in

Conversion program

COLLABORATORS

	TITLE : in		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Conversion program	November 20, 2024	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

in

1.1 The Front End

```
                           RIBlitzLibs v4.0
                           ================

                     Blitz Basic 2 Command libraries

                    (c)1994/95 Leading Edge Software

                         Libraries created by:
                             Steven Matty
                                  &
                           Stephen McNamara

                        Using Blitz Basic 2 by
                             Acid Software

                      *AmigaGuide Command Database*

                  Doc >>> AmigaGuide conversion program:
                           Stephen McNamara
```

1.2 RIAmosFuncLib

```
-----------------------------------------------------------------------------
====              RI AMOS Function Library V1.36 (C)1995     ====
-----------------------------------------------------------------------------

                          Written By Steven Matty
                        ©1995 Leading Edge Software

  This library was written primarily to emulate the functions that were
present in AM*S but not in Blitz Basic 2. It began life as a load of Blitz
Statements but was then converted to high speed 680x0. The library will
continually be expanded upon.

Donations are not requested, but is always welcome. You may freely
```

distribute this library as long as all documentation is included in an unmodified form. *NO* distribution with commercial packages/magazines without express written permission.

Command Index

Command List:

```
    success=Reserve(length) | (banknumber,length)
    success=SwapBank(bank,bank)
    Erase(banknumber)
    EraseAll
    address.l=Start(banknumber)
    length.l=Length(banknumber)
    success=BLoad(filename$) | (banknumber/address[,length,offset])
    success=BSave(filename$,banknumber/address,length)
    success=PLoad(filename$,banknumber/address)
    bytes.l=FileSize(filename$)
    banknumber=NextBank
    bytes.l=MemFree
    value.l=Max(value1.l,value2.l)
    value.l=Min(value1.l,value2.l)
    value.l=XOR(value1.l,value2.l)
    id.w=Lisa
    Reboot
    code.b=KeyCode
    CopyByte sourceaddress.l,destaddress.l,numbytes.l
    CopyWord sourceaddress.l,destaddress.l,numwords.l
    CopyLong sourceaddress.l,destaddress.l,numlongs.l
    FillMem sourceaddress.l,numberbytes.l[,bytevalue.l]
    CachesOff bitmask
    CachesOn  bitmask
    number.l=Timer
    ResetTimer
    success=Rename(source$,dest$)
    success=MakeDir(dirname$)
    tagnumber.l=GTTagValue[(tagnumber)]
    success=CludgeShapes(shapenum#,numshapes,address)
    success=CludgeSound(soundnumber,address)
    success=FindVolume(volumename$)
    status=BlitterDone
    WaitBlitter
    devicename$=DeviceName$(volumename$)
    BlitterNasty On|Off
    FuncLibVersion
```

```
************ NOTE ***************
* AS FROM THIS VERSION (V1.36)   *
* THERE WILL BE NO MORE COMMANDS *
* ADDED. INSTEAD, A NEW LIBRARY  *
* CALLED RIAMOSPROFUNC WILL BE   *
* RELEASED. THIS IS DUE TO LARGE *
* LIBRARY SIZE AND THE FACT THAT *
* BLITZ V1.90 DOES NOT INCLUDE   *
* A LINKER.                      *
**********************************
```

```
********************************* NOTE **************************************
* VALID BANKS RANGE FROM 0-49 INCLUSIVE. DO NOT USE A VALUE GREATER THAN 49 *
* OR IT WILL BE INTERPRETED AS AN ADDRESS RATHER THAN A BANKNUMBER          *
*****************************************************************************


                          MEMORY AND FILE ACCESS
                          ======================
```

1.3 RIAmosFuncLib

```
Function: Reserve
--------------------------------------------------------------------------------
Mode   : Amiga/Blitz
Syntax : success=Reserve(length) | (banknumber,length[,requirements])

  If only length if specified, then this functions returns the number of the
  bank allocated or -1 for failure.

  This will attempt to reserve <length> bytes of memory. If succesfull,
it will return -1. If unsuccessfull, 0 is returned.

The optional <requirements> parameter specifies which type of memory you
want :
    %1=PUBLIC
    %10=CHIP
    %100=FAST
    %100000000=LOCAL
    %1000000000=24BITDMA
    %10000000000=KICK
    %10000000000000000=CLEAR
    %1000000000000000000=REVERSE
    %10000000000000000000000000000000=NO_EXPUNGE

OR the values together for different combinations.

EXAMPLE:
  suc=Reserve(0,1024,%10)   ; Reserve 1k of Chip Mem returns -1
  suc=Reserve(1024)   ; Reserve 1k of Any Mem returns 1
```

1.4 RIAmosFuncLib

```
Statement: Erase
--------------------------------------------------------------------------------
Mode   : Amiga/Blitz
Syntax : Erase banknumber

  The Erase command will erase the specified memory bank.

EXAMPLE:
  suc=Reserve(0,1024,%10)   ; Reserve 1k of Chip Mem
  Erase 0
```

1.5 RIAmosFuncLib

```
Statement: EraseAll
--------------------------------------------------------------------------------
Mode   : Amiga/Blitz
Syntax : EraseAll

  This command will erase ALL allocated memory banks.

EXAMPLE:
  suc=Reserve(0,1024,%10)   ; Reserve 1k of Chip Mem
  suc=Reserve(1,2048,0)   ; Reserve 1k of ANY Mem
  EraseAll
```

1.6 RIAmosFuncLib

```
Function: BLoad
--------------------------------------------------------------------------------
Mode   : Amiga
Syntax : success=BLoad(filename$) | (filename$,bank/addr[,length,offset,memtype])

  If only filename$ is specified, then the next available bank is allocated,
  and the command returns the number of the bank for success or -1 for failure.

  If bank is specified, then the file is loaded into that bank. If address
is specified then it is loaded to the address. Valid banks are 0-49.
If the bank does not exist, Blitz will reserve a bank for you.
If the bank does exist, Blitz will erase the bank from memory, and
allocate a new one.
The return result is -1 for success, or 0 for failure  (not enough RAM,
file not exist). If offset is specified, then <length> bytes will be read
from the specified offset position in the file.
If memtype is specified, then the file is loaded into a memory block
of that particular memtype (see Reserve)
If you wish to leave either length/offset unspecified, simply use the
value 0

EXAMPLE:
  suc=BLoad("s:startup-sequence",0) ; returns -1
  suc=BLoad("c:dir",0,0,0,%10)    ; Loads into CHIP
  suc=BLoad("c:list")     ; returns 1
```

1.7 RIAmosFuncLib

```
Function: PLoad
--------------------------------------------------------------------------------
Mode   : Amiga
Syntax : success=PLoad(filename$,bank/address)

  This will attempt to load the executable file to the specified address.
  -1 is success, 0 is failure. The program must contain only a CODE
hunk and must be FULLY relocatable.

EXAMPLE:

  suc=PLoad("c:dir",0)
```

1.8 RIAmosFuncLib

```
Function: BSave
---------------------------------------------------------------------------------
Mode   : Amiga
Syntax : success=BSave(filename$,bank/address,length)

  This will save <length> bytes at bank/address to the file. Return result
is -1 for success, 0 for failure. If length > bank length then the length
of the bank is saved instead. If 0 is specified, the entire bank is saved.

EXAMPLE:
  suc=BLoad("c:dir",0,0,0,%10)    ; Loads into CHIP
  suc=BSave("ram:temp",0)
```

1.9 RIAmosFuncLib

```
Function: Start
--------------------------------------------------------------------------------
Mode   : Amiga/Blitz
Syntax : start_address.l=Start(banknumber.b)

  This will return the start address of the specified bank. (0=no bank)

EXAMPLE:
  suc=Reserve(0,1024,%10)
  NPrint Start(0)
  MouseWait
  End
```

1.10 RIAmosFuncLib

```
Function: Length
--------------------------------------------------------------------------------
Mode   : Amiga/Blitz
Syntax : length_of_bank.l=Length(banknumber.b)

  This will return the length of the specified bank in bytes. (0=No bank)

EXAMPLE:
  suc=Reserve(0,1024,%10)
  NPrint Length(0)
  MouseWait
  End
```

1.11 RIAmosFuncLib

```
Function: MemFree
---------------------------------------------------------------------------------
Mode   : Amiga/Blitz
Syntax : bytes.l=MemFree

  This will return the total amount of Public Free RAM available to the
system.

EXAMPLE:
  NPrint "Total bytes free = ",MemFree
  MouseWait
  End
```

1.12 RIAmosFuncLib

```
Function: NextBank
---------------------------------------------------------------------------------
Mode   : Amiga/Blitz
Syntax : bank.b=NextBank

  This will return the number of the first available bank (-1 if none free).

EXAMPLE:
  suc=Reserve(0,1024)
  suc=Reserve(0,2048)
  NPrint NextBank
  MouseWait
  End
```

1.13 RIAmosFuncLib

```
Statement: FillMem
--------------------------------------------------------------------------------
Mode   : Amiga/Blitz
Syntax : FillMem address.l,length.l[,value.b]

   This will fill 'length' bytes starting from the specified address with
   'value'. If 'value' is ommitted, 0 is filled.

EXAMPLE:
  suc=Reserve(0,1024)   ; Allocate some memory
  FillMem Start(0),Length(0)  ; Clear it
  MouseWait
  End
```

1.14 RIAmosFuncLib

```
Statement: CopyByte
---------------------------------------------------------------------------------
Mode   : Amiga/Blitz
Syntax : CopyByte source.l,dest.l,num.l

  This will copy <num> bytes from <source> to <dest>

EXAMPLE:
  CopyByte Start(0),Start(1),Length(0)
```

1.15 RIAmosFuncLib

```
Statement: CopyWord
---------------------------------------------------------------------------------
Mode   : Amiga/Blitz
Syntax : CopyByte source.l,dest.l,num.l

  This will copy <num> words from <source> to <dest>

EXAMPLE:
  CopyWord Start(0),Start(1),Length(0)/2
```

1.16 RIAmosFuncLib

```
Statement: CopyLong
---------------------------------------------------------------------------------
Mode   : Amiga/Blitz
Syntax : CopyByte source.l,dest.l,num.l

  This will copy <num> longwords from <source> to <dest>
```

```
EXAMPLE:
  CopyLong Start(0),Start(1),Length(0)/4
```

1.17 RIAmosFuncLib

```
Function: MakeDir
--------------------------------------------------------------------------------
Mode   : Amiga
Syntax : success=MakeDir(name$)

   This function attempts to create a directory called <name$>
   If it is unsuccessfull, 0 is returned else -1 is returned.

EXAMPLE:
  suc=MakeDir("RAM:MYDIR")
```

1.18 RIAmosFuncLib

```
Function: Rename
--------------------------------------------------------------------------------
Mode   : Amiga
Syntax : success=Rename(source$,dest$)

   This attempts to rename the file <source$> to <dest$>
   NOTE: It is not possible to rename across devices.
   -1 is returned if successfull, else 0.

EXAMPLE:
  suc=Rename("S:Startup-Sequence","S:Startup2") ; Do not run this!
```

1.19 RIAmosFuncLib

```
Function: Timer
--------------------------------------------------------------------------------
Mode   : Amiga/Blitz
Syntax : t.l=Timer

   This will return the number of 50ths of a second since startup or the
   last call to ResetTimer.

EXAMPLE:
  NPrint Timer
  VWait
  NPrint Timer
  MouseWait
  End
```

1.20 RIAmosFuncLib

```
Statement: ResetTimer
--------------------------------------------------------------------------------
Mode   : Amiga/Blitz
Syntax : ResetTimer

   This will recent the CIA timer to 0.

EXAMPLE:
  NPrint Timer
  VWait
  ResetTimer
  NPrint Timer
  MouseWait
  End
```

1.21 RIAmosFuncLib

```
Function: Lisa
--------------------------------------------------------------------------------
Mode   : Amiga/Blitz
Syntax : chipver=Lisa

   This will return the current Lisa chip version :

    $00 for OCS Denise
    $F7 for ECS Denise
    $F8 for AGA Lisa

EXAMPLE:
  Select Lisa
    Case 0
      NPrint "You have an OCS Machine!"
    Case $F7
      NPrint "You have an ECS Machine!"
    Case $F8
      NPrint "You have an AGA Machine!"
    Case $F9
      NPrint "You have a AAA Machine?!" ; Maybe... :)
  End Select
  MouseWait
  End
```

1.22 RIAmosFuncLib

```
Statement: Reboot
--------------------------------------------------------------------------------
Mode   : Amiga/Blitz
Syntax : Reboot

   This will perform a cold reboot

EXAMPLE:
  NPrint "Press mousebutton to reset.."
  MouseWait
  Reboot
```

1.23 RIAmosFuncLib

```
Function: FileSize
--------------------------------------------------------------------------------
Mode   : Amiga
Syntax : size.l=FileSize(filename$)

  This return the length (in bytes) of the file.

EXAMPLE:
  NPrint "Startup is ",FileSize("S:startup-sequence")," bytes long!"
  MouseWait
  End
```

1.24 RIAmosFuncLib

```
Function: XOR
--------------------------------------------------------------------------------
Mode   : Amiga/Blitz
Syntax : x.l=XOR(x.l,y.l)

  This will perform an Exclusive-Or operation between X and Y and put the
result back into X

EXAMPLE:
    x=XOR(%101,%100)

Will place %001 into X (%101 XOR %100 = %001)
```

1.25 RIAmosFuncLib

```
Function: Max/Min
-------------------------------------------------------------------------------
Mode   : Amiga/Blitz
```

```
Syntax : value=Max(first_var,second_var)
         value=Min(first_var,second_var)

  This will compare both values and return either the Higher of the values
  (Max) or the Lower (Min). This currently supports INTEGERs only.

EXAMPLE:
  NPrint Max(30,50)
  NPrint Min(30,50)
  MouseWait
  End
```

1.26 RIAmosFuncLib

```
Function: KeyCode
-------------------------------------------------------------------------------
Mode   : Amiga/Blitz
Syntax : keycode=KeyCode

    This will return the status of the keyboard in the form of a keycode.
    You will need to experiment to find out the desired keycode for
    a particular key.
    This merely peeks address $bfec01 and returns the value found.

EXAMPLE:
  NPrint KeyCode
  MouseWait
  End
```

1.27 RIAmosFuncLib

```
Statement/Function : CludgeShapes
-------------------------------------------------------------------------------
Mode   : Amiga/Blitz
Syntax : [success]=CludgeShapes(shape#,numshapes,address)

   This allows the creation of shapes through INCBIN statements. It
allocates chip memory for each shape and copies the data into this.
It does the same as LoadShapes except it grabs shapes from memory.

EXAMPLE:
  suc=BLoad("myshapes",0)
  suc=CludgeShapes(0,50,Start(0))
  MouseWait
  End
```

1.28 RIAmosFuncLib

```
Statement/Function : CludgeSound
--------------------------------------------------------------------------------
Mode   : Amiga/Blitz
Syntax : [success]=CludgeSound(sound#,address)

  This does the same for CludgeShapes but works on only 1 sound at a time
NOTE: Looped sounds are not currently supported! The sound must be a valid
8SVX sample.

EXAMPLE:
  suc=BLoad("mysound",0)
  suc=CludgeSound(0,Start(0))
  MouseWait
  End
```

1.29 RIAmosFuncLib

```
Function : FindVolume
--------------------------------------------------------------------------------
Mode   : Amiga
Syntax : success=FindVolume(volumename$)

  This will look to see if the specified volume is present, and returns
0 if it is not or -1 if it is. If the volume is not present, this function
will NOT bring up a Requester ("Please insert Volume...")
The ":" should not be included in the volumename.

This is useful for waiting for diskswaps when you have a BlitzMode display

EXAMPLE:
    <Blitzmode Statements>
    QAMIGA
    Repeat
      VWait
    Until FindVolume("DISK2")
    BLITZ
    <More statements>
```

1.30 RIAmosFuncLib

```
Function : DeviceName$
--------------------------------------------------------------------------------
Mode   : Amiga
Syntax : devname$=DeviceName$(volumename$)

  This will return the device name of the specified volume or "" if the
volume was not found. The ":" may or may not be included.

EXAMPLE:
    NPrint DeviceName$("WORK:")
```

1.31 RIAmosFuncLib

```
Function : BlitterDone
------------------------------------------------------------------------------
Mode   : Amiga/Blitz
Syntax : status=BlitterDone

  This checks to see if the Blitter has finished BLITting. -1=Yes, 0=No

EXAMPLE:
  Repeat
  Unti BlitterDone
```

1.32 RIAmosFuncLib

```
Statement : WaitBlitter
------------------------------------------------------------------------------
Mode   : Amiga/Blitz
Syntax : WaitBlitter

  This will halt program execution until the Blitter is ready for use.

EXAMPLE:
  Blit 0,0,0
  WaitBlitter
     ..
     ..
```

1.33 RIAmosFuncLib

```
Statement : BlitterNasty
------------------------------------------------------------------------------
Mode   : Amiga/Blitz
Syntax : BlitterNasty

  This will set the BlitterNasty hardware register bit, which means that
the Blitter has complete priority over the CPU. This function returns
the old status.

*NOTE* In order for this to be effective, place this command in a loop
after a VWait.
```

1.34 RIAmosFuncLib

```
Function : FuncLibVersion
-----------------------------------------------------------------------------
Mode   : N/A
Syntax : N/A

  This command does nothing (except return 0). Press HELP on the command
name for your current version (v1.36 or higher only)


=============================================================================
                              VERSION HISTORY
=============================================================================

Version 0.9
~~~~~~~~~~~
Internal release. BLoad/BSave/Reserve/Erase only... (Blitz Statements)

Version 1.0
~~~~~~~~~~~
First public release. Many new commands... (Assembly)

Version 1.1
~~~~~~~~~~~
Bundled in RIBlitzlibsv3.1.lha on Aminet. Also published on BUM7
Added a few extra commands

Version 1.2
~~~~~~~~~~~
Changed CacheOff to CachesOff. Now supports a bitmask for disabling
any caches you want. CachesOn added - so you can turn them back on.
Don’t know why it wasn’t in before - I guess it didn’t work before...

Version 1.3
~~~~~~~~~~~
Fixed bug in CopyByte/CopyWord/CopyLong - using DBRA so only 65535
copies could be done. (Thanks Steve Mc)

Version 1.31
~~~~~~~~~~~~
Fixed memory drain in FileSize (DosObject not freed)
  Reported by Ott Aloe from ’The Farm’ - Thanks Ott

Version 1.32
~~~~~~~~~~~~
Added CD command to change current directory. (Request by Ott)

Version 1.33
~~~~~~~~~~~~
Fixed bug I introduced in FileSize (was freeing DosObject before
getting the length of file)

Version 1.34
~~~~~~~~~~~~
Added single parameter option to Reserve (only length). Returns
```

```
bank number else -1.

