in

| COLLABORATORS | | | |
|---|---|---|---|
| | *TITLE* :<br><br>in | | |
| *ACTION* | *NAME* | *DATE* | *SIGNATURE* |
| WRITTEN BY | | August 6, 2022 | |

| REVISION HISTORY | | | |
|---|---|---|---|
| NUMBER | DATE | DESCRIPTION | NAME |
| | | | |

# Contents

# Chapter 1

# in

## 1.1 Table of Contents



## 1.2 Sorry!

- 21.jul.97 : The functions I2P and P2I totally screwed up the system, so they're completely re-written now, and are 100 percent OK.

- 14.jul.97 : Forgot to mention that you must FreeVec(), the structure returned by rtFileReq(), yourself.

## 1.3 How to make contact with the Fortunates

```
                          E-mail
                          -=====-

Bugs & Coding to....: -->  sonicwlf@knoware.nl  <--
Ideas & Support to..: -->  ralf@stack.nl        <--


                         Internet
                         -=======-

Fortunates Wolf Page: --> http://utopia.knoware.nl/users/sonicwlf/ <--
Drak's Page.........: --> http://www.stack.nl/~ralf/               <--
```


## 1.4 About

Welcome to the fortunates.library v1.4! This Amiga-library contains lots of handy and smart routines. Not only for the pure speed of the assembler language but to offer a more efficient way of coding.

It's mainly for the coders in Assembler, Blitz Basic 2 and maybe Amiga-E too (Check out if there are emodules in this archive).

Check the Fortunates Wolf Page for the most current revision:

--> http://utopia.knoware.nl/users/sonicwlf/ <--

## 1.5 Installation

Just put the file 'fortunates.library' in your LIBS: directory.

## 1.6 Credits

Ideas : Drak
Coding : Sonicwolf
Using : AsmOne v1.29 on a MC68060

First Release: 28-jun-1997
This Release : 06-aug-1997

## 1.7 System Requirements

```
  Central Processing Unit.: Motorola MC68020 or higher
* Floating Point Unit.....: Motorola MC68881/2 and MC68040/60
  Kickstart Version.......: 3.0 or higher
  Free Memory.............: a few Kilobytes
  Programming Language....: Assembler, Blitz Basic 2 or Amiga-E
```

```
                                *=LibFunction-dependantly
```

## 1.8 AutoDocs Menu

```
                                Table Of Contents

          BR1UnpackA

          BR1UnpackB

          FindChunk

          HiresStd

          I2P

          I2ABGR

          LibBases

          ListChunks

          MemFree

          P2I

          rtFileReq

          SaveRGB8

          UpdatePort
                      uL,sL = Unsigned Long, Signed Long
        uW,sW = Unsigned Word, Signed Word
        uB,sB = Unsigned Byte, Signed Byte
        Ptr   = 32bit Pointer
```

