

rmhenglish

COLLABORATORS

	<i>TITLE :</i> rmhenglish		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY		July 31, 2024	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1	rmhenglish	1
1.1	RexxMustHave 15.0	1
1.2	RexxMustHave Introduction	1
1.3	RexxMustHave Author	2
1.4	RexxMustHave Warning, Requirements, Installation and Distribution	2
1.5	RexxMustHave Terms	2
1.6	RexxMustHave Bugs	3
1.7	RexxMustHave Functions	3
1.8	RexxMustHave Functions - AddAppIcon	5
1.9	RexxMustHave Functions - AddCx	5
1.10	RexxMustHave Functions - AddLibrary	6
1.11	RexxMustHave Functions - AddPart	7
1.12	RexxMustHave Functions - AddTime	8
1.13	RexxMustHave Functions - AllocSignal	8
1.14	RexxMustHave Functions - and	9
1.15	RexxMustHave Functions - AppIconSignal	9
1.16	RexxMustHave Functions - ChangeMode	10
1.17	RexxMustHave Functions - CheckNotify	10
1.18	RexxMustHave Functions - CheckSignal	11
1.19	RexxMustHave Functions - CheckTimer	11
1.20	RexxMustHave Functions - CmpTime	12
1.21	RexxMustHave Functions - CompareDates	12
1.22	RexxMustHave Functions - CreateTempFile	13
1.23	RexxMustHave Functions - CreateTimer	13
1.24	RexxMustHave Functions - CxSignal	14
1.25	RexxMustHave Functions - date2GMT	14
1.26	RexxMustHave Functions - DeleteVar	15
1.27	RexxMustHave Functions - DosString	16
1.28	RexxMustHave Functions - ds2tv	16
1.29	RexxMustHave Functions - EasyRequest	17

1.30 REXXMustHave Functions - expand	17
1.31 REXXMustHave Functions - fault	18
1.32 REXXMustHave Functions - FilePart	19
1.33 REXXMustHave Functions - FormatDate	19
1.34 REXXMustHave Functions - FreeAppIcon	21
1.35 REXXMustHave Functions - FreeCx	21
1.36 REXXMustHave Functions - FreeSignal	22
1.37 REXXMustHave Functions - FreeNotify	22
1.38 REXXMustHave Functions - FreeTimer	23
1.39 REXXMustHave Functions - GetDate	23
1.40 REXXMustHave Functions - GetFileDate	24
1.41 REXXMustHave Functions - GetSysTime	24
1.42 REXXMustHave Functions - GetTZ	25
1.43 REXXMustHave Functions - GetUniqueID	26
1.44 REXXMustHave Functions - GetVar	26
1.45 REXXMustHave Functions - FindVar	27
1.46 REXXMustHave Functions - GMTOffset	27
1.47 REXXMustHave Functions - HandleAppIcon	28
1.48 REXXMustHave Functions - HandleCx	29
1.49 REXXMustHave Functions - help	30
1.50 REXXMustHave Functions - IoErr	31
1.51 REXXMustHave Functions - IsInteractive	31
1.52 REXXMustHave Functions - lock	31
1.53 REXXMustHave Functions - Match	32
1.54 REXXMustHave Functions - MatchPattern	33
1.55 REXXMustHave Functions - NameFromFile	33
1.56 REXXMustHave Functions - NotifySignal	34
1.57 REXXMustHave Functions - OpenURL	34
1.58 REXXMustHave Functions - or	35
1.59 REXXMustHave Functions - ParseConfig	36
1.60 REXXMustHave Functions - ParseDate	38
1.61 REXXMustHave Functions - ParsePattern	39
1.62 REXXMustHave Functions - PathPart	39
1.63 REXXMustHave Functions - PortSignal	40
1.64 REXXMustHave Functions - PortWait	40
1.65 REXXMustHave Functions - PrintFault	41
1.66 REXXMustHave Functions - ProgDir	41
1.67 REXXMustHave Functions - ProgramName	42
1.68 REXXMustHave Functions - ReadArgs	43

