you can see how many GUIs are loaded/open as well as Load, Open, or Unload GUI files.

From here you can also QUIT. Upon quitting, all xOnClose event commands are executed for windows which were open, and all xOnQuit event commands for all loaded files (if any are defined).

Note that the meaning of the GUI file name extensions, are :

```
MyFile.gc      - This is GUI Command file (.gc)  
MyFile.g       - This is a GUI help file, i.e. a GUI which is supposed  
                 to be loaded and in general operated from another GUI
```

There is no difference in the structure of the files. The difference is in their operation only.

You need not use the above naming conventions if you don't want to.

## 1.90 Stop

STOP - No arguments

STOP will stop all execution of commands, DEAD!  
When a STOP command is found, the program will return to its normal state of waiting for something to happen.  
Use it for getting out of trouble, from wherever you may be.

## 1.91 System

---

Things you can WAIT for or see if they EXIST (with the IFEXISTS command)

With the IFEXISTS and the WAIT commands, you can either wait, or see if the following system items exist :

FILE - Full path and filename

PORT - Public Message Port Name

SCREEN - Public Screen Name

GUI - Gui4Cli file (True if file is loaded)

WINDOW - Gui4Cli Window (True if the window of the given GUI is open)

VARIABLE - the name of an internal Gui4Cli variable - Env: variables can be checked by filename with the FILE keyword

DIRECTORY- name of the directory

DEVICE - A Dos Device such as DF0 - WITHOUT the : - i.e. Ram NOT Ram:

VOLUME - A Dos Volume - WITHOUT the :

ASSIGN - An assignment - WITHOUT the :

DOS - A Device OR a Volume OR an Assignment (i.e. any of the above 3)

Gui4Cli Files and Windows are referred to by their file names only and not the full path of the file.

Note : If you want to use RX to launch ARexx programs, you may want to check first that REXX has been started up.

REXX opens 2 ports REXX and AREXX, so you can do the following :

```
IfExists PORT REXX
; Ok - REXX is running
else
cli 'REXX'
endif
```

## 1.92 Text

Text L T W H Text Length BOX|NOBOX

This event is a simple text display with a BOX or with NOBOX drawn around it.

The L T W H are the same as for other gadgets.

The Text is the default text to be shown in the beginning. This can easily be changed with the update command.

Length, specifies the length of the buffer you want - i.e. the maximum number of characters likely to be shown in this text box. It is important that you set this number correctly, otherwise the text will be clipped.

BOX or NOBOX are keywords denoting whether you want a BOX drawn around the text or not.

IMPORTANT :

No commands other than Gadget modifiers (GadID, GadFont etc) can be attached to this event, as these will never be executed (since the gadget can not be selected and so never "happens")

## 1.93 Translation

### TRANSLATION

Translation is the name I have given to the process whereby Gui4Cli will substitute the contents of a variable for it's name, if the \$ (dollar sign character) is placed in front of the variable name.

Example :

```
SetVar   MyVar "This is my variable"  
Say      $MyVar
```

Here we put the string into the variable and then print the variable contents. The output will be :

```
This is my variable
```

Translation is a *very* important concept in Gui4Cli, and one which you *must* understand.

Translation is done automatically by Gui4Cli on ALL arguments of all EVENT COMMANDS (be they numbers or strings), making it very flexible.

You can, for example, say :

```
Update   $MyGui $MyGadgetID $MyNewValue
```

and Gui4Cli will update the given gadget in the given GUI, to the given NewValue, according to whatever these variables contain.

BTW : AmigaDOS also does this translation stuff, substituting the contents of an Environment (ENV:) variable, when ever a \$ sign is found.

## 1.94 TTGet

---

```
TTGET FullPath/IconName (Without the ".info" extension)
```

This command reads ALL the tooltypes in your GUI's icon and automatically converts them into

```
variables
```

. This is useful for any situation where you need to give your GUI a configuration.

Tooltypes for GUI icons MUST contain an equal "=" character. An example tooltype would be:

```
EDITOR=Work:Text/Editors/Editor
```

TTGET would understand this as:

```
VariableName=VariableContents
```

Everything BEFORE the "=" character becomes the NAME of the variable. VariableName must NOT contain spaces. Using the example tooltype, you would get a variable called:

```
$EDITOR
```

Everything AFTER the "=" would become the variable's CONTENTS. So we have a variable called \$EDITOR which contains:

```
Work:Text/Editors/Editor
```

The Variable's Name will always be converted to Upper case to avoid confusion, since Gui4Cli variables are case sensitive, and will be added to the GLOBAL variable list.

To use TTGET in your GUI, you must first initialize (set the default value) of the variable, preferably early on in the script. For example:

```
SetVar EDITOR "default value"
TTGet Dh0:MyDir/MyGui.gc
```

When using TTGET you must supply the FULL path and filename of your GUI icon, without the ".info" file extension.

## 1.95 Update

```
UPDATE GuiFile GadID Value
```

This command will update the specified gadget to display the new value you have specified.

GuiFile - The name of the GUI file the gadget resides in.

GadID - The ID number you have given to the gadget (see the GadID command)

Different gadgets have different update values you must give, as follows :

```
Buttons      - No update
Checkbox    - 1 = Checked, 0 = Unchecked
Sliders     - New current level of slider
Listview    - New line number           - 1st line is line 0
Radio Buttons - Number of chosen button - 1st choice is No 0
Cycler      - Number of current choice - 1st choice is No 0
TextIn     - New Text
Text       - New Text
```

example :

---> For a TextIn (xTEXTIN) type gadget in file MyGui.gc, having an ID of 3 :

```
Update MyGui.gc 3 "New Text"
```

---> For a Cycler type gadget..

```
Update MyGui.gc 5 2 ;This will make the 3rd choice the current one.
```

## 1.96 UseTopaz

USETOPAZ (No arguments)

Forces the GUI to use Topaz.font, no matter what gadget fonts or WinFonts are declared.

Use this to override fonts specified which you may not have.

## 1.97 Variables

Variables :

Variables are like small temporary files.

They provide a place where you can store some text, and refer to it whenever you need it, by using the name of the variable.

Gui4Cli has 3 types of Variables :

Normal Variables

- These are the standard Gui4Cli variables (Case Sensitive)

ENV: variables

- ENV: Variables are actually small files in the ENV: ↔  
directory

which is in your Ram disk.

### Internal Variables

- Internal variables are there to provide you with information  
You can read them, but you can NOT set them.

