rxsocket_eng_guide

COLLABORATORS

	TITLE : rxsocket_eng_guide	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		May 24, 2025	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

rxsocket_eng_guide

1.1 main

```
rxsocket.library 9.5

1. Warning
2. About this guide
3. Distribution
4. Author
5. Introduction
6. System requirements
7. Installation
8. Terms
9. Bugs
10. Structures
11. Functions
12. Function by type
13. Passing sockets
14. Thanks
15. Bibliography
16. Note
17. Inetd support
```

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 WARRANTY IS MADE,
ECCEPT THAT *THIS CODE IS TOTALLY BACKDOORS FREE*.

To check the integrity of this archive you must:
- download the rxsocket.md5.lha at http://users.iol.it/alfier/soft/rxsocket. ↩
   html
- store rxsocket.md5 files from the above archive in ram:rxsocket.md5
- unpack your rxsocket archive
- go into "rxsocket/" drawer
- start md5sum -cv ram:rxsocket.md5
```

The rxsocket.md5.lha archive always refers to the last
Aminet distribution, rather than to beta versions and/or
versions avaible at my home page.

1.3 guide

This guide contains links with inline ARexx macros wich
use rmh.library/OpenURL. They will work if and only if
you have both openurl.library and rmh.ibrary installed.

rmh.library is by me and it is included in this archive.

openURL.library is by Troels Walsted Hansen:

"This library was created to make it easier for application programmers to
include clickable URLs in their applications, about windows, etc. ..."

You can find it on aminet at comm/www/OpenURL20.lha

1.4 distribution

rxsocket.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.

1.5 author

I am Alfonso Ranieri .

My e-mail address is alfier@iol.it .

My home page is at http://users.iol.it/alfier/ .

You can find me on:
- #amigaita ircnet~;
- #amyita ircnet .

1.6 introduction

This library is a complete bridge to bsdsocket.library ,
so it is a powerfull ARexx API for internet comunication.

The functions of this library directly call bsdsocket.library
functions, which I'll name "original functions", but this doc

is not an introduction to bsdsocket.library but just a guide
to rxsocket.library functions.

The environment is macro-private: each macro opens bsdsocket.library
and what else must be private and stores a list of "things" that must
be freed on exit.

The way used to handle arguments and results is:

- when the original function wants a non-structure as argument,
  that argument is given to the function;

- when the original function wants a structure as argument, a
  valid ARexx variable name is the argument for that structure:
  various fields of that stem must be set by the user;

- when the original function returns an integer, that integer is returned;

- when the original function returns a structure, a valid ARexx variable
  name is passed as argument to the function and various fields of that
  stem are set by the function. An ARexx boolean is returned.

This is a general policy in my ARexx libraries to try to emulate the AmigaOS
tags programming way.

1.7 installation

Run the installation script.

1.8 requirements

The library needs AmigaOS, version >=2, and a TCP/IP stack.

The library is tested on:
- Miami 2.xx 3.xx MiamiDx any beta release
- Genesis (genesis.library 2.xx 3.xx)
- TermiteTCP 1.50

It works on TermiteTCP 1.50 , but some functions are not avaible with it.

All examples needs rmh.library. It is included in this archive.

1.9 terms

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

- socket : the named space created by socket(), Dup2Docket(), and so on;

- socketfd : the socket descriptor id as an integer value;

```
- addr or address: the Internet address, in dotted form.
  An Internet address is a 32 bits unsigned long, rappresented in the "dotted"
  form as "a.b.c.d" "a.b.c" "a.b" "a" or as a symbolic name.
  In this library addresses are passed/returned in dotted form, e.g. resolve()
  return the dotted form of its argument or -1.
  I wanted to use the original bsdsocket.library API, but ARexx integer
  implementation makes difficult to use integer as Internet addresses.

- 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
```

1.10 bugs

```
- This is not a rxsocket.library bug, but an Amitcp bug:
  DO NOT CREATE ROUTE SOCKET IF USING AMITCP; JUST A SIMPLE
  s=socket("ROUTE","RAW")
  BADLY CRASHES AMITCP !!!
```

1.11 structures

```
The main structures passed or returned to/from functions are:

the bsdsocket.library structures:
- struct hostent        returned by GetHostByName(), GetHostByAddr()
- struct servent        returned by GetServByName(), GetServByNumber()
- struct protoent       returned by GetProtoByName(), GetProtoByNumber()
- struct sockaddr_in    passed and returned to/from various functions

As I said above, I emulate structure as ARexx stemName; an example will help:

we want to get the local protoent of the echo TCP service, so we need a call  ↩
   to
GetServByName() passing it a stemName (see Terms):

    if ~GetGervByName("SE","echo","TCP") then do
        /* failure */
        say "no echo TCP service"
        exit
    end

    /* success */
    say "Name:"  se.ServName
    say "Port:"  se.ServPort
    say "Proto:" se.ServProto

    if se.ServAliases.num=0 then say "No alias"
    else do
        say "Aliases"
```