1.69 REXXMustHave Functions - ReadFile	44
1.70 REXXMustHave Functions - WriteFile	44
1.71 REXXMustHave Functions - ReadTextClip	45
1.72 REXXMustHave Functions - RealName	46
1.73 REXXMustHave Functions - requester	47
1.74 REXXMustHave Functions - SetComment	48
1.75 REXXMustHave Functions - AskToolType	48
1.76 REXXMustHave Functions - GettoolType	49
1.77 REXXMustHave Functions - PutToolType	49
1.78 REXXMustHave Functions - FrontMostScreen	50
1.79 REXXMustHave Functions - SetFileDate	50
1.80 REXXMustHave Functions - SetOwner	51
1.81 REXXMustHave Functions - SetIoErr	51
1.82 REXXMustHave Functions - SetREXXVar	52
1.83 REXXMustHave Functions - SetSignal	52
1.84 REXXMustHave Functions - SetStem	53
1.85 REXXMustHave Functions - SetSysTime	54
1.86 REXXMustHave Functions - SetVar	54
1.87 REXXMustHave Functions - signal	55
1.88 REXXMustHave Functions - StartNotify	55
1.89 REXXMustHave Functions - StartTimer	56
1.90 REXXMustHave Functions - StopTimer	57
1.91 REXXMustHave Functions - SubTime	57
1.92 REXXMustHave Functions - TimerSignal	58
1.93 REXXMustHave Functions - tv2ds	58
1.94 REXXMustHave Functions - VerifyHotkey	59
1.95 REXXMustHave Functions - wait	59
1.96 REXXMustHave Functions - WaitForChar	60
1.97 REXXMustHave Functions - WriteTextClip	60
1.98 REXXMustHave Functions - xor	61
1.99 REXXMustHave Functions - StemToVar	61
1.100 REXXMustHave Functions - VarToStem	61
1.101 REXXMustHave Functions - DOSWhatIs	62
1.102 REXXMustHave Functions - MultiAssign	63
1.103 REXXMustHave Functions - NamedSpaceCreate	63
1.104 REXXMustHave Functions - NamedSpaceFree	64
1.105 REXXMustHave Functions - NamedSpaceExport	64
1.106 REXXMustHave Functions - NamedSpaceImport	65
1.107 REXXMustHave Functions - NamedSpaceGetVar	66

1.108RexxMustHave Functions - GetRMHString	66
1.109RexxMustHave Functions - MacroNotifyCreate	67
1.110RexxMustHave Functions - MacroNotifyFree	69
1.111RexxMustHave Functions - MacroNotifyGetEvent	69
1.112RexxMustHave Functions - MacroNotifyInsert	70
1.113RexxMustHave Functions - MacroNotifyJoin	71
1.114RexxMustHave Functions - MacroNotifySync	71
1.115RexxMustHave Functions - LocalizeStrings	72
1.116RexxMustHave Functions - MacroEnv	73

Chapter 1

rmhenglish

1.1 RexxMustHave 15.0

RexxMustHave 15.0 © 2000, 2001 Alfonso Ranieri

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

1.2 RexxMustHave Introduction

Introduction

The name of the library stands for Rexx Must Have:
the library is a set of functions I think ARexx
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
- NamedSpace - sets of vars
- MacroNotify - trees of macros

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

Parse arguments and files with the most powerful ReadArgs around
and the very cute expand() function.

Help programmers with many functions that handle date, time,

environment vars and so on.

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

1.3 RexxMustHave Author

Author

I am Alfonso Ranieri

My e-mail address is alforan@tin.it

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

1.4 RexxMustHave Warning, Requirements, Installation and Distribution

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 RexxMustHave Terms

Terms

The main terms used are:

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

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

DAYS
MINUTE
TICK

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

SECS
MICRO

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 RexxMustHave Bugs

Bugs

1. This is an AmigaOS 3.5 bug:
if you drop an appicon into another appicon, you receive
a DOUBLECLICK event rather than a DROP one.

1.7 RexxMustHave Functions

Functions

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

DOS

ChangeMode	CreateTempFile	DosString
DOSWhatIs	Expand	IsInteractive
Lock	Match	MatchPattern
MultiAssign	NameFromFile	ParsePattern
ProgDir	ReadArgs	ReadFile
RealName	SetComment	SetOwner
WaitForChar	WriteFile	