When working with variables you use the following notation :

MyVariable - is the name of a Gui4Cli Variable called MyVariable  
\$MyVariable - is the contents of this Variable

.MyVariable - is the name of an ENV: Variable called .MyVariable  
\$.MyVariable - is its contents

ALL of the parameters of ALL EVENT COMMANDS are translated upon execution,  
This means that if you give the command :

```
SetScreen $ScreenName
... and if the Variable ScreenName contains the words "MyScreen",
SetScreen MyScreen
... will be the command that will be executed.
```

## 1.98 Variables\_Env

ENV: Variables

An ENV: Variable MUST start with a . (full stop character)

```
SetVar .MyVar "This is my env variable"
or
.MyVar = "This is my env variable"
```

will both create a small file in the env directory called ".MyVar"  
and containing the above string.

You can also declare/set an env variable with a double full stop :

```
..MyVar = "This is my env variable"
```

This will create a env: variable by the name of "MyVar"  
i.e. 2 fullstops will result in a variable without full stops.  
This comes in useful when you want access to env: variables  
set by other programs.

Env variables can be used just as if they were normal variables.  
The [start][length] notation however, will not work on them.

Env Variables are much slower than normal variables (although  
quite usable), but since they are actual files which AmigaDOS  
can refer to they are sometimes extremely useful.



## 1.99 Variables\_Int

INTERNAL VARIABLES :

-----

Internal variables do what functions would in other languages. They are there, so you can get various information about the system and the guis running.

You can *\*not\** set internal variables - only read them.

They can be accessed by placing a double dollar sign in front of them :

```
> curdir = $$G4C.DIR
```

for example, will store the name of the current dir Gui4Cli is at, in variable "curdir".

There are many such

Internal Variables  
you can access.

It is very important to understand how internal variables work, before using them extensively. Internal variables always try to show the values of the last thing that happened. You can, if you need these values later, store them into variables, as they are going to change the next time something happens.

Gui4Cli works in "Command Execution Cycles"  
(Sounds much more impressive than it is :)

This means that it sits around waiting for an event to happen, like the clicking of the mouse, a keypress etc..

When such an event happens, if it is declared in the gui it happens in, the commands attached to this event start executing. Gui4Cli may, while executing, jump to subroutines in other files, come back etc and it will finally stop. That's the end of the "command execution cycle" - The DEBUG mode will show this clearly.

All the internal variables below, will (unless otherwise stated) remain valid throughout the life of such a cycle. Thereafter the user might have done anything and you can't be sure that the values are correct.

For example, when a button is hit, the GAD.L etc values will be correct for that button, throughout the life of a command execution cycle (unless you issue an INFO command and thereby change the gadget that Gui4Cli thinks of as "current")

When something else happens, the above values are history. The values returned will be the ones of the new event.

```
example : (file name = test.gc)  
; Slider (gadget No 1) has a width of 100
```

```
xHSlider 10 10 100 15 Title Var 0 100 50 %21d
gadID 1
```

```
; button (gadget No 2) is 200 pixels wide
xButton 10 50 200 50 button2
gadID 2
; here $$GAD.W = 200 i.e. the width of the button which was just clicked
; now we tell Gui4Cli that we want information on the slider
Info GAD test.gc/1
; now $$GAD.W = 100 - i.e. the width of the slider
```

Only the INFO will change the value of the variables.

More on \$\$RETCODE & \$\$REXXRET :

-----  
There are certain commands which set \$\$RETCODE.

This tells you the "return code" of the command, i.e. whether the command succeeded or failed.

If you use a CLI, or SENDREXX command, for example, you can afterwards check \$\$RETCODE, to see if the command was successful. Usually a 0 means success.

Example :

```
CLI 'c:copy Myfile to ram:myfile'
if $$RETCODE = 0
  say "File copied"
else
  say "Error during copy"
endif
```

If you send ARexx commands, there may be a value returned by the program to which you sent the command. If so, another internal variable named \$\$REXXRET will contain this value (RESULT\_2)

## 1.100 Variables\_IntList

### INTERNAL VARIABLES

=====

These are the system internal variables, which are valid all the time.

G4C.PORT	Gui4Cli's port name (normally Gui4Cli)
G4C.VERSION	Gui4Cli version number => version 3.0 = 30
G4C.LOADED	No of guis loaded
G4C.OPEN	number of guis whose window is currently open
G4C.DIR	current directory
G4C.FAIL	current failat level
G4C.DEFSCREEN	The screen name which all guis get (default = '*')

---

G4C.OUTPUT        The default console output spec  
 G4C.FILEPATTERN The default ASL requester file pattern  
 G4C.EDITOR        The default editor name (c:Ed)  
 G4C.GUIPATTERN   The default gui extension (#?.gc - for the requester)  
 G4C.GUILOADDIR   The dir that the G4C load requester will open at

----- \$\$RETCODE and \$\$REXXRET -----  
 RETCODE is set by various commands, to show if the command succeeded or not. Almost always, 0 means ok. Look at the command list to see which commands will set this variable. REXXRET will show what the latest ARExx command returned.

RETCODE            return code from program or command (0 = ok)  
 REXXRET            stuff returned from arexx command (usually nothing)

----- WINDOWS -----  
 These are the values of the currently active window, or of the window you have specifically chosen with the  
       INFO  
       command

WIN.L              current window's left edge in screen  
 WIN.T              top edge  
 WIN.W              external width (i.e. full window)  
 WIN.H              external height  
 WIN.IW             internal width (i.e. inside window)  
 WIN.IH             internal height  
 WIN.SCREEN        name of window's screen  
 WIN.NAME           name of gui file  
 WIN.PATH           the path of the gui file

----- SCREENS -----  
 These values show you the data of the screen that the current gui is (or will) open on.

While the current gui is usually the active one this can be changed with the info command. So, you can see any public screen's data by say, making a dummy gui and setscreening it to the screen you want (don't open it), and then choosing it with the info command.

SCREEN.W           screen width  
 SCREEN.H           screen height  
 SCREEN.BAR        screen's title bar height  
 SCREEN.NAME        screen's name  
 SCREEN.DEPTH      number of bitplanes of screen's bitmap  
 SCREEN.COLORS     number of colors screen can show

----- MOUSE -----  
 These values show you the mouse data when the mouse was last clicked on (while any Gui4Cli window was active).