```
        do i = 0 to se.ServAliases.num-1
            say se.ServAliases.i
        end
    end
```

As you can see, GetServByName() sets, on success, the fields:
- SERVNAME
- SERVPORT
- SERVPROTO
- SERVALIASES.NUM
- SERVALIASES.0 , ... , SERVALIASES.last (last = SERVALIASES.NUM-1)

of the ARexx stem that has as root part that stemName you give the function as the first arguments.

If an array is part of the structure then the number of members are returned in X.Y.NUM and the mebers can be found in X.Y.0 , ... , X.Y.last last = X.Y.NUM -1; so if X.Y.NUM == 0 the array is empty.

For each structure the fields read or set by functions are:

- hostent
  - HOSTNAME
  - HOSTADDRTYPE
  - HOSTLENGTH
  - HOSTALIASES.NUM
  - HOSTALIASES.0, ... ,HOSTALIASES.last (last = HOSTALIASES.NUM-1)
  - HOSTADDRLIST.NUM
  - HOSTADDRLIST.0, ... ,HOSTADDRLIST.last (last = HOSTADDRLIST.NUM-1)

- servent
  - SERVNAME
  - SERVPORT
  - SERVPROTO
  - SERVALIASES.NUM
  - SERVALIASES.0, ... ,SERVALIASES.last (last = SERVALIASES.NUM-1)

- protoent
  - PROTONAME
  - PROTOPROTO
  - PROTOALIASES.NUM
  - PROTOALIASES.0, ... ,PROTOALIASES.last (last = PROTOALIASES.NUM-1)

To make life easier a lot of arguments have their human form and can be passed to functions (directly or in as stem field) as string.
They are expecially:

- FAMILY as in socket(family,type,protocol) or ADDRFAMILY the address family that actually has just the string form "INET"
  can be passed as integer too.

- type as in socket(family,type,protocol) the type of the socket has the string forms
  - "STREAM"
  - "DGRAM"

```
  - "RAW"
  - "RDM"
  - "SEQPACKET"
  can be passed as integer too.

- protocoll as in socket(family,type,protocol) the protocol of the socket has ↩
   the
  string forms
  - "IP"
  - "HOPOPTS"
  - "ICMP"
  - "IGMP"
  - "GGP"
  - "IPIP"
  - "TCP"
  - "EGP"
  - "PUP"
  - "UDP"
  - "IDP"
  - "TP"
  - "IPV6"
  - "ROUTING"
  - "FRAGMENT"
  - "RSVP"
  - "ESP"
  - "AH"
  - "ICMPV6"
  - "NONE"
  - "DSTOPTS"
  - "EON"
  - "ENCAP"
  - "DIVERT"
  - "RAW"
  can be passed as integer too.
```

1.12 functions

```
accept
addr2c
bind
CloseRxsCon
CloseSocket
connect
Dup2Socket
errno
ErrorString
GetHost
GetHostBYADDR
GetHostBYName
GetHostID
GetHostName
GetPeerName
GetProtoByName
GetProtoByNumber
GetServByName
```

```
GetServByPort
GetSocketBase
GetSocketBaseSingle
GetSocketEvents
GetSockName
GetSockOpt
help
HostErrorno
HostErrorString
InetAddr
InetCksum
InetNTOA
IoctlSocket
IsDotAddr
IsLibOn
IsOnSocks
IsUp
IsSocket
LastSocket
Listen
NextRxsReleased
ObtainSocket
OpenConnection
OpenRxsCon
QueryInterfaces
recv
RecvFrom
RecvFromUntil
RecvLine
ReleaseCopyOfSocket
ReleaseSocket
resolve
RxsCall
send
SendTo
SetRxSocketOpt
SetSocketBase
SetSocketBaseSingle
SetSocketSignals
SetSockOpt
ShutDown
socket
SysLog
SysLogCtl
WaitSelect
WriteRxsCon
```

1.13 functions by type

```
Sockets
CloseSocket
Dup2Socket
IsSocket
LastSocket
NextRxsReleased
```

```
ObtainSocket
ReleaseCopyOfSocket
ReleaseSocket
RxsCall
socket
ShutDown
GetSocketEvents

Sockets and socketbase options
IoctlSocket
GetSocketBase
GetSocketBaseSingle
SetSocketBase
SetSocketBaseSingle
GetSockOpt
SetSocketSignals
SetSockOpt

Rx/Tx
recv
RecvFrom
RecvFromUntil
RecvLine
send
SendTo
Listen
OpenConnection
WaitSelect

Connections
accept
bind
connect

Databases
GetHost
GetHostBYADDR
GetHostBYName
GetHostID
GetHostName
GetProtoByName
GetProtoByNumber
GetServByName
GetServByPort
QueryInterfaces
IsUp

Names
GetPeerName
GetSockName
InetAddr
InetNTOA
resolve

Low level
InetCksum
IsOnSocks
```

```
Errors and log
errno
ErrorString
HostErrorno
HostErrorString
SysLog
SysLogCtl

Various
addr2c
CloseRxsCon
IsDotAddr
IsLibOn
help
OpenRxsCon
SetRxSocketOpt
WriteRxsCon
```

1.14 accept

```
accept

Usage: sockfd=accept(socketfd,remote)
<socketfd/N>,<remote/V>

Accepts a connection on a socket after a bind and a listen.
Creates a new socket for the new connection and returns its socketfd.
Fills remote with the sockaddr_in fields of the connected peer.

Returns the socketfd, an integer >=0, or -1 for failure.
```

1.15 addr2c

```
Addr2C

Usage: packetAddr=Addr2C(addr)
<addr/N>

Converts an Internet address, e.g. as returned by resolve
to packed chars.
Usefull when you want to export an address into memory.
```

1.16 bind

```
bind

Usage: res=bind(socketfd,locale)
<socketfd/N>,<locale/V>
```

```
Assign a port number to a socket.
stem must be set as sockaddr_in, usually with ADDRADDR as 0,
but can be not-set at all, e.g. for DGRAM sockets.

Returns -1 for failure.

EXAMPLE

    sock = socket("INET","DGRAM","IP")
    if sock<0 then do
        say "cannot open socket:" errno()
        exit
    end

    local.ADDRFAMILY = "INET"
    local.ADDRADDR = 0
    local.ADDRPORT = 4000

    if bind(sock,"LOCAL")<0 then do
        say "cannot allocate port 4000:" Errno()
        exit
    end
```

1.17 closerxscon

```
CloseRxsCon

Usage res=CloseRxsCon(attempt)
<attempt/N>

Closes the global rxsocket console.
attemp can be 0 or a non negative integer (1) .
If no attemp is specified it is assumed to be 0 .
if attempt is 0, the function wait till all the console users
release it. If attemp is 0, the funtion doesn't wait and the
console is closed iff it is not busy .