## 1.9 AutoDocs

```
Name            f_MemFree()

Lib.Offset      -30 -($1E)

Synopsis        memfree,type = f_MemFree(void)
                D0.uL    D1.uL

Description     Returns the largest block of free memory in your Amiga.
                Register D1 will contain the memorytype, in case of
                FastRAM a NULL, else a non-NULL meaning ChipRAM.
```

```
Inputs          none

Result          memfree - Largest free memoryblock available
                type    - 0=FastRAM, 1=ChipRAM

Reg.Used        d0, d1, a1, a6

Note            none
```

```
Name            f_LibBases()

Lib.Offset      -36 (-$24)

Synopsis        bases = f_LibBases(void)
                A0.Ptr

Description     Returns a pointer to 9 frequently used Libraries:
                 Dos- Intuition- Graphics- Gadtools- Icon- Diskfont-
                 ASL- Reqtools- Workbench-base        (respectively)

                And do NOT deallocate the pointed area and/or libraries!

Inputs          none

Result          bases - Pointer to the first longword of 9

Reg.Used        a0

Note            So you must do this 9 times: <MOVE.L (A0)+,yourdest>
```

```
Name            f_ListChunks()

Lib.Offset      -42 (-$2A)

Synopsis        result,chunklist = f_ListChunks(filename)
                D0.uL  D1.Ptr                    D1.Ptr
```

```
Description    This will build a list of all chunks out of an IFF.
               And return the listpointer in D1, you'll have to FreeVec() it;

               Offset  0 : Len.4 : TypeOf IFF (eg.ILBM)
               Offset  4 : Len.4 : ChunkHeader (eg.BMHD)
               Offset  8 : Len.4 : ChunkOffset in file
               Offset 12 : Len.4 ...Next ChunkHeader & Offsets...
               Offset xx : Len.4 : NULL, meaning the end

               The function will return a result-code in D0 like:
               SUCCESFUL       0
               ERR_OPENINGFILE 1
               ERR_NOMEM       2
               ERR_READ        3
               ERR_NOTIFF      4

Inputs         filename  - pointer to the filename

Result         result    - a code (see above)
               chunklist - pointer to the list of chunks

Reg.Used       d0, d1, d2, d3, a0, a1

Note           - If the IFF-file is corrupt it will return a NULL.
               - ChunkOffset= Beginning of the Chunk including Header&Size.
               - There is only ONE "FORM" allowed.
```

```
Name           f_FindChunk()

Lib.Offset     -48 -($30)

Synopsis       result = f_FindChunk( chunklist, chunkid )
               D0.uL                 a0.Ptr     D0.uL

Description    Find a particular ChunkHeader in your specified chunklist.
               When found, then D0 contains the "ChunkOffset", or -1.

Inputs         chunklist - pointer to the ChunkList
               chunkid   - an IFF-compatible ID

Result         result    - a ChunkOffset, or -1 when it failed

Reg.Used       d0, d1, a0

Note           none
```

```
Name            f_rtFileReq()

Lib.Offset      -54 -($36)

Synopsis        result = f_rtFileReq( pref-struc )
                D0.uL                 a0.Ptr

Description     Gives a filerequester according to your preferences:

                PreferenceStructure:
                0  4  f_RTWindow      = A WindowPtr, or null (WB)
                4  4  f_RTMatchPat    = A MatchPatternStringPtr, or null
                8  1  f_RTWaitPointer = Set, or null
                9  1  f_RTMultiSelect = Set, or null for SingleFileSelection

                Then if D0 returns non-Null, d0 will contain a ptr to a:

                MultiSelection:
                0  4  Pointer         : Pointer to the stringpointers
                4  x  DirString       : The particular Directory ending with NULL
                x  4  StringPtr 1     : Pointer to the first filename  (NULL- ↩
                   ended)
                x  4  StringPtr 2     : Pointer to the second filename (NULL-en ↩
                   ....
                x  4  Stri..... x     : Pointer to the ...... filename (NU.....
                x  4  NULL            : No more strings

                or if you ordered a SingleFile-requester then d0 points to:

                SingleSelection:
                0  x  String          : The Path+Filename String (NULL-terminated ↩
                   )

                YOU MUST FreeVec() these pointers yourself. (After usage)

Inputs          pref-struc - pointer to a preference-structure

Result          result     - pointer to either a MultiSelection or FileSelection,
                              or a null when the user cancelled.

Reg.Used        D0-D7/A0-A6

Note            ToDo: DirSelection
```

```
Name            f_UpdatePort()

Lib.Offset      -60 (-$3C)

Synopsis        waitmask, userport = f_UpdatePort(window)
                D0.uL     A0.Ptr                  A0.Ptr

Description     Creates a waitmask for Wait() outof the window in A0.
                And it returns the userport in A0 (where the intuimessages arrive ↩
                    ).

Inputs          window - a windowpointer

Result          waitmask - only including the SigBit from you window
                userport - and the messageport for IDCMP

Reg.Used        d0,d1,a0

Note            This function is very useful when you often close and
                open (eg.) an 'option'-window; each time, the waitmask
                and the userport must be updated then.
                ( You can set, for instance, bit number 12 in the waitmask
                  meaning that the Wait() will be interrupted when the
                  user pressed Ctrl-C. Check "dos.i" for more Ctrls. )
```

```
Name            f_HiresStd()

Lib.Offset      -66 (-$42)

Synopsis        result = f_HiresStd(scr_width,scr_height)
                D0.uL               D0.uL     D1.uL

Description     Tests your screendimensions if it's qualified to contain
                'hires-laced' graphics ie. that the buttons may be double-height.

Inputs          scr_width  - width of your screen
                scr_height - height of your screen

Result          result - non-NULL for success, otherwise NULL

Reg.Used        d0,d1,d2,d3,d4

Note
```

```
Name            f_BR1UnpackA()

Lib.Offset      -72 (-$48)

Synopsis        result = f_BR1UnpackA(source,destination,unpackmax)
                D0.uL                 A0.Ptr A1.Ptr      D0.uL

Description     Unpacks ByteRun1-Encoded data pointed by A0 to the destination
                pointed in A1. D0 contains a the maximum unpacksize so
                that corrupted data won't exceed this limit.

Inputs          source    - ptr to ByteRun1-Encoded data (BODY-stuff)
                destination - ptr to your allocated piece of memory
                unpackmax   - maximum for unpacking

Result          result - NULL if succesful, else corrupted

Reg.Used        d0,d1,d2,d3,a0,a1

Note
```

```
Name            f_BR1UnpackB()

Lib.Offset      -78 (-$4E)

Synopsis        result = f_BR1UnpackA(source,destination,packedsize)
                D0.uL                 A0.Ptr A1.Ptr      D0.uL

Description     Unpacks ByteRun1-Encoded data pointed by A0 to the destination
                pointed in A1. D0 contains the amount of packed size; the size
                the BODY-chunk

Inputs          source    - ptr to ByteRun1-Encoded data (BODY-stuff)
                destination - ptr to your allocated piece of memory
                packedsize  - size of the ByteRun1-Encoded stuff

Result          result - NULL if succesful, else corrupted

Reg.Used        d0,d1,a0,a1,a2,a3
```