Version 1.35
~~~~~~~~~~~~
Added single parameter option to BLoad (only filename). Returns
bank number else -1.

Added NOTE to BlitterNasty docs because copperlist would trash
the bit every 50th.

Version 1.36
~~~~~~~~~~~~
Fixed stupid bug in BLoad. Meant that FNSLoad/ILBMLoad would not
work.
Added FuncLibVersion to display current lib version
```

1.35 RIAmosFuncLib: Command Index

```
                  Command index for library RIAmosFuncLib

                                Library Main

                           Number of commands: 32

            BlitterDone                          BlitterNasty
            BLoad                                BSave
            CludgeShapes                         CludgeSound
            CopyByte                             CopyLong
            CopyWord                             DeviceName$
            Erase                                EraseAll
            FileSize                             FillMem
            FindVolume                           FuncLibVersion
            KeyCode                              Length
            Lisa                                 MakeDir
            Max/Min                              MemFree
            NextBank                             PLoad
            Reboot                               Rename
            Reserve                              ResetTimer
            Start                                Timer
            WaitBlitter                          XOR
```

1.36 RIAnimLib

```
-----------------------------------------------------------------------------
====                  RI ANIM Library V1.3 (C)94/95      ====
-----------------------------------------------------------------------------

                        Written By Stephen McNamara
                        ©94/95 Leading Edge Software

                               Command Index
```

```
This library enables the playback of both Anim5 and Anim7 format
animations.  It allows you to playback animations at any co-ordinate in a
bitmap and supports different palettes for frames of the animation.  It
also allows you to playback animations from FAST ram, thus you can now play
massive animations that can only fit in FAST ram.

When playing back animations you must make sure that your display is
double-buffered.  Please refer to the Blitz manual for information about
how anims can be played back properly - or look at the example program
included with this file.

There has been some extensive testing of this library.  The result of this
is that all none problems with it have been fixed.  Bug fixes include loop
frame anims not looping properly and anims with separate palettes per frame
now play correctly.


Command list:
    [suc=]RIAnimInit(address,bitmap#,palette# [,xy_offset]|[,x,y])
    [suc=]RINextAnimFrame bitmap#
    AnimLoop

New commands:
    num.w=RIAnimFrameCount

Updated commands:
    RIAnimInit now accepts offset as x,y coordinates.
```

1.37 RIAnimLib

```
Statement/Function: RIAnimInit
-----------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: [suc=]RIAnimInit(address,bitmap#,palette# [,xy_offset]|[,x,y])

  This command attempts to take an animation held in memory (CHIP or FAST)
and identify it as a supported animation format.  If it identifies it okay
it will set up the animation by unpacking frame 1 of the anim onto the
specified bitmap and copying the palette to the specified palette object.
You must ensure that the bitmap is big and deep enough to actually hold the
animation.  At the moment there is no checking of the bitmap size.  The
palette object you give is automatically resized to the size of the palette
in the animation.

  The optional parameter allows you to play an animation at an offset into
a bitmap.   This command has been extended so that you can specific the
optional offset into the bitmap as either a byte value, or a x,y coordinate.
Given in offset form, you should use the following formula to calculate the
value to use:

      offset=(X/8)+(Y*(pixel_width/8))

      where: X and Y are your co-ordinates
             pixel_width is the width of your bitmap.
  Offset form is kept for compatibility with older versions of this
library.  You should unsure that your animation will never go off screen
when using the offset parameter(s).  Incorrect placement could cause a
crash of your machine.

  If used as a function, this command returns true for a successful
initialise or false for failure.
```

1.38 RIAnimLib

```
Statement/Function: RINextAnimFrame
------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: [suc=]RINextAnimFrame bitmap#

  This command attempts to unpack the next frame of a previously
initialised animation onto the specified bitmap.  It returns true or false
to say whether it succeeded or not.
```

1.39 RIAnimLib

```
Statement: AnimLoop
------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: AnimLoop ON|OFF

  This command allows you to control the looping mode of the animation.
With animloop off, playback of an animation will stop at the last frame of
it.  Any attempt to draw another frame will fail.  With it on, though, the
animation will loop around.

  Note: you must ensure that your animation has loop frames at the end of
it if you want to loop the animation around.  The reverse of this is true
for animloop off - the animation must not have loop frames if you don’t
want it to loop around.  If you select animloop off but have looping frames
in your anim then the animation will end by displaying a copy of frame 2
of the animation.
```

1.40 RIAnimLib

```
Function: RIAnimFrameCount
------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: numframes=RIAnimFrameCount

  This command allows you to count the number of frames in the currently
```

```
initialised animation.
```

1.41 RIAnimLib: Command Index

```
                    Command index for library RIAnimLib

                                Library Main

                           Number of commands: 4

                              AnimLoop
                              RIAnimFrameCount
                              RIAnimInit
                              RINextAnimFrame
```

1.42 RIAppLib

```
-----------------------------------------------------------------------------
====                RI App Library V1.4 (C)1994      ====
-----------------------------------------------------------------------------

                           Written By Steven Matty
                         ©1994 Leading Edge Software

                               Command Index

                             !!!! IMPORTANT !!!!
    *******************************************
    *   For New Features See Bottom Of Text   *
                *   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~   *
                *******************************************


This small library provides quick and easy to use commands for accessing
AppWindows, AppIcons and AppMenus.

An AppWindow is a window on the Workbench screen which will allow you
to drag file(s) from into it, instead of ploughing through file-requesters.

An AppMenu adds a menu item to the "Tools" menu of the Workbench. It is
normally used for when the program is 'sleeping' and the user wishes to
wake it up. In addition, if any files are selected and the menu item
is selected these are passed to the program.

An AppIcon is just like a normal file icon on the Workbench except it
allows you to drop file(s) onto it, or to double-click it to wake
up the program.

These features require at Workbench v2.0 or higher.

Command List:
```

```
  AppEvent()
  AppEventCode()
  AppEventID()
  AddAppWindow()
  AddAppIcon()
  AddAppMenu()
  DelAppWindow()
  DelAppIcon()
  DelAppMenu()
  NextAppFile()
  AppFile()
  AppNumFiles()
```

1.43 RIAppLib

```
Function : AppEvent
------------------------------------------------------------------------------
Modes  : Amiga
Syntax : status=AppEvent

This command checks to see whether or not an 'App'Event (e.g. File
dropped onto an AppIcon or Menu Item selected) has occurred.

This function will return 0 if no event has occurred, else
$80000 if :
  An AppMenu was selected
        An AppIcon was double-clicked
        A File Was Dragged Into An AppWindow
        A File Was Dragged Onto An AppIcon


********
```

1.44 function

```
* NOTE * : This function no longer returns the number of files
********   selected. $80000 is returned instead of -1.
           See AppNumFiles().

e.g.

  Repeat
    VWait
    appev.l=AppEvent    ; Has something happened
  Until appev
  If appev=$80000
    NPrint "An AppEvent Occurred! !"
  EndIf
```

1.45 RIAppLib

```
Function : AddAppWindow
--------------------------------------------------------------------------------
Modes  : Amiga
Syntax : success=AddAppWindow(windownumber)
```

This command attempts to make the window specified by 'windownumber' to become an AppWindow. -1 means success, 0 means failure. There is a limit of 16 AppWindows open at any one time.

1.46 RIAppLib

```
Function : AddAppIcon
--------------------------------------------------------------------------------
Modes  : Amiga
Syntax : success=AddAppIcon(id,text$,iconname$)
```

This command attempts to place an AppIcon onto the Workbench desktop.
ID is a unique identification number. Text$ is text to display underneath the AppIcon and Iconname$ is the name of the file to use the Icon imagery.
-1 means success, 0 means failure. There is a limit of 16 AppIcons.
e.g.

```
  suc=AddAppIcon(0,"QuickFormat","SYS:System/Format")
  If suc=0 Then End
```

1.47 RIAppLib

```
Function : AddAppMenu
--------------------------------------------------------------------------------
Modes  : Amiga
Syntax : success=AddAppMenu(id,text$)
```

This command tries to add 'text$' to the Tools menu of Workbench.
ID is a unique identification number. Returns -1 for success, 0 for failure.
There is a limit of 16 AppMenu items.

e.g.

```
  suc=AddAppMenu(0,"Wakey Wakey")
  If suc=0 Then End
```

1.48 RIAppLib

```
Function : AppEventCode
--------------------------------------------------------------------------------
Modes  : Amiga
Syntax : apptype=AppEventCode

This function will return the type of App object which caused the event.
0=No Event Occurred
1=AppWindow
2=AppIcon
3=AppMenu

e.g.

  Repeat
    VWait
    appev.l=AppEvent    ; Has something happened
  Until appev
  Select AppEventCode
    Case 1
      NPrint "An AppWindow caused this!"
    Case 2
      NPrint "An AppIcon caused this!"
    Case 3
      NPrint "An AppMenu caused this!"
  End Select
```

1.49 RIAppLib

```
Function : AppEventID
-------------------------------------------------------------------------------
Modes  : Amiga
Syntax : idnumber=AppEventID

  This will return the object ID number which caused the AppEvent.
  This ID number refers to the one which was used in
  AddAppIcon/AddAppWindow/AddAppWindow.

  -1 means that no AppEvent occurred.
```

1.50 RIAppLib

```
Function : NextAppFile
-------------------------------------------------------------------------------
Modes  : Amiga
Syntax : filename$=NextAppFile

  This will return the full path and filename for the file/drawer/volume
  which was selected when an AppEvent occurred.  If a directory was selected
  then a '/' is appended to file name. If a volume (e.g. a Disk) was
  selected then a ":" is appended.

  An empty string means nothing was selected.

  e.g.
  ; AppStuff initalized
  Repeat
    VWait
    appev.l=AppEvent
  Until appev=$80000      ; repeat until some files are selected.
  numfiles.l=AppNumFiles
  For n=1 To numfiles
    NPrint "File number "+str$(n)+" is "+NextAppFile
  Next n
```

1.51 RIAppLib

```
Function : AppNumFiles
------------------------------------------------------------------------------
Modes  : Amiga
Syntax : numfiles=AppNumFiles

  This will return the number of files selected when the AppEvent occurred.
```

1.52 RIAppLib

```
Function : AppFile
------------------------------------------------------------------------------
Modes  : Amiga
Syntax : filename$=AppFile(file#)

  This will return the full path and filename for the file/drawer/volume
  which was selected when an AppEvent occurred. The file# parameter
  specifies which file to return. If a directory was selected then a '/'
  is appended to file name. If a volume (e.g. a Disk) was selected then
  a ":" is appended.

  An empty string means nothing was selected.

  e.g.
  ; AppStuff initalized
  Repeat
    VWait
    appev.l=AppEvent
  Until appev=$80000      ; repeat until some files are selected.
  numfiles.l=AppNumFiles
  For n=1 To numfiles
    NPrint "File number "+str$(n)+" is "+AppFile(n)
  Next n
```

1.53 RIAppLib

```
Function: DelAppWindow
------------------------------------------------------------------------------
Modes  : Amiga
Syntax : success=DelAppWindow[(number)]

These commands will remove the AppWindow from the system and free up the
associated message port.
```

1.54 RIAppLib

```
Function: DelAppIcon
------------------------------------------------------------------------------
Modes  : Amiga
Syntax : success=DelAppIcon[(id)]

These commands will remove the AppIcon from the system and free up the
associated message port.
```

1.55 RIAppLib

```
Function: DelAppMenu
------------------------------------------------------------------------------
Modes  : Amiga
Syntax : success=DelAppMenu[(id)]

These commands will remove the AppMenu from the system and free up the
associated message port.

==============================================================================
                              VERSION HISTORY
==============================================================================

Version 1.0
~~~~~~~~~~~
First release using old commands


Version 1.1
~~~~~~~~~~~
Re-written from scratch now handles multiple files and is structured
more like the standard Intuition event handling.

Version 1.2
~~~~~~~~~~~
Fixed memory drain. There still seems to be some loss somewhere
Need to add WaitAppEvent which waits for AppEvents + Normal events
instead of a Repeat..VWait..Until loop.
```

```
Version 1.3
~~~~~~~~~~~
Re-wrote to allocate memory on run-time. Saved approx 1k off
library size.

Version 1.4
~~~~~~~~~~~
Changed AppEvent to return 0 for no event or $80000 for AppEvent.
No longer returns number of files in AppEvent. Added AppNumFiles
for this. Also changed Blitz Windowslib to allow WaitEvent to capture
AppEvents.
```

1.56 RIAppLib: Command Index

```
                     Command index for library RIAppLib

                                Library Main

                           Number of commands: 13

            AddAppIcon
            AddAppMenu
            AddAppWindow
            AppEvent
            AppEventCode
            AppEventID
            AppFile
            AppNumFiles
            DelAppIcon
            DelAppMenu
            DelAppWindow
            NextAppFile
            This function no longer returns the number of files
```

1.57 RICommoditiesLib

```
-----------------------------------------------------------------------------
====              RI Commodities Library V1.2 (C)1994     ====
-----------------------------------------------------------------------------

                          Written By Steven Matty
                        ©1994 Leading Edge Software

                               Command Index

Introduction
============

This library allows the easy use of Commodities. It requires Kickstart 2 or
higher.
```

1.58 RICommoditiesLib

```
Function : MakeCommodity
-----------------------------------------------------------------------------
Modes  : Amiga
Syntax : success=MakeCommodity(name$,title$,description$)

This command attempts to add your Commodity to the list of commodities.
A return value of -1 indicates success, 0 means failure. (not enough memory)

name$ refers to the name of the Commodity and it should be unique. This is
the name that appears when running the Commodity Exchange program.
title$ is the title of your program, e.g. "My Screen Blanker".
description$ is a brief description of your program.

The Commodity Exchange program will then have 'name$' in its list of
Commodities and when a user clicks on your commodity, it will display
the title$ and description$.
```

1.59 RICommoditiesLib

```
Function : SetHotKey
----------------------------------------------------------------------------
Modes  : Amiga
Syntax : success=SetHotKey(hotkey#,hotkeydescription$)

This will add a hotkey event to your commodity so that after a hotkey
has been pressed you can find out which one.

e.g.    success=SetHotKey(0,"lalt lshift a")
```

1.60 RICommoditiesLib

```
Function : HotKeyHit
----------------------------------------------------------------------------
Modes  : Amiga
Syntax : hitkeynum=HotKeyHit

This will return the number of the hot key which has been hit since the
last 'CommodityEvent' was called, or -1 if no such hotkey has been activated.
```

1.61 RICommoditiesLib

```
Function : CommodityEvent
----------------------------------------------------------------------------
Modes  : Amiga
```

```
Syntax : anyevent=CommodityEvent

This looks to see if either
  a) A hotkey has been pressed
  b) A message from Exchange has been received

and returns -1 if such an event occurred, of 0 is nothing has yet happened.
This should be inside a Repeat-Until loop, e.g.

  Repeat
    VWait
    ev.l=Event
    ce.l=CommodityEvent
    hk.l=HotKeyHit    ; This must be used after CommodityEvent
  Until ev or ce or hk
```

1.62 RICommoditiesLib

```
Statement : SetStatus
---------------------------------------------------------------------------
Modes  : Amiga
Syntax : SetStatus on|off

This sets the status of your Commodity to either Active (on) or Inactive
(off) - this can be seen by running the Commodities Exchange program.
```

1.63 RICommoditiesLib

```
Function : ExchangeMessage
---------------------------------------------------------------------------
Modes  : Amiga
Syntax : messnum.l=ExchangeMessage

This looks to see if the Commodities Exchange has issued you with as message,
e.g. Hide Interface, Show Interface. It returns the message ID of the incoming
message or 0 for no message.
```

1.64 RICommoditiesLib

```
Functions: CxAppear
-------------------------------------------------------------------------------
Modes : Amiga

This is used in conjunction with ExchangeMessage, ie

  em.l=ExchangeMessage
  Select em
```

```
    Case CxAppear
      Gosub _appear
    Case CxDisAppear
            Gosub _disappear
        End Select
```

The functions merely return the ID value associated with that particular Commodities Exchange message.