Notify

CheckNotify	FreeNotify	NotifySignal
StartNotify		

AppIcon

AddAppIcon	AppIconSignal	FreeAppIcon
------------	---------------	-------------

HandleAppIcon

Commodity

AddCx

VerifyHotkey

FreeCx

HandleCx

Faults

AddPart

IoErr

ProgramName

Fault

PathPart

SetIoErr

FilePart

PrintFault

Signals

AllocSignal

FreeSignal

SetSignal

xor

and

or

Signal

CheckSignal

PortSignal

Wait

Date and time

AddTime

CompareDates

ds2tv

GetDate

GetTZ

SetFileDate

StopTimer

tv2ds

CheckTimer

CreateTimer

FormatDate

GetFileDate

GMTOffset

SetSysTime

SubTime

CmpTime

Date2GMT

FreeTimer

GetSysTime

ParseDate

StartTimer

TimerSignal

Macros tree

MacroNotifyCreate

MacroNotifyInsert

MacroNotifyFree

MacroNotifyJoin

MacroNotifyGetEvent

MacroNotifySync

Vars

DeleteVar

SetVar

FindVar

GetVar

Vars sharing

NamedSpaceCreate

NamedSpaceGetvar

VarToStem

NamedSpaceExport

NamedSpaceImport

NamedSpaceFree

StemToVar

Various

AddLibrary

FrontMostScreen

Help

OpenURL

PutToolType

SetRexxVar

AskToolType

GetToolType

LocalizeStrings

ParseConfig

ReadTextClip

SetStem

EasyRequest

GetUniqueID

MacroEnv

PortWait

Requester

WriteTextClip

1.8 RexxMustHave Functions - AddAppIcon

AddAppIcon - creates an appicon.

Synopsis

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

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.

stem is an ARexx stem name; in it you may set the field:

```
SupportsOpen
SupportsCopy
SupportsRename
SupportsInformation
SupportsSnapshot
SupportsUnSnapshot
SupportsLeaveOut
SupportsPutAway
SupportsDelete
SupportsFormatDisk
SupportsEmptyTrash
PropagatePosition
NotifySelectState
```

The above have sense only under AmigaOS 3.5.

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

See

AppIconSignal FreeAppIcon HandleAppIcon .

1.9 RexxMustHave Functions - AddCx

AddCx - adds a commodity.

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:

- Unique - only one cx with name is allowed
- Notify - notify me if someone try to open another cx with name
- Showhide - receive APPEAR DISAPPEAR
- DiskInserted - receive a DISKINSERTED event when a disk is inserted
- 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 RexxMustHave Functions - 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 doesn'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 RexxMustHave Functions - 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 RexxMustHave Functions - 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 RexxMustHave Functions - AllocSignal

AllocSignal - allocates an Exec signal.

Synopsis

```
sigBit = AllocSignal()
-
```

Function

Allocates an returns an Exec signal bit.

Because ARexx scripts run as separate process, you may not free allocated signal before exiting. Anyway, remember that you may allocate up to 15 signals and that some of them are used by many objects such as ARexx ports, timers, RxMUI gui and so on. It means that is freeing signals when they are not needed is a good programming way.

Inputs

none

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

See
CheckSignal FreeSignal signal wait .

1.14 REXXMustHave Functions - 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 REXXMustHave Functions - AppIconSignal

AppIconSignal - returns an 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 RexxMustHave Functions - 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:

EXCLUSIVE
SHARED

Inputs

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

1.17 RexxMustHave Functions - 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 RexxMustHave Functions - CheckSignal

CheckSignal - checks signals.

Synopsis

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

Function

Checks the signals specified in mask.
Note that 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 RexxMustHave Functions - 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 RexxMustHave Functions - 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 RexxMustHave Functions - 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 RexxMustHave Functions - 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 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 RexxMustHave Functions - 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 or -1 if no signal is available

See

CheckTimer FreeTimer StartTimer StopTimer .

1.24 REXXMustHave Functions - CxSignal

CxSignal - returns 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 REXXMustHave Functions - 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 RexxMustHave Functions - DeleteVar

DeleteVar - deletes a DOS var

Synopsis

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

Function

Deletes the DOS var named name.

option is one or more of:

```
VAR
ALIAS
IGNORE
GLOBAL
BINARY
NTNULL
SAVE
```

The default is "VAR GLOBAL"

Inputs

name - the name of the var
options - see above

See

FindVar GetVar SetVar .

1.27 RexxMustHave Functions - 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 RexxMustHave Functions - 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 RexxMustHave Functions - EasyRequest