Returns an ARexx boolean .
See OpenRxsCon WriteRxsCon
```

1.18 closesocket

```
CloseSocket

Usage res=CloseSocket(socketfd)
<socketfd/N>

Closes a socket.

Returns -1 for failure.
The way how a socket is closed depends on its LINGER parameter value.
```

1.19 connect

```
connect

Usage: res=connect(socketfd,remote)
<socketfd/N>,<remote/V>

Connects the socket to the socketaddr_in as defined in remote .

Returns -1 for failure.

EXAMPLE

    sin.addrFamily = "INET"
    sin.addrPort   = 80
    sin.addrAddr   = addr /* from a call to resolve() */
    if connect(sockfd,"SIN")<0 then do
        say "connect: error" Errno()
        exit
    end
```

1.20 dup2socket

```
Dup2Socket

Usage: sockfd=Dup2Socket(socketfd)
<socketfd/N>

Duplicates an existing socket and returns the new socketfd.
A new internal socket resource is allocated.
It calls the original dup2socket() function with the second
argument as -1.

Returns the new socketfd or -1 for failure.
```

1.21 errno

```
errno

Usage: error=errno()
-

Returns the current error code.
```

1.22 errorstring

```
ErrorString
Usage: errorString=ErrorString(code)
[code/N]
```

```
Returns the error string associated with the error code.
If code is not specified, it is assumed to be the current
error code.
```

1.23 gethostbyaddr

```
GetHostByAddr

Usage: res=GetHostByAddr(host,addr)
<host/V>,<addr/N>

Fills "host" with a hostent data, host given as addr.

Returns an ARexx boolean.
HostErrorno() can be used to get the error code for failure.
```

1.24 gethostbyname

```
GetHostByName

Usage: res=GetHostByName(host,hostName)
<host/V>,<hostName>

Fills "host" with a hostent, host given as name.

Returns an ARexx boolean.
HostErrorno() can be used to get the error code for failure.
```

1.25 gethostid

```
GetHostID()

Usage: id=GetHostID()

Returns the unique identifier of current host.

NOTE:
This function is deprecated.
To get the ip of an interface use QueryInterface() .
```

1.26 gethostname

```
GetHostName

Usage: res=GetHostName(name)
<name/S>
```

```
Fills "name" with the current host name.

Returns an ARexx boolean.
```

1.27 getpeername

```
GetPeerName

Usage: res=GetPeerName(socketfd,remote)
<socketfd/N>,<remote/V>

Set remote with a sockaddr_in of the peer connected to a socket.

Returns an ARexx boolean.

No TermiteTCP.
```

1.28 getprotobyname

```
GetProtoByName

Usage: res=GetProtoByName(stem,protoName)
<stem/V>,<protoName>

Set stem with the protoent of the proto given as name.

Returns an ARexx boolean.
```

1.29 getprotobynumber

```
GetProtoByNumber

Usage: res=GetProtoByNumber(stem,protoID)
<stem/V>,<protoID/N>

Set stem with the protoent of the proto given as number.

Returns an ARexx boolean.
```

1.30 getservbyname

```
GetServByName

Usage: res=GetServByName(stem,serviceName,protoName)
<stem/V>,<serviceName>,<protoName>
```

```
Fills stem with the serventry of the service given as name and protocol.

Returns an ARexx boolean.

No TermiteTCP.
```

1.31 getservbyport

```
GetServByPort

Usage: res=GetServByPort(stem,portNumber,protoName)
<stem/V>,<potNumber/N>,<protoName>

Fills stem with the serventry of the of the service given as port
number and protocol.

Returns an ARexx boolean.

No TermiteTCP.
```

1.32 getsocketbase

```
GetSocketBase

Usage: res=GetSocketBase(stem)
<stem/V>

Gets some global parameters in the bsdsocket.library base.
The original bsdsocket.library function is SocketBaseTagList, which is used to
get/set; here we split it in 2 as GetSocketBase() and SetSocketBase().

You must set the field of stem you want to get, then call the function.
The function fills the fields you choosed with their current value.

The fields accepted are:
- "BREAKMASK"
- "DTABLESIZE"
- "ERRNO"
- "ERRNOSTRPTR"
- "HERRNOSTRPTR"
- "HERRNO"
- "SIGIOMASK
- "SIGURGMASK
- "LOGFACILITY"
- "LOGMASK"
- "LOGSTAT"

Return -1 for failure.

EXAMPLE

    drop a. /* to be sure we don't make a mass :-) */
```

```
a.ERRNOSTRPTR=40   /* must be numeric */
a.BREAKMASK=1      /* can be whatever you want */
a.HERRNOSTRPTR=2   /* must be numeric */

call GetSocketBase("A")
say a.ERRNOSTRPTR       ----->"Message too long"
say a.BREAKMASK         ----->4096
say a.HERRNOSTRPTR      ----->"Host name lookup failure"
```

1.33 getsocketbasesingle

```
GetSocketBaseSingle

Usage: res=GetSocketBaseSingle(opt,var,value)
<opt>,<var/V>,[value/N]

See GetSocketBase().
Just as GetSocketBase() but get a single option.
var is where the value will be written.
value is the value for ERRNOSTRPTR and HERRNOSTRPTR.

Return -1 for failure.
```

1.34 getsocketevents

```
GetSocketEvents

Usage: res=GetSocketEvents(stem)
<stem/V>

Retrieves asynchronous events of sockets, setting the fields:
- ACCEPT
- CLOSE
- CONNECT
- ERROR
- OOB
- READ
- WRITE
of the stem passed as argument, with an ARexx boolean.

Returns the sockefd of the socket interested in the asynchronous events
or -1 if no socket.

Errno() CAN'T be used to get info if failure.
```

1.35 getsocketname

```
GetSocketName

Usage: res=GetSocketName(socketfd,stem)
<socketfd/N>,<stem/V>

Sets stem as a sockaddr_in of the socket.