1.65 RICommoditiesLib

```
Functions: CxDisAppear
--------------------------------------------------------------------------------
Modes : Amiga
```

This is used in conjunction with ExchangeMessage, see CxAppear for more information.

1.66 RICommoditiesLib

```
Functions: CxEnable
--------------------------------------------------------------------------------
Modes : Amiga
```

This is used in conjunction with ExchangeMessage, see CxAppear for more information.

1.67 RICommoditiesLib

```
Functions: CxDisable
--------------------------------------------------------------------------------
Modes : Amiga
```

This is used in conjunction with ExchangeMessage, see CxAppear for more information.

1.68 RICommoditiesLib

```
Functions: CxKill
--------------------------------------------------------------------------------
Modes : Amiga
```

This is used in conjunction with ExchangeMessage, see CxAppear for more information.

1.69 RICommoditiesLib

```
Functions: CxChangeList
--------------------------------------------------------------------------------
Modes : Amiga
```

This is used in conjunction with ExchangeMessage, see CxAppear for more information.

1.70 RICommoditiesLib

```
Functions: CxUnique
--------------------------------------------------------------------------------
Modes : Amiga
```

This is used in conjunction with ExchangeMessage, see CxAppear for more information.

1.71 RICommoditiesLib

```
Functions: ExchangeAppear
--------------------------------------------------------------------------------
Modes : Amiga
```

To be used in conjunction with ExchangeMessage, ie

```
  em.l=ExchangeMessage
  If em
    If ExchangeAppear then Gosub _appear
    If ExchangeDisAppear then Gosub _dispappear
  EndIf
```

This is intended as an alternative way of acting upon Exchange Messages.

1.72 RICommoditiesLib

```
Functions: ExchangeDisAppear
--------------------------------------------------------------------------------
Modes : Amiga
```

To be used in conjunction with ExchangeMessage, see ExchangeAppear for more
information on usage.

1.73 RICommoditiesLib

```
Functions: ExchangeEnable
--------------------------------------------------------------------------------
Modes : Amiga
```

To be used in conjunction with ExchangeMessage, see ExchangeAppear for more
information on usage.

1.74 RICommoditiesLib

```
Functions: ExchangeDisable
--------------------------------------------------------------------------------
Modes : Amiga
```

To be used in conjunction with ExchangeMessage, see ExchangeAppear for more
information on usage.

1.75 RICommoditiesLib

```
Functions: ExchangeKill
--------------------------------------------------------------------------------
Modes : Amiga
```

To be used in conjunction with ExchangeMessage, see ExchangeAppear for more
information on usage.

1.76 RICommoditiesLib

```
Functions: ExchangeChangeList
--------------------------------------------------------------------------------
Modes : Amiga
```

To be used in conjunction with ExchangeMessage, see ExchangeAppear for more
information on usage.

1.77 RICommoditiesLib

```
Functions: ExchangeUnique
--------------------------------------------------------------------------------
Modes : Amiga


To be used in conjunction with ExchangeMessage, see ExchangeAppear for more
information on usage.
```

1.78 RICommoditiesLib: Command Index

```
                 Command index for library RICommoditiesLib

                                Library Main

                           Number of commands: 20

          CommodityEvent                      CxAppear
          CxChangeList                        CxDisable
          CxDisAppear                         CxEnable
          CxKill                              CxUnique
          ExchangeAppear                      ExchangeChangeList
          ExchangeDisable                     ExchangeDisAppear
          ExchangeEnable                      ExchangeKill
          ExchangeMessage                     ExchangeUnique
          HotKeyHit                           MakeCommodity
          SetHotKey                           SetStatus
```

1.79 RICompactDisklib

```
------------------------------------------------------------------------------
----              RI Compact Disc Library V1.3 (C)94/95                   ----
------------------------------------------------------------------------------

                Written By Stephen McNamara & Steven Matty
                       ©94/95 Leading Edge Software

                               Command Index

Command list:
    OpenCD
    CloseCD
    CDDoor On/Off
    CDPlayTrack track#,numtracks
    CDReadTOC
    CDStatus
    CDStop
    CDVolume volume,lengthoffade
    CDNumTracks
    CDFirstTrack
    CDLastTrack
    CDTrackLength track#
    CDFlush
    CDPause On/Off
    CDRewind
    CDFastForward
    CDNormalSpeed
    CDSpeed mode
    CDUpdateInfo
    CDTrackMins [offset]
    CDTrackSecs [offset]
    CDTrackPlaying
```

1.80 RICompactDisklib

```
Statement/Function: OpenCD
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: [suc=]OpenCD

  Attempts to open the cd.device for use my the library.  If used as a
function it returns true or false to say whether the device was opened
successfully.  You must use this command before you attempt to use any of
the other commands in this library.
```

1.81 RICompactDisklib

```
Statement/Function: CloseCD
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: [suc=]CloseCD

  You must close the cd.device before your program ends.  Close the device
by using this command.
```

1.82 RICompactDisklib

```
Statement: CDDoor
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: CDDoor On/Off

  Controls the status of the cd tray on your cd drive.  Giving a value of
On (non-zero) with this command will cause the tray to open, Off will cause
the tray to close
```

1.83 RICompactDisklib

```
Statement/Function: CDPlayTrack
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: CDPlayTrack track#,numtracks

  Use this command to make the cd drive play one or more audio tracks on
the currently inserted compact disc.  Tracks are numbered from one but you
should make sure that track one is an audio track, since CD-ROMs store
program data on track one.  The numtracks arguement allows you to play
more than one track without extra commands.  When the cd player reaches the
end of the track it will move straight onto the next track automatically if
you specified to play more than one.
  This command can return a value to you if desired. ~A return value of
true means that the command succeeded, else false means failure.
```

1.84 RICompactDisklib

```
Statement/Function: CDReadTOC
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: [suc=]CDReadTOC

  Read the table of contents off the current CD.  This most be done before
you attempt to obtain information about tracks/try to play a track.  This
command can optionally return true or false to say whether or not it
succeeded.
```

1.85 RICompactDisklib

```
Function: CDStatus
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: status=CDStatus

  Returns the status information for the cd.device.  This data includes the
current status of the cd drive, and whether or not there is a compact disc
inserted into it.  The return value is a binary number, with the following
bits being of interest:

Name                Bit number     Meaning
------------------------------------------
CDSTSB_CLOSED     0      Drive door is closed
CDSTSB_DISK     1      A disk has been detected
CDSTSB_SPIN     2      Disk is spinning (motor is on)
CDSTSB_TOC      3      Table of contents read.  Disk is valid.
CDSTSB_CDROM      4      Track 1 contains CD-ROM data
CDSTSB_PLAYING      5      Audio is playing
CDSTSB_PAUSED     6      Pause mode (pauses on play command)
```

```
CDSTSB_SEARCH     7      Search mode (Fast Forward/Fast Reverse)
CDSTSB_DIRECTION    8      Search direction (0 = Forward, 1 = Reverse)
```

It is possible to get more than one bit set at a time in the variable so you should not do straight comparisions with the return value. Use the & operator to test for different statuses, e.g.

```
    If (CDStatus & %1) then NPrint "CD tray is closed!"
```

1.86 RICompactDisklib

```
Statement: CDStop
------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: CDStop
```

Causes the cd player to stop playing the current track.

1.87 RICompactDisklib

```
Statement/Function: CDVolume
------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: CDVolume volume,lengthoffade
```

Gotta find out :)

1.88 RICompactDisklib

```
Function: CDNumTracks
------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: num=CDNumTracks
```

Get the total number of tracks on the currently inserted compact disc. Should be used only after the table of contents has been read using CDReadTOC.

1.89 RICompactDisklib

```
Function: CDFirstTrack
------------------------------------------------------------------------
Modes : Amiga/Blitz
```

```
Syntax: num=CDFirstTrack

  Returns the first track on the disc available for playing using the
CDPlayTrack command.
```

1.90 RICompactDisklib

```
Function: CDLastTrack
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: num=CDLastTrack

  Returns the last track on the disc available for playing using the
CDPlayTrack command.
```

1.91 RICompactDisklib

```
Function: CDTrackLength
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: l=CDTrackLength(track#)

  Returns the length in seconds of the selected track.  The track# should
be checked to make sure that it exists on the compact disc.
```

1.92 RICompactDisklib

```
Statement: CDFlush
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: CDFlush

Gotta find out :)
```

1.93 RICompactDisklib

```
Statement: CDPause
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: CDPause On/Off

  This command is used to either make the cd player pause on the currently
playing track, or restart after being paused.  If you set pause on whilst
```

```
a track is not playing, and then attempt to play a track the cd player will
go straight into pause mode.
```

1.94 RICompactDisklib

```
Statement: CDRewind
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: CDRewind

  Set the cd player into rewind mode.
```

1.95 RICompactDisklib

```
Statement: CDFastForward
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: CDFastForward


  Set the cd player into fastforward mode.
```

1.96 RICompactDisklib

```
Statement: CDNormalSpeed
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: CDNormalSpeed

  Restore the cd player to normal playing speed.
```

1.97 RICompactDisklib

```
Statement: CDSpeed
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: CDSpeed speed

  Set the cd player speed directly using the value in the speed parameter.
```

1.98 RICompactDisklib

```
Statement: CDUpdateInfo
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: CDUpdateInfo

  This command is used to update the current track information whilst a
compact disc is actually playing.  After it has been called, the commands
CDTrackMins, CDTrackSecs and CDTrackPlaying will return information about
the current track.
```

1.99 RICompactDisklib

```
Function: CDTrackMins
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: num=CDTrackMins[(offset)]

  Returns the current time from start of the track for the currently
playing cd track.  The optional parameter offset can take the value of 0 or
1.  IF offset=1 is passed, the time returned will reflect the playing time
from the start of the compact disc, rather than from the start of the
track.
```

1.100 RICompactDisklib

```
Function: CDTrackSecs
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: num=CDTrackSecs[(offset)]

  Returns the current time from start of the track for the currently
playing cd track.  The optional parameter offset can take the value of 0 or
1.  IF offset=1 is passed, the time returned will reflect the playing time
from the start of the compact disc, rather than from the start of the
track.
```

1.101 RICompactDisklib

```
Function: CDTrackPlaying
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: num=CDTrackPlaying

  Returns the number of the currently playing cd track.
```

1.102 RICompactDisklib: Command Index

```
                 Command index for library RICompactDisklib

                                Library Main

                           Number of commands: 22

            CDDoor                              CDFastForward
            CDFirstTrack                        CDFlush
            CDLastTrack                         CDNormalSpeed
            CDNumTracks                         CDPause
            CDPlayTrack                         CDReadTOC
            CDRewind                            CDSpeed
            CDStatus                            CDStop
            CDTrackLength                       CDTrackMins
            CDTrackPlaying                      CDTrackSecs
            CDUpdateInfo                        CDVolume
            CloseCD                             OpenCD
```

1.103 RICopperFXLib

```
-----------------------------------------------------------------------------
----                RI CopperFX Library V1.3 (C)94/95                   ----
-----------------------------------------------------------------------------

                        Written By Stephen McNamara
                        ©94/95 Leading Edge Software

                               Command Index

This is a library of commands that assist in setting it custom copperlists
for your blitz mode games.  It interfaces with the display library and so
can only be used in conjunction with CopList objects.  The commands in this
library insert copper instructions into the custom space in a Coplist
object - you must therefore have custom space in your CopList if you want
to use them.
Custom space is given to the coplist object during initialisation - it is
the last parameter of the InitCopList command.

AGA warning: Three of the commands in this library are AGA only
             (A1200/A400/CD32).  They should not be used on non-AGA
             machines.

Command list:

    CopperReset coplist#,startline[,ccoffset]
    DoColSplit cols_adr,numlines,colour_register
    RedoColSplit cols_adr,numlines,ccoffset
    CopperEnd
    CopperInfoBlock
    CopperCommand copins1,copins2
    CopperMove register,value
    CopperWait x,y
    CopperSkip x,y
    GetCCOffset
    CopperAGACol
```

1.104 RICopperFXLib

```
Statement: CopperReset
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: CopperReset coplist#,startline[,ccoffset]

This command sets up the copper library to work on a certain coplist
object. It must be used before you can use any of the commands in this
library.  coplist# is the number of the coplist you want to effect,
startline is the vertical start position to store (for the commands
DoColSplit and RedoColSplit).  The optional ccoffset parameter allows you
to specify an offset into the custom area of the copperlist as a start
position for the library.  The ccoffset parameter is given in the form of
the number of copper instructions from the start of the custom area.
```

1.105 RICopperFXLib

```
Statement/Function: DoColSplit
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: DoColSplit cols_adr,numlines,colour_register

  This command is AGA only at the moment.  What it does is produce a nice
aga fade going down the screen.  The colours to fade from/to are given in
the form of 6 longwords, the address of which is pointed to by cols_adr.
The following structure could be used to store the colours:

    Newtype.colourinfo
      r1.l
      g1.l
      b1.l
      r2.l
      g2.l
      b2.l
    End Newtype

You would then assign a variable to be of type .colourinfo, and set the
colour values in it.  It would then be passed to the DoColSplit command
using the & operator to pass the address of the variable:

    Deftype.colourinfo cols
    cols\r1=0,0,0,255,255,255
    DoColSplit &cols,256,0

The split will start at the current y counter value (set by CopperReset)
```

```
and will go on for numlines vertical lines.  It will effect the colour
register supplied, which maybe any aga register.  The Y counter will be
moved down to the end of the colour split after this command has finished,
meaning that you can do multiple splits one after the other easily.
```

1.106 RICopperFXLib

```
Statement/Function: RedoColSplit
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: RedoColSplit cols_adr,numlines,cc_offset

This command must be used after the DoColSplit.  What it allows you to do
is quickly update the colour information set up by the DoColSplit command
without rebuilding the whole colour split.  The parameters are the same
except that cc_offset replaces the colour register parameter.  For this
command to work, you must start it at the same custom address as the
DoColSplit was started at.  This parameter is for you to pass the address
to start at too the library.  An easy way to do this is to store the
current cc_offset BEFORE calling DoColSplit:

    pos.w=GetCCOffset
    DoColSplit &cols,256,0
    ;
    ; Change colours values in cols variable here!
    ;
    RedoColSplit &cols,256,pos
```

1.107 RICopperFXLib

```
Statement/Function: CopperEnd
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: CopperEnd

This command is used to tidy up the copperlist after you have finished
adding custom commands.  It is necessary if you're ever executing any WAIT
commands (including DoColSplit) after vertical position 255.  After this
position extra code is required to make sure the CopList display terminated
properly.  If you don't use it after going over 255 vertically, you will
get screen corruption in your display.
```

1.108 RICopperFXLib

```
Statement/Function: CopperInfoBlock
--------------------------------------------------------------------------
Modes : Amiga/Blitz
```

```
Syntax: ad.l=CopperInfoBlock

Returns the address of the internal library information.  This command is
primarily for debugging by me.  The data held within the structure is
private, and no assumptions should be made about it by the user of this
library.
```

1.109 RICopperFXLib

```
Statement: CopperCommand
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: CopperCommand copins1,copins2

This command allows you to manually insert copper instructions into the
current set coplist object.  The copper instruction is given as two words
which are stored straight into the coplist.
```

1.110 RICopperFXLib

```
Statement: CopperMove
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: CopperMove register,value

This command allows you to insert a move instruction into the copperlist.
The first parameter should be a hardware register address (given as an
offset from $0), the second should be a value to move into it.  The value
parameter must be a word.
```

1.111 RICopperFXLib

```
Statement: CopperWait
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: CopperWait x,y

This command allows you to insert a wait instructino into the copperlist.
The horizontal and vertical position to wait for are given by x,y.  The
copper has a horizontal resolution though of 4 low resolution pixels,
thus your x coordinate will be rounded down to the nearest multiple of 4.
```

1.112 RICopperFXLib

```
Statement: CopperSkip
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: CopperSkip x,y

This command allows you to insert a wait instructino into the copperlist.
The horizontal and vertical position to wait for are given by x,y.  The
copper has a horizontal resolution though of 4 low resolution pixels,
thus your x coordinate will be rounded down to the nearest multiple of 4.
```

1.113 RICopperFXLib

```
Function: GetCCOffset
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: offset=GetCCOffset

Gets the current custom copper instruction offset.  Used if you want to
keep track of how far through your custom area you are, or in conjunction
with Do/RedoColSplit.  The return value is the number of instructions from
the start of the custom area.
```