EasyRequest - shows an Intuition requester

Synopsis

```
res = EasyRequest(text,title,gadgetText,screenName,flags,idcmp)
<text>,[title],[gadgetText],[screenName],[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 RexxMustHave Functions - 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:

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
OWNERUID

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 RexxMustHave Functions - 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 RexxMustHave Functions - 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 RexxMustHave Functions - 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:

- %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 leading 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 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 RexxMustHave Functions - FreeAppIcon

FreeAppIcon - deletes 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 RexxMustHave Functions - 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 RexxMustHave Functions - 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 RexxMustHave Functions - 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 RexxMustHave Functions - 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 RexxMustHave Functions - 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

GMT

if supplied, the function tries to convert the date in

GMT format:

1. ENV:TZ exists, the date is converting according to it
2. if ENV:TZ does not exist, the date is converting according to the default locale
3. if ENV:TZ does not exist, and the default locale can't be

opened for any reason, the date is not converted

Inputs

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

Result

res - an integer:
0 - GMT was specified but the date was not converted
1 - GMT was not specified or it was and the date was converted

See

CompareDates ds2tv FormatDate GetFileDate ParseDate SetFileDate tv2ds .

1.40 RexxMustHave Functions - 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 RexxMustHave Functions - 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 RexxMustHave Functions - GetTZ

GetTZ - reads ENV:TZ

Synopsis

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

Function

Reads ENV:TZ, if it exists, and sets in stem 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 present

Inputs

stem - an ARexx stem name

Result

res - an integer:

- 0 - TZ doesn't exist (no fields is set)
- 1 - TZ exists

See

date2GMT GetDate GMTOffset .

1.43 RexxMustHave Functions - 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 RexxMustHave Functions - 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:

```
VAR  
ALIAS  
IGNORE  
GLOBAL  
BINARY  
NTNULL  
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 FindVar SetVar .

1.45 RexxMustHave Functions - FindVar

FindVar - reads a local var.

Synopsis

```
var = FindVar(name,options)
<name>,[options]
```

Function

Gets the value of the local var name.

options is one or more of:

```
VAR
ALIAS
IGNORE
BINARY
```

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 GetVar SetVar .

1.46 RexxMustHave Functions - 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

See

date2GMT GetDate GetTZ .

1.47 RexxMustHave Functions - 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:

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 pending and sets the fields:

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

DOUBLECLICK

COPY

RENAME

INFORMATION

SNAPSHOT

UNSNAPSHOT

LEAVEOUT

PUTAWAY

DELETE

FORMATDISK

EMPTYTRASH

SELECTED

UNSELECTED

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)
        class = "DROP";
        drop=1;
    }
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 .

1.48 REXXMustHave Functions - 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:

```

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 function 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

DISKINSERTED
a disk was inserted

DISKREMOVED
a disk was removed

where $i = 0, \dots, n$ $n = \text{numMsg}-1$

Inputs

cxID - a cx id
handle - an ARexx stem name

Result

numMsg - the number of messages pending

See

AddCx CxSignal FreeCx .

1.49 RexxMustHave Functions - help

help - returns a RexxMustHave functions help string.

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

1.50 RexxMustHave Functions - 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.

1.51 RexxMustHave Functions - 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

1.52 RexxMustHave Functions - 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:

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 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

1.53 RexxMustHave Functions - 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:

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 .

1.54 RexxMustHave Functions - 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:

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 .

1.55 RexxMustHave Functions - 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

1.56 RexxMustHave Functions - 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 .

1.57 RexxMustHave Functions - 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:

```
SHOW
BRINGTOFRONT
NEWWINDOW
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 syntax.
Pretty good: Amiga always needs a bit more of confusion."

1.58 RexxMustHave Functions - 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 .

1.59 RexxMustHave Functions - 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:

```
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 uppercased

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
```

1.60 RexxMustHave Functions - 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 buggiest 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 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. 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 .
```

1.61 RexxMustHave Functions - 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 .

1.62 RexxMustHave Functions - 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 path part of path

See

AddPart FilePart .

1.63 REXXMustHave Functions - 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 in the macro.

Result

signal - the signal of port

1.64 REXXMustHave Functions - PortWait

PortWait - waits for a port to appear.

Synopsis

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

Function

Wait for portName to appear for secs seconds.

If secs is 0, the function waits for ever.

The function is breakable 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(NOTIFYPORT,10)=0 then do
    say "sorry, NOTIFYPORT not opened in last 10 seconds"
    exit
end
```

1.65 RexxMustHave Functions - 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

1.66 RexxMustHave Functions - ProgDir

ProgDir - sets PROGDIR: for the current macro

Synopsis

```
hd = ProgDir()
-
```

Function

Sets the HomeDir ("PROGDIR:") for the current macro to the path to the macro itself and returns its name.

After a call to this function, you may use the name "PROGDIR:" as a valid absolute path.

If the function is called more than once, nothing happens.

