rmhenglish

COLLABORATORS			
	TITLE : rmhenglish		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		May 24, 2025	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

rmhenglish

1.1 Index

1.2 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.
```

1.3 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:
- manipulate:
  . appicon     standard AmigaOS app icon
  . commodities standard AmigaOS cx
  . notify      standard AmigaOS notify on file or clipboard changes
  . timers      timer.device timers
- alloc signal and use them in a standard AmigaOS signals events driven
  programming style
- parse arguments and files with the most powerfull ReadArgs around
  and the very cute examine .
```

- 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 "object",e.g. timers or notifies, are named with unique integers.

1.4 requirements

The library needs AmigaOS, version >=2.

1.5 installation

Use the installation script.

1.6 author

I am Alfonso Ranieri .

My e-mail address is alfier@iol.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://users.iol.it/alfier .

1.7 distribution

rmh.library is FreeWare.

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

1.8 terms

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

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

```
  set;

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

- 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
```

1.9 bugs

```
None serious known.
```

1.10 functions

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

 Dos related
 ChangeMode
 CreateTempFile
 DosString
 expand
 IsInteractive
 lock
 MatchPattern
 NameFromFile
 ParsePattern
 ReadArgs
 RealName
 SetComment
 SetOwner

 Notify
 CheckNotify
 FreeNotify
 NotifySignal
 StartNotify

 AppIcon
 AddAppIcon
 AppIconSignal
 FreeAppIcon
 HandleAppIcon

 Commodity
 AddCx
```

```
CxSignal
FreeCx
HandleCx
VerifyHotkey

Faults
AddPart
AllocSignal
and
CheckSignal
fault
FilePart
FreeSignal
IoErr
or
PathPart
PortSignal
PortWait
PrintFault
ProgramName
SetIoErr
SetSignal
signal
wait
WaitForChar
xor

Timer
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
GetUniqueID
help
OpenURL
ParseConfig
ReadTextClip
requester
SetRexxVar
SetStem
WriteTextClip
```

1.11 addappicon

```
AddAppIcon

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

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

"name" is the name of the appicon.
"ico" 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.

Returns an unique ID.

See AppIconSignal FreeAppIcon HandleAppIcon .
```

1.12 addcx

```
AddCx

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

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

"name" is the name of the commodity as in Exchange
"text" is the text as in Exchange
"descr2 is the description as in Exchange
"flags" is one or more of:
- UNIQUE    only one cx with name is allowed
- NOTIFY    notify me if someone try to open another cx with name
- SHOWHIDE  receive APPEAR DISAPPEAR
"hotkey" is a valid hotkey description. If it is specified, the
cx will receive a HOTKEY event when this sequence is used.
```

```
 The functions returns:
 - 0  a UNIQUE cx with named "name" already exists
 - 1  invalid hotkey description
 - otherwise cxID

 See CxSignal FreeCx HandleCx .
```

1.13 addlibrary

```
AddLibrary

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

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

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

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

So use is:

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

1.14 addpart

```
AddPart

Usage: complete = AddPart(path,file)
<path>,<file>

Adds "path" to "file".

Returns the complete path to file.

See FilePart PathPart .
```

1.15 addtime

```
AddTime

Usage: call AddTime(time1,time2)
<time1/V>,<time2/V>

Adds "time1" to "time2", result in "time1", both timeval structures.

See CmpTime GetSysTime SubTime SetSysTime .
```

1.16 allocsignal

```
AllocSignal

Usage: sigBit = AllocSignal()
-

Allocates an returns a signal bit.
The mask to wait for is 2**sigBit.

Returns -1 for failure

See CheckSignal FreeSignal signal wait .
```

1.17 and

```
and

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

And till to 15 integer.

See or wait .
```

1.18 appiconsignal

```
AppIconSignal

Usage: signal = AppIconSignal(appIconID)
<appIconID/N>

appIconID is the appicon ID returned by AddAppIcon .
Returns the signal of an appicon.
The mask to wait for is signal.

See AddAppIcon FreeAppIcon HandleAppIcon .
```

1.19 changemode

```
ChangeMode

Usage: res = ChangeMode(file,mode)
<file>,<EXCLUSIVE|SHARED>

Changes the mode of a file or a lock opened in this macro.
Returns an ARexx boolean.
```