1.114 RICopperFXLib

```
Statement: CopperAGACol
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: CopperAGACol register,r,g,b

Setting AGA colours is a pain in the arse.  This instruction though allows
you to do it easily in your copperlist by doing all the extra work for you.
Just supply the colour register number to move the data into and the r,g,b
values.  This command generates 4 copper instructions inside your
copperlist.
```

1.115 RICopperFXLib: Command Index

```
                  Command index for library RICopperFXLib

                                Library Main

                           Number of commands: 11

           CopperAGACol                        CopperCommand
           CopperEnd                           CopperInfoBlock
           CopperMove                            CopperReset
           CopperSkip                            CopperWait
           DoColSplit                            GetCCOffset
           RedoColSplit
```

1.116 RIDebugLib

```
------------------------------------------------------------------------------
====                  RI Debug Library V1.21 (C)1994/5                    ====
------------------------------------------------------------------------------

                        Written By Stephen McNamara
                      ©1994/5 Leading Edge Software

                               Command Index

This library is an extension for the Blitz Basic runtime error debugger by
Leading Edge Software.  It allows your program to give the debugger a set
of simple instructions that are invaluable whilst debugging a program.  They
can only be used in conjunction with version 1.9+ of Blitz Basic 2, and the
updated Acid library debug.obj.

You should note that these commands can *ONLY* be used in amiga mode since
they require the debugger to immediately respond to them.  When in Blitz
mode, multitasking is disabled so the debugger is unable to react to the
commands.  When compiling, Blitz will tell you if you try and use the
commands in Blitz mode.

Additional commands in this library require the related update of the
debugger.  Currently this libraries version number is 1.21, you should have
a debugger version greater than or equal to this number.


A note about variable tracing
-----------------------------

Variable tracing is only performed whilst the debugger is either single
stepping a blitz program, or TRACING a program.  When a program is running
on its own, no update of any windows in the debugger is performed.

Command list:
               AddvarTrace variable,variable$,display_mode
               DelVarTrace variable$
               VarTraceWindow
               DisAsmWindow

New commands:
               CopperTrace address[,offset]

New commands (V1.21):
    ProcControl On/OFf
```

1.117 RIDebugLib

```
Statement : AddVarTrace
------------------------------------------------------------------------------
Modes  : Amiga
Syntax : AddVarTrace var,variable$,display_mode

This command adds a variable trace to the debuggers list of traces.  The
parameter 'var' is the actual variable to add to the list, variable$ is the
name which will be printed in the variable window in the debugger (usually
the same as the variable name) and display_mode is the prefered output mode
for the variables value.

The string variable$ will be displayed inside the variable trace window.
This will normally be the name of your variable, but on occasion you might
want some extra info with the name.  In these cases, you can make the
variable$ anything you like, for example "a (counter)" means that we're
tracing variable a but we want to remember that is being used as a counter
inside the program.

The output mode can take the following values, depending of course on the
type of variable:

Bytes/Words/Longs:        0=nocare (default output will be selected)
                          1=decimal
                          2=hexadecimal
                          3=binary

Quicks/Floats:            0=nocare
                          1=decimal

Strings:                  0=nocare (defaults to no length/maxlen data)
                          1=no length/maxlen data
                          2=length/maxlen data displayed

The command will automatically work out the 'type' of your variable and
ensure that the proper output mode is selected.

You should note that you can add the same variable more than once if you
like.  This will be useful if you want to display a variables value in more
than output mode.  For example, you could display the byte sized variable
MYVAR in both decimal and hexadecimal by 'adding' it twice.
```

1.118 RIDebugLib

```
Statement : DelVarTrace
------------------------------------------------------------------------------
Modes  : Amiga
Syntax : DelVarTrace variable$

This command instructs the debugger to remove a variable, identified by the
string variable$, from its trace list.  The debugger will look for the name
variable$ and delete *ALL* occurences of this name from the list.  If you
```

added the variable trace with a different name from the actual name of the
variable, you must ensure that the variable$ matchs that which you used to
add the variable.

1.119 RIDebugLib

```
Statement : VarTraceWindow
-------------------------------------------------------------------------------
Modes  : Amiga
Syntax : VarTraceWindow
```

This command instructs the debugger to open its variable trace window.
This can save the bother of going to the debugger separately and opening
the window yourself.

1.120 RIDebugLib

```
Statement : DisAsmWindow
-------------------------------------------------------------------------------
Modes  : Amiga
Syntax : DisAsmWindow
```

This command instructs the debugger to open its disassembly window. The
disassembly window will open at the address of the command following
DisAsmWindow. This can be helpful in cases like statements/functions that
are totally assembly since you cannot evaluate the address of a label thats
inside a statement/function.

1.121 RIDebugLib

```
Statement : CopperTrace
-------------------------------------------------------------------------------
Modes  : Amiga
Syntax : CopperTrace address[,offset]
```

This command instructs the debugger to open its copper window. If the
offset parameter is passed with the command, the library assumes that
'address' points to a coplist object (e.g. address=addr coplist(0)), it
then adds the offset and takes the longword at that address as the start
position for the window. Thus, if you wanted to open the copper window
right at the start of coplist 0 you'd do:

```
    CopperTrace Addr Coplist(0),4
```

See the coplist object in the debugger for more information about offsets.

1.122 RIDebugLib

```
Statement : ProcControl
------------------------------------------------------------------------------
Modes  : Amiga
Syntax : ProcControl On/Off

  This command allows you to switch the debuggers procedure control on or
off.  If on, the debugger will not step/trace inside of statements and
functions.  Instead it will execute them as single commands.
  This command is actually the same as toggling the gadget on the debugger
screen.
```

1.123 RIDebugLib: Command Index

```
                    Command index for library RIDebugLib

                                Library Main

                           Number of commands: 6

                               AddVarTrace
                               CopperTrace
                               DelVarTrace
                               DisAsmWindow
                               ProcControl
                               VarTraceWindow
```

1.124 RIEncryptLib

```
-----------------------------------------------------------------------------
====                RI Encrypt Library V1.2 (C)1994      ====
-----------------------------------------------------------------------------

                        Written By Stephen McNamara
                        ©1994 Leading Edge Software

                               Command Index

Commands in the library:

    Encrypt memadr,len[,wheel1,wheel2,wheel3]
    value.w=GetWheel(n)
    Decrypt memadr,len,wheel1,wheel2,wheel3
```

1.125 RIEncryptLib

```
Statement: Encrypt
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: Encrypt memadr,len[,wheel1,wheel2,wheel3]

  This will encrypt a block of memory starting at the address and running
through to addresslength-1.  The optional wheel parameters allow you to
specify the start positions of the three wheels.  If you leave these out
then the wheels' start positions will be randomised.
```

1.126 RIEncryptLib

```
Function: GetWheel
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: value=GetWheel(n)

  This will tell you the position that wheel n stopped at after encrypting
a file.  n can range from 1 to 3 - YOU MUST REMEMBER THESE POSITIONS IF
YOU WANT TO DECRYPT THE FILE (at the moment at least).
```

1.127 RIEncryptLib

```
Statement: Decrypt
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: Decrypt memadr,len,wheel1,wheel2,wheel3

    Same Encrypt except that it does the opposite and the wheel positions
ARE NOT OPTIONAL.  The positions should be the ones you wrote down after
encrypting the file.


>> END
```

1.128 RIEncryptLib: Command Index

```
                   Command index for library RIEncryptLib

                                Library Main

                           Number of commands: 3

                                  Decrypt
                                  Encrypt
                                  GetWheel
```

1.129 RIFNSLib

```
----------------------------------------------------------------------------
====                    RI FNS Library V1.0(C)1994/5       ====
----------------------------------------------------------------------------

                        Written By Stephen McNamara
                        ©1994 Leading Edge Software

                               Command Index
```

This Blitz2 library prints proportional fonts in either Amiga or Blitz mode. It uses my own (rather primitive) font file format, details of which can be found at the end of this text file. Fonts can be upto 64 pixels wide and any height (although the font editor is limited to 64 pixels at the present moment). Fonts can be output in upto 256 colours (AGA!) and in the following ways: bold, centred, underlined, right-aligned or just standard left-aligned.

Note: a default font (PERSONAL.8) is built into this library and can be used by simply using font number 0. You do not have to install this font, it is automatically available for your use. A second point is to make is that the library is set up with a clipping rectangle of 0,0 to 0,0. Thus you have to use either FNSClip, FNSClipOutput or FNSOutput (with the optional clip parameter) to set the clipping rectangle before you try to print anything.

Please feel free to critisise (or praise!) this library, send me anything you want to say about it at:

      SIS3149@SIS.PORT.AC.UK

These are all the FNS library commands:

```
      InstallFNS
      RemoveFNS
      FNSClip
      FNSClipOutput
      FNSSetTab
      FNSInk
      FNSOrigin
      FNSOutput
      FNSPrint
      FNSHeight
      FNSLength
      FNSLoad
      FNSSlot
      FNSUnderline
      FNSVersion
      FNSWidth
      FNSPrefs
      FNSUnload
```

Note: All return values will be words except when using InstallFNS and
      FNSVersion.

```
                                Control Codes
                                =============

The FNS library now supports an additional control code for a return
character (Ascii 10). You can now print, using this control code, multiple
lines of text in one go. If you have special print options on, for example
centering, then separate lines of text will automatically be centered
below each other.

Example usage:

  a$="Hello to all you people"+chr$(10)+"out there!"
  FNSPrefs %1,1
  FNSPrint 0,160,100,a$

  This will print "Hello to all you people" and "out there!" on separate
  lines of the destination bitmap. Both lines will be centered.

The control code to changeing ink colour during line printing is still
the same (Ascii 1). See the section on FNSPrint for more
information about it.


FNS Font file format:
=====================

Header: 256 bytes.
  0-3   : 'FNS.' - file identifier - looked for by InstallFNS
  4-5   : height of font (#word)
  6-7   : width of font in multiples of 16 (#word)
  8-9   : underline position (offset from top of font, #word)
  10-11 : size of data for each font character
    [ (WIDTH/8) * height ]
  32-255: byte giving widths of each character in the font.
    These bytes doesn't really hold the width, rather
    they hold the value to add to the X position of the
    character to get to the position to print the next
    character at (!).

  256-EOF:character data starting at ASCII 32 (space)
```

1.130 RIFNSLib

```
Statement: FNSSetTab
-------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: FNSSetTab tab_width

Use this command to set the tab spacing used when printing. The value
given should be the spacing IN pixels.
```

1.131 RIFNSLib

```
Function: FNSLoad
---------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: suc=FNSLoad (filename$,font#)

This command is used to load a font from disk and automatically install
it for use by the FNS commands.  Filename$ should be the full name of
the file to load (path$+file$) and font# should be 0<= and >=15.  This
command returns a value of -1 for failure or the font number the font
was installed as (see InstallFNS).  A failure could either be a load
error or an installation error.

You should make sure that the file you load IS an FNS font file.

IMPORTANT NOTE: to use this command, you must have
our RIAM*S library installed on your copy of Blitz2.
Running it without this library could, and probably will, cause a major
crash of your computer.

Also note that if you do an ERASEALL (RIAM*S library command for
erasing banks), you will DELETE your font from memory!
```

1.132 RIFNSLib

```
Statement: FNSUnLoad
---------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: FNSUnLoad font#

This command is used to remove a font installed with the FNSLoad
command. When this command runs it automatically removes the font
entry in the FNS commands and deletes the memory that the font file is
held in. There is no need to do this at the end of a program as the
RIAM*S library automatically frees up all allocated
memory.
```

1.133 RIFNSLib

```
Function: FNSSlot
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: address.l=FNSSlot

FNSSlot returns the adres of 16 longwords. These longwords are the actual
adresses of fonts in memory. This command is really just for testing
purposes.
```

1.134 RIFNSLib

```
Function: InstallFNS
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: font_num.b=InstallFNS(font_num.b,address.l)

This is used to install a font so that it is available for use by
the output routines.  Font_num should be a number >=0 and <=15,
address should be the address in memory of the FNS font file.
This function will check that the address given does contain a FNS
font (it will look for the header 'FNS.'), if it cannot find the font
or something else goes wrong it will return a 0 to you, otherwise it
will return the number the font was installed as.

Note: The font number you give is automatically ANDED with $F when you
      call this function, thus if you supply a number greater that 15
      you could actually overwrite a previously installed font.

See: RemoveFNS
```

1.135 RIFNSLib

```
Statement: RemoveFNS
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: RemoveFNS font#

This command simply removes an installed font from the list of font
held internally by the FNS routines.  There is no real need to remove
fonts as installing fonts takes up no memory, except of course the
actual font data.  You do not need to remove FNS fonts before ending a
program.

See: InstallFNS
```

1.136 RIFNSLib

```
Statement: FNSPrint
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: FNSPrint font_num.b,x.w,y.w,a$/string_address
        [,preferences,colour]

This command prints the string a$ in an FNS font at the position X,Y.
Font_num is the number of a previously installed FNS font, the output
of this command is sent to the current FNS bitmap (see FNSOutput). You
can setting a drawing rectangle on the currently used bitmap to limit
the output of the font - see FNSClip for more info.
```

```
Instead of a string, though, you can give the address of a null
terminated string in memory.  Also, you can change the colour that text
is being output in in the current string by putting the character ASCII 1
followed by a byte value from 0-255 specifying the colour to change to.

The optional parameters are for controlling how the text is output.
They automatically overide the default setting but are not permanent,
i.e. the default output style and colour are restored after the line
has been output.  Use FNSInk and FNSPrefs to set the default font
output mode.

  See: FNSOuput,FNSInk,FNSPrefs,FNSOrigin,FNSClip
```

1.137 RIFNSLib

```
Statement: FNSOutput
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: FNSOutput bitmap#[,clip_update]

This command selects a bitmap for use by the FNS routines, the bitmap
must be a previously reserved Blitz 2 bitmap object.  After this
command all FNS font printing will occur on the selected bitmap.  The
optional parameter allows you to update the clipping rectangle for
output at the same time as setting the output bitmap.  Setting
clip_update to a non-zero value will cause the clipping area to
automatically be set to the dimensions of the selected bitmap.

NOTE:
-----
This command MUST be used before you attempt to use FNSPrint.
The maximum depth of the bitmap for printing is 8 bitplanes since this
is all Blitz 2 currently supports.

See: FNSClip,FNSClipOutput
```

1.138 RIFNSLib

```
Statement: FNSInk
-------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: FNSInk colour#

This sets the output colour for the FNS font drawing routines. The
number range is dependant on the depth of the destination bitmap, the
max posible range, though, is limited to 0 to 255 colours.  The FNS
output routines will attempt to draw in all the bitplanes of the
selected bitmap, any extra bits in the ink colour will be ignored.

See: FNSPrefs
```

1.139 RIFNSLib

```
Statement: FNSPrefs
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: FNSInk preferences[,colour#]

This sets the output prefs for the FNS font drawing routines but at
the same time also sets the colour for the FNS routines (optional).
At the moment the following options are available, the bits of the
preferences byte are used to select the different options:

                            bit 0: Centred text
                            bit 1: Bold text
          bit 2: Underline
          bit 3: Right aligned

See: FNSInk,FNSPrint,FNSLength
```

1.140 RIFNSLib

```
Function: FNSHeight
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: height.w=FNSHeight(font_num)

This routine returns the height of a previously installed FNS font.
Font_num should be >=0 and <=15.

See: FNSUnderline,FNSWidth
```

1.141 RIFNSLib

```
Function: FNSUnderline
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: under_pos=FNSUnderline(font_num)

This routine returns the underline position of the selected FNS font.
Font_num should be >=0 and <=15.

See: FNSHeight,FNSWidth
```

1.142 RIFNSLib

```
Function: FNSWidth
---------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: width.w=FNSWidth(font_num)

This routine returns the width in multiples of 16 of the selected FNS
font.  Font_num should be >=0 and <=15.

See: FNSHeight,FNSUnderline
```

1.143 RIFNSLib

```
Statement: FNSClip
---------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: FNSClip x1,y1,x2,y2

This command is used to limit the output of the FNSPrint command.  The
co-ordinates given should describe a rectangle that is to be used to
clip the output.  This rectangle can be thought of as a window on the
bitmap - no printing can occur outside of the window.
X1,Y1 are the top left corner of the clipping rectangle and X2,Y2 are
the bottom right corner.  Please note that both X co-ordinates should be
multiples of 16 and that X2 should be the heightest multiple of 16 that
you do not wish output to occur at.  Thus if your bitmap is 320x256 then
you would use the following to set the clipping rectangle to the full
bitmap:
    FNSClip 0,0,320,256

See: FNSClipOutput,FNSOutput
```

1.144 RIFNSLib

```
Statement: FNSClipOutput
---------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: FNSClipOutput

This command is used to quickly set the clipping rectangle for the FNS
commands to the full size of a bitmap.

See: FNSClip,FNSOutput
```

1.145 RIFNSLib

```
Statement: FNSOrigin
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: FNSOrigin [x,y]

This command is used to set an origin co-ordinate for printing output.
Whenever you use FNSPrint, the origin co-ordinates are added (as words)
to the co-ordinates you give for output.  I.e. setting the origin at
100,0 and printing at co-ordinates 0,0 will cause the output to be at
100,0.

Using this command without any parameters will cause the origin to
be reset to the position 0,0.
Note: This command does not affect the use of the FNSClip command.
```