Result

hd - the name of the HomeDir

Example

```
call ProgDir()
call Open("in", "PROGDIR:conf/configuration", "R")
```

1.67 RexxMustHave Functions - 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:

FULL

if supplied the complete path to the macro is returned
if not supplied, just the macro name is returned

NOEXT

works like no FULL, but only chars before the first ".",
if present, are returned

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
```

1.68 RexxMustHave Functions - ReadArgs

ReadArgs - standard AmigaDOS arguments parsing.

Synopsis

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

Function

Calls dos/ReadArgs().

Arguments:

template is the template of the arguments

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

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

args is the arguments line for an online arguments parsing

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

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

parm.i.value
the value of the argument

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

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

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, default "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

end
```

See

fault IoErr PrintFault .

1.69 RexxMustHave Functions - ReadFile

ReadFile - reads a file

Synopsis

```
chars = ReadFile(file,maxSize/N)
<file>,[maxSize/N]
```

Function

Reads and returns up to maxSize chars from a file.

Inputs

file - an AmigaDOS file name
maxSize - max chars to read 0<=maxSize<=65535 default 65535

Result

chars - the first (max) 65535 chars of file

1.70 RexxMustHave Functions - WriteFile

WriteFile - writes to a file

Synopsis

```
res = WriteFile(file,buf,opt)
<file>,<buf>,[opt]
```

Function

Writes buf to file.

opt may be one or more of:

APPEND

append buf at the end of file

NTCREATE

do not create file if it doesn't exist

NEWLINE

add a newline at the end of buf

Inputs

file - an AmigaDOS file name

buf - the chars to write

opt - options

Result

res - an ARexx boolean

1.71 RexxMustHave Functions - ReadTextClip

ReadTextClip - reads 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

```

1.72 RexxMustHave Functions - 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:

```

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)

```

Note

```

Sometime
    rname: procedure
    parse arg a

```

```
o=pragma("D",PathPart(a))
return AddPart(pragma("D",o),FilePart(a))

is better.
```

1.73 RexxMustHave Functions - 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

1.74 RexxMustHave Functions - 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.

1.75 RexxMustHave Functions - AskToolType

AskToolType - reads an icon tooltype

Synopsis

```
res = AskToolType(icon,name,flags)
<icon>,<name>,[flags]
```

Function

Reads and returns the tooltype name of the icon
icon .

flags may be KEY in which case, the function
doesn't return the value of the tooltype, but just an
ARexx boolean to indicate if the tooltype is present.

Inputs

icon - an icon name without the .info
name - the name of the tooltype to read
flags - flags to controll the operation

Result

res - the value of the tooltype:
an empty string - the tooltype was not found and
KEY was not specified in flags
0 - the tooltype was not found and KEY was specified in
flags
1 - the tooltype was found and KEY was specified in
flags
the value of the tooltype - the tooltype was found and KEY
was not specified in flags

See

GetToolType PutToolType

1.76 RexxMustHave Functions - 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

See

AskToolType PutToolType

1.77 RexxMustHave Functions - PutToolType

PutToolType - writes an icon 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 icon 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:

WBDISK
WBDRAWER
WBTOOL
WBPROJECT
WBGARBAGE
WBDEVICE
WBKICK
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

See

AskToolType GetToolType

1.78 RexxMustHave Functions - 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

1.79 RexxMustHave Functions - 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 .

1.80 RexxMustHave Functions - 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.

1.81 RexxMustHave Functions - 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

1.82 RexxMustHave Functions - 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())

1.83 RexxMustHave Functions - 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)

1.84 REXXMustHave Functions - SetStem

SetStem - sets 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

1.85 RexxMustHave Functions - 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 .

1.86 RexxMustHave Functions - 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:

```
VAR
ALIAS
IGNORE
GLOBAL
BINARY
NTNULL
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 FindVar GetVar .

1.87 RexxMustHave Functions - 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 .

1.88 RexxMustHave Functions - 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` .

1.89 RexxMustHave Functions - 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 .

1.90 RexxMustHave Functions - 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 .

1.91 RexxMustHave Functions - 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 .

1.92 REXXMustHave Functions - 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

1.93 REXXMustHave Functions - 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 .

1.94 RexxMustHave Functions - 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

1.95 RexxMustHave Functions - 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 .

1.96 RexxMustHave Functions - 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

1.97 RexxMustHave Functions - WriteTextClip

WriteTextClip - writes text in a 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

1.98 RexxMustHave Functions - 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 .

1.99 RexxMustHave Functions - StemToVar

StemToVar - copies a stem to local vars

Synopsis

```
call StemToVar(stem)
<stem>
```

Function

Copies all the fields of the stem 'stem' to local vars.

Inputs

stem - an ARexx stem name

See

VarToStem

1.100 RexxMustHave Functions - VarToStem

VarToStem - copies local vars to stem

Synopsis

```
call VarToStem(patt,stem)
[patt],[stem/V]
```

Function

Copies all the local vars that match 'patt' or just all, if patt is not supplied, as ARexx vars, setting them as fields of 'stem' if present.