MOUSE.X            mouse x position in screen  
 MOUSE.Y            mouse y position

---

MOUSE.WX            x position inside currently active window  
 MOUSE.WY            y position inside currently active window  
 MOUSE.COLOR        Color number of pixel at mouse position

----- ALL GADGETS -----

Made current upon being clicked upon or by being chosen with the INFO command. These variables are valid for all types of gadgets.

GAD.L                gadget left edge in window  
 GAD.T                gadget top edge  
 GAD.W                gadget width  
 GAD.H                gadget height  
 GAD.GUI              gui file name gadget resides in  
 GAD.TYPE             type of gadget (i.e. XBUTTON, XLISTVIEW etc)  
 GAD.ID                gadget's ID  
 GAD.VAR              the name of the variable the gadget will store it's value in.  
 GAD.VALUE            gadget's value (varies - same value as you would  
                          give with the update command)

----- PALETTES -----

A palette is made current by being clicked on or chosen with info.

COLOR.R              Palette's chosen color - red value  
 COLOR.G              green  
 COLOR.B              blue  
 COLOR.NUMBER        chosen color number  
 COLOR.TOTAL         total number of colors

----- LISTVIEWS -----

Listviews are special. They are made current upon being clicked on, or upon being specifically chosen with the LVUSE command.

All types of Listviews have lines. The vars \$\$LV.LINE, \$\$LV.TYPE and \$\$LV.REC, refer to the CURRENT LINE, which is the line that was last clicked on or the line that was last acted upon with an LVMULTI, LVSEARCH, LVFIND or LVGO command.

Each Listview keeps track of it's own current line, so you can have many listviews and use the LVUSE command to switch between them. The current line of each listview will be remembered when you get back to it (with an LvMulti-next command, for example).

LV.LINE                current line No (or "" for no current line)  
 LV.TYPE                type of current line (DEV,VOL,ASN,DIR,FILE,NONE)  
 LV.REC                content of current line  
 LV.SEL                = 1 if line is selected, 0 if not  
 LV.MODE                LV mode (NUM,TXT,MULTI,DIR)  
 LV.DIR                directory LV - current dir  
 LV.TOTAL              total number of lines in LV  
 LV.GUI                the LV's gui name  
 LV.ID                 the LV's ID

----- IMAGES -----

Images can not "happen" so they become current \*only\* by being chosen with the info command.

```
IMAGE.W           current image's width
IMAGE.H           height
IMAGE.DEPTH       depth in bitplanes
IMAGE.COLORS      number of colors (as per bitplanes)
```

----- PARSEVAR -----  
 These variables become valid after calling the ParseVar command. They will remain valid throughout the life of G4C, until an other parsevar command is issued.

```
PARSE.n           ex: $$PARSE.0, $$PARSE.1 etc, are words No 0, 1,
                  etc respectively. Note the first word is No 0.
PARSE.TOTAL       The total number of words parsed. (Maximum is 30)
```

----- KEY CODES -----  
 These are the codes & qualifiers of the last key pressed. You can learn what code each key has, by setting DEBUG ON (in the new debug mode) and pressing any key.

```
RAWKEY.CODE       code of last raw key (arrow, help F1-F10) pressed
RAWKEY.QUAL       it's qualifier (i.e. if key was shifted, alt etc)

VANKEY.CODE       code of last vanilla key pressed (except control-key)
VANKEY.QUAL       it's qualifier
VANKEY.LETTER     the letter of the key pressed
```

#### ----- OBSOLETE VARIABLES

These are the old variables names which you shouldn't use anymore, although they still work, for compatibility. They have their new equivalents.

```
LOADED_GUI
OPEN_GUI
LASTLV
CURRENT_DIR
```

## 1.101 Variables\_Normal

### NORMAL VARIABLES :

=====

Normal variables are the most common in Gui4Cli. They are the ones you can declare with a simple statement like :

```
myvar = 3           - or
myvar = 'This is my variable' - or
```

```
SetVar myvar 'This is my variable'
```

You can thereafter refer to their contents by putting a \$ sign in front :

```
> say '$myvar'
> should print out : This is my variable
```

```
*** Important : Normal variables are CASE SENSITIVE
(i.e. myvar is not the same as MyVar)
```

Note that you can hit CONTROL-V to get a full listing of all variables currently declared by all guis loaded. Furthermore, you can "Set Debug On" to see when and how variables are set or read while the guis are executing commands.

#### NORMAL VARIABLE TYPES:

```
=====
```

Normal variables are kept in lists.  
Each GUI has such a list, where it keeps all it's variables.

You can get/set the variables of your gui's list, by just declaring:

```
> MyVar = "Something"
```

If you want to get/set another Gui's variables, you can do this:

```
> OtherGui.gc/MyVar = "SomethingElse"
This will set variable "MyVar" of Gui "OtherGui.gc" to "SomethingElse"
```

Similarly, to get the results, you would do:

```
Say $MyVar          --> would print: "Something"
Say $OtherGui.gc/MyVar --> would print: "SomethingElse"
```

#### GLOBAL Variables :

```
-----
```

There is also a "GLOBAL" variable list, which is always present and which can be accessed from any gui, anytime, by putting a '\*' character in front of the name.

So, to set a Global variable, you would do :

```
> *MyGlobalVar = "Something"
and to read it:
> Say $*MyGlobalVar
```

Global variables are usefull since they are accessible from everywhere. I.e. you do not have to rely on a certain gui being loaded to have them available.

#### The VARPATH command :

```
-----
```

If you are building a multi-gui application you can use the

```
VarPath
command, with which you can declare a "Variable Search Path" and ↔
in
```

effect "merge" the variables of many guis.

#### FULL VARIABLE NOTATION :

=====

Although for normal use you just declare the variable name as above, normal variables may have the following full template :

```
gui/name[+-start][+-length]
```

and Global variables

```
*name[+-start][+-length]
```

In all the above, only the name is required. All others are optional.