1.20 checknotify

```
CheckNotify

Usage: res = CheckNotify(notifyID)
<notifyID/N>

Checks if a notify notified the macro (the object content changed).

notifyID is the notify ID returned by StartNotify .
Returs an arexx boolean.

See FreeNotify NotifySignal StartNotify .
```

1.21 checksignal

```
CheckSignal

Usage: rec = CheckSignal(mask)
<mask/N>

Check the signals specified in "mask".
Checked signals are cleared.

Returns the signals set.

See AllocSignal FreeSignal signal wait .
```

1.22 checktimer

```
CheckTimer

Usage: res = CheckTimer(timerID)
<timerID/N>

Check if a timer completed.

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

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

Returs an arexx boolean.

See CreateTimer FreeTimer StartTimer StopTimer .
```

1.23 cmptime

```
CmpTime

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

Compares "time1" to "time2", both timeval structures.
Returns:
<0 time1<time2
 0 time1=time2
>0 time1>time2

See AddTime GetSysTime SubTime SetSysTime .
```

1.24 comparedates

```
CompareDates

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

Compares 2 dates set as DateStamp.

Returns:
<0 date1>date2
 0 date1==date2
>0 date1<date2

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

1.25 createtempfile

```
CreateTempFile

Usage: name = CreateTempFile()
-

Opens an unique temporary file in T: and return it's
complete name.
"Temporary" means that at the end of the macro the file
will 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.
```

1.26 createtimer

```
CreateTimer

Usage: timerID = CreateTimer()
-

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.

Returns the ID of the timer.

See CheckTimer FreeTimer StartTimer StopTimer .
```

1.27 cxsignal

```
CxSignal

Usage: signal = CxSignal(cxID)
<cxID/N>

cxID is the cx ID returned by AddCx .
Returns the signal of a cx.
The mask to wait for is signal.

See AddCx FreeCx HandleCx .
```

1.28 date2gmt

```
date2GMT

Usage: res = date2GMT(date)
<date/V>

Reads the "date" as a datestamp structure and converts it to the
GMT format according to TZ in ENV: .
If TZ is not found, the date is untouched.
The function takes care of the presence of the daylight in TZ .

Returns:
0 TZ is not present in ENV:
1 TZ is present in ENV: and the date was converted

See GetDate GetTZ GMTOffset .
```

1.29 deletevar

```
DeleteVar

Usage: call DeleteVar(name,[options])
<name>,[options]

Deletes the var named "name" .

If present, options is one or more of:
- VAR
- ALIAS
- IGNORE
- GLOBAL
- BINARY
- NTNULL
- SAVE
e.g. "VAR GLOBAL"
The default is "VAR GLOBAL"

See GetVar SetVar .
```

1.30 dosstring

```
DosString

Usage: string = DosString(code)
[code/N]

If code is not specified, it is assumed to be the current IoErr .

Return a localized dos string.
```

1.31 ds2tv

```
ds2tv

Usage: call ds2tv(from,to)
<from/V>,[to/V]

Convert "from", a DateStamp, to a TimeVal, writing it in "from" or
in "to" if given.

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

1.32 easyrequest

```
EasyRequest
```

```
Usage: res = EasyRequest(text,title,gadgetText,screenName,flags,idcmp)
<text>,[title],[gadgetText],[screeName],[NOFALLBACK],[idcmp/N]

Creates and shows and intuition easy requester.

- "text" is the text of the requester

- "title" is the title (default "Arexx Macro Request")

- "gadgetText" is the string for gadgets text; each gadgetText must be
  separeted by the other with a | (e.g. "a|b|c" creates 3 gadgets a b c)
  (default " OK ")

- "screenName" is the name of a pubblic screen where to open the requester

- "NOFALLBACK", if present as the 5th argument makes the function to not
  open the requester on the default pubblic screen if the screenName
  dosn't exist, but to fail

- "idcmp" is an integer value of IDCMP flags wich will close the requester

"text" and "gadgetText" can contain a % but only followed by an another % .
Gadget are counted from left to right and first is number 1, last 0.

Returns:

-30   the gost window was not created (means screen not found or too few memory ↩
   )
-20   the requester was not created
 -1   idcmp received ( RC is set to the idcmp received)
  0   last gadget pressed
n>=0  gadget number n pressed

BUG
The requester is not patchable.

See requester .
```