1.146 RIFNSLib

```
Function: FNSLength
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: a=FNSLength (font#,a$[,prefs])

This command is equivalent of the basic command a=len(a$) except that
it returns the x size, in pixels, of the string if it were to be printed
in the font font#.  The optional preferences parameter allows you to
adjust the output of the string, if you specify no preferences then this
function will use the previously selected preferences to calculate the
string length.  Using preferences allows you to account for things like
bold text output.

See: FNSPrefs
```

1.147 RIFNSLib

```
Function: FNSVersion
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: a.q=FNSVersion

This command allows you to test the version number of the FNS library
that your program is being compiled with.  It returns a quick float
value and so you should use a quick float variable for the answer.  This
doc file was written for version 1.0 of the library.


FNS Font file format:
=====================

Header: 256 bytes.
```

```
  0-3   : 'FNS.' - file identifier - looked for by InstallFNS
  4-5   : height of font (#word)
  6-7   : width of font in multiples of 16 (#word)
  8-9   : underline position (offset from top of font, #word)
  10-11 : size of data for each font character
    [ (WIDTH/8) * height ]
  32-255: byte giving widths of each character in the font.
    These bytes doesn't really hold the width, rather
    they hold the value to add to the X position of the
    character to get to the position to print the next
    character at (!).

  256-EOF:character data starting at ASCII 32 (space)
```

1.148 RIFNSLib: Command Index

```
                     Command index for library RIFNSLib

                                Library Main

                           Number of commands: 18

            FNSClip                                 FNSClipOutput
            FNSHeight                               FNSInk
            FNSLength                               FNSLoad
            FNSOrigin                               FNSOutput
            FNSPrefs                                FNSPrint
            FNSSetTab                               FNSSlot
            FNSUnderline                            FNSUnLoad
            FNSVersion                              FNSWidth
            InstallFNS                              RemoveFNS
```

1.149 RIFxLib

```
-----------------------------------------------------------------------------
====               RI FX Library V1.2 (C)1994       ====
-----------------------------------------------------------------------------

             Written By Stephen McNamara (help from Steven Matty)
                        ©1994 Leading Edge Software

                                Command Index

Note: The library has had a lot of the commands inside it expanded so that
they work on any size bitmap.  At the moment the following, though, will
only work on lorez bitmaps: ZoomX8, Derez and ZoomXY

None of the commands in this library use the blitter chip.
Also note that the maximum bitmap depth for these functions is 8.

Command list:
    FadeInBitmap source#,dest#,delay[,offset1,offset2,height]
    ClearBitmap source#,delay[,offset,height]
    ZoomX2 source#,dest#,add_source,add_dest,width,height
    ZoomX4 source#,dest#,add_source,add_dest,width,height
    ZoomX8 source#,dest#,add_source,add_dest,width,height
    addval.w=ADDValue(bitmap#,x,y)
    InitZoomXY source#,dest#,add_source,add_dest
    ZoomXY xzoom_value,yzoom_value,height
    Derez source#,dest#,add_source,add_dest,derez_value,height


This two commands have been removed from this library to reduce its size.
If you need or want these commands then just mail me or Steve and we'll
sort something out for you.
(Slow)    PlanarToChunky bitmap_addr,dest_address,width,height,depth
(Slow)    ChunkyToPlanar source_address,bitmap_addr,width,height,depth
```

1.150 RIFxLib

```
Statement: FadeInBitmap
------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: FadeInBitmap source#,dest#,delay[,offset1,offset2,height]

  This is used to make an any width, any height, bitmap appear on another
  one in a nice way.  Source# and dest# should be bitmap object numbers
   and delay is the 'slow-down' value for the fade.  This is necessary
  because this routine works very fast - at full speed it looks just like
  a slow screen copy.  You should note that the delay is taken as being a
  word, thus don't pass 0 or you'll actually get a delay of 65535.  This
  routine will adjust itself to take into account the depth of the bitmap,
  WARNING: the depth of the destination bitmap should be AT LEAST as big
  as the depth of the source# bitmap because the depth of the fade is taken
  from the source# bitmap.
  The optional parameters in this command allow you to set respectively:
  the source bitmap y offset, the destination bitmap y offset and the
  height of the fade (in pixels).  If these parameters are left out then
  the fade automatically occurs across the full size of the bitmap.

  See: ClearBitmap
```

1.151 RIFxLib

```
Statement: ClearBitmap
------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: ClearBitmap source#,delay[,offset,height]

  This is used to clear an any width, any height, bitmap in a very pleasant
  way.  The parameters are the same as for FadeInBitmap except that
  only one bitmap is needed.  The delay parameter i used for the same
  reason as in FadeInBitmap - to slow down the effect.  The optional
  parameters allow you to set a y start value for the clear and the
  height (in pixels) of the clear.

  See: FadeInBitmap
```

1.152 RIFxLib

```
Statement: ZoomX2
========================================================================
Modes : Amiga/Blitz
Syntax: ZoomX2 source#,dest#,add_source,add_dest,width,height

  This command does a very fast X2 zoom.  It works with two bitmaps - one
  source and one dest (note: these can be the same bitmap but you should
  be careful that the zoom is not done over the source data).  The two
  parameters add_source and add_dest allow you to specify the position of
  the start of the zoom, they specified as byte offsets from the top left
  corner of the bitmaps (byte 0).  These values can be calculated by the
  following method:

    add_source=(Y x BITMAP_WIDTH (in bytes) + (X / 8)

  or by using the built in command ADDValue.  Width and height are both
  specified in pixels.

  NOTE: There is no clipping on this command - be careful not to zoom off
        the edges of bitmaps.
  You can zoom from a bitmap to a different size bitmap BUT the
  destination bitmap must be as deep as the source and big enough
  to hold the zoomed data.

  See: ZoomX4, ZoomX8 and ADDValue
```

1.153 RIFxLib

```
Statement: ZoomX4
========================================================================
Modes : Amiga/Blitz
Syntax: ZoomX4 source#,dest#,add_source,add_dest,width,height

  This is exactly the same as ZoomX2 except that a times 4 zoom is done
  by this command.

  Note: You can zoom from a bitmap to a different size bitmap BUT the
  destination bitmap must be as deep as the source and big enough
  to hold the zoomed data.

  See: ZoomX2, ADDValue
```

1.154 RIFxLib

```
Statement: ZoomX8
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: ZoomX8 source#,dest#,add_source,add_dest,width,height

  This is exactly the same as ZoomX2 except that a times 8 zoom is done
  by this command

  See: ZoomX2, ADDValue
```

1.155 RIFxLib

```
Function: ADDValue
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: addval.w=ADDValue(bitmap#,x,y)

  This function can be used the calculate the add_source and add_dest
  values used in all the zoom commands.  Just give the bitmap number, x
  co-ordinate and the y co-ordinate and you'll get an answer back that can
  be used straight in the ZoomXn commands.

  See: ZoomX2, ZoomX4, ZoomX8 and ZoomXY
```

1.156 RIFxLib

```
Statement: InitZoomXY
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: InitZoomXY source#,dest#,add_source,add_dest

  This command initialises the ZoomXY routine to the bitmaps you want it
  to work on.  You MUST use this routine before calling ZoomXY.  The
  parameters are the same as the first four parameter for the ZoomXn
  commands - source and dest bitmaps and add_source/dest values.

  See: ZoomXY
```

1.157 RIFxLib

```
Statement: ZoomXY
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: ZoomXY xzoom_value,yzoom_value,height

  This command does a zoom based on the values you give it.  You should
  note, though, that zoom values should be integer values (no fractional
  part).  The height is the height in pixels that the source data should be
  zomed to.  Please note that this command is different to the other zoom
  commands in that the output of it is clipped to fit inside 320 pixels.

  This command should only be used after InitZoomXY has been called.
  This routine has an extra feature in that if you give both zoom values
  as 1 then a bitmap copy is done from the source to the dest using the
  offsets given and the height.

  See: InitZoomXY
```

1.158 RIFxLib

```
Statement: Derez
------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: Derez source#,dest#,add_source,add_dest,derez_value,height

  This command is used to derez a low resolution bitmap onto another one.
  The bitmaps are source# and dest#, add_source and add_dest are used to
  control the start position of the derez (see ZoomX2 and ADDValue to see
  how these are calculated).  The derez value if obviously the amount that
  each pixel will be derezed to in both the x and y directions, the height
  is the height of the derez - the derez is clipped to fit inside this in
  the y direction and inside 320 pixels in the x direction.
  This routine has an extra feature in that if you give derez_value as 1
  then a bitmap copy is done from the source to the dest using the offsets
  given and the height.
```

1.159 RIFxLib

```
Statement: ReduceX2
------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: ReduceX2 source#,dest#,add_source,add_dest,width,height

  This command halves the given rectangle of one bitmap and pastes it onto
the destination bitmap.  Width should be a multiple of 16, width and height
should describe a rectangular area that will be reduced (these values
should be in pixels).

  See ZoomX2 and other commands for more information about the syntax of
this command.
```

1.160 RIFxLib: Command Index

```
                          Command index for library RIFxLib

                                    Library Main

                                Number of commands: 10
                                    ADDValue
                                    ClearBitmap
                                    Derez
                                    FadeInBitmap
                                    InitZoomXY
                                    ReduceX2
                                    ZoomX2
                                    ZoomX4
                                    ZoomX8
                                    ZoomXY
```

1.161 RIGfxLib

```
------------------------------------------------------------------------------
====                RI GFX Function Library V1.2 (C)1994     ====
------------------------------------------------------------------------------

                Written By Stephen McNamara & Steven Matty
                        ©1994 Leading Edge Software

                               Command Index

This library contains commands for the control of palette objects inside
Blitz2.  These are just simple commands that allow either interrogation of
the palette objects are modifications to the colour values contained in
them.  After changing the palette with these commands, you’ll have to do
either a USE PALETTE or DISPLAYPALETTE (whichever is applicable to what
you’re doing) to make the changes come into effect on your screen.

Or send us anything you’ve written........


Command list:
      PaletteInfo palette#
      r.w=PalRed (Colour#)
      g.w=PalGreen (Colour#)
      b.w=PalBlue (Colour#)
      r.w=AGAPalRed (Colour#)
      g.w=AGAPalGreen (Colour#)
      b.w=AGAPalBlue (Colour#)
      PalAdjust dest_palette#,ration.q[,start_col,end_col]
      FillPalette palette#,r,g,b[start_col,end_col]
      AGAFillPalette palette#,r,g,b[start_col,end_col]

New commands:
      [suc=]CopyColour source_pal#,dest_pal#,source_col#,dest_col#
      [suc=]SaveCMAP palette#,filename$
      CPUCls bitmap#
```

1.162 RIGfxLib

```
Statement: PaletteInfo
------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: PaletteInfo Palette#

  This command is used to specify the palette object that all palette
interrogations should look at.  The majority of the commands use this
palette object as the source for their data, e.g. PalRed(1) will look at
the red value of colour 1 of the palette last used in a PaletteInfo
command.
```

1.163 RIGfxLib

```
Function: PalRed
------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: r.w=PalRed (Colour#)

  This command is used to get the red value of colour number Colour#.  You
should use the PaletteInfo command to specify what palette this command
takes its information from.
  The value returned will be from 0 to 15
```

1.164 RIGfxLib

```
Function: PalGreen
------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: g.w=PalGreen (Colour#)

  This command is used to get the green value of colour number Colour#.
You should use the PaletteInfo command to specify what palette this command
takes its information from.
  The value returned will be from 0 to 15
```

1.165 RIGfxLib

```
Function: PalBlue
------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: b.w=PalBlue (Colour#)

  This command is used to get the blue value of colour number Colour#.  You
should use the PaletteInfo command to specify what palette this command
takes its information from.
  The value returned will be from 0 to 15
```

1.166 RIGfxLib

```
Function: AGAPalRed
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: r.w=AGAPalRed (Colour#)

  This command is used to get the red value of colour number Colour#.  You
should use the PaletteInfo command to specify what palette this command
takes its information from.
  The value returned will be from 0 to 255, this number of shades, though,
can only be displayed on an AGA machine.
```

1.167 RIGfxLib

```
Function: AGAPalGreen
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: g.w=AGAPalGreen (Colour#)

  This command is used to get the green value of colour number Colour#.
You should use the PaletteInfo command to specify what palette this command
takes its information from.
  The value returned will be from 0 to 255, this number of shades, though,
can only be displayed on an AGA machine.
```

1.168 RIGfxLib

```
Function: AGAPalBlue
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: b.w=AGAPalBlue (Colour#)

  This command is used to get the blue value of colour number Colour#.  You
should use the PaletteInfo command to specify what palette this command
takes its information from.
  The value returned will be from 0 to 255, this number of shades, though,
can only be displayed on an AGA machine.
```

1.169 RIGfxLib

```
Statement: PalAdjust
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: PalAdjust dest_palette#,ration.q[,start_col,end_col]

  This command is used to multiple all the colours, or a range of colours,
in a palette object, by a ratio.  The dest_palette# arguement is used to
```

```
give a destination for the adjusted colour information.  This destination
should be a pre-reserved palette and should be AT LEAST as big and the
source palette.  The source palette is taken as being the palette last used
in the PaletteInfo command.
  The ratio should be given as either a quick value or a float and should
be below one for a fade or above to lighten a palette.  If you give a ratio
of 1 then a palette copy will occur.
  The optional start and end parameters let you specify the range of
colours to adjust.  Only this range of colours, though, will be adjusted
and stored in the destination palette.
```

1.170 RIGfxLib

```
Statement: FillPalette
------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: FillPalette palette#,r,g,b[start_col,end_col]

  This command lets you fill a given palette object with specific r,g,b
values.  The values given should be between 0 to and 15.  Optionally, you
can give start and end colour numbers to set a range for the fill.  You
should be careful, though, because when you specify a range, no checking is
done (at the moment) to make sure that you don't exceed the colour limit of
the palette.
  You should note that this command does not work on the palette last
PaletteInfo'ed.
```

1.171 RIGfxLib

```
Statement: AGAFillPalette
------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: AGAFillPalette palette#,r,g,b[start_col,end_col]

  This command is identical to FillPalette except that it lets you specify
AGA shade values for the r,g,b parameters.
  See FillPalette for more information.
```

1.172 RIGfxLib

```
Statement/Function: CopyColour
------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: [suc=]CopyColour source_pal#,dest_pal#,source_col#,dest_col#

  This will attempt to copy a colour entry in a palette to another entry,
which can be in a separate palette or the same.  If used as a function,
then it will return -1 for success, or 0 for failure.  The command fails if
either of the colour numbers is out of the range of the relevant palette.
```

1.173 RIGfxLib

```
Statement/Function: SaveCMAP
--------------------------------------------------------------------------
Modes : Amiga
Syntax: [suc=]SaveCMAP palette#,filename$

  This command will save out the given palette as an IFF file, with just a
BMHD and CMAP.  This file can be loaded into graphics packages like DPaint.
It will return -1 for success in saving, or 0 for failure.
```

1.174 RIGfxLib

```
Statement: CPUCls
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: CPUCls bitmap#

  Does a clear of a bitmap using the CPU.  This command, unlike the Acid
command Cls, only clears to colour 0.  On accerelated machines, though, it
out performs the Cls instruction.


>> END
```

1.175 RIGfxLib: Command Index

```
                     Command index for library RIGfxLib

                                Library Main

                           Number of commands: 13

            AGAFillPalette                      AGAPalBlue
            AGAPalGreen                         AGAPalRed
            CopyColour                          CPUCls
            FillPalette                         PalAdjust
            PalBlue                             PaletteInfo
            PalGreen                            PalRed
            SaveCMAP
```

1.176 RIListLib

```
-----------------------------------------------------------------------------
====               RI LinkList Library V0.91ß (C)1995     ====
-----------------------------------------------------------------------------

                        Written By Steven Matty
                      ©1995 Leading Edge Software

                             Command Index

NOTICE:
=======

This library is currently in the very EARLY testing stage, so PLEASE do not
rely on the stability/continuity in the future. It can and probably WILL change
in the future. Especially dont rely on the structure of the linked list nodes
to remain the same - they have already changed since V0.9ß.

All commands can be optionally used as functions, eg. suc=<command>(params)
(EXCEPT: AddrListItem)

WARNING!
========
Do NOT use list numbers of 5 or higher! Your machine will crash if you try.
Future version will check for this.


Command List:

    InitList list#,var
    AddListItem list#
    DelListItem list#
    SetListItem list#,var
    GetListItem list#,var
    NextListItem list#
    PrevListItem list#
    FirstListItem list#
    LastListItem list#
    AddFirstListItem list#
    AddLastListItem list#
    ad.l=AddrListItem(list#)
```

1.177 RIListLib

```
Statement/Function: InitList
------------------------------------------------------------------------------
Mode   : Amiga/Blitz
Syntax : InitList list#,var

This command will initialize a linked list for your use. It will also
add the first node (which will contain the contents of <var>).
list# must be less than five. This command optionally returns 0 for failure,
or -1 for success.
```

