rmhenglish

COLLABORATORS			
	TITLE : rmhenglish		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		July 31, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

rmhenglish

1.1 RexxMustHave - Version 6.0 @ 1999, 2000 Alfonso Ranieri

```
RexxMustHave - Version 6.0 © 1999, 2000 Alfonso Ranieri

1. Introduction
2. Author
3. WRID
4. Terms
5. Bugs
6. Functions
```

1.2 introduction

```
 1. Introduction


The name of the library stands for Rexx Must Have:
the library is a set of functions I think ARexx macros should have.

The library offers functions to:

    o manipulate:
        . appicon      standard AmigaOS app icon
        . commodities  standard AmigaOS cx
        . notify       standard AmigaOS notify on file or clipboard
                       changes
        . timers       timer.device timers

    o allocate signal and use them in a standard AmigaOS signals events
      driven programming style

    o parse arguments and files with the most powerful ReadArgs around
      and the very cute examine .

    o help programmer with many functions that handle date, time,
      environment vars and so on.
```

All resources created are automagically freed at the exit of the macro.

All objects, e.g. timers or notifies, are named with unique integers.

1.3 author

2. Author

I am Alfonso Ranieri

My e-mail address is alforan@tin.it

You can find me on irc at:
- #amigaita ircnet
- #amyita ircnet

You can find last version of this library at my home page:
http://web.tiscalinet.it/amiga/

1.4 Warning, Requirements, Installation and Distribution

3. Warning, Requirements, Installation and Distribution

Warning
THIS SOFTWARE AND INFORMATION ARE PROVIDED AS IS.
ALL USE IS AT YOUR OWN RISK, AND NO LIABILITY OR
RESPONSIBILITY IS ASSUMED. NO WARRANTIES ARE MADE.

Requirements
The library needs AmigaOS, version or higher.

Installation
Use the installation script.

Distribution
rmh.library is FreeWare.

You are free to distribute it as long as the original archive
is kept intact. Commercial use or its inclusion in other
software package is prohibited without prior written consents
from the Author.

1.5 terms

4. Terms

```
The main terms used are:

    o stem or stemName: a valid ARexx variable name e.g. var,
      var.0, var.name

   o DateStamp: a stem set by macro or returned by functions, with
     the fields:
        . DAYS
        . MINUTE
        . TICK
     set;

   o TimeVal: a stem set by macro or returned by functions, with
     the fields:
        . SECS
        . MICRO
     set;

   o types of arguments: the types used are:
       D   any data            --
       N   numeric             /N
       S   symbol              /S ARexx valid symbol
       V   stemName            /V ARexx valid symbol as S but
                                  with length<20
```

1.6 bugs

```
5. Bugs


   1. for same strange reason, maybe same patch to dos.library,
      the default flags in SetVar() and GetVar() is taken as
      "LOCAL" rather than "GLOBAL", so specify "GLOBAL"   when
      using SetVar() GetVar();

   2. I did my best to make ParseDate() as safe as I could; if
      you have any problem using it, please report it to me.
```

1.7 functions

```
6. Functions


   The functions have also the "RMH_"FunctionName form to avoid
   conflicts with functions from other libraries.


        DOS
            ChangeMode       CreateTempFile    DosString         Expand
            IsInteractive    Lock              Match             MatchPatter
            NameFromFile     RealName          SetComment        SetOwner
            ParsePattern     ReadArgs          ReadFile          WaitForChar
```

```
Notify
    CheckNotify      FreeNotify        NotifySignal       StartNotify

AppIcon
    AddAppIcon       AppIconSignal     FreeAppIcon        HandleAppIcon

Commodity
    AddCxCxSignal    FreeCx            HandleCx           VerifyHotkey

Faults
    AddPart          Fault             FilePart           IoErr
    PathPart         PrintFault        ProgramName        SetIoErr

Signals
    AllocSignal      and               CheckSignal        FreeSignal
    or               PortSignal        SetSignal          Signal
    Wait             xor

Date and time
    AddTime          CheckTimer        CmpTime            CompareDates
    CreateTimer      Date2GMT          ds2tv              FormatDate
    FreeTimer        GetDate           GetFileDate        GetSysTime
    GetTZ            GMTOffset         ParseDate          SetFileDate
    StartTimer       StopTimer         SubTime            SetSysTime
    TimerSignal      tv2ds

Vars
    DeleteVar        GetVar            SetVar

Various
    AddLibrary       EasyRequest       FrontMostScreen    GetToolType
    GetUniqueID      Help              OpenURL            ParseConfig
    PortWait         PutToolType       ReadTextClip       Requester
    SetRexxVar       SetStem           WriteTextClip
```

1.8 addappicon

```
AddAppIcon - creates an appicon.


Synopsis
  appIconID = AddAppIcon(name,icon)
  <name>,[icon]


Function
```

```
  Adds an appicon on the workbench screen.
  An appicon is a standard AmigaOS appicon.

  name is the name of the appicon.
  icon is the name of a info file, without the .info suffix.
  If icon is not given or the icon can't be found, the system
  tools default icon is used.

  The appicon is freed at the exit of the macro if it wasn't yet.


Inputs
  name - the name of the appicon
  icon - the icon file to use


Result
  appIconID - the appicon ID, 0 for failure


See
  AppIconSignal FreeAppIcon HandleAppIcon .
```

1.9 addcx

```
AddCx - adds a ommodity.


Synopsis
  cxID = AddCx(name,title,desc,flags,hotkey)
  <name>,<title>,<descr>,[flag],[hotkey]


Function
  Adds a commodity.
  A commodity is a standard but limited AmigaOS commodity.

  flags is one or more of:
      o UNIQUE       only one cx with name is allowed
      o NOTIFY       notify me if someone try to open another cx with name
      o SHOWHIDE     receive APPEAR DISAPPEAR
      o DISKINSERTED receive a DISKINSERTED event when a disk is inserted
      o DISKREMOVED  receive a DISKREMOVED event when a disk is removed

  hotkey is a valid hotkey description. If it is specified, the
  cx will receive a HOTKEY event when this sequence is used.


Inputs
    name   - the name of the commodity as in Exchange
    text   - the text as in Exchange
    descr2 - the description as in Exchange
    flags  - see above
    hotkey - a hotkey description
```

```
Result
  cxID - an integer:
         . 0  a UNIQUE cx with named name already exists
         . 1  invalid hotkey description
         . otherwise cxID


See
  CxSignal FreeCx HandleCx .
```