1.33 expand

```
expand

Usage: num = expand(stem,pattern,[FILE|DIR])
<stem/V>,<pattern>,[FILE|DIR]

Reads entries which match given pattern.

Normally every kind of entries are read, but if the 3nd argument is:
- FILE only files are read
- DIR  only dir are read

Entries are written in fields.i,...,fields.x where x=num-1.
The fields set are:
- DIRENTRYTYPE   as TYPE but numeric
- TYPE           "FILE" or "DIR"
```

```
- DATE           datestamp structure
- ENTRYTYPE
- PROTECTION
- SIZE
- NUMBLOCKS
- COMMENT
- OWNERUID
- OWNERGID

Returns the numbers of entries that match pattern.
```

1.34 fault

```
fault

Usage: fault(code,msg)
[code/N],[msg]

Builds and returns the string:

<msg>":" <dos error string of code>

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

See Printfault IoErr DosString
```

1.35 filepart

```
FilePart

Usage: file = filepart(path)
<path>

Returns the file part of "path" .

See AddPart PathPart
```

1.36 formatdate

```
FormatDate

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

Returns a formatted date string.

If "date" is present, it must be a stem set as a DateStamp and the date
is read from it. If it is not present, the date is the current system
date.
```

```
If "fmt" is present, it must be a valid locale/FormatDate format string.
If it is not present, it is the "short date format" of the locale.

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

If "locale2 is given, the functions tries to open it to format
the date in that locale way. If it is not present, locale is the
default locale of the system.

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

1.37 freeappicon

```
FreeAppIcon

Usage: call FreeAppIcon(appIconID)
<appIconID/N>

Frees an appicon created with AddAppIcon().

appIconID is the appicon ID returned by AddAppIcon .

Always returns 1.
```

```
See AddAppIcon AppIconSignal HandleAppIcon .
```

1.38 freecx

```
FreeCx

Usage: call FreeCx(cxID)
<cxID/N>

Frees a cx created with AddCx().

cxID is the cx ID returned by AddCx .
Always returns 1.

See AddCx CxSignal HandleCx .
```

1.39 freesignal

```
FreeSignal

Usage: call FreeSignal(signal)
<signal/N>

Free a signal bit allocated by AllocSignal().
DON'T PLAY WITH THIS FUNCTION!!!

See AllocSignal CheckSignal signal wait .
```

1.40 freenotify

```
FreeNotify

Usage: call FreeNotify(notifyID)
<notifyID/N>

Frees a notify .

notifyID is the notify ID returned by StartNotify .

See CheckNotify NotifySignal StartNotify .
```

1.41 freetimer

```
FreeTimer

Usage: call FreeTimer(timerID)
```

```
<timerID/N>

Free a timer .
timerID is the timer ID returned by CreateTimer .

See CheckTimer CreateTimer StartTimer StopTimer .
```

1.42 getdate

```
GetDate

Usage: res = GetDate(date,"GMT")
<date/V>,["GMT"]

Reads the system date and set "date" as a DateStamp struct.
If "GMT" is specified as the second argument, the date is
returned in GMT format according to the TZ definition in ENV: .
If "GMT" is specified, but TZ is not present, the date returned
is the current system date, but not converted to GMT value.

Returns:
0 "GMT" was specified but TZ is not present in ENV:
1 "GMT" was not specified or it was and TZ is present in env:

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

1.43 getfiledate

```
GetFileDate

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

Reads the date of "fileName" and set "date" as a DateStamp struct.

Returns:
0 the file was not found
1 success

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

1.44 getsystime

```
GetSysTime

Usage: call GetSysTime(time)
<time/V>

Sets "time" as a TimeVal from the current system time.
```

```
See AddTime CmpTime SubTime SetSysTime .
```

1.45 gettz

```
GetTZ

Usage: res = GetTZ(var)
<stem/V>

Reads "ENV:TZ" , if any, and sets in "var" the fields:
- daylight boolean, the daylight string is present;
- timezone number of seconds to ADD to convert to GMT ;
- tzstn    timezone string;
- tzdtn    daylight string (if any);

Returns:
- 0 TZ doesn't exist (no fields is set);
- 1 TZ exist.

See date2GMT GetDate GMTOffset .
```

1.46 getuniqueid

```
GetUniqueID

Usage: id = GetUniqueID()
-

Each call to this function returns an unique integer.
```

1.47 getvar

```
GetVar

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

Gets the value of the var "name" .