1.178 RIListLib

```
Statement/Function: AddListItem
-------------------------------------------------------------------------------
Mode   : Amiga/Blitz
Sytnax : AddListItem list#

This command will attempt to add another item onto your list at the current
position. This command optionally returns 0 for failure (no list initalized or
not enough RAM) or -1 for success.
```

1.179 RIListLib

```
Statement/Function: DelListItem
-------------------------------------------------------------------------------
Mode   : Amiga/Blitz
Syntax : DelListItem list#

This command will attempt to delete the current item from your list. If the
current item is the ONLY item then this command will fail. (To be fixed in
later version). -1 means success.
```

1.180 RIListLib

```
Statement/Function: SetListItem
-------------------------------------------------------------------------------
Mode   : Amiga/Blitz
Syntax : SetListItem list#,var

This command will store the contents of <var> into the current node (created
by AddNode). -1 means success.
```

1.181 RIListLib

```
Statement/Function: GetListItem
-------------------------------------------------------------------------------
Mode   : Amiga/Blitz
Syntax : GetListItem list#,var

This command will store the contents of the current list item into your variable
specified by <var>. -1 means success
```

1.182 RIListLib

```
Statement/Function: NextListItem
--------------------------------------------------------------------------------
Mode   : Amiga/Blitz
Syntax : NextListItem list#

This command will move you onto the next item in your list. 0 indicates
failure (no list/end of list) or -1 for success.
```

1.183 RIListLib

```
Statement/Function: PrevListItem
--------------------------------------------------------------------------------
Mode   : Amiga/Blitz
Syntax : PrevListItem list#

This command will move you back to the previous item in your list. 0 indicates
failure (no list/start of list) or -1 for success.
```

1.184 RIListLib

```
Statement/Function: FirstListItem
--------------------------------------------------------------------------------
Mode   : Amiga/Blitz
Syntax : FirstListItem list#

This command will take you to the first item in your list. 0 indicates failure
(no list) or -1 for success.
```

1.185 RIListLib

```
Statement/Function: LastListItem
--------------------------------------------------------------------------------
Mode   : Amiga/Blitz
Syntax : LastListItem list#

This command will take you to the last item in your list. 0 indicates failure
(no list) or -1 for success.
```

1.186 RIListLib

```
Statement/Function: AddFirstListItem
--------------------------------------------------------------------------------
Mode   : Amiga/Blitz
Syntax : AddFirstListItem list#

This command will insert a node to the front of your list. 0 indicates failure
(no list/no RAM) or -1 for success.
```

1.187 RIListLib

```
Statement/Function: AddLastListItem
--------------------------------------------------------------------------------
Mode   : Amiga/Blitz
Syntax : AddLastListItem list#

This command will add a node to the end of your list. 0 indicates failure (no
list/no RAM) or -1 for success.
```

1.188 RIListLib

```
Function: AddrListItem
--------------------------------------------------------------------------------
Mode   : Amiga/Blitz
Syntax : ad.l=AddrListItem(list#)

This function will return the address of the current node in your list.



Version Details
===============

Version 0.9 (22/1/95)
---------------------
First Release. No Docs.


Version 0.91 (29/1/95)
----------------------
Modified format of nodes to mirror exec lists. Should be ok to use
in Gadtools Listviews etc...(UNTESTED).
Added AddrListItem command which returns address of current item.
```

1.189 RIListLib: Command Index

```
                    Command index for library RIListLib
```

```
                                Library Main

                           Number of commands: 12

           AddFirstListItem                     AddLastListItem
           AddListItem                          AddrListItem
           DelListItem                          FirstListItem
           GetListItem                          InitList
           LastListItem                         NextListItem
           PrevListItem                         SetListItem
```

1.190 RIPackLib

```
-----------------------------------------------------------------------------
====                RI Pack Library V1.2 (C)1994       ====
-----------------------------------------------------------------------------

                Written By Stephen McNamara & Steven Matty
                        ©1994 Leading Edge Software

                               Command Index

This library contains commands for the unpacking of ILBM's (IFF pictures)
and the grabbing of their palettes (CMAP chunks).  Nearly all the commands
in this library can be used as either STATEMENTS or FUNCTIONS.  Usage is
identical in both cases but if used as a function then the command will
return:
    FALSE for failure
    TRUE for success


Command list:
      UnpackIFF address.l,bitmap#[,lines,offset]
      suc=UnpackIFF (address.l,bitmap#[,lines,offset])
      ILBMPalette address.l,palette#
      suc=ILBMPalette (address.l,palette#)
      ILBMGrab address.l,bitmap#,palette#
      LoadIFF filename$,bitmap#[,palette#]
      suc=LoadIFF (filename$,bitmap#[,palette#])
      DeIce source_address,dest_address
      suc=DeIce (source_address,dest_address)
      val.l=ChunkHeader (A$)
```

1.191 RIPackLib

```
Statement/Function: UnpackIFF
-------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: UnpackIFF address.l,bitmap#[,lines,offset]
  suc=UnpackIFF (address.l,bitmap#[,lines,offset])

  This command is used to unpack an IFF picture file from memory onto a
```

bitmap. Address.l should point to the START of the iff file header in memory (either CHIP or FAST mem can be used), bitmap should be the number of a previously initialised bitmap. The optional lines parameter allows you to specify the number of lines to unpack from the IFF file.
  This command checks the size of the bitmap against the size of the IFF before it unpacks the IFF onto it. Checks are made for width, height and depth of the bitmap and the following is done:

(size=WIDTH, HEIGHT and DEPTH)

```
    BITMAP 'size' < IFF 'size' : unpack aborted
    BITMAP 'size' = IFF 'size' : pic is unpacked
    BITMAP 'size' > IFF 'size' : pic is unpacked
```

  Extra aborts can be caused by:
    - not using a previously installed bitmap
    - given the optional lines parameter as 0 or less
    - not giving ADDRESS.l as a pointer to a valid IFF ILBM header

  When using the optional parameters, you should note that if you try to unpack more lines than the IFF has, the unpack routine will automatically stop at the last line of the IFF. It will not reject the UnpackIFF command. Also note that the offset is a byte offset from the start of the bitplanes. You can use the AddValue command to calculate this value.

  NOTE: you should save your IFF pictures with the STENCIL OFF because at the moment this routine does not check to see if STENCIL data is present in the IFF file.

1.192 RIPackLib

```
Statement/Function: ILBMPalette
-----------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: ILBMPalette address.l,palette#
  suc=ILBMPalette (address.l,palette#)
```

  This command is used to grab the palette from a IFF picture file held in memory (CHIP or FAST mem). Address.l should be given as the address of either an IFF file in memory or a CMAP chunk in memory. When you use the SAVE PALETTE command from inside an art program (e.g. DPaint) or from inside Blitz2, the program saves out a CMAP chunk which gives details about the palette. The CMAP chunk is also saved with IFF picture files to give the palette of the picture.
  This command will look at the address you gave and try and find a CMAP chunk from the address given to address+5120. If it finds a chunk it will grab the palette into the given palette object. If the palette object already contains palette information then this information is deleted. This routine looks in the CMAP chunk and reserves the palette object to have the same number of colour entries.
  This command will fail if it doesn't find a CMAP chunk.

1.193 RIPackLib

```
Statement: ILBMGrab
--------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: ILBMGrab address.l,bitmap#,palette#

  This command lets you grab both the palette and the graphics from an IFF
picture file with just one command.  It returns to success parameter to
say whether or not it succeeded in grabbing the data, so if you need to know
if the grabbing was successful you'll have to use the separate commands
for grabbing palettes and graphics.

  NOTE: this command essentially just calls both UnpackIFF and ILBMPalette
so everything said about these commands is relevent for ILBMGrab.
```

1.194 RIPackLib

```
Statment/Function: LoadIFF
--------------------------------------------------------------------------
Modes : Amiga
Syntax: LoadIFF filename$,bitmap#[,palette#]
  suc=LoadIFF (filename$,bitmap#[,palette#])

  This command is a direct replacement for Blitz2's LoadBitmap.  It is a
lot faster than Blitz's command since it loads the file into memory and
then unpacks it from there.  Thus you need to ensure that you have enough
free memory to load the IFF into before trying to use this command.
  This command is also more stable than Blitz's since it checks for the
existence of the file before trying to load it in.
  The optional parameter allows you to load in the palette of the IFF
picture.  Refer to UnpackIFF and ILBMPalette for more information about
unpacking the graphics and grabbing the palettes.

IMPORTANT NOTE: to use this command you must have our FUNC library
installed in your copy of Blitz2.  Use of this command without this library
will probably lead to a bad crash of your Amiga!
```

1.195 RIPackLib

```
Statement/Function: DeIce
--------------------------------------------------------------------------
Modes : Amiga
Syntax: DeIce source_address,dest_address
  suc=DeIce (source_address,dest_address)

  This is a command from my (Stephen McNamara) past.
  It is used to unpack data files packed by my favourite Atari ST packer -
PACK ICE v2.40.  I've put it into Blitz because still have loads of files
that I've packed with it.  To use it, source_address should (obviously)
```

```
contain the address of the data, dest_address should be where to unpack the
data to.  In the function form, this command returns either 0 for unpack
failed or -1 for success.
  Note: The size of the data unpacked is the long word at source_address+8
(I think, or is it 4?) if anybody is interested......
```

1.196 RIPackLib

```
Function: ChunkHeader
------------------------------------------------------------------------
Modes : Amiga
Syntax: val.l=ChunkHeader (A$)

  This command was put in by me (Stephen McNamara) before I realised Blitz
already had a command that does exactly the same.  I've left it in just
because I want to.  It is useful when looking through IFF files for chunks
(e.g. ILBM, CMAP, etc.) as it gives you a longword value to look for in
memory to find the chunk.  The string should be a four character string
(e.g. CMAP), you'll be returned the longword value of the string.
  This command does the job of the following bit of Blitz2 code:

    a$="CMAP"
    val.l=Peek.l(&a$)


>> END
```

1.197 RIPackLib: Command Index

```
                    Command index for library RIPackLib

                                Library Main

                           Number of commands: 6

                                ChunkHeader
                                DeIce
                                ILBMGrab
                                ILBMPalette
                                LoadIFF
                                UnpackIFF
```

1.198 RIReqLib

```
-----------------------------------------------------------------------------
====                   RI REQ Library V0.9 (C)1994       ====
-----------------------------------------------------------------------------

                                   Written By Steven Matty
```

©1994 Leading Edge Software

Command Index

The well known Req.Library for the Amiga is one of the best file requesters around, so I wrote this small lib to enable Blitz users to have Req requesters in their programs with the minimum of hassle.

* PLEASE NOTE * That this library must have at least v2.2 of the Req.Library available.

Command List:

```
  ReqOutput()
  ReqFileRequest()
  ReqFileLoc()
```

1.199 RIReqLib

```
Statement: ReqOutput
--------------------------------------------------------------------------------
Modes  : Amiga
Syntax : ReqOutput windownumber
```

This command sets the ReqLib.library to put all requesters onto the window specified by <windownumber>. If this command is not called then the requesters will appear on the Default Public Screen.

1.200 RIReqLib

```
Function: ReqFileRequest
--------------------------------------------------------------------------------
Modes  : Amiga
Syntax : pathname$=ReqFileRequest([title$[,flags]])
```

This opens up the standard file requester. If <title$> is given then the text will appear on the requester title bar.
The optional <flags> parameter specifies a flag setting (see below) for use. If this is omitted then the last flag setting is used.

1.201 RIReqLib

```
Function: ReqFileLoc
--------------------------------------------------------------------------------
Modes  : Amiga/Blitz
Syntax : memorylocation.l=ReqFileLoc
```

This simply returns the address in memory where the Req.Library file requester stucture is located.

FLAGS
=====

Below is a list of possible flag settings and a brief description of each.

```
#FRQSHOWINFOB     = %1   ;Set to show .info files. Default is not.
#FRQEXTSELECTB    = %10   ;Extended select.  Default is not.
#FRQCACHINGB      = %100   ;Directory caching.  Default is not.
#FRQGETFONTSB     = %1000   ;Font requester rather than a file requester.
#FRQINFOGADGETB   = %10000   ;Hide-info files gadget.
#FRQHIDEWILDSB    = %100000   ;DON'T want 'show' and 'hide' string gadgets.
#FRQABSOLUTEXYB   = %1000000   ;Use absolute x,y positions rather than centering  ←
   on mouse.
#FRQCACHEPURGEB   = %10000000   ;Purge the cache whenever the directory date stamp ←
    changes if this is set.
#FRQNOHALFCACHEB  = %100000000   ;Don't cache a directory unless it is completely  ←
   read in when this is set.
#FRQNOSORTB       = %1000000000   ;DON'T want sorted directories.
#FRQNODRAGB       = %10000000000  ;DON'T want a drag bar and depth gadgets.
#FRQSAVINGB       = %100000000000  ;Are selecting a file to save to.
#FRQLOADINGB      = %1000000000000  ;Are selecting a file(s) to load from.
#FRQDIRONLYB      = %10000000000000  ;Allow the user to select a directory, rather ←
    than a file.
```

STRUCTURE
=========

Below is a description of the Req.Library file requester structure.

```
STRUCTURE AFileRequester,0
  UWORD frq_VersionNumber   ;MUST BE REQVERSION!!!!!!!!!!!!!!!!!

  ;You will probably want to initialize these three variables.
  APTR  frq_Title     ; Hailing text
  APTR  frq_Dir       ; Directory array (must be DSIZE+1 characters long)
  APTR  frq_File      ; Filename array (must be FCHARS+1 characters long)
  ;If you initialize this variable then the file requester will place the complete ←
      path name in here on exit.
  APTR  frq_PathName      ; Complete path name array - (must be DSIZE+FCHARS+2  ←
     long)
  ;If you want the file requester to pop up on your custom screen, put one of your ←
      window pointers here.
  ;Or better yet, you can leave this field zeroed and put a pointer to one of your ←
      windows in the
  ;pr_WindowPtr field in your process structure.
  APTR  frq_Window      ; Window requesting or NULL
  ;Initialize these to the number of lines and columns you want to appear in the  ←
     inner window that
  ;displays the file names.  If you leave these set to zero then default values  ←
     will be used.
```

```
UWORD frq_MaxExtendedSelect   ; Zero implies a maximum of 65535, as long as ↩
   FRQEXTSELECT is set.
UWORD frq_numlines      ; Number of lines in file window.
UWORD frq_numcolumns      ; Number of columns in file window.
UWORD frq_devcolumns      ; Number of columns in device window.
ULONG frq_Flags     ; Various - umm - flags.  See above for more info.
UWORD frq_dirnamescolor ;These five colors will all default
UWORD frq_filenamescolor  ;to color one if you don't specify
UWORD frq_devicenamescolor  ;a color (ie; if you specify color zero).
UWORD frq_fontnamescolor  ;If you want color zero to be used, specify
UWORD frq_fontsizescolor  ;color 32, or some other too large number
        ;which mods down to zero.

UWORD frq_detailcolor   ;If both of these colors are specified as
UWORD frq_blockcolor    ;zero then the block pen will be set to one.

UWORD frq_gadgettextcolor ;The color for the text of the five boolean gadgets.   ↩
   Defaults to 1.
UWORD frq_textmessagecolor  ;The color for the message at the screen top.   ↩
   Defaults to 1.
UWORD frq_stringnamecolor ;The color for the words Drawer, File, Hide and Show.  ↩
    Defaults to 3.
UWORD frq_stringgadgetcolor ;The color for the borders of the string gadgets.   ↩
   Defaults to 3.
        ;Unfortunately it is not possible to specify
        ;the color of the actual text in an Intuition
        ;string gadget.
UWORD frq_boxbordercolor  ;The color for the boxes around the file and directory ↩
    areas.  Defaults to 3.
UWORD frq_gadgetboxcolor  ;The color for the boxes around the five boolean  ↩
   gadgets.  Defaults to 3.

STRUCT  frq_RFU_Stuff,36    ;This area, which is reserved for
          ;future use, should all be zero.

STRUCT  frq_DirDateStamp,ds_SIZEOF  ; A copy of the cached directories date  ↩
   stamp.
          ; There should never be any need to change this.

UWORD frq_WindowLeftEdge;   ;These two fields are only used when the
UWORD frq_WindowTopEdge;    ;FRQABSOLUTEXY flag is set.  They specify
          ;the location of the upper left hand
          ;corner of the window.

UWORD frq_FontYSize     ;These fields are used to return the selected
UWORD frq_FontStyle     ;font size and style, only applicable when the
          ;font bit is set.

;If you set the extended select bit and the user extended selects, the list of  ↩
   filenames will start from here.
APTR  frq_ExtendedSelect    ; Linked list of ESStructures if more than one  ↩
   filename is chosen.
;All of the following variables you shouldn't need to touch.  They contain  ↩
   fields that the file
;requester sets and likes to preserve over calls, just to make life easier for  ↩
   the user.
STRUCT  frq_Hide,WILDLENGTH+2   ; Wildcards for files to hide.
```

```
  STRUCT  frq_Show,WILDLENGTH+2   ; Wildcards for files to show.
  WORD  frq_FileBufferPos   ; Cursor’s  position  and first
  WORD  frq_FileDispPos     ; displayed character number in
  WORD  frq_DirBufferPos    ; the three string gadgets.  No
  WORD  frq_DirDispPos      ; need  to initialized these if
  WORD  frq_HideBufferPos   ; you don’t want to.
  WORD  frq_HideDispPos
  WORD  frq_ShowBufferPos
  WORD  frq_ShowDispPos

;         The  following  fields are PRIVATE!  Don’t go messing with them or
; wierd  things may/will happen.  If this isn’t enough of a warning, go read
; the one in intuition.h, that should scare you off.

  APTR  frq_Memory      ; Memory allocated for dir entries.
  APTR  frq_Memory2     ; Used for currently hidden files.
  APTR  frq_Lock      ; Contains lock on directories being read across calls.
  STRUCT  frq_PrivateDirBuffer,DSIZE+2  ; Used for keeping a record of which
          ; directory we have file names for.
  APTR  frq_FileInfoBlock
  WORD  frq_NumEntries
  WORD  frq_NumHiddenEntries
  WORD  frq_filestartnumber
  WORD  frq_devicestartnumber
  LABEL frq_SIZEOF
```

Enjoy!