#### THE RULES OF THE GAME :

-----

- As we said, the 'gui' part, is the name of the Gui that the variable belongs to. If no 'gui' name is given the variable is automatically assumed to belong to the gui it resides in.
- [start] is the character of the variable you want to start at.
- [length] is how many characters to consider
- If you give a [start] which is more than the length of the variable you'll get "".
- If you give a length that will pass the end of the variable, then only the actual length of the variable will be considered. Variables will *not* be expanded to accommodate [start] or [length].
- If the [start] is omitted it means '0' i.e. from the start of the var.
- If the [length] is omitted it means the rest of the variable.
- If [start] is negative it means from the *end* of variable.
- If [length] is negative it means "count from right to left"

#### EXAMPLES :

-----

```
var = '0123456789'
res = $var[3][3]
```

res will now be = '345'

If these variables were in an other file called "mygui" and you wanted to refer to them, you would do :

```
mygui/var = '0123456789'
res = $mygui/var[3][4]
```

now res will equal '3456'  
or if 'var' was a global variable:

```
*var = '0123456789'
res = ${var}[3][4]
```

ok, up to now ?..

```
res = $var[3]
```

res is now '3456789' since we did not give a length.

```
res = $var[3][200000000]
```

res is still '3456789' since res will not be expanded.  
And what's more, it works both ways:

```
var[3][2] = 'xx'
or
var[3][2] = 'xxxxxxxxxxxxxxxxxxxxxx'
```

will both set var to: '012xx56789'

More stuff :

```
var[-1][1]      ; is the last character of a variable
var[-3][3]      ; are the last 3 characters of a variable
var[3][-3]      ; are the first 3 characters of a variable
var[0][3]       ; are, again, the first 3 chars
```

and so on...

REALLY ESOTERIC STUFF :

-----

- In the "Gui/Name[start][length]" notation, for internal, pain-in-the-ass related reasons, the "Gui" and "Name" parts can *not* themselves be variables. (There are ways of circumventing this if needed - see below).

However, you can use variables for [start] and [length] :

```
var = "01234556789"
c = 0
while $c < 10
  say '$var[$c][$c]\n'
  c == $c + 1
endwhile
```

Important : the variables used for [start][length] *must* be simple direct variables - i.e. they can not contain other variables in them.

If you need to have a variable name made up of other variables, you can use an intermediary variable to construct the full name, like this:

```
; Say you want to have the 'Gui' part of the name as a variable..
gui = MyGui.gc
temp = "\$$gui/MyVar" ; will store "$MyGui.gc/MyVar" in temp
say "$temp"          ; will translate temp
```



; (as variable within variable)

## 1.102 VarPath

VARPATH VariableSearchPath

This is a very important command, since it allows you to "Merge" the variables of many guis together. It is however, a little complicated and should not be used if you don't understand it.

First of all, read the section on  
Normal Variables  
to understand it.

Now.. If you're making an application which will have many guis, you may want to "Merge" their variables, so you don't have to state the GUI part of the name each time, or remember in which of your guis a certain variable was declared

```
> VarPath "gui1/gui2"
```

will search your gui's variables (always done first) and if not found will continue and search the vars of gui1 then gui2

So if you declare :

```
MyVar = 5
```

- Gui4Cli will search your gui for variable MyVar.
- If not found, it will search gui1, then gui2.
- If it finds a variable by that name, it will set it to 5.
- If it doesn't find it, it will create a new variable for \*your\* gui and set it to 5

Similarly, in reading a variable:

```
say $MyVar
```

- Will search your gui, then the varpath, and if not found will return with "" (nothing)

You can also have the GLOBAL variable list in your varpath. This is denoted with the '\*' character :

```
> VarPath "gui1/*/gui2/gui3"
```

- will search your gui first (always), then gui1, then the global variable list, then gui2 then gui3.

The normal way to use Private vars in a multi-gui application, would be to declare VARPATH with all the names of the guis the application uses after it and access the global variables (if needed) using the \*MyVar system.

You don't have to declare the name of \*your\* gui in the varpath. This will ALWAYS be checked first. If you do declare it, then it will be searched twice.

If the variable is not found anywhere in the varpath and needs to be created, then it will be added to \*your\* gui's variables. (Unless specifically declared, as in - MyOtherGui.gc/MyVar = 5)

### 1.103 Wait

```

        WAIT
SYSTEM
        Name Timeout

```

WAIT will wait until the specified SYSTEM items (FILE, PORT etc) are ready, or until the timeout is reached.

If the item are already ready, the command will return immediately.

This command will set the \$\$RETCODE internal variable to show you if the given item appeared within the TimeOut limit you specified. 0 = OK.

This command is useful when starting up programs, or when writing to files. It does the same job as the IFEXISTS command, but in a different way.

The Timeout is in ticks..

example :

```
WAIT    PORT Gui4Cli 100
```

- wait for a public message port called "Gui4Cli". If it hasn't appeared in 100 ticks, return, setting \$\$RETCODE to an error status (i.e. over 0).

### 1.104 While

```

        WHILE      Argument
OPERATOR
        Argument
ENDWHILE
AND, OR

```

You use this command as follows :

```
var = 1           ; set var equal to 1
WHILE $var < 10  ; while var is less than 10
  say '$var\n'   ; print the current value of var
  var == $var + 1 ; add 1 to var
ENDWHILE         ; go back to WHILE and do it again,
                 ; until $var is 10
```

- this will print out numbers 1-9

The arguments and operators have the same meaning as for IF, so see it for explanation. OR and AND can be attached to WHILE, just like to IF.

You can have While within While, up to 126 levels.

A while statement is often called a "LOOP" because it just keeps looping and re-doing the code in between the while/endwhile until the condition you set in the while statement has been satisfied.

This is *\*very\** useful in many situations, so learn how to use it!

## 1.105 WinBackGround

```
WINBACKGROUND SOLID|PATTERN|ICON|IMAGE APen|IconName BPen
```

This command allows you to have nice backgrounds for your window and you can thank Michael Van Elst for giving me the needed backfill hook code for it.

The background used can be :

```
SOLID : A Solid color which will be of color APen.
- ex : WinBackground SOLID 3 0
Here the last argument (0) is disregarded.
```

```
PATTERN : A Hash pattern made up of 2 colors.
- ex : WinBackground PATTERN 3 1
Here, the background is a pattern made up of colors No 3 and 1
```

```
ICON : An Icon which will be tiled to the background of your window.
- ex : WinBackground ICON DF0:MyIcon 0
Here, again, the last argument is disregarded.
The 2nd argument is the full path & name of the icon, without
the .info extension.
```

```
IMAGE : An image, which *MUST* have already been loaded with the
LOADIMAGE command, will be tiled to your windows background.
- ex : WinBackground IMAGE ImageAlias(name) 0
```

NOTE :

The GadTools library was not designed to be used with colored backgrounds, so some gadgets (such as listviews or text-in gadgets) have parts of them which should (if this was a perfect world) have been blanked out by GadTools, instead filled in with the Background you give.

It's not much of a problem, but let's hope it changes..

## 1.106 WinBig

WinBig L T W H Title

Defines a window at Left(L), Top(T), Width(W), Height(H)  
Title is a string that will be shown in the TitleBar.