If present, options is one or more of:
- VAR
- ALIAS
- IGNORE
- GLOBAL
- BINARY
- NTNULL
- SAVE
e.g. "VAR GLOBAL"
The default is "VAR".
```

```
See DeleteVar SetVar .
```

1.48 gmtoffset

```
GMTOffset

Usage: gmo = GMTOffset(locale)
[locale]

Read the GMT offset in minutes from the locale.
OBSOLETE: use GetTZ that reads ENV:TZ (AmigaOS locale has no daylight field) .

See date2GMT GetDate GetTZ .
```

1.49 handleappicon

```
HandleAppIcon

Usage: numMsg = HandleAppIcon(appIconID,handle)
<appIconID>,<handle/V>

Handles an appicon.

appIconID is the appicon ID returned by AddAppIcon .

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

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

If the class is DROP there are set the fields:
- .i.DROPNUM  the number of icons dropped
- .i.NAME.j   the name of the j-th icon
- .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

See AddAppIcon AppIconSignal FreeAppIcon .
```

1.50 handlecx

```
HandleCx

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

Handles a cx.

cxID is the cx ID returned by AddCx .

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

The functions returns the number of the messages that were pending
and sets the fields:
- .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
where i = 0,...,n n = numMsg-1

See AddCx CxSignal FreeCx .
```

1.51 help

```
help

Usage: args = help(funName)
<funName/S>

Returns the arguments types mask of the function "funName" .
```

1.52 ioerr

```
IoErr

Usage: err = IoErr()
-

Returns the current I/O error code.
```

1.53 isinteractive

```
IsInteractive

Usage: res = IsInteractive(file)
<file>

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

Returns an ARexx boolean.
```

1.54 lock

```
lock

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

Creates a lock with a logic name "logic", on "name" , of mode:
- EXCLUSIVE
- 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 usefull it is :-) .

The lock can be unlock via 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 refering a
lock rather than a file in the ARexx io function, because a
NIL: file is also created, e.g. if you do a
call lock(ram,"ram:")
call writeln(ram,"hello")
nothing happens.

The function ChangeMode() and NameFromFile() now works on locks too.

Returns an ARexx boolean.
```

1.55 match

```
Match

Usage: res = MatchPattern(pattern,string,[CASE])
<string1>,<string2>,[CASE]

Verifies if "string1" matches "string2" .
```

```
"string1" is a AmigaDOS pattern string ( NOT the result of ParsePattern() ).
If the 3rd argument CASE is given, the matching is case sensitive.

See MatchPattern .
```

1.56 matchpattern

```
MatchPattern

Usage: res = MatchPattern(pattern,string,[CASE])
<pattern>,<string>,[CASE]

Verifies if "string" matches "pattern" .

"pattern" is be the result of ParsePattern().
If the 3rd argument CASE is given, the matching is case sensitive.

See Match ParsePattern .
```

1.57 namefromfile

```
NameFromFile

Usage: signal = NameFromFile(file,var)
<file>,<var/S>

Writes in "var" the name of "file" , which must be the logical name
of a file or a lock opened in this macro.

Returns an ARexx boolean.
```

1.58 notifysignal

```
NotifySignal

Usage: signal = NotifySignal(notifyID)
<notifyID/N>

Returns the signal of a notify.
notifyID is the notify ID returned by StartNotify .
The mask to wait for is signal.

See CheckNotify FreeNotify StartNotify .
```

1.59 openurl

```
OpenURL

Usage: res = OpenURL(url,flags)
<url>,[flags]

ARexx bridge to openurl.library/URL_OpenA.
Stop using script to send url to browsers, just use this function et voila.

"url" is the url to open.
"flags" is one or more of:
- SHOW
- BRINGTOFRONT
- NEWWINDOW
- LAUNCH

If "flags" is not specified, global OpenURL preferences are used.

Returns:
- -1 OpenURL.library is not installed in the system;
-  0 OpenURL.library couldn't contact any browser
-  1 success

Read openurl.library documentation.
```

1.60 or

```
or

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

Or till to 15 integers.

See and wait .
```

1.61 parseconfig

```
ParseConfig

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

Read a configuration files.
A configuration file is an ascii file made of lines as
<option> {argument}.