1.10 addlibrary

```
AddLibrary - adds a library.


Synopsis
  res = AddLibrary(<lib1>,{lib})
  <lib1>,{lib}


Function
 Adds to the ARexx libraries list till to 15 libraries with
 the query offset at -30 and priority 0.

 Don't use this function with libraries that doen't have the ARexx
 query function offset at -30 (e.g. with openurl.library)!!!

 Each library is first checked to be already in the ARexx library
 list, the a try to open it is made.


Inputs
  lib1 - first library to add (required)
  libi - ith library to add (optional)
  ...


Result
  res - an integer:
          . 0   all ok
          . >0  n-th lib can't be added. The var "Result" is set to the name  ↩
             that failed.


Example

  ...
  if Addlibrary(name1,name2)>0 then do /* failure */
      say "can't add '"Result"'"
      exit
  end
  ...
```

1.11 addpart

```
AddPart - adds a path to a filename.


Synopsis
  complete = AddPart(path,file)
  <path>,<file>


Function
 Adds path to file.


Inputs
  path - the path
  file - the file name


Result
  complete - complete path to file.


See
  FilePart PathPart .
```

1.12 addtime

```
AddTime - adds 2 timeval.


Synopsis
  call AddTime(time1,time2)
  <time1/V>,<time2/V>


Function

 Adds time1 to time2, result in time1.


Inputs

  time1 - a timeval to adds to time2 and where to store the result
  time2 - a timeval to add to time1


Result
  none


See
  CmpTime GetSysTime SubTime SetSysTime .
```

1.13 allocsignal

```
AllocSignal - allocates an Exec signal.


Synopsis
  sigBit = AllocSignal()
  -


Function
 Allocates an returns a signal bit.


Inputs
  none


Result
  sigBit - the signal bit or -1 for failure.


See
  CheckSignal FreeSignal signal wait .
```

1.14 and

```
and - ands integers


Synopsis
  res = and(val1,val2,..)
  <val1/N>,<val2/N>,{val/N}


Function
 Ands up to 15 integer.


Inputs
  val1 - an integer (required)
  val2 - an integer (required)
  vali - i-th integer to and (optional)


Result
  res - the inclusive ands of the arguments


See
  or wait .
```

1.15 appiconsignal

```
AppIconSignal - returns a appicon signal.


Synopsis
  signal = AppIconSignal(appIconID)
  <appIconID/N>


Function
  appIconID is the ID of an appicon created with AddAppIcon().
  The functions returns the signal to wait for appicon events.


Inputs
  appIconID - an appicon ID


Result
  res - the signal of the appicon


See
  AddAppIcon FreeAppIcon HandleAppIcon .
```

1.16 changemode

```
ChangeMode - changes the mode of a file or a lock.


Synopsis
  res = ChangeMode(file,mode)
  <file>,<mode>


Function
  Changes the mode of a file or a lock opened in this macro.

  mode is one of:
      o EXCLUSIVE
      o SHARED


Inputs
  file - the logical name of a file
  mode - the new mode, see above
```

1.17 checknotify

```
CheckNotify - checks a notify.
```

```
Synopsis
  res = CheckNotify(notifyID)
  <notifyID/N>


Function
  notifyID is a notify created with StartNotify().

  The function checks if the notify notified the macro (the object
  content changed).


Inputs
  notifyID - the ID of a notify


Result
  res - an ARexx boolean.


See
  FreeNotify NotifySignal StartNotify .
```

1.18 checksignal

```
CheckSignal - checks signals.


Synopsis
  rec = CheckSignal(mask)
  <mask/N>


Function
  Checks the signals specified in mask.
  The signals are cleared.


Inputs
  mask - the signals to check.


Result
  res - the signals in mask that are set.


See
  AllocSignal FreeSignal signal wait .
```

1.19 checktimer

```
CheckTimer - checks a timer


Synopsis
  res = CheckTimer(timerID)
  <timerID/N>


Function
  timerID is the ID of a timer create by CreateTimer().

  The function checks if the timer completed.

  If the timer was not started, the function returns 1, as the timer
  completed.


Inputs
  timerID - the ID of a timer


Result
  res - an Arexx boolean.


See
  CreateTimer FreeTimer StartTimer StopTimer .
```

1.20 cmptime

```
CmpTime - compares 2 timeval


Synopsis
  res = CmpTime(time1,time2)
  <time1/V>,<time2/V>


Function
 Compares time1 to time2, 2 timeval structures.


Inputs
  time1 - a timeval
  time2 - a timeval


Result
  res - an integer:
          . <0 time1<time2
          .  0 time1=time2
          . >0 time1>time2
```

```
See
  AddTime GetSysTime SubTime SetSysTime .
```

1.21 comparedates

```
CompareDates - compares 2 dates


Synopsis
  res = CompareDates(date1,date2)
  <date1/V>,<date2/V>


Function
 Compares date1 and date2, 2 DateStamp.


Inputs
  date1 - a DateStamp
  date2 - a DateStamp


Result
  res - an integer:
          . <0 date1>date2
          .  0 date1==date2
          . >0 date1<date2


See
  ds2tv FormatDate GetDate GetFileDate ParseDate SetFileDate tv2ds .
```

1.22 createtempfile

```
CreateTempFile - creates a temporary file.


Synopsis
  name = CreateTempFile()
  -


Function
  Opens an unique temporary file in T: and return it's complete
  name.

  Temporary means that at the exit of the macro the file is be
  deleted.

  The file is not open in the macro; if you need to write to it,
  just do a open() with the name returned by this function.
```

```
Inputs
  none


Result
  name - the DOS name of the file (not the logical one)
```

1.23 createtimer

```
CreateTimer - creates a timer.


Synopsis
  timerID = CreateTimer()
  -


Function
  Creates a timer.
  A timer can be used to create timeout, to wait for specific amount of time,
  to break a wait loop and so on.


Inputs
  none


Result
  timerID - the ID of the timer


See
  CheckTimer FreeTimer StartTimer StopTimer .
```

1.24 cxsignal

```
CxSignal - return the signal of a commodity.


Synopsis
  signal = CxSignal(cxID)
  <cxID/N>


Function
  cxID is the ID of a commodity created with AddCx().

  The function returns the signal of the commodity to wait for
  commodity events.
```

```
  Inputs
    cxID - the ID of a commodity.


  Result
    signal - the signal of the commodity


  See
    AddCx FreeCx HandleCx .
```

1.25 date2gmt

```
  date2GMT - converts to GMF format


  Synopsis
    res = date2GMT(date)
    <date/V>


  Function
    date is a DateStamp.

    The function convert it to the GMT format, according to ENV:TZ. If
    ENV:TZ doesn't exist, the date remains untouched.

    The function takes care of the presence of the daylight in TZ.


 Inputs
    date - an ARexx stem name set as Datestamp


 Result
    res - an integer:
            . 0 TZ is not present in ENV:
            . 1 TZ is present in ENV: and the date was converted


 See
    GetDate GetTZ GMTOffset .
```