Returns -1 for failure.
```

1.36 getsockopt

```
GetSockOpt

Usage: res=GetSockOpt(socketfd,level,parm,stem)
<socketfd/N>,<level>,<name>,<stem/V>

Sets stem with value of the parm associated with a socket at level "level".

Levels are:
- "SOCKET"
- "IP"

Valid parms for "SOCKET" are:
- "DEBUG"
- "REUSEADDR"
- "REUSEPORT"
- "KEEPALIVE"
- "DONTROUTE"
- "LINGER"
- "BROADCAST"
- "OOBINLINE"
- "TYPE"
- "ERROR"

The value is written in stem.
If  "LINGER", the fields "ONOFF", "LINGER" of stem are set.

Valid parms for "IP" are:
- "HDRINCL"
- "IPOPTIONS"    just a boolean not an options buffer
- "TTL"
- "TOS"

Returns -1 for failure.
```

1.37 help

```
help

Usage: helpString=help(funName)
<funName>
```

```
Returns the arguments mask string of rxsocket.library function "funName".
```

1.38 hosterrorno

```
HostErrorno

Usage: error=HostErrorno()
-

Returns current host-lookup error.
```

1.39 hosterrorstring

```
HostErrorString

Usage: errorString=HostErrorString(code)
[code/N]

Returns string associated with host-lookup error code.
If code is not specified, it is assumed to be the current
host-lookup error code.
```

1.40 inetaddr

```
InetAddr

Usage: inetAddr=InetAddr(addr)
<addr/N>

Converts IP from dotted form to integer addr.
The STRING returned IS NOT an ARexx number, but just a string of decimal  ↩
   digits.

Returns -1 on error (bad dotted addr).
```

1.41 inetcksum

```
InetCksum

Usage: cksum=InetCksum(data,len)
<data>,[len/N]

Computes an Internet checksum on data for len bytes.
If no len, chekcsum is computed on all data.
The chekcsum is "the 16 bit one's complement of the
one's complement sum of all 16 bit words of 'data'
```

```
for 'len' bytes"; if 'len' is odd a padding byte is
added at the end of data.
```

1.42 inetntoa

```
InetNTOA

Usage: addrString=InetNTOA(hostName)
<addr>

Converts IP address from integer addr to dotted form.
```

1.43 ioctlsocket

```
IoctlSocket

Usage: res=IoctlSocket(socketfd,parm,data,var)
<socketfd/N>,<parm>,<data>,[var/V]

Set or get socket attributes.

The function has 3 differemt form.

To set a socket attribute:
- FIOASYNC      value is /N
- FIONBIO       value is /N
the function must be used in the form
res = IoctlSocket(socketfd,attr,value)
<socketfd/N>,<attrs>,<value>
e.g.
res = IoctlSocket(sock,"FIONBIO",1)
sets as socket non blocking

To get a socket attribute:
- SIOCATMARK
- FIONREAD
the function must be used in the form
res = IoctlSocket(socketfd,attr,var)
<socketfd/N>,<attr>,<var/V>
e.g.
res = IoctlSocket(s,"FIONREAD","A")
gets the number ready to be read in a

To get an interface attribute:
- SIOCGIFADDR
- SIOCGIFDSTADDR
- SIOCGIFBRDADDR
- SIOCGIFNETMASK
- SIOCGIFFLAGS
- SIOCGIFMETRIC
- SIOCGIFMTU
- SIOCGIFPHYS
```

```
the function must be used in the form
res = IoctlSocket(socketfd,attr,name,var)
<socketfd/N>,<attr>,<name>,<var/V>
e.g.
res = IoctlSocket(s,"SIOCGIFADDR","mi0","A")
gets the IFAddr of mi0 (if it exists) and write it in a as a dotted addr.

Returns -1 for failure.

EXAMPLES:
look at ioctl.rexx in examples drawner.

NOTE:
I didn't want to let user set any interface attributes.
I think this is dangerous. Anyway it might change.
```

1.44 isdotaddr

```
IsDotAddr

Usage: res=IsDotAddr(addr)
<addr>

Tests if addr is a "good dotted internet address form".

Returns an ARexx boolean.
```

1.45 islibon

```
IsLibON

Usage: res=IsLibON(name)
[name]

Tests on what stack the rxsocket is working on or if a library is present,
returning an ARexx boolean.

Name is a string made of one or more of the following words separeted by space ↩
    (s):
- MIAMI       running on Miami
- AMITCP      running on AmiTCP (~"MIAMI TTCP")
- TTCP        running on TermiteTCP
- USERGROUP   a usergroup.library is present
- GENESIS     Genesis is installed

If no argument is given, the function returns a string describing
the stack in use. If no stack is running an empty string is
returned.

Nota Bene: Genesis is tested if no stack is running.
```

1.46 isonsocks

```
IsOnSocks

Usage: res=IsOnSocks(wrapper)
[wrapper]

Tests if the stack is running under a socks, e.g. you set a socks
in Miami socks.
It works with Miami and Genesis socks wrapper with no argument.

It flushes memory and searches in ExecBase library list.

Returns an ARexx boolean.
```

1.47 isup

```
IsUp

Usage: res=IsUp(interface)
<interface>

Tests if an interface is up.

Returns:
-1 the specified interface doesn't exist
 0 the interface is down
 1 the interface is up
```

1.48 issocket

```
IsSocket

Usage res=IsSocket(socketfd)
<socketfd/N>

Tests if a socketfd is a valid socket descriptor.

Returns an ARexx boolean.
```

1.49 lastsocket

```
LastSocket()

Usage: socketfd=LastSocket()
--

Returns the last socketfd active in the macro, or -1 if none.
This function is usefull at a beginning of a macro that is supposed
```

```
to be started by RxsCall() or inetd, to check if it has a socket
already opened.
```

1.50 listen

```
listen

Usage: res=listen(socketfd,backlog)
<socketfd/N>,<backlog/N>

Tells system that socket wants to accept connection.
backlog is the max number of connection accepted.