Note

Name f_I2P()

Lib.Offset -84 (-$54)

Synopsis f_I2P(src, planeptrs,wpr, bpr, planes, height)
A0.Ptr A1.Ptr D0.uL D1.uL D2.uL D3.uL

Description Transfer interleaved data to a planar-based place. Most IFF-ILBMs are interleaved, so when you f_BR1UnpackX them, they must be planar if you want to have the picture be shown on Amiga. (But it's possible to have an interleaved screen, if there's enough contigues ChipRAM, but mostly.. planar is preferred)

So that's why you must supply a pointer to a list of pointers pointing to all single planes. Be sure that their sizes are large enough (or just the same) eg. 320x256/8

Inputs source - ptr to the source, eg. the unpacked BODY
planeptrs - ptr to a list containing pointers to YOUR allocated memory locations.
wpr - words-per-row eg. 320 pixels = 20 words (word-aligned !)
Calculate it like this: (width+15)/16
bpr - Get this from BytesPerRow(BitMap) if your destination are bitplanes (eg. VGA-modes are sophisticated aligned).
Otherwise, in case of eg.allocated memory, it's (wpr *2).
height - height/number of rows
planes - = depth (if there's a mask, there's an EXTRA bitplane )

Result none, it always works or you supplied wrong parameters..

Reg.Used assume ALL

Note

```
Name            f_P2I()

Lib.Offset      -90 (-$5A)

Synopsis        result = f_P2I(planeptrs,dest   wpr,  height,planes)
                D0.uL          A0.Ptr    A1.Ptr D0.uL D1.uL  D2.uL

Description     Transfer planar data to an interleaved place. Most
                IFF-ILBMs are interleaved, so when you f_BR1Pack them, they
                must be interleaved if you want to have the picture saved.

Inputs          planeptrs - ptr to a list containing pointers to your
                            memory locations containing each plane
                dest      - ptr to the destination (width*height/8*bitplanes)
                wpr       - words-per-row eg. 320 pixels = 20 words (word-aligned ↩
                    !)
                            Calculate it like this: (width+15)/16
                height    - height or number of rows
                planes    - the number of pointers

Result          result - NULL if succesful, else corrupted data

Reg.Used        d0,d1,d2,d3,d4,d5,d6,a0,a1,a2,a3

Note
```

```
Name            f_I2ABGR()

Lib.Offset      -96 (-$60)

Synopsis        f_I2ABGR(source,destination,width,height,planes,bpr,  lineskipper ↩
    ,[CMAP])
                         A0.Pt  A1.Ptr      D0.uL D1.uL  D2.uL  D3.uL D4.uL  ↩
                                  [A2.Ptr]

Description     Convert interleaved data to three red/green/blue guns.
                A pixel is stored in a longword with the format: Alpha-Blue-Green ↩
                    -Red.
                (ie. The precise bits represent their plane-activity from 0 to  ↩
                    23)
                So that's the reason why it's not A-R-G-B.

Inputs          source      - pointer to the interleaved (picture)data
                destination - ptr to the destination (width*height*4)
                width       - width in pixels
                height      - height in pixels
                planes      - number of planes to convert
```

```
                bpr         - width in bytes (word-aligned!) ( (w+15)/16 *2 )
                lineskipper - bytes, to skip each scanline (eg. 320/8 * 8 (8 ↩
                   planes))
                CMAP        - pointer to an area with sizeof #colors*4. Load your
                              chunk CMAP at the start of this area.
                              Remember that this area will be shapeshifted,
                              and btw. that you only have to supply this pointer
                              if the picture has 8 or less bitplanes + a CMAP.

Result          If YOU give sane parameters, it all will be allright

Reg.Used        ALL

Note            It's possible to get only for instance the blue gun.
                Just add eg. 320/8*16 to A0 and set planes to 8.
                                (!In case of a 24-bit picture!)
```

```
Name            f_SaveRGB8()

Lib.Offset      -102 (-$66)

Synopsis        f_SaveRGB8(source,width,height,filename,displayid)
                           A0.Ptr D0.uL D1.u   D2.uL    D3.uL

Description     Save your ABGR-buffer to disk in IFF-RGB8 format. (24bit)

Inputs          source    - pointer to an ABGR
                width     - in pixels
                height    - in pixels
                filename  - ptr to a string
                displayid - a displayid for the CAMG-chunk

Result          always succesful

Reg.Used        assume all

Note            This format is the fastest of all!
```

## 1.10 To Do

```
- Chunky2Planar
- Planar2Chunky
- SneakPreview24 ( 24 bitplanes)
- ShowImage      (1-8 bitplanes + HAM)
- PackBR1
- Spline1D

- A GadTools-Replacing Button-System

    E-mail your wishes to:
-->  sonicwlf@knoware.nl  <--
```

## 1.11 History

```
06-jul-97:
-=========-
Revision 2 : f_ListChunks
             f_FindChunk
             f_rtFileReq
             f_UpdatePort

28-jun-97:
-=========-
First release, two functions: MemFree and LibBases. (No Bugs)
```