1.26 deletevar

```
  DeleteVar - delete a DOS var


  Synopsis
    call DeleteVar(name,options)
    <name>,[options]
```

```
Function
  Deletes the DOS var named name.

  option is one or more of:
      o VAR
      o ALIAS
      o IGNORE
      o GLOBAL
      o BINARY
      o NTNULL
      o SAVE
  The default is "VAR GLOBAL"


Inputs
  name - the name of the var
  options - see above


See
  GetVar SetVar .
```

1.27 dosstring

```
DosString - returns a localized I/O error string.


Synopsis
  string = DosString(code)
  [code/N]


Function
  Returns a localized I/O error string.

  If code is omitted, it is assumed to be the current IoErr .


Inputs
  code - the I/O error coed

Return
  string - the localized string associated with code
```

1.28 ds2tv

```
ds2tv - converts DateStamp to timeval


Synopsis
  call ds2tv(from,to)
```

```
  <from/V>,[to/V]


Function
  Converts from, a DateStamp, to a TimeVal, writing it in from or
  in to, if supplied.


Inputs
  from - an ARexx stem name set as DateStamp
  to - an ARexx stem name where function will set as timeval


Result
  none


See
  CompareDates FormatDate GetDate GetFileDate ParseDate SetFileDate tv2ds .
```

1.29 easyrequest

```
EasyRequest - shows an Intuition requester


Synopsis
  res = EasyRequest(text,title,gadgetText,screenName,flags,idcmp)
  <text>,[title],[gadgetText],[screeName],[flags],[idcmp/N]


Function
  Creates and shows and intuition easy requester.

 text and gadgetText can contain a % but only followed by an another % .


Inputs
  text - the text of the requester
  title - the title (default "ARexx Macro Request")
  gadgetText - the string for gadgets text; each gadgetText must be
               separated by the other with a | (e.g. "a|b|c" creates 3 gadgets a ↩
                   b c)
               (default " OK ")
  screenName - the name of a public screen where to open the requester
  flags - controls the function, one of
             . NOFALLBACK
               if present makes the function do not open the
               requester on the default public screen if the
               screenName doesn't exist, but to fail

  idcmp - an integer value of IDCMP flags which will close the requester

Returns:
  res - an integer:
          .   -1   idcmp received ( RC is set to the idcmp received)
```

```
           .    0   last gadget pressed
           . n>=0  gadget number n was pressed.
                   Gadget are counted from left to right and first is number 1,  ↩
                      last 0.


See
  requester .
```

1.30 expand

```
expand - returns directory entries.


Synopsis
  numb = expand(stem,pattern,flags)
  <stem/V>,<pattern>,flags


Function
  Reads entries which match given pattern.

  flags is one of:
      o FILE
        only files are read
      o DIR
        only dir are read

 Entries are written in fields.i,...,fields.x where x=num-1.

 The fields set are:
      o DIRENTRYTYPE   as TYPE but numeric
      o TYPE           FILE or DIR
      o DATE           DateStamp structure
      o ENTRYTYPE
      o PROTECTION
      o SIZE
      o NUMBLOCKS
      o COMMENT
      o OWNERUID
      o OWNERGID


Inputs
  stem - an ARexx stem name
  pattern - an AmigaDOS pattern
  flags - see above


Result
  numb - the number of the entries that match pattern.
```

1.31 fault

```
fault - returns a fault string.


Synopsis
  string = fault(code,msg)
  [code/N],[msg]


Function
  Returns the string:

      msg: DoString(code)

 If code is omitted, it is assumed to be the current IoErr.

 If msg is not specified it is assumed to be the macro name with no extension.


Inputs
  code - an I/O error code
  msg - the message to prefix


Result
  string - the fault string


See
  Printfault IoErr DosString
```

1.32 filepart

```
FilePart - returns the file part of a path


Synopsis
  file = filepart(path)
  <path>


Function
  Returns the file part of path.


Inputs
  path - the complete path to a file



Result
  file - the file part of path
```

```
See
  AddPart PathPart
```

1.33 formatdate

```
FormatDate - converts DateSTamp to string.


Synopsis
  date = FormatDate(date,fmt,locale)
  [date/V],[fmt],[locale]


Function
  Converts date a DateStamp to  a string, according with
  the format flags in fmt and the locale.

  If date is supplied, it must be a stem set as a DateStamp and the
  date is read from it. If it is omitted, the date is the
  current system date.

  If fmt is supplied, it must be a valid locale/FormatDate format
  string. If it is omitted, it is the "short date format" of the
  locale.

  Valid fmt commands are:
      o %a - abbreviated weekday name
      o %A - weekday name
      o %b - abbreviated month name
      o %B - month name
      o %c - same as "%a %b %d %H:%M:%S %Y"
      o %C - same as "%a %b %e %T %Z %Y"
      o %d - day number with leading 0s
      o %D - same as "%m/%d/%y"
      o %e - day number with leading spaces
      o %h - abbreviated month name
      o %H - hour using 24-hour style with leading 0s
      o %I - hour using 12-hour style with leading 0s
      o %j - julian date
      o %m - month number with leading 0s
      o %M - the number of minutes with leading 0s
      o %n - insert a linefeed
      o %p - AM or PM strings
      o %q - hour using 24-hour style
      o %Q - hour using 12-hour style
      o %r - same as "%I:%M:%S %p"
      o %R - same as "%H:%M"
      o %S - number of seconds with leading 0s
      o %t - insert a tab character
      o %T - same as "%H:%M:%S"
      o %U - week number, taking Sunday as first day of week
      o %w - weekday number
      o %W - week number, taking Monday as first day of week
      o %x - same as "%m/%d/%y"
```

```
        o %X - same as "%H:%M:%S"
        o %y - year using two digits with leading 0s
        o %Y - year using four digits with leading 0s

    If locale is supplied, the functions tries to open it to format
    the date in that locale way. If it is omitted, locale is the
    default locale of the system.


  Inputs
    date - an ARexx stem name set as DateStamp
    fmt - a format flags string, see above
    locale - a locale name


  Result
    date - a date string


  See
    CompareDates ds2tv GetDate GetFileDate ParseDate SetFileDate tv2ds .
```

1.34 freeappicon

```
  FreeAppIcon - delets an appicon.


  Synopsis
    call FreeAppIcon(appIconID)
    <appIconID/N>


  Function
    Deletes an appicon created with AddAppIcon().

    appIconID is the appicon ID returned by AddAppIcon().


  Inputs
    appIconID - an appicon D


  Result
    none


  See
    AddAppIcon AppIconSignal HandleAppIcon .
```

1.35 freecx

```
FreeCx - deletes a commodity.


Synopsis
  call FreeCx(cxID)
  <cxID/N>


Function
  Deletes a commodity created with AddCx().



Inputs
   cxID - the commodity ID returned by AddCx().


Result
   none


  See
   AddCx CxSignal HandleCx .
```