If Left is -1, the window is centered on the width of the screen  
If Top is -1, the window is centered on the height of the screen.

If Width is negative, then the width of the window will be the size of the screen it opens on, minus this negative number. Same for the Height.

Example : WinBig -1 -1 -1 -10 "My Window"

This is a window centered horizontally and vertically on the screen, of a size -> Width = ScreenWidth-1, and Height = ScreenHeight-10.

If Width or Height are more than the screen size, they are reduced so that the window fits in the screen.

Windows always have the following 4 gadgets, unless told otherwise (by the WinType command) :

- The close button - Closes the window
- The Drag Bar - This consists of all parts of the window's Title bar not taken up by other gadgets.
- The zoom button - Zips the window to the size defined by the WinSmall command, or, if this command is not defined to size 100, 11 at 0, 11.
- The Depth arrangement gadget

Window Types can be changed with the "WinType" command, with which you can define which of the above gadgets you want and make the window borderless and/or backdrop and/or resizable.

All windows have a number of  
Default Shortcuts

## 1.107 WinFont

```
WINFONT  FontName  Size  UL|BD|IT
```

If you do not specify any gadget fonts (with the GADFONT command), then the default font (the Screen's font) will be used.

You may change this, by declaring a global default font (for this GUI) with this command.

FontName - The name of the font e.g. Topaz.font

Size - The size of the font. If the size is not found, the diskfont library will scale the font to the size requested, but this will not look good.

UL|BD|IT - A Mask (see WinType command) specifying whether the font will be UL (Underlined), BD (Bold) or IT (Italic)  
010 for example is a Bold font.

## 1.108 WinOnMouse

```
WinOnMouse  LeftOffset  TopOffset
```

This command instructs the window to disregard the Left and Top offsets given in the WinBig command and instead open relative to the current mouse position.

LeftOffset & TopOffset are the offset from the current mouse position that the window's Left & Top will be.

This is a useful command for making small borderless sub-windows.

## 1.109 WinOnWin

```
WinOnWin  GuiName  LeftOffset  TopOffset
```

This command instructs the window to disregard the Left and Top offsets given in the WinBig command and instead open relative to a "GuiName's" window.

GuiName : The name of the Gui File whose window will be used  
LeftOffset : Left Offset from the GuiName window position  
TopOffset : Top Offset from GuiName window position

## 1.110 WinOut

WinOut "ConsoleSpecification"

Specifies a console window to be used to send any output of CLI commands that are called by the program (if any). It is best NOT to use this command, unless you must, since each new console that you specify with WinOut, takes up extra memory.

If there is output, and no console window has been specified, the default window will be used.

Note that only the CLI command gets a window to output in. RUN does not get a window.

If Gui4Cli is started from a shell, the shell itself is the default window. If you give the command Run >NIL: (or >NIL: <NIL:) Gui4Cli, there is no default window!

If run from workbench, Gui4Cli will define a default window, and output, if any, will go there. The default window is :

```
"con:0/12/450/80/Gui4Cli Output/AUTO/CLOSE/WAIT"
```

You may change this with the TOOLTYPE/CLI options or the SET command.

Gui4Cli messages (error reports etc) will be output to the default window.

## 1.111 WinShortcuts

All windows have the following keyboard commands :

Use them when the window is active by pressing the CTRL key and the letter, together.

Control J - Will cause the active GUI to jump to the next available Public Screen, if any. An event - xONJUMP NewScreenName - will be called every time this happens (if defined), placing the name of the new screen in the above variable so that you can SetScreen any other accompanying GUIs to this new screen also.

Control W - This shortcut will bring up a requester, asking you whether to save the current window size as the default SMALL or NORMAL (Big) window size. Thereafter, every time the GUI is loaded, it will be automatically resized to these sizes. (see also the info on Resizing)

Control V - Will print out a list of all the variables currently in use.

Control E - Edit current window : Starts the Editor specified in the Tool Types or C:Ed, passing it the File name of the Window Program as argument.

Control R - Reload current window : Quits current window and reloads it. Very useful for seeing where your gadgets are and moving them around till you get them right.  
xOnQuit, xOnOpen etc commands are not executed.

Control Q - Quits current window.

Control C - Closes current window.

Control N - Activates next open window and brings it to the front.

Control F - Brings current window to the Front

Control B - Pushes current window to the back

Control S - Opens the Status requester which lets you Quit Gui4Cli or Load, Reload, Unload, Open or Close any of the windows that you may have loaded.

ESCAPE - This is the PANIC! button. It will stop whatever gui4cli is doing at the time (but not any programs that it may have launched via the cli, run, launch etc commands..)

Visual editing shortcuts :

-----

Control G - Will save the gui just as it is at present, with the gadgets/graphics at their current positions. Be sure to read the Visual Editing section before trying this.

Control H - Will select/place a gadget or graphic under the current cursor position. Use this in cases where Control-MouseClick doesn't work.

## 1.112 WinSmall

WinSmall L T W H

Specifies the position and size of the window, when the user clicks on the window's zoom gadget - you don't have to declare it.

L T W H are the sizes - there are the following weird values you can also use.

Left => if -1 then the small window will be placed flush to the right side of the screen.

Top => if -1 it will be placed under the screen bar  
if -2 it will be placed at the bottom of the screen

Width => if -1 the window will be wide enough to fit the title.

Height => if < 0 the height will be the size of the Screen title bar.  
This size is also the minimum size it will ever be.

The default size, if you do not declare a winsmall command, is :

0, -1, 150, -1

## 1.113 WinType

WinType MASK

WinType lets you specify all the gadgets in the window.

The MASK argument is a series of 8 numbers which can be 1 or 0

Each number has the following meaning ( 1=Yes, 0=No )

1st : Close Gadget  
2nd : DragBar  
3rd : Zoom Gadget  
4th : Depth arrangement Gadget  
5th : Borderless  
6th : Backdrop  
7th : Right Resize bar - makes window resizable  
8th : Bottom Resize bar - makes window resizable

example :

WinType 11110001

- This is a standard window with Close, DragBar, Zoom and Depth gadgets since the first 4 numbers are "1".
- It is not Borderless or Backdrop, since the next 2 numbers are 0s.
- It is resizable, having the resize gadget on the bottom border, since the 7th number is 0 and the 8th is 1.

Note that if you want a borderless window, you should give "" as the window title, otherwise the title (and part of the window bar) will be shown.