Returns -1 for failure.
```

1.51 nextrxsreleased

```
NextRxsReleased

Usage: key=NextrxsReleased()
-

Returns the next released socket in a child macro called via RxsCall()
from this macro.

See RxsCall() .

Returns a valid key to be used with ObtainSocket() .
If there are no (no more) sockets to get, key is null() .

Examples:

call RxsCall(fun,,"OBTAIN")
key=NextRxsReleased()
do i=0 while key~=null()
    s.i=ObtainSocket(key)
end
```

1.52 obtainsocket

```
ObtainSocket

Usage: sockfd=ObtainSocket(key,family,type,protocol)
<key>,[family],[type[,[protocol]

The function is needed when you want to pass a socket from a macro to another.
It obtains a previously released socket.
Only rxsocket.library released sockets can be obtained.
Key is the key returned by ReleaseSocket() or ReleaseCopyOfSocket().
The way used to handle a safe socket release-obtain is:
```

```
- when a socket is released by ReleaseSocket() or ReleaseCopyOfSocket(), the
  socket is still is linked in a list that belongs to the macro where it was  ↩
     created.
- if a released socket is not obtained it is freed at the exit of the macro  ↩
   where
  it was created;
- if a release socket was obtained with ObtainSocket() it belongs to the
  environment of the macro where it was obtained;
- if ObtainSocket() fails for any reasones, the socket is still in the macro  ↩
   where it
  was created;
- when a socket is released, it can't be used before it is obtained.
- key is the result of ReleaseSocket() , ReleaseCopyOfSocket() or  ↩
   NextRxsReleased ()
  and consists of a packed char of length 8, or 4 on failure.
  Key can be tested with a comparation to null() as we usually do with  ↩
     messages
  from an ARexx port. Key passed to ObtainSocket() is checked to test its  ↩
     coerence,
  anyway, please, don't "play" with it.

Usally ReleaseSocket() is used in a "concurrent service" after a accept() and  ↩
   the
key is given as argument of a macro that must handle the new connection. The
first thing that macro should do is to obtain the socket with a call to
ObtainSocket() and tell the "parent macro" about the result of the operation
(e.g. with an ARexx message on an ARexx port).

If you specify one or more of family , type , proto , the socket is obtained
only if it matches them .

Returns -1 for failure or the socketfd of the obtained socket.
```

1.53 openconnection

```
OpenConnection

Usage: sockfd=OpenConnection(proto,localPort,host,remotePort,stem)
<proto>,<localPort>,[host],[remoteport],[stem/V]

A function that creates a socket binds and/or connects it.

Let's see the different forms:
- proto can be the string "TCP" or "UDP"
- port1 and port2 are service name or port numbers
  if and only if they are service names, they are resolve by getserbyname()

res=OpenConnection(proto,port1)
    resolve port1 if it is a service name
    create a socket
    bind the socket to port1

res=OpenConnection(proto,port1,hostName)
    resolve port1 if it is a service name
    resolve hostName
```

```
    create a socket
    connect the socket to hostName:port1

res=OpenConnection(proto,port1,hostName,port2)
    resolve port1 if it is a service name
    resolve port2 if it is a service name
    resolve hostName
    create a socket
    bind the socket to port1
    connect the socket to hostName:port2

Did you understand?

Take a look at the examples.
Read some docs for the differeces beetwen conneting a socket of type TCP or  ↩
   UDP.

Have fun!

Returns:

 -5   socket can't be bound, e.g. port already bound
 -4   port2 not resolved
 -3   port1 not resolved
 -2   host lookup failure
 -1   bsdbsdsocket.library error
 >=0  socket number id, if success

If present as the 5th argument and on connection, stem is filled as  ↩
   socketaddr_in.
```

1.54 openrxscon

```
OpenRxsCon

Usage res=OpenRxsCon()
-

Tries to open the global rxsocket console.
The global rxsocket console ("console") is a console
to be used for debuggin porpouse.

It's default description is
CON:0/10/280/120/RXSocket Console/WAIT/AUTO/CLOSE
but if the ENV:RXSCON is found, its content is used .

Once the console is opened, rxsocket.library CANNOT be flused
until it is closed via CloseRxsCon() .

Returns an ARexx boolean (1 means the console was opened or it was
already opened).
See CloseRxsCon WriteRxsCon
```

1.55 queryinterfaces

```
QueryInterfaces

Usage: res = QueryInterfaces(stem)
<stem/V>

Reads the interface list and writes interfaces attributes in stem .
Returns the number of the found interfaces, so 0 means none, or -1
that means that the function was not able to create the DGRAM socket
needed for the query. Anyway a positive results means success.

If res is positive, interface attributes are written in
stem.i,...,stem.j where j=res-1, e.g.

res = QueryInterface("INTERFACES")
if res>=0 then do
    say "Found" res "interface(s)"
    do i=0 to res-1
        say interfaces.i.name
    end
end
else say "not able to find any interface"

The attributes are:
- Name
- Family the family of the interface as integer (2 stands for INET)
- AFAddr the address of this interface
- PPaddr
- BAddr
- NMask
- Metric
- MTU
- IFWire
- Flags the decimal value of flags, then
  - up
  - broadcast
  - debug
  - loopback
  - pointtopoint
  - notrailers
  - running
  - noarp
  - promisc
  - allmulti
  - oactive
  - simplex
  - link0
  - link1
  - link2
  - multicast
```

1.56 recv

```
recv

Usage: res=recv(socketfd,buff,len,flags)
<socketfd/N>,<buff/S>,[len/N],[flags]

Receives data from a connected socket. It receives max len bytes and fills ↩
   buff
with the data received.