Inputs

patt - an AmigaDOS pattern
stem - an ARexx stem name

See

StemToVar

1.101 RexxMustHave Functions - DOSWhatIs

DOSWhatIs - finds out what a name is

Synopsis

```
what = DOSWhatIs(name)
<name>
```

Function

Finds out what a name is.

Inputs

name - a name

Result

what - a string:
DEVICE
DIRECTORY
VOLUME
LATE
NONBINDING
PRIVATE
empty string

1.102 RexxMustHave Functions - MultiAssign

MultiAssign - retrieves multi assignments

Synopsis

```
res=MultiAssign(name,stem)
<name>,<stem/V>
```

Function

Given an assignment name, the functions retrieves the list of all the multi assignments relative to 'name'.

Inputs

name - an assignment
stem - where to write the results

Result

res - an ARexx boolean

Example

```
if MultiAssign("s:", "ass") then do
  say s "is associated with" ass.num "drawer(s) "
  do i=0 to ass.num-1
    say i ass.i
  end
end
```

1.103 RexxMustHave Functions - NamedSpaceCreate

NamedSpaceCreate - creates a named space

Synopsis

```
res=NamedSpaceCreate(name,opts)
<name>,[opts]
```

Function

Creates a NamedSpace. A NamedSpace (called NS) is a global endpoint for data sharing among ARexx macros.

A NS exists till the macro that created it is running.

The NS name is unique. NS names comparison is not case sensitive.

A NS space contains data in the form of couples NAME/VALUE.

A NS space may be PRIVATE, in which case the datas may be changed only by the macro that created it, or PUBLIC in which case the datas may be modified by anyone.

If supplied, 'opt' may be one or more of:

- o PRIVATE - the NS is private

Inputs

name - the name of the NS
opt - options for the NS

Result

res - an integer, 0 for success

See

NamedSpaceExport	NamedSpaceFree	NamedSpaceGetvar
NamedSpaceImport	StemToVar	VarToStem

1.104 RexxMustHave Functions - NamedSpaceFree

NamedSpaceFree - disposes a NS

Synopsis

```
res=NamedSpaceFree(name)
<name>
```

Function

Disposes a NS. Only the NS creator may use this function.

Inputs

name - the name of the NS

Result

res - an integer, 0 for success

See

NamedSpaceCreate	NamedSpaceExport	NamedSpaceGetvar
NamedSpaceImport	StemToVar	VarToStem

1.105 RexxMustHave Functions - NamedSpaceExport

NamedSpaceExport - exports var(s) to a NS

Synopsis

```
res=NamedSpaceExport(name,pattern,opts)
<name>,[pattern],[opts]
```

Function

Exports var(s) to a NS. Once you created a NS, you want to use it for sure. It means setting some couple name/value in the NS. To do that, you may simply "export" some macro var to the NS.

pattern is an AmigaDOS pattern; if supplied only the vars matching it are exported; if not supplied all vars are exported.

opts is one or more of:

LOCAL

only the local vars are exported

GLOBAL

only the global vars are exported

BOTH

both local and global vars are exported (default)

Inputs

name - the name of the NS
pattern - an AmigaDOS pattern
opts - options

Result

res - an integer, 0 for success

See

NamedSpaceCreate	NamedSpaceFree	NamedSpaceGetvar
NamedSpaceImport	StemToVar	VarToStem

1.106 RexxMustHave Functions - NamedSpaceImport

NamedSpaceImport - imports var(s) from a NS

Synopsis

```
res=NamedSpaceImport(name,pattern)
<name>,[pattern]
```

Function

Imports var(s) from a NS. Once you created a NS, you want to use it for sure. It means retrieving some datas from the NS. To do that, you may simply "import" some data

from the NS.

pattern is an AmigaDOS pattern; if supplied only the names in the NS matching it are imported; if not supplied all names are imported.

Inputs

name - the name of the NS
pattern - an AmigaDOS pattern

Result

res - an integer, 0 for success

See

NamedSpaceCreate	NamedSpaceFree	NamedSpaceGetvar
NamedSpaceImport	StemToVar	VarToStem

1.107 RexxMustHave Functions - NamedSpaceGetVar

NamedSpaceGetVar - gets a single NS var

Synopsis

```
value=NamedSpaceGetVar(name,var)
<name>,<var>
```

Function

Gets a single NS var.