## 1.114 xAppIcon

xAPPICON L T IconName Title Variable ON|OFF

This command defines an AppIcon on the workbench.

L and T are the position that the appicon will appear at on the workbench.  
Negative values, allow the Workbench to decide where to put them.

---



IconName : The name of the icon to be used (no .info extension)

Title : The Title that will appear beneath the icon

Variable : The name of the variable where the names of any files whose icons were selected when the user clicked on the AppIcon, will be placed.

ON|OFF : The starting state of the Gadget. If OFF, it will not appear if ON, it will.  
You can declare a GadID for this event and SetGad it ON or OFF at any time.

### 1.115 xAppMenu

xAPPMENU AppMenuName Variable ON|OFF

This Event will place an AppMenuItem at the Tools Menu of the workbench.

The Variable is where the names of any files whose icons were selected when the user chose the AppMenuItem, will be placed.

ON or OFF is the starting state of the menu item. If it is OFF, it will not be shown.

You can give an ID number (see the GadID command) to this event and SetGad it ON or OFF whenever you want.

### 1.116 xAppWindow

xAPPWINDOW Variable

Variable : The name of the variable where the names of any files whose icons were dropped into the AppWindow will be placed.

When you declare this event, you also make the window an AppWindow. No other action is needed.

An AppWindow can not (currently) be switched ON or OFF.

Only one xAppWindow event can be declared per GUI. If you declare 2, the 2nd one will be disregarded.

### 1.117 xArea

---

```
xAREA L T W H COMP|BOX|NONE
```

This is a area hit gadget, like an invisible Button.

Nothing will be shown unless the gadget is hit, whereupon it will

```
COMP = complement the area, or  
BOX   = draw a box around it, or  
NONE  = leave it alone.
```

This event is useful, if you want to make something like an image map, for example.

## 1.118 xButton

```
xBUTTON L T W H Title
```

Specifies a Button Type gadget, at (L)eft, (T)op and of size (W)idth, (H)eight with the "Title" text on it.

Example :

```
xButton 10 20 100 15 "My Button"
```

Buttons are buttons - you just press them and they do their commands  
Icons (xICON) can be used in exactly the same way.

## 1.119 xCheckBox

```
xCheckBox L T W H Title Variable OnText OffText ON|OFF
```

Specifies a checkbox type gadget at Left, Top, sized Width Height.

The Title will be placed next to the gadget.

Variable is the name of the variable that will hold the value.

Checkbox type gadgets have two states : Checked (ON) & Unchecked (OFF)

If the gadget is ON, OnText will be placed in Variable, otherwise, OffText.

ON|OFF specify the starting state of the gadget, when the window first opens.

Example :

```
xCheckBox 30 15 16 16 "CheckBox" ChkVar "Box On" "Box Off" ON
```

Here, if the user checks the gadget, then the variable ChkVar will contain the words "Box On"

\*\*\*\* NEW :

---

CheckBox gadgets are now resizable !!!  
 So the width and height you specify HAVE meaning.  
 The default size that Commodore eh... Escom eh... VisCorp..  
 suggests, is width=26, height=11 (for HighRes non-interlace)

If this means that you have to change all your GUIs... ehm.. sorry :-)

## 1.120 xCycler

```
xCYCLER L T W H Title Variable
```

This is a cycler at L, T of Width, Height. It's value will be placed in Variable. The Title (if any) will be placed to the left.

You MUST tell a cycler what fields you want it to have, i.e. what will be written on it's face every time the user clicks it.

You do this by attaching to it CSTR (Cycler-String) commands, as follows :

```
xCYCLER L T W H Title Variable
CSTR Title Value
CSTR Title Value
.. etc .., up to 12 CSTRs
```

The Title of the CSTR is what will be placed on the face of the button and the value is what will go into the Variable.

The 1st field in cyclers & radio buttons is No 0.

## 1.121 xIcon

```
xICON L T IconName (no .info)
```

Icons act like buttons. Although they take up more space than the respective brush would take, they are very easy to implement.

The (L)eft and (T)op are where the icon will be placed and IconName is the full path of the icon file you want to use, without the ".info" extension

## 1.122 xLVHook

Listview Hooks :

```
xLVDirHook HookID/GadID
xLVHook HookID/GadID
```

These two events act according to what happens to the ListView which has the same "HookID".

The HookID of a listview is the same as it's GadID number, unless you want to be picky and change it with the

LVDirHook

gadget

modifier (You no longer need to use the LVDirHook modifier - just use the GadID instead - it's better).

xLVDirHook - will "Happen" if the Dir ListView that has the same HookID changes directories (by the user double clicking on it). When this happens, you can, for example, refresh the place where you display the path the dir-listview is currently at.

xLVHook - will happen if the MULTI or DIR listview that has the same HookID is clicked once (otherwise, the LV itself will only happen if the user double-clicks on it)

The \$\$LV\_DIR, \$\$LV\_GUI, and \$\$LV\_ID

Internal Variables

will

contain the appropriate values.

ex :

```
; This is a DIR listview with an ID of 1
xListview 0 10 200 100 '' Variable sys: 0 DIR
GadID 1
```

```
; This is it's dirhook - happens whenever a dir is double-clicked
xLVDirHook 1
SetWinTitle MyGui '$$LV.DIR'
```

```
; This is it's simple-hook - happens whenever the list is single-clicked
xLVHook 1
Say 'File chosen is $Variable\n'
```

## 1.123 xMenu

```
xMENU MenuName ItemName SubItemName Shortcut
```

This is a menu.

```
MenuName      : The Menu Titles shown in the screen bar.
ItemName      : An Item in the above Menu
SubItemName   : A SubItem in the above Item of the above Menu.
ShortCut      : A letter to be used as a short cut (Amiga Key + Letter)
```

You MUST define a MenuName and an ItemName in each xMENU command.

If there is no SuItemName declared, i.e. if you write "" (empty string) then the Event is a MenuItem.

If you give a SubItemName, then the Event is a SubItem. In this case, if there is no ItemName by that name, it will be created.

You may give a "" for shortcut also.

\*\*\* NEW Wonderful stuff :

You can now have icons in your menus! (isn't gadtools great ?)  
All you have to do is declare the ItemName or SubItemName you want to be an icon by preceding it with a # character.

example :

```
xMENU Project #MyIcon "" ""
- will put the icon named MyIcon.info in the Project menu.
```

Note that the MenuName can NOT be an icon.

Menus may be a little confusing at first, till you get it. Try them out in all combinations and see what happens.