1.36 freesignal

```
FreeSignal - frees a signal bit.


Synopsis
  call FreeSignal(signal)
  <signal/N>


Function
 Frees a signal bit allocated with AllocSignal().


Inputs
  signal - a signal bit


Result
  none


See
  AllocSignal CheckSignal signal wait .
```

1.37 freenotify

```
FreeNotify - frees a notify.


Synopsis
  call FreeNotify(notifyID)
  <notifyID/N>


Function
  Frees a notify.


Inputs
  notifyID - the notify ID returned by StartNotify()


Result
  none


See
  CheckNotify NotifySignal StartNotify .
```

1.38 freetimer

```
FreeTimer - frees a timer.


Synopsis
  call FreeTimer(timerID)
  <timerID/N>


Function
  Frees a timer.


Inputs
  timerID - the timer ID returned by CreateTimer()


Result
  none


See
  CheckTimer CreateTimer StartTimer StopTimer .
```

1.39 getdate

```
GetDate - returns current date.
```

```
Synopsis
  res = GetDate(date,flags)
  <date/V>,[flags]


Function
  Reads the system date and set date as a DateStamp struct.

  flags is one of
      o GMT
        If GMT is specified, the date is returned in the GMT format
        according to the ENV:TZ.
        If GMT is specified, but ENV:TZ doesn't exist, the date
        returned is the current system date, but not converted to
        the GMT format.


Inputs
  date  - an ARexx stem name set by the function as a DateStamp
  flags - see above

Retult
  res - an integer:
          . 0 GMT was specified but ENV:TZ doesn't exist
          . 1 GMT was not specified or it was and ENV:TZ exists


See
  CompareDates ds2tv FormatDate GetFileDate ParseDate SetFileDate tv2ds .
```

1.40 getfiledate

```
GetFileDate - returns a file date


Synopsis
  res = GetFileDate(fileName,date)
  <fileName>,<date/V>


Function
 Reads the date of fileName and set date as a DateStamp.


Inputs
  fileName - the name of a file
  date     - an ARexx stem name set by the function as a DateStamp


Result
  res - an ARexx integer:
          . 0 the file was not found
          . 1 success
```

```
Bug
  Should accept a GMT flag.


See
  CompareDates ds2tv FormatDate GetDate ParseDate SetFileDate tv2ds .
```

1.41 getsystime

```
GetSysTime - returns the system time.


Synopsis
  call GetSysTime(time)
  <time/V>


Function
  Sets time as a TimeVal from the current system time.


Inputs
  time - an ARexx stem name set by the function as a timeval



Result
  none


See
  AddTime CmpTime SubTime SetSysTime .
```

1.42 gettz

```
GetTZ - reads ENV:TZ


Synopsis
  res = GetTZ(stem)
  <stem/V>


Function
  Reads ENV:TZ, if it exists, and sets in stem the fields:
      o daylight - boolean, the daylight string is present
      o timezone - number of seconds to ADD to convert to GMT
      o tzstn    - timezone string;
      o tzdtn    - daylight string, if present
```

```
Inputs
  stem - an ARexx stem name


Result
  res - an integer:
          .  0 TZ doesn't exist (no fields is set)
          . - 1 TZ exist


See
  date2GMT GetDate GMTOffset .
```

1.43 getuniqueid

```
GetUniqueID - returns an unique integer


Synopsis
  id = GetUniqueID()
  -


Function
  Each call to this function returns an unique integer.


Inputs
  none


Result
  id - an unique integer
```

1.44 getvar

```
GetVar - reads a system var.


Synopsis
  var = GetVar(name,options)
  <name>,[options]


Function
  Gets the value of the var name.

  options is one or more of:
      o VAR
      o ALIAS
      o IGNORE
```

```
      o GLOBAL
      o BINARY
      o NTNULL
      o SAVE
 The default is "VAR".


Inputs
  name    - the name of the var to read
  options - see above


Result
  var - the content of the var or an empty string


See
  DeleteVar SetVar .
```

1.45 gmtoffset

```
GMTOffset - returns the GMT offset as set in a locale.


Synopsis
  gmo = GMTOffset(locale)
  [locale]


Function
 Reads the GMT offset in minutes from the locale.


Inputs
  locale - a locale name


Result
  gmo - the gmt offset

Note
 The function is deprecated: use GetTZ that reads ENV:TZ (AmigaOS
 locale has no daylight field) .


See
  date2GMT GetDate GetTZ .
```

1.46 handleappicon

```
HandleAppIcon - handles an appicon.
```

```
Synopsis
  numMsg = HandleAppIcon(appIconID,handle)
  <appIconID>,<handle/V>


Function
  Handles an appicon.

  appIconID is the appicon ID returned by AddAppIcon().

  The following fields of handle can be set:
      o WAIT    - wait for messages from the appicon, default 1
      o CTRLC   - wait for a ctrl-c as well, default 0
      o SIGNALS - wait for this signals too, default 0

  The functions returns the number of the messages pending and sets
  the fields:
      o handle.i.CLASS that can be:
          . DOUBLECLICK - the user doubleclicked the icon
          . DROP        - icons were dropped over this appicon
                          i = 0,...,n n = numMsg-1

 If the class is DROP there are set the fields:
      o .i.DROPNUM  the number of icons dropped
      o .i.NAME.j   the name of the j-th icon
      o .i.LOCK.j   a boolean set if the above has a lock
                (e.g. if another appicon is dropped over an appicon
                the lock will be 0)
 where j = 0,...,m m = .i.DROPNUM-1


Bug
  With AmigaOS 3.5, an AppIcon dropped into another AppIcon is
  returned as a DOUBLECLICK.


 See
  AddAppIcon AppIconSignal FreeAppIcon .
```

1.47 handlecx

```
HandleC - handles a cx


Synopsis
  numMsg = HandleCx(cxID,handle)
  <cxID>,<handle/V>


Function
  Handles a cx
  cxID is the cx ID returned by AddCx() .

 The following fields of handle can be set:
```

```
  o WAIT    wait for messages from the cx - default 1
  o CTRLC   wait for a ctrl-c as well - default 0
  o SIGNALS wait for this signals too - default 0

 The functions returns the number of the messages that were pending
 and sets the fields:
  o .i.CLASS that can be:
      - HOTKEY       the hotkey was pressed
                     (only if you specified hotkey in AddCx())
      - DISABLE      pressed the gadget Disable in Exchange
      - UNABLE       pressed the gadget Unable in Exchange
                     (only if you specified NOTIFY in AddCx() flags)
      - KILL         pressed the gadget Remove in Exchange
      - UNIQUE       someone opened a cx with the same name
      - APPEAR       pressed the gadget Show in Exchange
                     (only if you specified SHOWHIDE in AddCx() flags)
      - DISAPPEAR    pressed the gadget Hide in Exchange
                     (only if you specified SHOWHIDE in AddCx() flags)
      - LISTCHG      someone changed the cx list
      - DISKINSERTED a disk was inserted
      - DISKREMOVED  a disk was removed
   where i = 0,...,n n = numMsg-1


Inputs
  cxID - a cx id
  handle - an ARexx stem name


Result
  numMsg - the number of messages pending


See
  AddCx CxSignal FreeCx .
```