Inputs

name - the name of the NS
var - the name of the var to get

Result

value - the value of var, or an empty string if it doesn't exist

See

NamedSpaceCreate	NamedSpaceExport	NamedSpaceFree
NamedSpaceImport	StemToVar	VarToStem

1.108 RexxMustHave Functions - GetRMHString

GetRMHString - returns a string associated with a RMH code

Synopsis

```
string=GetRMHString(code)
<code/N>
```

Function

Returns a string associated with a RMH code.

Actually, the following strings are defined:

51	[NO_MEM]	too few memory
52	[ARG_MISS]	required argument missed
53	[BAD_NUMBER]	bad number
54	[BAD_VALUE]	bad value
55	[NO_NAMEDSPACE]	NamedSpace not found
56	[NO_VAR]	Var not found
57	[PRIVATE_NAMEDSPACE]	NamedSpace is private

Inputs

code - the code to retrieves the string associated to

Result

string - the string associated with 'code'

Note

At the moment, only NamedSpace functions returns a RMH error code.

1.109 RexxMustHave Functions - MacroNotifyCreate

MacroNotifyCreate - creates a MacroNotify object

Synopsis

```
sig=MacroNotifyCreate(name)
<name>
```

Function

Creates a MacroNotify object. A MacroNotify object is a named space that exists till the creator macro runs. A MacroNotify represents a tree of macros.

This is how it works:

```
o Macro A creates the MacroNotify object:
  sig=MacroNotifyCreate(name)
```

sig is a signal to wait on for MacroNotify events

Macro A waits for event with

```

rec=Wait(sig)
if and(rec,sig)>0 then call GetMacrosEvents()
...
GetMacrosEvents:
  ev=MacroNotifyGetEvent(name)
  do while ev~=""
    parse var ev ev more
    select
      when ev="STARTED" then do
        /* a macro started, more is the macro ID */
      end
      when ev="ENDED" then do
        /* a macro ended, more is the macro ID */
      end
      when ev="INFO" then do
        /* a macro send us infos, more is an info string */
      end
      when ev="ATTEMPT" then do
        /* someone tried to create a MacroNotify already
           create by us, more is the name */
      end
    end
  end
  ev=MacroNotifyGetEvent(name)
end

```

- o Macro B joins the club:


```
res=MacroNotifyJoin(name)
```
- o When macro B joins the club, macro A is informed with a STARTED event
- o Macro B may send infos to macro A via:


```
res=MacroNotifyInsert(name,"INFO",more)
```
- o When macro B exits, it leaves the club and sends us a a ENDED event
- o If macro A exits and macro B is still running, macro B is notified via a break_d (2**12) signal. It may be trapped via a signal on break_d
- o If macro A wants to notify macro B, macro A may do that via:


```
call MacroNotifySync(name)
```

Of course, in the example above, there may be more than a macro B and even more than a macro A. It means that more macros may join a single MacroNotify and that a single macro may join more MacroNotifies.

Inputs

name - the name of the MacroNotify. The name comparison is not case sensitive. The name of a MacroNotify is unique. You may obtain an unique name via
 name=ProgramName("NOEXT") || time(s)
 and tricks like that.

Result

sig - a signal to wait on for event, 0 is failure

See

MacroNotifyFree	MacroNotifyGetEvent	MacroNotifyInsert
MacroNotifyJoin	MacroNotifySync	

1.110 RexxMustHave Functions - MacroNotifyFree

MacroNotifyFree - disposes a MacroNotify object

Synopsis

```
res=MacroNotifyFree(name)
<name>
```

Function

Disposes a MacroNotify object. All the macros that join the club are notified with a break_d. Only the MacroNotify creator may use this function.

Inputs

name - the name of the MacroNotify. The name comparison is not case sensitive.

Result

res - an ARexx boolean

See

MacroNotifyCreate	MacroNotifyGetEvent	MacroNotifyInsert
MacroNotifyJoin	MacroNotifySync	

1.111 RexxMustHave Functions - MacroNotifyGetEvent

MacroNotifyGetEvent - gets next event from a MacroNotify

Synopsis

```
ev=MacroNotifyGetEvent(name)
<name>
```

Function

Gets next event from a MacroNotify.

When you receive a MacroNotify signal, you must call this function, till it returns an empty string. Only the MacroNotify creator may use this function.

Inputs

name - the name of the MacroNotify. The name comparison is not case sensitive.