## 1.124 xOnJump

xOnJump Variable

With V2.1 you can press CONTROL-J on a GUI and this will cause it to jump to the next available Public Screen.

When this happens, this event will be executed, if defined.  
The Variable will contain the name of the new screen.

This allows you to SetScreen any other accompanying GUIs to the same screen.

## 1.125 xOnKey

xONKEY #R or #V or Letter or #KeyValue

Commands attached to this event will be executed when the specified key is pressed.

You may specify any of the following :

```
xOnKey L      do commands on pressing of the letter L
              (or any letter)
```

```
xOnKey #13   do commands on hitting ENTER - or other values
              (13 is the ASCII value of the ENTER key)
```

xONKEY #R       do commands upon the pressing of any raw key  
                  (raw keys are the function keys, the arrows, help etc)

xONKEY #V       do commands upon the pressing of any "normal" or  
                  as they say, "vanilla" key.

Raw keys are the arrows, help & function keys - all others are called "Vanilla" keys (God knows why..)

The internal vars \$\$RAWKEY.CODE, \$\$RAWKEY.QUAL, \$\$VANKEY.CODE, \$\$VANKEY.QUAL, \$\$VANKEY.LETTER, will contain the relevant key codes and qualifiers (qualifiers are the shift, alt etc keys).

This is useful for the #R and #V commands, which will "happen" whenever \*any\* key of their type is pressed. You can then see what the key pressed was (by looking at the internal vars) and react accordingly.

To find out what code number each key has, use SET DEBUG ON, to enter the debug mode and while in it, the values of any keys pressed will printed out.

(see also the GadKey command)

## 1.126 xOnLoad\_etc

xONLOAD etc

These are very important events, which take place :

xOnLOAD       -> upon loading of the GUI  
xOnOPEN       -> upon opening of the GUI's window  
xOnCLOSE      -> upon Closing of the GUI's window (with GuiClose or Close gadget)  
xOnQUIT       -> upon Quitting of the GUI

xONRELOAD     -> When you call GuiLoad, if the file you want loaded is already loaded, then nothing will happen.  
              You can declare this event if you want something to happen.

For a simple gui you may have something like the following :

```
xOnLoad
  MyVar = something ; set any variables you need to set first..
  SetGad MyGui.gc 1 OFF ; set any gadgets on/off/show/hide etc
  GuiOpen MyGui.gc ; to open the gui window upon loading
```

```
xOnClose
  GuiQuit MyGui.gc ; to quit (unload) the gui upon the closing of
                   ; it's window.
```

The above is only an example. You may have many commands, actions etc upon the happening of any of the above events.

These events (apart from xOnReload) can also take ARGUMENTS

These arguments can be defined in the event's command line:

```
> xONLOAD arg1 arg2... arg6
```

This is specially usefull in xOnLoad, as it allows you to make small guis and use them as "functions". Examples of such fuctions can be found in the GUIs:tools/rtn directory.

## 1.127 xOnReturn

xONRETURN LaunchID

This event will be executed when a LAUNCH

command with

the same LaunchID as this event, has finished executing.

## 1.128 xOn\_Various

OTHER Events :

-----

These are various events (things that the user may do), for which commands may be declared.

The commands attached to these events are executed every time the given event happens. No arguments are passed.

xOnRMB -> upon clicking of the Right Mouse Button - Only works if there are no menus declared.

xOnACTIVE -> When the window is activated by the user

xOnINACTIVE -> When the window is made inactive

xONDOUBLECLICK -> Upon the double-clicking of the window, on an area not taken up by a gadget. - If this event is not declared, the default is to bring the window to front.

There are also the following events :

xONFAIL -> The commands of this event are executed if the return code of a program launched by a CLI command, is equal to or exceeds 10, or any other ErrorNumber you have set with

the FAILAT command.

- xBEFORE -> Attach commands to be executed before every other execution starts. This will be called every time a gadget is clicked on, and be executed before the gadget's commands. It does not apply to the above events (xOnLOAD etc)
- xAFTER -> Same as xBEFORE, but these commands are executed after the Gadget commands.
- xONDISKIN -> Happens whenever a disk is inserted (currently there is no way to find out \*which\* disk was inserted, except by checking with the \$\$DISK.xxx internal variables)
- xONDISKOUT -> Whenever a disk is removed.
- xONFART -> not yet implemented...

## 1.129 xPalette

xPALETTE L T W H

This is a palette gadget from which you can choose colors.

Every time a color is selected, the palette will "happen" and the internal variables COLOR.R, COLOR.G, COLOR.B, COLOR.NUM, COLOR.TOTAL will contain the R G B values, the color number and the total number of colors.

Look at the Internal Variables for more info.

The number of colors is taken from the screen that the window with the palette gadget will open on.

You can update a palette by choosing another color :  
> update GuiName gadID NewColorNumber

Look at the guis:tools/palette.gc gui to see an example

## 1.130 xPipe

xPIPE PipeName ON|OFF

This is the event that will enable you to use the output that cli programs produce.

A PIPE: is an amigados device which exists in all versions of the OS from V2.0 and up (AFAIK). In order to be able to

---



use it you MUST have the file PIPE in your DEVS/DOSDRIVERS directory and the Queue-handler in your L: dir. You probably already have them set up as above, as this is the standard Amiga OS set-up.

What a pipe: does, is provide a "holding place" for the output of CLI programs, until Gui4Cli can get around to reading it.

PipeName - should be the name of a pipe: file  
 ON|OFF - is the starting state of the pipe (You can give the xPIPE a gadid and set it ON|OFF - or CLEAN it.

```
xPIPE PIPE:AnyName ON
```

Thereafter, you can launch/run/cli any command and redirect it's output to this pipe: file, like so:

```
RUN 'c:search >PIPE:AnyName dh0:MirrorOnWall FairestOfAll'
```

Now, if the c:search program produces any output, this will go to the PIPE: file you stated and the above xPIPE event will "happen".

Inside the xPIPE event, there is one internal variable which will be valid : \$\$PIPE.TXT. Every time the xPIPE event happens it will mean that an other line of text is ready for you and stored into this variable.

----- example:

```
; a button to start the search
xButton 10 10 60 12 'Search'
  run 'c:Search >PIPE:SearchResults dh0:Universe TrueLove'

; the pipe event that the results will be sent to
xPIPE PIPE:SearchResults ON
  say '$$PIPE.TXT\n'
```

In the above example, every time a line of text is output from the c:search command, it will be printed.