1.48 help

```
help - returns a RexxMustHave functions help strings.


Synopsis
  string = help(funName)
  <funName/S>


Function
  Returns the arguments types mask of the function funName.


Inputs
  funName - a RexxMustHave function name


Result
```

```
  string - a help string
```

1.49 ioerr

```
IoErr - returns and clears DOS I/O error.


Synopsis
  err = IoErr()
  -


Function
  Returns and clears the current I/O error code.


Inputs
  none


Result
  err - the I/O error.
```

1.50 isinteractive

```
IsInteractive - checks if a file is interactive.


Synopsis
  res = IsInteractive(file)
  <file>


Function
  Checks if file is interactive, e.g. like a stdout console.


Inputs
  file - a filename


Result
  res - an ARexx boolean
```

1.51 lock

```
lock - creates a lock.


Synopsis
```

```
  res = lock(logic,name,mode)
  <logic>,<name>,[mode]


Function
  Creates a lock on name, with a logic name logic.

  mode is one of:
      o EXCLUSIVE
      o SHARED (default)


  The function just works as internal ARexx open() , but rather than
  opening a file, creates a lock.

  Many people asked me to insert in rmh.library a lock() function,
  so here it is, but I don't know how useful it will be.

  The lock can be unlock via the standard ARexx close() function,
  anyway it is at macro exit.

  ARexx sees the lock as a normal file, e.g. it appears in show(f) .
  Anyway, there is no problem if you use a logic name referring a
  lock rather than a file in the ARexx io functions, because of a
  NIL: file is also created, e.g. if you do a

      call lock(ram,"ram:")
      call writeln(ram,hello)

  nothing happens.


Inputs
  logic - the logical name of the lock
  name  - the name of the AMigaDOS file/drawer to lock
  mode  - the mode of the lock


Result
  res - an ARexx boolean
```

1.52 match

```
Match - AmigaDOS pattern matching.


Synopsis
  res = MatchPattern(pattern,string,flags)
  <pattern>,<string>,flags


Function
  Verifies if string matches pattern.
```

```
  pattern is a AmigaDOS pattern string (NOT the result of ParsePattern()).

  flags is one or more of:
      o CASE
        the matching is case sensitive.


Inputs
  pattern - a pattern string
  string  - the string to match
  flags   - see above


Result
  res - an ARexx boolean


See
  MatchPattern .
```

1.53 matchpattern

```
MatchPattern - AmigaDOS pattern matching.


Synopsis
  res = MatchPattern(pattern,string,flags)
  <pattern>,<string>,flags


Function
  Verifies if string matches pattern.


  pattern is be the result of ParsePattern().

  flags is one or more of:
      o CASE
        the matching is case sensitive.

Inputs

  pattern - a pattern, the result of ParsePattern().
  string  - the string to match
  flags   - see above


Result
  res - an ARexx boolean


 See
  Match ParsePattern .
```

1.54 namefromfile

```
NameFromFile - returns a file name from a logical name.


Synopsis
  signal = NameFromFile(logical,var)
  <logical>,<var/S>


Function
  logical is a logical name of a file or a lock opened in the macro.
  The functions writes the AmigaDOS name of logical in var.


Inputs
  logical - a logical name of a file or a lock
  var     - an ARexx var name


Result
  res - an ARexx boolean
```

1.55 notifysignal

```
NotifySignal - returns the signal of a notify


Synopsis
  signal = NotifySignal(notifyID)
  <notifyID/N>


Function
  Returns the signal of a notify.


Inputs
  notifyID - a notify ID


Result
  signal - the notify signal


See
  CheckNotify FreeNotify StartNotify .
```

1.56 openurl

```
OpenURL - opens an url.
```

```
  Synopsis
    res = OpenURL(url,flags)
    <url>,[flags]


  Function
    ARexx bridge to openurl.library/URL_OpenA.

    Stop using script to send url to browsers, just use this function
    et voila.

    flags is one or more of:
        o SHOW
        o BRINGTOFRONT
        o NEWWINDOW
        o LAUNCH

    If flags is omitted, global OpenURL preferences are used.

   url is the url to open.


  Inputs
    url   - the url to open
    flags - see above


  Result
    res - an integer
            . -1 OpenURL.library is not installed in the system;
            .  0 OpenURL.library couldn't contact any browser
            .  1 success

Note
    Troels put an ARexx interface in OpenURL, so use
    openurl.library/OpenUrl() instead of this function.

    P.S.

        Troels, great idea to use the same function name, with a
        different arguments sintax.
        Pretty good: Amiga always needs a bit more of confusion.
```

1.57 or

```
  or - ors integers


  Synopsis
    res = or(val1,val2,...)
    <val1/N>,<val2/N>,{val/N}


  Function
```

```
  Ors up to 15 integers.

  The functions is usefull to or signals to wait, as in
      recSig=Wait(or(signal1,signal2))


Inputs
  val1 - the first integer to or
  val2 - the second integer to or
  ...
  valn - the n-th integer to or


Result
  res - the or of the arguments


See
  and wait .
```

1.58 parseconfig

```
ParseConfig - parses a configuration file.


Synopsis
  res = ParseConfig(file,stem,mode)
  <file>,<stem/V>,[mode]


Function
  Parses a configuration files.

  A configuration file is an ascii file made of lines as

      <option> {argument}.

  During parsing are ignored:
      o empty lines
      o lines beginning with # or ;
      o lines after the 1024th
      o chars after a ;
      o char after the 256th

  The functions write in stem
         o .i       - option uppercased
         o .i.value - {args}, empty if none
         o .i.line  - the line number (from 1) of the option
                      (NOTA BENE: .i.line~=i usually)

  mode is one or more of:
      o SIMPLECOMMENT   - # doesn't start a comment
      o NOUPPER         - option is not uppercased
      o NOSTRIPSPACES   - every sequence of 2+ spaces or tabs in
                          args is translated in a single space
```

```
Inputs
  file - the file to parse
  stem - an ARexx stem name
  mode - see above



Result
  res - an integer:
          .  -1  file not found
          . >=0  number of valid lines.


Example

  lets suppose a.config is:

  ### Configuration file for a
  ###
  NoGui
  MODE sync
  Wait yes ;wait the child to end
  #

  After the call
      res=ParseConfig("a.config","CONF")

  you will have:

      res             3

      conf.0          NOGUI
      conf.0.value
      conf.0.line     3

      conf.1          MODE
      conf.1.value    sync
      conf.1.line     4

      conf.2          WAIT
      conf.2.value    yes
      conf.2.line     5
```