If len is not specified it is assumed to be 256 .

Flags is one or more of:
- "OOB"
- "PEEK"
- "DONTROUTE"
- "EOR"
- "TRUNC"
- "CTRUNC"
- "WAITALL"
- "DONTWAIT"
- "EOF"
- "COMPAT"
e.g. "OOB PEEK".

Returns -1 for failure or bytes read length.
```

1.57 recvfrom

```
RecvFrom

Usage: res=RecvFrom(socketfd,buff,len,flags,remote)
<socketfd/N>,<buff/S>,[len/N],[flags],[remote/V]

Receives data from a socket. It receives max len bytes and fills buff
with the data received.

If len is not specified it is assumed to be 256 .

If present, remote must be set as sockaddr_in.

Flags is one or more of:
- "OOB"
- "PEEK"
- "DONTROUTE"
- "EOR"
- "TRUNC"
- "CTRUNC"
- "WAITALL"
- "DONTWAIT"
- "EOF"
- "COMPAT"
e.g. "OOB PEEK".

Returns -1 for failure or bytes read length.
```

1.58 recvfromuntil

```
RecvFromUntil

Usage: res=RecvFromUntil(socketfd,buff,len,stopData,flags,remote)
<socketfd/N>,<buff/S>,<len/N>,<stopData>,[flags],[remote/V]

Receives data from a socket until the stopData data occurr.
Very funny function that wait for a string to uccur and then
returns the data received till that string (but without that string).
Next recv of any kind will return data AFTER the stop string.

It receives max len bytes and fills buff with the data
received.

If present, remote must be set as sockaddr_in.

Flags is one or more of:
- "OOB"
- "PEEK"
- "DONTROUTE"
- "EOR"
- "TRUNC"
- "CTRUNC"
- "WAITALL"
- "DONTWAIT"
- "EOF"
- "COMPAT"
e.g. "OOB PEEK".

Returns -1 for failure or bytes read length +1, so 1 means eof.
```

1.59 recvline

```
RecvLine

Usage: res=RecvLine(socketfd,buff,len,flags,remote)
"<socketfd/N>,<buff/S>,[len/N],[flags],[remote/V]

Receives a line from a socket. It receives max len bytes and fills buff with
the data received.

If len is not specified it is assumed to be 256 .

If present stem must be set as sockaddr_in.

Flags is one or more of:
- "OOB"
- "PEEK"
- "DONTROUTE"
- "EOR"
- "TRUNC"
- "CTRUNC"
- "WAITALL"
```

```
- "DONTWAIT"
- "EOF"
- "COMPAT"
e.g. "OOB PEEK".

Returns -1 for failure or bytes read length.

This is a relly bad non buffered readline. Don't use it so much!
This function doesn't work on MiamiDx 0.9.
It was patched, so that if no remote is given, it uses recv() rather
then recvfrom(). So if ou are using this function with a STREAM
socket don't pass it remote.
```

1.60 releasecopyofsocket

```
ReleaseCopyOfSocket

Usage: key=ReleaseCopyOfSocket(socketfd)
<socketfd/N>

Releases a copy of a socket to the public.
Returns a key string to be used with ObtainSocket().

See ObtainSocket().
```

1.61 releasesocket

```
ReleaseSocket

Usage: key=ReleaseSocket(socketfd)
<socketfd/N>

Releases a socket to the public.
Returns a key string to be used with ObtainSocket().

See ObtainSocket().
```

1.62 resolve

```
resolve

Usage: addr=resolve(host)
<host>

Converts IP address from name to dotted form (or from dotted form to itself) .
The functions first tries inet_addr() and then a gethosbyname().

Returns -1 or address of host.
```

1.63 rxscall

```
RXSCall

Usage: res = RxsCall(macro,socketfd,flags)
<macro>,[socketfd/N],flags

This function calls macro , and creates a socket by releasing a copy of
socketfd and calling a special ARexx macro process handler that tries to
obtain that socket.

The socketfd in the called macro is usually DIFFERENT from the one passed.
The function LastSocket() returns the last socketfd created in the macro,
so if the macro was started by this function, LastSocket() always returns a
value>=0.

If socketfd is negative (-1) or it is not specified, no socket is passed.

Local vars, e.g. created bye rmh.library/SetVar() are passed to the child  ↩
   macro.

Let's name the macro in which this function is used "parent" and the macro
called "child" .

flags can be one or more of:
- SYNC      usually the child is called async; if you specify this flag,
            the parent waits for the child to end;
            note that other flags may force it;
- STRING    child is a macro-string rather than a macro file name;
- FUN       child is called as a function;
- RESULT    a result is expected from child; SYNC is forced;
- OBTAIN    every socket released in child, but not obtained at child exit,
            is passed to parent, that can obtain it via NextRxsReleased() ;
            this is the suggested way to share sockets among macros;
- NOERR     if an error occurs in child, it is usually reported to the parent;
            with this flag, you will not be bored by errors occurred in
            child, and if an error occurred, it is written in RC; note that
            this has sense only if SYNC was specified.
- OPENCON   if child has no stdin/stdout, forces the global rxsocket console
            to be opened, so that it will be the stdin/stdout of the macro

NOTA BENE: the socket is always duplicate, so if parents does not need it
           my suggestion is to immediatly close it, expecially if it is a
           STREAM one.

Returns:
- a result from child if RESULT was specified
- 0 if SYNC was not specified and there was no error in child
```

1.64 send

```
send

Usage: res=send(socketfd,data,flags)
```

```
<socketfd/N>,<data>,[flags]

Sends data to a connected socket.

Flags is one or more of:
- OOB
- PEEK
- DONTROUTE
- EOR
- TRUNC
- CTRUNC
- WAITALL
- DONTWAIT
- EOF
- COMPAT
e.g. "OOB PEEK".