During parsing are ignored:
- empty lines
- lines beginning with # or ;
- lines after the 1024th
- chars after a ;
```

```
- char after the 256th

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

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

Returns:
- -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
#

res=ParseConfig("a.config","CONF")

--> 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
```

1.62 parsedate

```
ParseDate

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

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, others
generate ARexx error 18. It means 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 easly without this protection.
(I think locale.library/ParseDate is the buggest function in AmigaOS :)

If "fmt" is not given, it is the "short date format" of the locale.

If "stem" is given, it is set as a DateStamp.

If "locale" is given, the functions tries to open the locale to parse
the date in that locale way. If it is not present, "locale" is the
default locale of the system.

The function was optimized 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 in "string".
With this, it should be very stable now.

Returns an arexx boolean.

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

1.63 parsepattern

```
ParsePattern

Usage: patt = ParsePattern(pattern,CASE)
<pattern>,[CASE]

Creates a pattern to use with MatchPattern().
If specified, the second argument must the string CASE.

Returns the pattern to use in MatchPattern.

See MatchPattern .
```

1.64 pathpart

```
PathPart

Usage: path = PathPart(path)
<path>

Returns the path part of "path" .

See AddPart FilePart .
```

1.65 portsignal

```
PortSignal

Usage: signal = PortSignal(portName)
<portName>

Return the signal of a in-macro-created port.
The mask to wait for is signal.
```

1.66 portwait

```
PortWait

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

Wait for "portName" to appears for "secs" seconds.
If "secs2 is 0, wait till the port appears.
It is stoppable via ctrl-c.

Returns the seconds the function waited: 0 means the port DIDN'T appear.

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

BUG:
If the is opened and closed very fast, this function may not
see that the port was opened (humm maybe it is better :) .
```

1.67 printfault

```
PrintFault

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

If code is not specified, it is assumed to be the current IoErr .
If msg is not specified it is assumed to be the macro name with no extension.
stderr must be a logic name of a file created in this macro,
if it is not specified it is assumed to be STDERR or STDOUT if STERR
does not exist.

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

1.68 programname

```
ProgramName

Usage: pname = ProgramName(mode)
[FULL|NOEXT]

Returns the "program name" of the macro.

If FULL is specified the complete path is returned.
If FULL is not specified just the macro name is returned.
NOEXT works like no FULL, but only chars before the first '.' ,
if present, are returned.
```

1.69 readargs

```
ReadArgs

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

Calls dos/ReadArgs().

- "template" is the template of the arguments

- "help" is the help string to prompt when a ? is given (default "template"  ↩
   itself)

- "var" is the stem name where to write the arguments (default "PARM")

- "args" is the arguments line for an online arguments parsing, not from STDIN

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

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

- i.value the value of the argument

- i.flag arexx boolean indicating if the arguments was give

- i.mult   number of MULTI if /M givien for argument i
  i.value.j multi value j of arguments i (j = 0 ... i.mult)

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 .

1.70 readtextclip

```
ReadTextClip

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

Writes in "var" the text content of the clipborad unit "clip" .
Default value for clip is 0.

Returns:
-1 clip len is greater than 65535, only first 65535 chars are returned in var
 0 failure
 1 success
```

1.71 realname

```
RealName

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

Tries to get the real name for the specificied logicName .
This is very usefull 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:
- REQ if the device doesn't exist , a requester will be show

Returns: the real name or an empty string for failure (the reasone
can be found via IoErr() , DosString() ) .

Examples:
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)
```

1.72 requester

```
requester

Usage: res = Requester(msg1,IDCMP,msg2,msg3)
<msg1>,[IDCMP/N],[msg2],[msg3]

Opens a DOS requester, with 2 gadgets and wait for gadgts or
"IDCMP" , if specified.

Returns:
0 "Cancel" gadget pressed
1 "Accept" gadget pressed or IDCMP cames

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 .
```

1.73 setcomment

```
SetComment

Usage: res = SetComment(file,comment)
<file>,<comment>

Sets the "comment" of "file" . "file2 is an AmigaDOS file name with path, not
an ARexx file name.

Returns an ARexx boolean.
```

1.74 setfiledate

```
SetFileDate

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

Sets the date of "fileName" as defined in "date" a DateStamp .

Returns an arexx boolean.

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

1.75 setowner

```
SetOwner

Usage: res = SetOwner(file,GID,UID)
<file>,<GID/N>,<UID/N>

Sets the group id and the user id of "file" .
"file" is an AmigaDOS file with path not an ARexx file name.
GID and UID are words and can be read with expand() .