1.59 parsedate

```
ParseDate - parses a date.


Synopsis
  res = ParseDate(string,fmt,stem,locale)
  <string>,[fmt],[stem/V],[locale]
```

```
Function
  Verify if string is a well formatted date according to fmt
  and converts it to a DateStamp.

  If fmt is present, it must be a valid locale/ParseDate format string:
  only
      %a %A %b %B %d %e %h %H %I %m %M %p %S %y %Y

  are accepted, all others generate ARexx error 18.
  It means that after a "%" there must be one of:
      a A b B d e h H I m M p S y Y

  Sorry, but it would have crashed very easily without this
  protection (I think locale.library/ParseDate is the buggest
  function in AmigaOS).

  If fmt is omitted, it is the "short date format" of locale.

  If stem is supplied, it is set as a DateStamp.

  If locale is supplied, the functions tries to open it to parse
  the date in that locale way.

  If it is omitted, locale is the default locale of the system.


  The function was optimised to not block if trash data are present
  in string: it matches the number of the words present in
  fmt and passes to locale.library/ParseDate only that
  number of words. With this, it should be very stable now.


Inputs
  string - the string to parse
  fmt    - the format of the date
  stem   - an ARexx stem name
  locale - a locale name


Result
  res - an ARexx boolean.


See
  CompareDates ds2tv FormatDate GetDate GetFileDate SetFileDate tv2ds .
```

1.60 parsepattern

```
ParsePattern - compiles an AmigaDOS pattern.


Synopsis
  patt = ParsePattern(pattern,flag)
  <pattern>,[flag]
```

```
Function
  Creates a pattern to use with MatchPattern().

  flag is one or more of:
      o CASE


Inputs
  pattern - the pattern to compile
  flag    - see above

Return
  patt - the compiled pattern


See
  MatchPattern .
```

1.61 pathpart

```
PathPart - returns the path of a complete path to a file.


Synopsis
  pathPart = PathPart(path)
  <path>


Function
  Returns the path part of path.


Inputs
  path - a complete path o a file.


Result
  pathPart - the parth part of path


See
  AddPart FilePart .
```

1.62 portsignal

```
PortSignal - returns the signal of a port.


Synopsis
  signal = PortSignal(portName)
  <portName>
```

```
Function
  Returns the signal of a in-macro-created port.


Inputs
  portName - the name of a port created n the macro.


Result
  signal - the signal of port
```

1.63 portwait

```
PortWait - waits for a port to appear.


Synopsis
  secs = PortWait(portName,secs)
  <portName>,[secs/N]


Function
  Wait for portName to appears for secs seconds.

  If secs is 0, the function waits for ever.


  The function is breakkable via a ctrl-c.

  Returns the seconds the function waited: 0 means the port didn't
  appear.


Inputs
  portName - the name of the port to wait for
  secs     - timeout in seconds


Result
  secs - number of seconds waited


Example

  if PortWait(NOTFYPORT,10)=0 then do
      say "sorry, NOTIFYPORT not opened in last 10 seconds"
      exit
  end
```

1.64 printfault

```
PrintFault - prints a fault string.


Synopsis
  call PrintFault(code,msg)
  [code/N],[msg],[stderr]


Function
  Prints to stderr the same string fault() would return.

  If code is omitted, it is assumed to be the current IoErr.

  If msg is omitted, it is assumed to be the macro name with no extension.

  stderr must be a logic name of a file created in the macro,

  The message is printed into:
      o stderr - if it is supplied,
      o "STDERR" - if such a file was opened in the macro
      o "STDOUT" - otherwise


Inputs
  code   - an I/O error code
  msg    - the head of message
  stderr - where to print the message
```

1.65 programname

```
ProgramName - returns the name of the macro


Synopsis
  pname = ProgramName(mode)
  [mode]


Function
  Returns the program name of the macro.

  mode is one of:
      o FULL
        if supplied the complete path to the macro is returned
        if not supplied, just the macro name is returned
      o NOEXT
        works like no FULL, but only chars before the first ".",
        if present, are returned
      o PATH
        returns the path part of the macro


Inputs
  mode - see above
```

```
Result
  pname - the name of the macro


Example

  Let's suppose the function is called from the macro
  "Work:Inet/Amirc/Rexx/ban.amirx"

  ProgramName()           --> ban.amirx
  ProgramName("NOEXT")    --> ban
  ProgramName("FULL")     --> Work:Inet/Amirc/Rexx/ban.amirx
  ProgramName("PATH")     --> Work:Inet/Amirc/Rexx
```

1.66 readargs

```
ReadArgs - standard AmigaDOS arguments parsing.


Synopsis
  res = ReadArgs(template,help,stem,args)
  <template>,[help],[stem/V],[args]


Function
  Calls dos/ReadArgs().

  Arguments:
      o template is the template of the arguments

      o help is the help string to prompt when a ? is given
        (default template itself)

      o stem is the stem name where to write the arguments
        (default PARM)

      o args is the arguments line for an online arguments parsing

  If "STDERR" is opened (e.g. with open(STDERR,"CONSOLE:","W")) all
  intermedie I/O operations are made with it.


  The function writes (lets supposed stem is "PARM"),
  counting them arguments from left to right:

      o parm.i.value
        the value of the argument

      o parm.i.flag
        an ARexx boolean that indicates if the arguments was supplied

      o parm.i.mult
        number of MULTI if /M given for argument i
```

```
      o parm.i.value.j
        multi value j of arguments i (j = 0 ... i.mult)


Inputs
  template - the template
  help     - the help string
  stem     - an ARexx stemName, deafult "PARM"
  args     - inline arguments string


Result
  res - an ARexx boolean


Example

  /* */

  if ~ReadArgs("FILE/M/A,BUFFER=BUFF/N,QUICK/S") then do
      call PrintFault(IoErr(),ProgramName())
      exit
  end

  if parm.1.flag then say "BUFFER:" parm.1.value
  if parm.2.flag then say QUICK

  say "FILE(s):" parm.0.mult
  do i = 0 to parm.0.mult

      num = expand(F,parm.0.value.i)
      do j = 0 to num-1
          say f.j
      end

  end


See
  fault IoErr PrintFault .
```

1.67 readfile

```
ReadFile - reads a file


Synopsis
  chars = ReadFile(file)
  <file>


Function
  Reads and returns max 65535 chars from a file.
```

```
Inputs
  file - an AmigaDOS file name


Result
  chars - the first (max) 65535 chars of file
```