Steve.

1.202 RIReqLib: Command Index

```
                     Command index for library RIReqLib

                                Library Main

                          Number of commands: 3

                               ReqFileLoc
                               ReqFileRequest
                               ReqOutput
```

1.203 RISortLib

```
------------------------------------------------------------------------------
----               RI String Sort Library V1.3 (C)94/95                   ----
------------------------------------------------------------------------------


                        Written By Stephen McNamara
                        ©94/95 Leading Edge Software
```

```
                                Command Index

This library allows you to sort a linked list of items.  It works
only with linked lists, and at present can only sort items into alphabetical
order based on a string in the item.
The sorting routine used in this library is very simple and crude.  This
library should not be used to sort in speed critical situations due to the
inefficiency of the sorting method.  The library will, though, be fast
enough for most situations.


Command list:
    StringSort linkedlist(),sizeof.type[,offset]
    =ListBase (linkedlist)
    StringSortItem linkedlist(),sizeof.type[,offset]
    StringSortDir direction
```

1.204 RISortLib

```
Statement: StringSort
------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: StringSort linkedlist(),sizeof.type[,offset]

This is the basic sort command.  Its first parameter is a linked list, the
second is the sizeof each item in this list (e.g. the size of they type or
newtype that each item is).  The optional offset parameter allows you to
specify an offset into each item, this offset should be the offset for the
string you want to sort by.  If the offset parameter is missing, an offset
of 0 will be assumed.

This command sorts the whole of the linked list, starting from the very
first item.

Example:

    Newtype.listitem
      pad.w
      text$
    End Newtype

    Dim List myitems.listitem(10)

    AddItem myitems() : myitems()\text="Hello"
    AddItem myitems() : myitems()\text="World"

    ;Sort list myitems(), string is offset 2 from start of type
    StringSort myitems(),SizeOf.listitem,2

    ResetList myitems()
    While NextItem(myitems())
      NPrint myitems()\text
    Wend
    MouseWait
    End
```

1.205 RISortLib

```
Function: ListBase
------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: ad.l=ListBase(linkedlist())

This command returns the base address of the linked list supplied.  This
address holds data for the linked list, and pointers to the first item and
current item in the list.  This command will not be of any use to most
people, rather it is included for debugging purposes.
```

1.206 RISortLib

```
Statement: StringSortItem
------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: StringSortItem linkedlist(),sizeof.type[,offset]

This is basically the same command as StringSort except that this command
sorts the linked list from the *current* list item rather than the first
list item.  Thus it can be used to only sort a part of a list.  Apart from
this the command is the same as StringSort.
```

1.207 RISortLib

```
Statement: StringSortDir
------------------------------------------------------------------------
Modes : Amiga/Blitz
Syntax: StringSortDir direction

Set the direction of sorting.  A direction of zero causes strings to be
sorted into ascending order (smallest to largest), non-zero selects
descending order (largest to smallest).
```

1.208 RISortLib: Command Index

```
                    Command index for library RISortLib

                                Library Main
```

```
                              Number of commands: 4

                                  ListBase
                                  StringSort
                                  StringSortDir
                                  StringSortItem
```

1.209 RIToolTypesLib

```
----------------------------------------------------------------------------
====               RI ToolTypes Library V1.2 (C)1994       ====
----------------------------------------------------------------------------

                           Written By Stephen McNamara
                           ©1994 Leading Edge Software

                                  Command Index

This library contains commands to allow the reading, comparing and setting
of tooltypes in a .info file.  All tooltype names are case insignificant
but as a general sort of rule they should really be completely uppercase.

This library attempts to open the system Icon.library, if the opening of
this library fails ALL commands in this library will be unusable.  Almost
every function in this library relies on the Icon.library completely.

Command list:
      GetIconObject filename$
      PutIconObject filename$[,icontype]
      FreeIconObject
      FindToolValue tooltype$
      FindToolNumber toolnum#
      MatchToolValue tooltype$,value$
      PutToolValue tooltype$,value$
      NewToolType tooltype$,value$
      ClearToolTypes

New commands for this version:
      SetIconHit width#,height#
      ShapeToIcon shape#[,shape#]
      SetIconType type#
      IconRender value#
      IconDefaultTool tool$
      bool=FindToolType(tool$)

Changed commands:
      FindToolValue - now returns "" if the tooltype was
  found but did not have a value (e.g. DONOTWAIT).  You should
  now use FindToolType to check for the existance of a tooltype
  and then use FindToolValue to get its value.
      PutIconObject - now has an optional parameter that  lets you set
  the type of the file.  See SetIconType for more information
  about possible values for this command.
```

1.210 RIToolTypesLib

```
Statement/Function: GetIconObject
-----------------------------------------------------------------------
Modes  : Amiga
Syntax : GetIconObject filename$
   suc.l=GetIconObject (filename$)

  This command reads in a .info file from disk.  The filename given will
have '.info' added to the end of it and will be loaded into memory (chip or
fast depending on what is available for allocation) as a diskobject.
Please refer to the Amiga hardware includes for information about the
diskobject structure (or see your Blitz Basic Amigalibs resident file).

  If used as a function, this command will return either FALSE for failure
or the address of the allocated diskobject in memory.
```

1.211 RIToolTypesLib

```
Statement/Function: PutIconObject
-----------------------------------------------------------------------
Modes  : Amiga
Syntax : PutIconObject filename$[,icontype]
   suc.l=PutIconObject (filename$)

  This command takes a diskobject structure reserved and initialised by
GetIconObject and saves it out to disk as a .info file for the specified
file.  All current tooltypes and values will be saved with the file.
  The optional parameter allows you to set the type of the file associated
with the .info file.  See SetIconType for possible values for this
parameter.  Note that if you leave out this parameter the icontype will not
be changed.
```

1.212 RIToolTypesLib

```
Statement/Function: FreeIconObject
-----------------------------------------------------------------------
Modes  : Amiga
Syntax : FreeIconObject
   suc.l=FreeIconObject

  This command will free up the diskobject that is currently being used.
It will not save out any tooltype changes and will free up the memory
without ANY changes being made to the .info file loaded from disk.
  All changes will be lost when you use this command!
```

1.213 RIToolTypesLib

```
Function: FindToolValue
------------------------------------------------------------------------
Modes  : Amiga
Syntax : toolval$=FindToolValue(tooltype$)

  This function returns the value of the selected tooltype.  The return
value is a string, and is the part of the tooltype string after the "=" in
the tooltype entry.  The tooltype$ string that you pass can be in either
lower case or uppercase since all testing in done in uppercase, although as
a general rule, all tooltypes should be in uppercase.
  This function will return a null string if the named tooltype was not
found in the list of tooltypes for the file.  If the selected tooltype did
not have an actual value (e.g. DONOTWAIT) then this function will also
return a null string - you can though use a combination of this command and
FindToolType to cover this situation.
```

1.214 RIToolTypesLib

```
Function: FindToolNumber
------------------------------------------------------------------------
Modes  : Amiga
Syntax : toolval$=FindToolNumber(tooltype$)

  This command will return the FULL tooltype string in the selected
tooltype position.  If the tooltype number does not exist then "" will be
returned.

  Example:  tooltypes: "DONOTWAIT"
         "CLOCKX=157"

  FindToolNumber(0) will return "DONOTWAIT"
  FindToolNumber(1) will return "CLOCKX"
  FindToolNumber(49) will return ""
```

1.215 RIToolTypesLib

```
Function: MatchToolValue
------------------------------------------------------------------------
Modes  : Amiga
Syntax : suc.l=MatchToolValue(tooltype$,value$)

  This command searchs the current list of tooltypes for the selected
tooltype and, if found, attempts to match the values of it with the given
value.  This command uses the operating system call MatchToolType(), it
is able to cope with a tool having more than one value,

      e.g. LANGUAGE=ENGLISH|FRENCH
      (the | is used to show OR, thus this tooltype
       means that LANGUAGE equals ENGLISH or FRECH)
    When using match toolvalue with this tooltype, TRUE will be
    returned when you use value$="ENGLISH" or "FRENCH" but not
    (I think) both.

  You should note that for this command, the case of VALUE$ is
insignificant.
```

1.216 RIToolTypesLib

```
Statement/Function: SetToolValue
------------------------------------------------------------------------
Modes  : Amiga
Syntax : SetToolValue tooltype$,value$
   suc.l=SetToolValue (tooltype$,value$)

  This command will attempt to set a tooltype that is currently defined to
the specified value.  When used as a function, this command will return
TRUE for success or FALSE for failure, possible failures include: no icon
file loaded and tooltype not found.  When used, this command attempts to
allocate memory to store the new tooltype information in, it does not
attempt to free up the old memory allocated to the tooltype.  This means
that you should keep alterations of tooltypes to a minimum.  The best way
to manage tooltypes is:

    1. Open the icon
    2. Read the tooltypes
    3. Close the icon
    4. ... do your program ...
    5. Open the icon
    6. Alter the tooltypes
    7. Save the icon

  Using this series of events, you'll keep memory usage (which will be
fairly small anyway...) to the very minimum.
```

1.217 RIToolTypesLib

```
Statement/Function: NewToolType
------------------------------------------------------------------------
Modes  : Amiga
Syntax : NewToolType tooltype$,value$
   suc.l=NewToolType (tooltype$,value$)

  This command allocates a new tooltype in the currently loaded .info file
and sets its value.  No check is done to see is the tooltype already
exists and the new tooltype is added to the end of the current list of
tooltypes.
```

1.218 RIToolTypesLib

```
Statement: ClearToolTypes
--------------------------------------------------------------------------
Modes  : Amiga
Syntax : ClearToolTypes

  This command is used to clear all the tooltype information from the
currently loaded .info file.  It does not attempt, though, to free up all
the memory reserved to store tooltype names and values, you should
therefore not used this command too many times in a row.  Once you have
used this command, any attempt to read tooltype values will fail.
```

1.219 RIToolTypesLib

```
Statement: SetIconHit
--------------------------------------------------------------------------
Modes  : Amiga
Syntax : SetIconHit width#,height#

  This command sets the size of the 'hit-box' around the image in the
currently loaded .info file.  This is only of use if your info file has an
image associated with it.  You should note that the hit box should never be
smaller, horizontally or vertically, than the actual size of the image.
  When Workbench renders an image for a file onto a window, it
automatically puts a 3d box border around it.  The size of the hit box
determines the size of this border.  Your image will always be located in
the top left border of the hit box.
```

1.220 RIToolTypesLib

```
Statement: ShapeToIcon
--------------------------------------------------------------------------
Modes  : Amiga
Syntax : ShapeToIcon shape#[,shape#]

  This command lets you change the images associated with the currently
loaded .info file.  What it does is to set up the .info file in memory so
that when it is saved out next, the images you give are saved out with it.
Using this command does not actually copy any shape data around memory, all
it does it place a pointer in the .info to the shape data.  You should
therefore not delete a shape WITHOUT first saving the .info file to disk
(that is of course if you want to keep your changes).
  When you use this command, the hit box area for the .info file is
automatically set to the size of the first shape given.  It is important,
therefore, that the second shape is not larger than the first.  When you
give a second shape, this shape is set up to be the 'alternate render'
image, this means that this is the second image associated with the .info
file (remember the two windows in the IconEditor?)
```

1.221 RIToolTypesLib

```
Statement: SetIconType
--------------------------------------------------------------------------
Modes  : Amiga
Syntax : SetIconType type#

  This command lets you specify the type of the file associated with the
currently loaded .info file.  The type describes whether or not the file is
a tool or project etc...., and can take the following values:

      1 Disk
      2 Drawer
      3 Tool
      4 Project
      5 Trashcan

  This command is identical to the menu in the IconEditor 'Type'.
```

1.222 RIToolTypesLib

```
Statement: IconRender
--------------------------------------------------------------------------
Modes  : Amiga
Syntax : IconRender mode#

  This command lets you specify what Workbench should do to the icons
image when the user clicks on it.  It lets you choose whether a separate
image should be displayed or whether the current image should just be
modified.  Mode# is made up of several different values that should be
added together to create different effects, these are:

      0 Complement the select box
      1 Draw a box around the image
      2 Draw the alternate image
      3 Don't highlight
      4 Double image icon

  Thus if you wanted an icon to change to a second image when selected, and
the icon has a second image, you would set the render to 6 (4+2).  This
would mean that you had a second image (4) and that you wanted it to be
displayed when you select the icon (2).
  Note: when you use ShapeToIcon with two shape numbers the IconRender is
automatically set to 6.
```

1.223 RIToolTypesLib

```
Statement: IconDefaultTool
--------------------------------------------------------------------------
Modes  : Amiga
```

```
Syntax : IconDefaultTool tool$

  This command lets you set the default tool for the current .info file.
The default tool only applies for project files (see SetIconType) and is
the program that is run when you double click the icon file (e.g. all
Blitz2 source code files saved out with icons have the default tool
'Blitz2:Blitz2').
  This command can be used to make a file saved out by your program
double-clickable.  I have used it myself to make map files saved out from
my editor automatically load the editor when selected.
```

1.224 RIToolTypesLib

```
Statement: FindToolType
--------------------------------------------------------------------------
Modes  : Amiga
Syntax : bool=FindToolType (tool$)

  This command simply returns true or false to say whether or not the given
tooltype was found in the currently loaded .info file.

>>END
```

1.225 RIToolTypesLib: Command Index

```
                  Command index for library RIToolTypesLib

                                Library Main

                           Number of commands: 15

           ClearToolTypes                      FindToolNumber
           FindToolType                        FindToolValue
           FreeIconObject                      GetIconObject
           IconDefaultTool                     IconRender
           MatchToolValue                      NewToolType
           PutIconObject                       SetIconHit
           SetIconType                         SetToolValue
           ShapeToIcon
```

1.226 RITrackDiskLib

```
-------------------------------------------------------------------------------
====              RI TrackDisk Library V1.2 (C)1994       ====
-------------------------------------------------------------------------------

                          Written By Steven Matty
                        ©1994 Leading Edge Software
```

```
                              Command Index

Command List:

    success=OpenDisk(unit#)
    MotorOn unit#
    MotorOff unit#
    CloseDisk unit#
    success=ReadSector(unit#,sector#,buffer[,numsectors])
    success=WriteSector(unit#,sector#,buffer[,numsectors])
    success=FormatTrack(unit#,track#,buffer[,numtracks])
    success=WriteBoot(unit#[,bootdata])
```

1.227 RITrackDiskLib

```
Statement/Function : OpenDisk
---------------------------------------------------------------------------
Modes  : Amiga
Syntax : success=OpenDisk(unit#)

This attempts to open unit 'unit#' of the trackdisk.device, for use with
the other Statement/Functions in this library. A return value of 0 indicates  ←
   failure,
-1 indicates success.
```

1.228 RITrackDiskLib

```
Statement : MotorOn
---------------------------------------------------------------------------
Modes  : Amiga
Syntax : MotorOn unit#

This attempts to switch the drive motor on of the previously opened
trackdisk unit (called with OpenDisk). You must call this Statement/Function
before attempting to ReadSector/WriteSector/FormatTrack/WriteBoot
```

1.229 RITrackDiskLib

```
Statement : MotorOff
---------------------------------------------------------------------------
Modes  : Amiga
Syntax : MotorOff unit#

This turns the drive motor of 'unit#' off.
```

1.230 RITrackDiskLib

```
Statement/Function : ReadSector
---------------------------------------------------------------------------
Modes  : Amiga
Syntax : [success=]ReadSector(unit#,sector#,buffer[,numsectors])

This attempts to read 'numsectors' sectors from a trackdisk device which
has been opened with OpenDisk and has its Motor On. If numsectors is
omitted then 1 sector is read. The data is read into the memory location
pointed to by 'buffer'.

WARNING! Please MAKE SURE the MOTOR is _ON_ otherwise, all hell will break
loose!!!
```

1.231 RITrackDiskLib

```
Statement/Function : WriteSector
---------------------------------------------------------------------------
Modes  : Amiga
Syntax : [success=]WriteSector(unit#,sector#,buffer[,numsectors])

This is the same as ReadSector except........... it writes!
(and no, I am not being lazy by not typing any decent docs)
```