Returns -1 for failure or legth of  data send.
```

1.65 sendto

```
SendTo

Usage: res=SendTo(socketfd,data,flags,remote)
<socketfd/N>,<data>,[flags],[remote/V]

Send data to a socket.

If present, remote must be set as sockaddr_in.

Flags is one or more of:
- OOB
- PEEK
- DONTROUTE
- EOR
- TRUNC
- CTRUNC
- WAITALL
- DONTWAIT
- EOF
- COMPAT
e.g. "OOB PEEK".

Returns -1 for failure or length of data send.
```

1.66 setrxsocketopt

```
SetRxSocketOpt

Usage: call SetRxSocketOpt(options)
<options>
```

```
Sets global parameter in rxsocket.library for the current macro.
Actually, options defiined ares:
- HALT    every blocking functions, can be broken via "hi"
          e.g. connect() will be broken by "hi"
          HALT re-sets this option ON again, after a NOHALT
- NOHALT  set HALT by "hi" OFF
```

1.67 setsocketbase

```
SetSocketBase

Usage: res=SetSocketBase(stem)
<stem/V>

Sets global parameter in the bsdsocket.library base.
The original bdsocket.library function is SocketBaseTagList, which is use to
get/set; here we split it in 2 as GetSocketBase() and SetSocketBase().

You must set the field of "stem" with the value you want to set, then call the
function.
The fields are:
- BREAKMASK
- DTABLESIZE
- ERRNO
- HERRNO
- SIGEVENTMASK
- SIGIOMASK
- SIGURGMASK
- LOGFACILITY
- LOGMASK
- LOGSTAT

Returns -1 for failure.
```

1.68 setsocketbasesingle

```
SetSocketBaseSingle

Usage: res=SetSocketBaseSingle(opt,value)
<opt>,<value/N>

See SetSocketBase()
Just as SetSocketBase() but sets only one opt.
opt is the option name
value is the value to set, only numeric for now.

Returns -1 for failure.
```

1.69 setsocketsignals

```
SetSocketSignals

Usage: call SetSocketSignals(intrMask,ioMask,urgMask)
[intrMask/N],[ioMask/N],[urgMask/N]

Tells the stack which signals to use for SIGINT, SIGIO and SIGURG.
The same can be set by SetSocketBase()

Returns always 1.
```

1.70 setsockopt

```
SetSockOpt

Usage: res=SetSockOpt(socketfd,level,parm,value,value2)
<socketfd/N>,<level>,<parm>,<value>,[value2/N]

Sets value of the opt name associated with a socket at level "level".

Levels are:
- SOCKET
- IP
- TCP

Parms for level "SOCKET" are:
- DEBUG           N
- REUSEADDR       N
- REUSEPORT       N
- KEEPALIVE       N
- DONTROUTE       N
- LINGER          N
- BROADCAST       N
- OOBINLINE       N
- SNDBUF          N
- RCVBUF          N
- SNDLOWAT        N
- RCVLOWAT        N
- SNDTIMEO        N
- RCVTIMEO        N
- TYPE            N
- ERROR           N
- EVENTMASK       D

If parm is "EVENTMASK", value is one or more of:
- ACCEPT
- CLOSE
- CONNECT
- ERROR
- OOB
- READ
- WRITE
e.g. "CONNECT ERROR".
```

```
If parm is "LINGER", "SNDTIMEO" or "RCVTIMEO" the 5th argument can be passed
(default 0).

Valid opt for level IP are:
- HDRINCL        N
- TTL            N
- TOS            N

Valid opt for level TCP are:
- NODELAY         N
- MAXSEG          N
- NOPUSH          N
- NOOPT           N

Returns -1 for failure.
```

1.71 shutdown

```
ShutDown

Usage: res=ShutDown(socketfd,how)
<socketfd/N>,<how/N>

Causes all or part of a full-duplex connection on the socket to be shut down.
If how is 0, further receives will be disallowed.
If how is 1, further sends will be disallowed.
If how is 2, further sends and receives will be disallowed.

Returns -1 for failure.
```

1.72 syslog

```
SysLog

Usage: res=SysLog(message,level,facility)
<message>,[level],[facility]

message is a string that can't contain %c if c is different from
- m  %m is the string related to the current errno
- %  %% is a %
The function checks for other form and generates ARexx error 18 if
if find them.

Writes a message to syslog.
Level is on of:
- EMERG
- ALERT
- CRIT
- ERR
- WARNING
- NOTICE
- INFO
```

```
- DEBUG

Facility is on of:
- KERN
- USER
- MAIL
- DAEMON
- AUTH
- SYSLOG
- LPR
- NEWS
- UUCP
- CRON
- AUTHPRIV
- FTP

Always returns 1.
```

1.73 syslogctl

```
SysLogCtl

Usage: res=SysLogCtl(logpointer,logmask,facility,opts)
[logpointer],[logmask],[facility],[opts]

Controlls syslog.

logpointer is a string (copied) to be a tag for the message
(generated in the calling macro.

logmask is one of:
- EMERG
- ALERT
- CRIT
- ERR
- WARNING
- NOTICE
- INFO
- DEBUG
It is a LOG_UPTO() filter mask.

facility is one of:
- KERN
- USER
- MAIL
- DAEMON
- AUTH
- SYSLOG
- LPR
- NEWS
- UUCP
- CRON
- AUTHPRIV
- FTP
```

```
opts is one or more (separated by space(s)) of:
- PID
- CONS
- ODELAY
- NDELAY
- NOWAIT
- PERROR

Returns an ARexx boolean.
```

1.74 socket

```
socket

Usage: sockfd=socket(family,type,protocol)
<family>,<type>,<protocol>

Creates an endpoint for communication and returns a descriptor.
Adds to the local-macro list of open sockets a new link so that
resource can be freed at macro exit.

Returns a socketfd as integer that can be used in every function
wich needs a "socketfd" argument.