1.68 readtextclip

```
ReadTextClip - read text iff clipboard content.


Synopsis
  res = ReadTextClip(var,clip)
  <var/S>,[clip/N]


Function
  Writes in var the text content of the clipboard unit clip.
  Default value for clip is 0.


Inputs
  var  - an ARexx var name
  clip - the clipboard unit to read


Result
  res - an integer:
          . -1 clip len is greater than 65535, only first 65535
               chars are returned in var
          .  0 failure, e.g. clip unit contains no text
          .  1 success
```

1.69 realname

```
RealName - returns a real path.


Synopsis
  realName = RealName(logicName,flags)
  <logicName>,[flags]


Function
 Tries to get the real name for the specified logicName .

 This is very useful to try to "resolve" a generic volume name
 not in AmigaDOS format to a device name .

 The device MUST be mounted, or it fails.
```

```
 flags is one of:
      o REQ
        if the device doesn't exist, a requester will be shown


Inputs
  logicName - a logic name
  flags     - see above


Result
  realName - real path to logicName


Example

  RealName("ram:") --> RAM:
  RealName("s:") --> HD0:s
  RealName("StorageCD#1:") --> CD0:
  RealName(":") --> HD2:
  RealName("") --> "current dir"
  RealName("Sys:Disk.info") --> HD0:Disk.info
  RealName("NOTEXISTS_DEVICE:") --> "" (IoErr() --> 218)
  RealName("NOTEXISTS_DEVICE:","REQ") --> "" (IoErr() --> 218 , shows a  ↩
     requester)
  RealName("NOTEXISTS_FILE") --> "" (IoErr() --> 205)
```

1.70 requester

```
requester - shows an AmigaDOS requester.


Synopsis
  res = Requester(msg1,IDCMP,msg2,msg3,screen,flags)
  <msg1>,[IDCMP/N],[msg2],[msg3],[screen],[flags]


Function
 Opens a DOS requester, with 2 gadgets and waits for gadgets or
 IDCMP.


Inputs
  msg1       - first line
  IDCMP      - idcmp to wait too
  msg2       - second line
  msg3       - third line
  screenName - the name of a public screen where to open the requester
  flags      - controls the function, one of
                  . NOFALLBACK
                      if present makes the function do not open the
                      requester on the default public screen if the
                      screenName doesn't exist, but to fail
```

```
Result
  res - a integer:
          . 0 "Cancel" gadget pressed
          . 1 "Accept" gadget pressed or one of IDCMP came


Example

  disk="Disk_bla_bla:"
  res = 1
  call pragma(W,NULL)
  do while ~exists(disk) & res
      call pragma(W,1)
      res = requester(Insert disk "in any drive",x2d(8000))
      call pragma(W,NULL)
  end


 See
  EasyRequest .
```

1.71 setcomment

```
SetCommen - sets a file comment


Synopsis
  res = SetComment(file,comment)
  <file>,<comment>


Function
  Sets the comment of file.

  file is an AmigaDOS file name with path,
  not an ARexx logical file name.

 Returns an ARexx boolean.
```

1.72 gettooltype

```
GetToolType - reads an icon tooltype


Synopsis
  res = GetToolType(icon,name,var,default)
  <icon>,<name>,<var/S>,[default]


Function
  Reads the tooltype name of the icon icon,
```

```
  writing its value in var and using default,
  if name can't be found.


Inputs
  icon - an icon name without the .info .
  name - the name of the tooltype to read
  var - where to write the tooltype value
  default - write this one in var if tooltype was not found

Result
  res - a boolean to indicate if the icon and tooltype were found
```

1.73 puttooltype

```
PutToolType - writes an icoon tooltype


Synopsis
  res = PutToolType(icon,name,value,type,tool)
  <icon>,<name>,<value>,[type],[tool]


Function
  Sets the tooltype name of the icon icon to
  value.

  If icon can't be found and type is supplied,
  a new brand icond of type type is created, and if
  tool is supplied, its default tool is set to tool .

  icon is an icon name without the .info .

  type is one of:
      o WBDISK
      o WBDRAWER
      o WBTOOL
      o WBPROJECT
      o WBGARBAGE
      o WBDEVICE
      o WBKICK
      o WBAPPICON


Inputs
  icon - icon name without the .info .
  name - the tooltype to write
  value - the value to set the tooltype to
  type - create an icon of this type, if icon was not found
  tool - set the default tool to this tool in the brand new icon


Result
  res - an ARexx boolean
```

1.74 frontmostscreen

```
FrontMostScreen - returns the name of the frontmost pub screen


Synopsis
  screen = FrontMostScreen()
  -


Function
  Returns the name of the frontmost public screen, or an empty string
  if the frontmost screen is not public or it is public but private.


Result
  screen - a public screen name or an empty string
```

1.75 setfiledate

```
SetFileDate - sets a file date.


Synopsis
  res = SetFileDate(fileName,date)
  <fileName>,<date/V>


Function
 Sets the date of fileName to the date defined in date a DateStamp.



Inputs
  fileName - a file name
  date     - an ARexx stem name set as DateStamp


Result
  res - an ARexx boolean.


See
  CompareDates ds2tv FormatDate GetDate GetFileDate ParseDate tv2ds .
```

1.76 setowner

```
SetOwner - sets UID and GID of a file.


Synopsis
  res = SetOwner(file,GID,UID)
```

```
    <file>,<GID/N>,<UID/N>


  Function
   Sets the group-id and the user-id of file.

   file is a complete path to a file.

   GID and UID are words;
   they and can be read with expand().


  Inputs
    file - a complete path to a file
    GID  - group-id (0...65535)
    UID  - user-id (0...65535)


  Result
    res - an ARexx boolean.
```

1.77 setioerr

```
  SetIoEr - sets the current I/O error


  Synopsis
    call SetIoErr(code)
    <code/N>


  Function
    Sets the current I/O error code.


  Inputs
    code - the new I/O error code
```

1.78 setrexxvar

```
  SetRexxVar - sets a foreign ARexx var.


  Synopsis
    res = SetRexxVar(pkt,var,value)
    <pkt>,<var/S>,<value>


  Function
    pkt is an ARexx message received on a port.
```

```
  The function sets var to value in the
  environment of the macro that sent pkt.

  The function just checks if pkt is a good ARexx
  msg.

  Don't use this function with a message send in an async way.

 Any bad pkt generates ARexx error 17.


Inputs
  pkt   - an ARexx message
  var   - the var to set
  value - the value to set var to


Result
  res - an ARexx boolean (may fail iff pkt==Null())
```

1.79 setsignal

```
SetSignal - checks and sets signals.


Synopsis
  sig = SetSignal(new,mask)
  <new/N>,<mask/N>


Function
  Queries and modifies the state of the received signals as
  specified in mask .

  Returns the signals set.


Inputs
  new  - the new value for the signals set
  mask - the signals


Result
  sig - the signals set


Example

  To query all signals:
      sig = SetSignal(0,0)

  To query and clear the ctrl-c signal:
      sig = SetSignal(0,2**12)
```