Returns an ARexx boolean.
```

1.76 setioerr

```
SetIoErr

Usage: call SetIoErr(code)
<code/N>

Sets the current I/O error code.
```

1.77 setrexxvar

```
SetRexxVar

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

Sets "var" to "value" in a foreign ARexx macro environment.
Let's suppose you received a message "pkt" in a macro at a port.
With this function you are be able to set "var" to "value" in
the macro sender of the "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.

Returns an ARexx boolean (0 is returned iff pkt==Null() ) .
Any bad "pkt" generates ARexx error 17.
```

1.78 setsignal

```
SetSignal

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

Queries and/or modifies the state of the recevived signals as specified in
mask .

Returs the signals set .

Examples:

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

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

1.79 setstem

```
SetStem

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

Set the "field" of "stem" to "data" .
This function is usefull to void calling INTERPRET too often .
```

1.80 setsystime

```
SetSysTime

Usage: call SetSysTime(time)
<time/V>

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

Always returns 1.

See AddTime CmpTime GetSysTime SubTime .
```

1.81 setvar

```
SetVar

Usage: res = SetVar(name,value,options)
<name>,<value>,[options]
```

```
Sets the var "name" to "value" .

If present, options is one or more of:
- VAR
- ALIAS
- IGNORE
- GLOBAL
- BINARY
- NTNULL
- SAVE
e.g. "VAR GLOBAL"
The default is "VAR GLOBAL"

See DeleteVar GetVar .
```

1.82 signal

```
signal

Usage: call signal(task,signals)
<task/N>,<signals/N>

Signals "task" with "signals" .

DON'T PLAY WITH THIS FUNCTION!!!

See AllocSignal CheckSignal FreeSignal wait .
```

1.83 startnotify

```
StartNotify

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

Creates and starts a notify.
Notification can be for files or clip units.
When the object changes, the macro is signalled with a signal
that can be obtained by NotifySignal().
If "name" is the string "CLIP", the notification occurs on the
clip unit "unit" (default 0), otherwise "name2 must be a valid AmigaDOS
COMPLETE file name.
If the notify is not freed in the macro, it is freed at exit.

Returns:
- <0 failure (e.g. "name" does not exists) . IoErr() can be used to find
     the reasone - ONLY FOR FILE
- >0 id of the notify

See CheckNotify FreeNotify NotifySignal .
```

1.84 starttimer

```
StartTimer

Usage: timerID = StartTimer(timerID,secs,micros)
<timerID/N>,[secs/N],[micros/N]

Starts a timer to wait for "secs2 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 just make a
wait to Wait() with its signal bit obtained with timerSignal().
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.

Always returns 1.

See CheckTimer CreateTimer FreeTimer StopTimer .
```

1.85 stoptimer

```
StopTimer

Usage: call StopTimer(timerID)
<timerID/N>

Stop a timer.
timerID is the timer ID returned by CreateTimer .
Returns always 1, even if the timer was not started.

See CheckTimer CreateTimer FreeTimer StarTimer .
```

1.86 subtime

```
SubTime

Usage: call SubTime(time1,time2)
<time1/V>,<time2/V>

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

See AddTime CmpTime GetSysTime SetSysTime .
```

1.87 timersignal

```
TimerSignal

Usage: signal = TimerSignal(timerID)
<timerID/N>

Returns the signal of a timer.
The mask to wait for is signal.
```

1.88 tv2ds

```
tv2ds

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

Convert from "from" , a  TimeVal, to DateStamp, wrinting in "from" or
in "to" , if present.

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

1.89 verifyhotkey

```
VerifyHotkey

Usage: res = VerifyHotkey(hotkey)
<hotkey>

Verifies if "hotkey" is a valid Amiga Cx hotkey description.
```

1.90 wait

```
wait

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

Waits for "signals" or "secs" seconds and "micros" microseconds .

Returns the signals received or 0 on timeout.

See AllocSignal and CheckSignal FreeSignal or signal .
```

1.91 waitforchar

```
WaitForChar

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

Waits for a char from an interactive "file" for "timeout" seconds.
Returns -1 (eof) on end of file.
```

1.92 writetextclip

```
WriteTextClip

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

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

1.93 xor

```
xor

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

XOR till to 15 integers.

See and or .
```