Returns -1 for failure.
```

1.75 waitselect

```
WaitSelect

Usage: res=WaitSelect(stem,secs,micro,signals)
<stem/V>,[secs/N],[micro/N],[signals/N]

Waits for events on sockets or a timeout or exec signals.

An example will help.

Let's suppose we have 2 sockets, sf1 and sf2, and we want to controll
if something happens about them. We do:

    WAIT.READ.0 = sfd1  /* to wait for ready to be read event */
    WAIT.READ.1 = sfd2

    WAIT.WRITE.0 = sfd1 /* to wait for ready to be written event */
    WAIT.WRITE.1 = sfd1

    WAIT.EX.0 = sfd1    /* to wait for exceptions events*/
    WAIT.EX.1 = sfd2

     /* we wait for the events above, or 10 seconds or a signal in sig mask */
     res = WaitSelect("WAIT",10,0,sig)
```

```
     /* res may be:
        < 0  error
        = 0  no events on sockets
        > 0  number of ready sockets

       To test wich sockets is ready we make a boolean test on WAIT.0.READ
       and so on
    */

    if WAIT.0.READ then ... /* socket sfd1 is ready to be read */

    The function returns:
    -1 an error occurre
     0 timeout or signal
    >0 number of ready socket for *all* the READ, WRITE, EX.

    The function sets:
    - i.READ   ARexx boolean
    - i.WRITE  ARexx boolean
    - i.EX     ARexx boolean
    - SIGNALS  received signals

    A value of -1 as socket descriptor in READ.i, WRITE.i or EX.i
    is skipped. if no socket descriptor is set, no socket is waited for.

Returns -1 for failure.
```

1.76 writerxscon

```
WriteRxsCon

Usage res = WriteRxsCon(msg)
<msg>

Write msg to the global rxsocket console.
If the console was not opened, it is opened .
If msg doesn't end with a newline ( '\n' , "A"x ), a newline
is added.

Returns an ARexx boolean .
See OpenRxsCon CloseRxsCon
```

1.77 passing sockets

```
Passign sockets means:
- exporting sockets TO another macro
- importing sockets FROM another macro

The general mechanism ReleaseSocket/ObtainSocket can be used to
manage import/export:
- macro A :
  - create socket s
```

```
    - release socket s via ReleaseSocket()
    - send to macro B a message containing the key returned by ReleaseSocket()
      (or call macro B with the key as an argument)
  - macro B :
    - wait for a message from macro A conteining the key to pass to ObtainSocket ↩
      ()
      (or wait to be started from macro A with the key as an argument)
    - tries to obtain the socket via ObtainSocket()
    - reply the message with the result of the operation
      (or in same way tells A it obtained the socket, e.g. via a signal)
  - macro A :
    - wait for answer from macro B
    - test the result: if failure, re-obtain the socket and handles it

A simpler mechanism to EXPORT ONE socket is to use RxsCall funtion:
- macro A :
  - create socket s
  - call macro B via RxsCall(B,s)
  - close socket s ( RxsCall() dup the socket )
- macro B :
  - get the socket via LastSocket() function

From version 9.1 there is a way to IMPORT sockets from a macro:
- macro A :
  - call macro B via RxsCall(B,,"OBTAIN")
- macro B :
  - create its sockets
  - release them via ReleaseSocket()
- macro A :
  - obtain the socket released by macro B via NextRXSReleased() function
With this mechanism you can, e.g, use a "child" macro to connect
to a host and obtain a "connected" socket from the "child" .
```

1.78 thanks

```
  - thanks to shido for his gift "Hi shido! A lot of ovetti for you :-)";

  - thanks to [X_MaN] who introduced me into the irc world and Internet in
    general;

  - thanks to Kruse for his wonderfull Miami ;

  - thans to poing for his help;

  - thanks to Wiz for his help in Genesis support;

  - thans to Amiga "May it leave for ever" .
```

1.79 bibliography

```
  - Quite all rfc ;
```

- "Unix Network Programming" - W. Richard Stevens PTR Prentice Hall;

- socket.library autodoc from MiamiSDK, AmiTCPSDK and TermiteTCPSDK.

1.80 note

Pointers to deallocate the local environment in the library base is saved
in the fields pr_ExitCode and pr_ExitData of the Process structure of
the macro. At exit a chain of pr_ExitCode(pr_ExitData) is called.
Details are avaible on request.

When a function is not avaible, the user is informed via a requester
and an ARexx error 15 (function not found) is returned.

IsLibOn() can be used to test the environment.

rxsocket.library offers an API for other ARexx libraries that needs
to use bsdsocket.library functions in a clean way.
rxsocket.library SDK are avaible on request.

1.81 inetdsupport

With rxsocket.library you can easly create full functional inetd
server.

A little program called "rxs" is supplied. It should be installed
in c: (as the install script does) or in SYS:Rexxc .

It launches an ARexx macro in a special way, so that it can
obtain the socket from inetd.

In inetd database you must
- use rxs (complete path) as "Service"
- use rxs as "Name"
- specify the name of the macro (and its arguments) as "Args"

rxs template is:
"CON=CONSOLE/S,MACRO/A/F"

CONSOLE macros called from inetd have no stdin/stdout, anyway
they can be forced to use rxsocket global console as stdin/stdout
for debugging porpouses. This switch forces the global rxsocket
console to be opened, if it is not. See OpenRxsCon .
rxsco is a litle program to open the console via shell;
rxscc close the console.

MACRO is the name of the macro with its arguments

In the macro, you can obtain the socket passed by inetd, via
LastSocket() (NOTA BENE: this must be called BEFORE other
sockets are created):
...

```
s=LastSocket()
if s>=0 then /* ok I was called from inet and the socket is s*/
...
```

If LastSocket() returns -1, the macro was NOT started from inetd and has no socket. In this event, the macro can choose to run as a stand-alone service rather than an inetd service.