1.232 RITrackDiskLib

```
Statement/Function : FormatTrack
---------------------------------------------------------------------------
Modes  : Amiga
Syntax : [success=]FormatTrack(unit#,track#,buffer[,numtracks])

This does a TD_FORMAT on the specified track number. Buffer should point
to the area of memory which the track should be formatted with. I don't
know why this Statement/Function exists - but hey, it might come in useful.
```

1.233 RITrackDiskLib

```
Statement/Function : WriteBoot
---------------------------------------------------------------------------
Modes  : Amiga
Syntax : [success=]WriteBoot(unit#[,buffer])

This writes 1k of data to the bootblock of the specified disk unit.
The optional buffer parameter should point to an area of memory with which
to write the bootblock.
```

1.234 RITrackDiskLib

```
Statement : CloseDisk
---------------------------------------------------------------------------
Modes  : Amiga
Syntax : CloseDisk unit#

This closes the trackdisk.device of the specified unit#. The Motor is
automatically switched off if it is already on.
```

1.235 RITrackDiskLib: Command Index

```
                  Command index for library RITrackDiskLib

                                Library Main

                           Number of commands: 8

                                CloseDisk
                                FormatTrack
                                MotorOff
                                MotorOn
                                OpenDisk
                                ReadSector
                                WriteBoot
                                WriteSector
```

1.236 RIZoneJoyLib

```
-----------------------------------------------------------------------------
====               RI ZoneJoy Library V1.5 (C)1994/95       ====
-----------------------------------------------------------------------------

                 Joystick Routines Written By Steven Matty
                 Zone Routines Written By Stephen McNamara
                        ©1994 Leading Edge Software

                               Command Index

This library contains commands for setting up zones and testing the status
of the joysticks attached to the Amiga.

New additions to this library allow you to have multiple lists of zones
(refered to as zonetables in this doc).  To maintain compatibility with
older versions of the library, zonetable 0 is equivalent of the original
list of zones used in the library.  You cannot adjust the size of zonetable
0 (its size is 256 zones), nor can you delete it.  The new zonetables can
be from 1 to 65536 in size, there are 16 available zonetable numbers.

All commands that change or test zones will work on the last zonetable that
was selected with the command UseZoneTable.  The default table is number 0.
```

```
Command list:
      ZoneInit [zone_num]|[start_zone,end_zone]
      Setzone zone#,x1,y1,radius
      Setzone zone#,x1,y1,x2,y2
      a.w=Zone(x,y)
      a.w=ZoneTest(start_zone[,end_zone],x,y)
      ad.l=ZoneTable
      size.l=ZoneTableSize (table#)
      UseZoneTable table#
      NewZoneTable table#,size
      FreeZoneTable table#

      jf.b=JFire(joy#)
      jv.b=JVert(joy#)
      jh.b=JHoriz(joy#)
      af.b=AllFire [bit_pattern]

Additions for this version (1.5):

  JAdaptorStatus On/Off
  x1=GetZoneX1(zone#)
  y1=GetZoneY1(zone#)
  x2=GetZoneX2(zone#)
  y2=GetZoneY2(zone#)
```

1.237 RIZoneJoyLib

```
Statement: ZoneInit
-------------------------------------------------------------------------
Modes  : Amiga/Blitz
Syntax : ZoneInit [zone_num]|[start_zone,end_zone]

  This command is used to clear any zones currently set.  The optional
  parameters allow you to select either a single zone or a range of zones
  to reset.
```

1.238 RIZoneJoyLib

```
Statement/Function: Setzone
-------------------------------------------------------------------------
Modes  : Amiga/Blitz
Syntax : Setzone zone#,x1,y1,radius
         Setzone zone#,x1,y1,x2,y2

  This command lets you set up zones for testing.  The first version is
  used when you want to set up a circular zone and the second when you want
  a rectangular one.  With rectangular zones, x1,y1 should be the top left
  corner of the rectangle and x2,y2 should be the bottom left.

  If used as a function, this command returns TRUE or FALSE to say whether
  or not the change was made.

  Note: The max zone number for zonetable 0 is 255.
  A zone number outside the range of the current table will
        cause this command to abort.
        Zones can be defined in any order.
        Circular zones are used in exactly the same way as rectangular
        ones.
```

1.239 RIZoneJoyLib

```
Function: Zone
------------------------------------------------------------------------
Modes  : Amiga/Blitz
Syntax : a.w=Zone(x,y)

  This command takes the co-ordinates x,y and checks to see if they are
  inside any of the defined zones.  The zones are searched in order,
  starting at 0 and going through to the size of the zonetable-1.  This
  command will return the first zone that the co-ordinates were found to
  be inside, you should note that both types of zones are tested
  (rectangular and circular).

  This command returns either -1 for not inside a zone or the zone number.
```

1.240 RIZoneJoyLib

```
Function: ZoneTest
------------------------------------------------------------------------
Modes  : Amiga/Blitz
Syntax : a.w=ZoneTest(start_num[,end_num],x,y)

  This command is the same as the Zone command except that it allows you
  to select either one individual zone to test or a range of zones.  You
  should, though, ensure that end_num if greater than start_num.

  This command returns either -1 for not inside a zone or the zone number.
```

1.241 RIZoneJoyLib

```
Function: ZoneTable
------------------------------------------------------------------------
Modes  : Amiga/Blitz
Syntax : ad.l=ZoneTable

  This function returns the address in memory of the zone information
  storage area for the current zonetable. The zones are stored one after
  the other, with each zone taking up 8 words (16 bytes) in the data area,
  making a total size of 2048 bytes.  They are stored in the following way:

    Rectangular:    +0: x1
        +2: y1
        +4: x2
        +6: y2

    Circular: +0: x1
        +2: y1
        +4: radius of zone
        +6: -1 <-- this is set to show that the
                   zone is circular.

    Undefined zone: +0: -1
        +2: -1
        +4: -1
        +6: -1

  The first longword (4 bytes) of the zonetable is used to hold the size,
in zones, of the table (thus the true size of the zonetable is 4+number of
zones*8).
```

1.242 RIZoneJoyLib

```
Function: ZoneTableSize
------------------------------------------------------------------------
Modes  : Amiga/Blitz
Syntax : size.l=ZoneTableSize

  This function returns the size, in zones, of the current zonetable.  It
is equivalent of doing: size.l=peek.l(ZoneTable).
```

1.243 RIZoneJoyLib

```
Statement/Function: NewZoneTable
------------------------------------------------------------------------
Modes  : Amiga/Blitz
Syntax : NewZoneTable table#,size

  This command will attempt to allocate a new zonetable with the given
table number.  If the table already exists it will be deleted.  The maximum
size for a zonetable is 65536 zones.  If used as a function, this command
will return FALSE for failure or TRUE for success.  You should note that
all zones are automatically reset in the new table and that creating a
table does not make it the current table, this must be done with
UseZoneTable.
  Valid zonetable numbers range from 0 to 15.

  IMPORTANT NOTE: you cannot define the size of zonetable 0.  You cannot
```

```
use this command to alter it in any way.
```

1.244 RIZoneJoyLib

```
Statement/Function: UseZoneTable
------------------------------------------------------------------------
Modes  : Amiga/Blitz
Syntax : UseZoneTable table#

  This command is used to change the current zonetable to the selected one.
If used as a function, it will return TRUE for success or FALSE for
failure.
  Valid zonetable numbers range from 0 to 15.
```

1.245 RIZoneJoyLib

```
Statement/Function: FreeZoneTable
------------------------------------------------------------------------
Modes  : Amiga/Blitz
Syntax : FreeZoneTable table#

  This command is used to free a zonetable from memory.  If used as a
function, it will return TRUE or FALSE.  When successfully called, this
command will free the zonetable and change the currently used zonetable to
table number 0.
  Valid zonetable numbers range from 0 to 15.

  IMPORTANT NOTE: you cannot free zone table 0.
```

1.246 RIZoneJoyLib

```
Function: JFire
------------------------------------------------------------------------
Modes  : Amiga/Blitz
Syntax : jf.b=JFire(joy#)

  This command tests the fire button status of the joystick joy#, where
  joy# is between 1 and 4.  You should note that, as with all the joystick
  commmands, joy#=1 refers to the Amiga's joystick port, joy#=2 refers to
  the mouse port, and joy#=3 or joy#=4 refer to the four player adapter
  ports.

  This command returns 0 for fire button not pressed or -1 for pressed
```

1.247 RIZoneJoyLib

```
Function: JHoriz
--------------------------------------------------------------------------
Modes  : Amiga/Blitz
Syntax : jh.b=JHoriz(joy#)

  This command is used to test the horizontal direction of the selected
  joystick.  It returns:

    0: No horizontal direction
          -1: Joystick left
         1: Joystick right
```

1.248 RIZoneJoyLib

```
Function: JVert
--------------------------------------------------------------------------
Modes  : Amiga/Blitz
Syntax : jv.b=JVert(joy#)

  This command is used to test the vertical direction of the selected
  joystick.  It returns:

    0: No vertical direction
          -1: Joystick up
         1: Joystick down
```

1.249 RIZoneJoyLib

```
Function: AllFire
--------------------------------------------------------------------------
Modes  : Amiga/Blitz
Syntax : af.b=AllFire [(bit_pattern)]

  This command is used to test the fire button status of all four
  joysticks.  It returns a byte with the first four bits giving the
  joystick status, false=fire button not pressed, true=fire button pressed.
  The following bits belong to joysticks:

    bit 0: joystick 1 (joystick port)
    bit 1: joystick 2 (mouse port)
    bit 2: joystick 3 (four player adaptor)
    bit 3: joystick 4 (four player adaptor)

  The optional bit pattern can be used to restrict the testing of the fire
  buttons.  If a bit in the pattern is clear (false) then the joystick it
  belongs to will not have its fire button tested,

    e.g. AllFire (%0011) will test joysticks 1 and 2 and return the
```

```
             result.  It will return false for joysticks 3 and 4.
```

1.250 RIZoneJoyLib

```
Statement/Function: JAdaptorStatus
------------------------------------------------------------------------
Modes  : Amiga/Blitz
Syntax : JAdaptorStatus On/Off
   oldstatus=JAdaptorStatus(On/Off)

This command toggles the reading of the four player adaptor for the
following commands:

    AllFire
    Jvert
    JHoriz
    JFire

When the status is off, these commands will return 0 when you attempt to
read from joysticks 3 and 4.  When on the testing will be performed
normally.  Default status for the adaptor is on.
```

1.251 RIZoneJoyLib

```
Function: GetZoneX1
------------------------------------------------------------------------
Modes  : Amiga/Blitz
Syntax : x1=GetZoneX1 (zone#)

This command returns the x start position for the specified zone in the
currently used zone table.  If the zone number supplied goes outside the
size of the zonetable, then this command returns -1.  It also returns -1 if
the zone is undefined.
```

1.252 RIZoneJoyLib

```
Function: GetZoneY1
------------------------------------------------------------------------
Modes  : Amiga/Blitz
Syntax : y1=GetZoneY1 (zone#)

This command returns the y start position for the specified zone in the
currently used zone table.  If the zone number supplied goes outside the
size of the zonetable, then this command returns -1.  It also returns -1 if
the zone is undefined.
```

1.253 RIZoneJoyLib

```
Function: GetZoneX2
--------------------------------------------------------------------------
Modes  : Amiga/Blitz
Syntax : x2=GetZoneX2 (zone#)

This command returns the x end position for the specified zone in the
currently used zone table.  If the zone number supplied goes outside the
size of the zonetable, then this command returns -1.  It also returns -1 if
the zone is undefined.

Note: For circular zones, this command will return the radius of the circle
squared.
```

1.254 RIZoneJoyLib

```
Function: GetZoneY2
--------------------------------------------------------------------------
Modes  : Amiga/Blitz
Syntax : y2=GetZoneY2 (zone#)

This command returns the y end position for the specified zone in the
currently used zone table.  If the zone number supplied goes outside the
size of the zonetable, then this command returns -1.  It also returns -1 if
the zone is undefined.

Note: For circular zones this command will always return -1


--------------------------------------------------------------------------

Version details:

27/1/95
  - V1.5
  - Fixed comparison prob in both 'circular:, changed BPL into
    BGE.
  - Fixed _zonetest000 - was getting x1,y,x2,y2 in wrong order
  - Fixed _zonetest020 circular zones - same prob as above
  - Added:
        GetZoneX1/X2/Y1/Y2 for zone interrogating...
25/1/95
  - Added JAdaptorStatus for disabling/enabling player
    3 & 4 joystick reading.  If disabled, commands will return
    0 for these joysticks.
  - Added fourplayer checking to AllFire.
23/11/94
  - BIG bug in ZoneInit2 - was moving #0 into (a1) instead
      of (a0)
3/9/94
  - Added 020 specific zone testing
  - Added commands ZoneMode and SetZoneMode (for 020 support)
  - Speed increase on Jfire: replaced branches and moveqs with
     SEQ
  - Improved jvert and jhoriz to remove inefficiency

>>END
```

1.255 RIZoneJoyLib: Command Index

```
                   Command index for library RIZoneJoyLib

                                Library Main

                           Number of commands: 18

            AllFire                                 FreeZoneTable
            GetZoneX1                               GetZoneX2
            GetZoneY1                               GetZoneY2
            JAdaptorStatus                          JFire
            JHoriz                                  JVert
            NewZoneTable                            Setzone
            UseZoneTable                            Zone
            ZoneInit                                ZoneTable
            ZoneTableSize                           ZoneTest
```

1.256 Library Index

```
                     Libraries included in database: 18

                       Total number of commands: 218

           RIAmosFuncLib                           RIAnimLib
           RIAppLib                                RICommoditiesLib
           RICompactDisklib                        RICopperFXLib
           RIDebugLib                              RIEncryptLib
           RIFNSLib                                RIFxLib
           RIGfxLib                                RIListLib
           RIPackLib                               RIReqLib
           RISortLib                               RIToolTypesLib
           RITrackDiskLib                          RIZoneJoyLib

                              Full Command List
```

1.257 Full Command List

```
                             Full Command List
                             -----------------

            AddAppIcon
            AddAppMenu
            AddAppWindow
```

```
AddFirstListItem
AddLastListItem
AddListItem
AddrListItem
ADDValue
AddVarTrace
AGAFillPalette
AGAPalBlue
AGAPalGreen
AGAPalRed
AllFire
AnimLoop
AppEvent
AppEventCode
AppEventID
AppFile
AppNumFiles
BlitterDone
BlitterNasty
BLoad
BSave
CDDoor
CDFastForward
CDFirstTrack
CDFlush
CDLastTrack
CDNormalSpeed
CDNumTracks
CDPause
CDPlayTrack
CDReadTOC
CDRewind
CDSpeed
CDStatus
CDStop
CDTrackLength
CDTrackMins
CDTrackPlaying
CDTrackSecs
CDUpdateInfo
CDVolume
ChunkHeader
ClearBitmap
ClearToolTypes
CloseCD
CloseDisk
CludgeShapes
CludgeSound
CommodityEvent
CopperAGACol
CopperCommand
CopperEnd
CopperInfoBlock
CopperMove
CopperReset
CopperSkip
CopperTrace
```

```
CopperWait
CopyByte
CopyColour
CopyLong
CopyWord
CPUCls
CxAppear
CxChangeList
CxDisable
CxDisAppear
CxEnable
CxKill
CxUnique
Decrypt
DeIce
DelAppIcon
DelAppMenu
DelAppWindow
DelListItem
DelVarTrace
Derez
DeviceName$
DisAsmWindow
DoColSplit
Encrypt
Erase
EraseAll
ExchangeAppear
ExchangeChangeList
ExchangeDisable
ExchangeDisAppear
ExchangeEnable
ExchangeKill
ExchangeMessage
ExchangeUnique
FadeInBitmap
FileSize
FillMem
FillPalette
FindToolNumber
FindToolType
FindToolValue
FindVolume
FirstListItem
FNSClip
FNSClipOutput
FNSHeight
FNSInk
FNSLength
FNSLoad
FNSOrigin
FNSOutput
FNSPrefs
FNSPrint
FNSSetTab
FNSSlot
FNSUnderline
```

```
FNSUnLoad
FNSVersion
FNSWidth
FormatTrack
FreeIconObject
FreeZoneTable
FuncLibVersion
GetCCOffset
GetIconObject
GetListItem
GetWheel
GetZoneX1
GetZoneX2
GetZoneY1
GetZoneY2
HotKeyHit
IconDefaultTool
IconRender
ILBMGrab
ILBMPalette
InitList
InitZoomXY
InstallFNS
JAdaptorStatus
JFire
JHoriz
JVert
KeyCode
LastListItem
Length
Lisa
ListBase
LoadIFF
MakeCommodity
MakeDir
MatchToolValue
Max/Min
MemFree
MotorOff
MotorOn
NewToolType
NewZoneTable
NextAppFile
NextBank
NextListItem
OpenCD
OpenDisk
PalAdjust
PalBlue
PaletteInfo
PalGreen
PalRed
PLoad
PrevListItem
ProcControl
PutIconObject
ReadSector
```