1.80 setstem

```
SetStem - set a compound field.


Synopsis
  call SetStem(stem,field,data)
  <stem/V>,<field/V>,<data>


Function
  The function does:

      stem.field=data


  The function is useful to avoid using INTERPRET too often.


Inputs
  stem  - an ARexx stem name
  field - the field of stem to set
  data  - the value to set


Result
  none
```

1.81 setsystime

```
SetSysTime - sets the system time.


Synopsis
  call SetSysTime(time)
  <time/V>


Function
  Sets the system time as defined in time, a TimeVal.


Inputs
  time - an ARexx stem name set as timeval


Result
  none


See
  AddTime CmpTime GetSysTime SubTime .
```

1.82 setvar

```
SetVar - sets an AmigaDOS var.


Synopsis
  res = SetVar(name,value,options)
  <name>,<value>,[options]


Function
  Sets the var name to value .

  options is one or more of:
      o VAR
      o ALIAS
      o IGNORE
      o GLOBAL
      o BINARY
      o NTNULL
      o SAVE
 The default is "VAR GLOBAL".


Inputs
  name    - a var name
  value   - the value to set name
  options - see above


Result
  res - an ARexx boolean


See
  DeleteVar GetVar .
```

1.83 signal

```
signal - signals a task


Synopsis
  call signal(task,signals)
  <task/N>,<signals/N>


Function
 Signals task with signals.


Inputs
  task    - the task to signal
  signals - the signals mask to signal
```

```
Result
  none

Note
  DON'T PLAY WITH THIS FUNCTION.


See
  AllocSignal CheckSignal FreeSignal wait .
```

1.84 startnotify

```
StartNotify - creates a notify.


Synopsis
  notifyID = StartNotify(name,unit)
  <name>,[unit/N]


Function
  Creates and starts a notify.

  Notification can be for files or clip units.

  When name changes, the macro is signaled with a signal that can
  be obtained with NotifySignal().

  If name is the string "CLIP", the notification occurs on the
  clip unit unit (default 0), otherwise name must be a valid
  AmigaDOS complete path to a file.

  If the notify is not freed in the macro, it is freed at exit.


Inputs
  name - "CLIP" or a complete path to a file
  unit - if name is "CLIP", the clip unit (default 0)


Result
 res - an integer:
          . <0 failure (e.g. name does not exists) . IoErr() can
               be used to find the reason - ONLY FOR FILE
          . >0 id of the notify


See
  CheckNotify FreeNotify NotifySignal .
```

1.85 starttimer

```
StartTimer - starts a timer.


Synopsis
  call StartTimer(timerID,secs,micros)
  <timerID/N>,[secs/N],[micros/N]


Function
  Starts a timer to wait for secs seconds and micros microseconds.

  timerID is the timer ID returned by CreateTimer().

  The timer is started async, so you can go on doing what you want
  after this call.

  To wait for the timer to complete obtain its signal via
  TimerSignal() and Wait() for it.

  If the timer was already started, it is stopped and re-started
  with the new timeout.

  If the timer is not stopped or not freed, it is stopped and freed
  at the exit of the macro.


Inputs
  timerID - the ID of a timer
  secs    - the seconds to wait
  micros  - the microseconds to wait


Result
  none


See
  CheckTimer CreateTimer FreeTimer StopTimer .
```

1.86 stoptimer

```
StopTimer - stops a timer


Synopsis
  call StopTimer(timerID)
  <timerID/N>


Function
 Stops a timer.
```

```
 timerID is the timer ID returned by CreateTimer().


Inputs
  timerID - a timer ID


Result
  none


See
  CheckTimer CreateTimer FreeTimer StarTimer .
```

1.87 subtime

```
SubTime - subtracts 2 timeval


Synopsis
  call SubTime(time1,time2)
  <time1/V>,<time2/V>


Function
 Subtracts time1 to time2, result in time2, both timeval structures.


Inputs
  time1 - the timeval to subtract from time2
  time2 - the timeval to which subtract time1


Result
  none


See
  AddTime CmpTime GetSysTime SetSysTime .
```

1.88 timersignal

```
TimerSignal - returns a timer signal


Synopsis
  signal = TimerSignal(timerID)
  <timerID/N>


Function
 Returns the signal of a timer.
```

```
Inputs
  timerID - a timer ID


Result
  signal the signal of timerID
```

1.89 tv2ds

```
tv2ds - converts from timeval to DateStamp


Synopsis
  call tv2ds(from,to)
  <from/V>,[to/V]


Function
 Convert from from , a  TimeVal, to DateStamp, writing in from or in to , if  ↩
    present.


Inputs
  from - an ARexx stem name set as timeval
  to   - an ARexx stem name


Result
  none


See
  CompareDates ds2tv FormatDate GetDate GetFileDate ParseDate SetFileDate .
```

1.90 verifyhotkey

```
VerifyHotkey - verifies an hotkey string.


Synopsis
  res = VerifyHotkey(hotkey)
  <hotkey>


Function
 Verifies if hotkey is a valid Amiga Cx hotkey description.


Inputs
  hotkey - the string to verify
```

```
Result
  res - an ARexx boolean
```

1.91 wait

```
wait - waits for signals or timeout.


Synopsis
  received = wait(signals,secs,micros)
  <signals/N>,[secs/N],[micros/N]


Function
 Waits for signals or secs seconds and micros microseconds .


Inputs
  signals - the signals to wait
  secs    - timeout
  micros  - timeout


Result
  received - the signals received or 0 on timeout.


 See
  AllocSignal and CheckSignal FreeSignal or signal .
```

1.92 waitforchar

```
WaitForChar - waits for chars.


Synopsis
  res = WaitForChar(file,timeout)
  <file>,<timeout/N>


Function
  Waits for a char from an interactive file for timeout microseconds.


Inputs
  file    - the file to wait chars from
  timeout - microseconds timeout


Result
```

```
 res - an integer: the char read or -1 (eof) on end of file
```

1.93 writetextclip

```
WriteTextCli - write text in the clipboard


Synopsis
  call writetextclip(text,clip)
  <text>,[clip/N]


Function
  Writes text in the clipboard unit clip .
  Default value for clip is 0.


Inputs
  text - the text to write
  clip - the clip number
```

1.94 xor

```
xor - exclusive-ors integers.


Synopsis
  res = xor(val1,val2,...)
  <val1/N>,<val2/N>,{val/N}


Function
  XOR up to 15 integers.


Inputs
  val1 - required integer
  val2 - required integer
  valn - optional integer


Result
  res - the xor of the arguments


See
  and or .
```