----- Handling a pipe -----

You can start off a gui with the pipe ON or OFF.

If you give the pipe event a GADID, you can then switch it ON or OFF or CLEAN it, at any time. These will have the following effect :

```
SetGad gui id ON      :
  Will start (create) the pipe.
```

```
SetGad gui id OFF    :
  Will continue sending text to the xPIPE event and when all
  the text has been sent, it will delete the pipe.
```

SetGad gui id CLEAN :

Will immediately stop sending text to the xPIPE event, clean up any text that was in the pipe and go back to waiting for some text to be sent to it. It will not quit the pipe.

If you want to immediately stop and quit the pipe, you can declare "SetGad gui id CLEAN" and then again "SetGad gui id OFF".

----- \*\* IMPORTANT \*\* -----

In order to implement the PIPE:, Gui4Cli will start a new process which will sit quietly in the background and monitor the PIPE: file. If any text comes, it will read it and send it to Gui4Cli line by line.

That means there is quite a lot of overhead involved in each xPIPE you declare (about 10k), so don't fill the place with them..

YOU MAY WELL ask, "why should I use a pipe and not a simple file in ram: to redirect stuff to and then read it from there.."

- Well, a pipe: is actually a file, but one which you can read while an other program may be writting to it (i.e. outputing text in to the pipe).

- Moreover, a pipe will notify you when it has something to tell you, where as with a file you have to go check yourself.

- (Unless I add an xOnNotify event - which I'm thiiiiis much away from persuading myself to do...)

## 1.131 xRadio

xRADIO L T W H Variable Spacing

xRadios are similar to cyclers, in that they have up to 12 fields. These are denoted with RSTR :

```
xRADIO L T W H Variable Spacing
RSTR Title Value
RSTR Title Value
.. etc .., up to 12 CSTRs
```

The spacing is the distance that the radio buttons will have from each other (vertically), in pixels.

\*\*\*\* NEW :

Radio Buttons now have WIDTH & HEIGHT, which are resizable. Sorry if you have used them in your GUIs.. just correct the syntax

People who know a lot more than me and you about this, suggest that the size for HiRes non-interlace screens, should be :  
Width = 17, Height = 9

## 1.132 xRoutine

```
xROUTINE RoutineName
```

This is a "procedure" or "gosub" in other languages. The commands attached to this event will never be executed, unless called by another event.

Say you define an xROUTINE with the name of "Hello" :

```
xROUTINE Hello
```

Then from another event in the program you can call this routine with :

```
GoSub GuiName Hello
```

The commands attached to xROUTINE "Hello" will be executed and when finished, control will be returned to the command after "Gosub GuiName Hello"

Starting with V3.2, Routines can take/return

Arguments

Arguments may be accesses as internal variables \$\$ARG.0 to \$\$ARG ←

.5, or

even better, defined in the routine's definition line :

```
> XROUTINE MyRoutine Arg1 Arg2 SomeArgName Arg6
```

## 1.133 xSlider

```
xVSlider L T W H Title Variable Min Max Current ShowStr
```

```
xHSlider L T W H Title Variable Min Max Current ShowStr
```

These two gadgets are Sliders :

xVSlider is a vertical slider and xHSlider a horizontal one.

L T W H Title and Variable have the same meaning as other gadgets.

Min : The minimum value of the slider

Max : The maximum Value

Current : The starting Value of the slider

ShowStr : This is a printf style string specifying the format of the value to be shown next to the slider :

You can specify "%3d" or "%5d Widgets" for example.

The simplest is to specify "%5d" (the Number 5 means

the maximum length of the number that the slider can show).

The Title will be placed to the left of the gadget unless told otherwise by the GadTitle command

## 1.134 xTextIn

```
xTEXTIN L T W H Title Variable StartingText Bufflength
```

This is one of these little boxes where you can input text or numbers. L T W H is the size of the box & Title will be placed to the left.

StartingText is the text the window will open with.

Bufflength is the size of the buffer :

A "buffer" is a place where the program can put the things you write into this gadget. Since the program does not know how long the text you write into this gadget will be, you have to tell it beforehand.

A buffer of size 100 means that it is 100 characters long.

The gadget will not allow you to enter more characters.

Used to be that this gadget would not realize that you had entered anything, unless and until you pressed the ENTER key.

Through excellent programming on my part, this is all ancient history now.. :)

## 1.135 Index of ram:Gui4Cli.guide

Table of Contents:

Action

Advantages

All\_the\_Commands

Append

AppVar

Assign

BreakTask

CalcVar

CD

---

---

ChangeArg  
ChangeGad  
ChangeIcon  
Command\_Lines  
Copy  
Counter  
CutVar  
c\_Gui  
Delay  
Delete  
DelVar  
DirList  
DoCase  
Extract  
EZReq  
FailAt  
Fonts  
Font\_Sensing  
GadFont  
GadHelp  
GadID  
GadKey  
GadTitle  
GadTxt  
Gauge  
GoSub  
Graphics  
GuiEdit

---

---

GuiLoad  
GuiRename  
GuiScreen  
GuiWindow  
If  
IfExists  
Images  
Important\_Topics  
Info  
Introduction  
JoinFile  
KillScreen  
Launch  
Licence  
ListView  
LTWH  
LVAction  
LVDir  
LVHook  
LVMulti  
LV\_Commands  
MakeDir  
MakeScreen  
Mark  
NewFile  
Operation  
Operator  
Options  
ParseVar

---

---

PartRedraw  
Passing\_Args  
Paths  
Programming  
Quit  
Redraw  
Rename  
ReqFile  
ReSize  
Run  
Say  
Screen  
SendRexx  
Set  
SetColor  
SetGad  
SetGadValues  
SetScreen  
SetStack  
SetVar  
SetWintitle  
Status  
Stop  
System  
Text  
Translation  
TTGet  
Update

---

---

UseTopaz  
Variables  
Variables\_Env  
Variables\_Int  
Variables\_IntList  
Variables\_Normal  
VarPath  
Wait  
While  
WinBackGround  
WinBig  
WinFont  
WinOnMouse  
WinOnWin  
WinOut  
WinShortcuts  
WinSmall  
WinType  
xAppIcon  
xAppMenu  
xAppWindow  
xArea  
xButton  
xCheckBox  
xCycler  
xIcon  
xLVHook  
xMenu  
xOnJump

---



xOnKey

xOnLoad\_etc

xOnReturn

xOn\_Various

xPalette

xPipe

xRadio

xRoutine

xSlider

xTextIn

---