Result

ev - a MacroNotify event

See

MacroNotifyCreate	MacroNotifyFree	MacroNotifyInsert
MacroNotifyJoin	MacroNotifySync	

1.112 RexxMustHave Functions - MacroNotifyInsert

MacroNotifyInsert - inserts an event in a MacroNotify

Synopsis

```
res=MacroNotifyInsert(name,ev,more)
<name>,<ev>,[more]
```

Function

Inserts an event in a MacroNotify.

ev may be one of:

- o INFO

Inputs

name - the name of the MacroNotify. The name comparison is not case sensitive.

ev - the type of the event to insert

more - depends on the type of the event

Result

res - an ARexx boolean

See

MacroNotifyCreate	MacroNotifyFree	MacroNotifyGetEvent
-------------------	-----------------	---------------------

MacroNotifyJoin

MacroNotifySync

1.113 RexxMustHave Functions - MacroNotifyJoin

MacroNotifyJoin - joins a MacroNotify

Synopsis

```
res=MacroNotifyJoin(name)
<name>
```

Function

Joins a MacroNotify.

Inputs

name - the name of the MacroNotify. The name comparison is not case sensitive.

Result

res - an ARExx boolean

See

MacroNotifyCreate
MacroNotifyInsert

MacroNotifyFree
MacroNotifySync

MacroNotifyGetEvent

1.114 RexxMustHave Functions - MacroNotifySync

MacroNotifySync - signals all MacroNotify macros

Synopsis

```
res=MacroNotifySync(name)
[name]
```

Function

Signals with a break_d (2**13) all the macros that join a MacroNotify. Only the MacroNotify creator may use this function.

This function may be use to sync and/or to break all the macros that joint a MacroNotify.

Inputs

name - the name of the MacroNotify. The name comparison is not case sensitive. If it is omitted, all the MacroNotifies created in this macro are interested.

Result

res - an ARexx boolean

See

MacroNotifyCreate

MacroNotifyFree

MacroNotifyGetEvent

MacroNotifyInsert

MacroNotifyJoin

1.115 RexxMustHave Functions - LocalizeStrings

LocalizeStrings - localizes strings

Synopsis

```
res=LocalizeStrings(stem,cat,upTo)
<stem/V>,<cat>,[upTo/N]
```

Function

Localizes an array of strings.

I always hate the ARexx interface of locale.library, mainly for the following reasons:

1. The open catalog is left opened, if you don't close.
2. If you have to read, let's say, 100 localized strings, you must call GetCatalogString() 100 times, which means loosing a lot of time.

So, I wrote this little function to make localization of ARexx scripts easier.

An example will help: let's suppose your script use the strings:

```
"host not found"
"connecting to..."
"can't connect to"
"error from server:"
"server result:"
```

To simply localize them, do

```
strings.0="host not found"
strings.1="connecting to..."
strings.2="can't connect to"
strings.3="error from server:"
strings.4="server result:"
call LocalizeStrings("strings","mycatalog")
```

For any strings found, the functions will try to

obtain the localized string from "mycatalog".

If the string is found, it is replaced in the environment of the calling script.

If the catalog cannot be opened or the string is not found, nothing happen.

The codes read from the catalog go from 0 to the last string found, or to upTo if supplied.

Inputs

stem - where to read/Write strings
cat - the catalog name
upTo - read till this code

Result

res - an ARexx boolean to indicate if the catalog was found

1.116 RexxMustHave Functions - MacroEnv

MacroEnv - reads and sets macro environment parameters

Synopsis

```
call MacroEnv(stem,opt)
<stem/V>,[opt]
```

Function

Reads and sets parameters regarding the macro environment.

The function should be called at the beginning of the macro.

It reads, setting the fields of stem:

Name - the name of the macro
NoExtName - the name of the macro without extension
FullName - the complete name of the macro (with path)
Path - the path of the drawer the macro resides in
Parent - the full name of the macro that called this (if any)
WB? - called from wb? (have sense only if rxs was the tooltype)
Socket? - is there a bsdsocket.library?
Sock? - does the macro have one socket at least?
Sock - the socket id or -1

It sets:

PROGDIR: - macro process ProgDir is set so that the AmigaDOS facility PROGDIR: may be used as an absolute path
CurrentDir - macro process CurrentDir is set to PROGDIR:

STDERR - a logical file named STDERR is opened.
The relative AmigaDOS file is CONSOLE: if it
may be opened, STDIN otherwise. STDERR is used
by PrintFault() and ReadArgs().

opt may be one or more of:

PROGDIR - sets ProgDir
CD - set CurrentDir (only valid if PROGDIR is present)
STDERR - open STDERR

Inputs

stem - where to write parameters
opt - options

Result

The function fails with ARexx error 3 or returns 0.