rxsocketeng

COLLABORATORS			
	TITLE : rxsocketeng		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		May 28, 2025	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

rxsocketeng

1.1 main

```
--------------------------------------------------------------------------------

                      RXSOCKET.LIBRARY version 2.1

--------------------------------------------------------------------------------

                              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, ECCEPT THAT
              "THIS CODE IS TOTALLY BACKDOORS FREE".

--------------------------------------------------------------------------------


 Introdution
 Installation
 System Requirements
 Author
 Distribution
 Terms
 Bugs
 Structures
 Functions
 Thanks
 Bibliography
 To do
 Note
 Changes
 Inetd support

--------------------------------------------------------------------------------
```

1.2 introdution

The goal of this library is to offer a set of socket functions to ARexx-macro
to makes easy to access the Internet from ARexx.

The functions of this library directly call bsdsocket.library functions, which
I'll call "original functions", so this doc does not want to be an introdution
to bsdsocket.library but just an explanation about the rxsocket.library functions.

The environment is macro-private. Each macro opens the bsdsocket.library and
has a list of "things" that must be freed on exit.

The way used to handle argument or results is:

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

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

- when the original bsdsocket.library returns an integer, that integer is returned ↩
  ;

- when the original bsdsocket.library returns a structure, a valid ARexx varable
  name is passed as argument to the function and various fields of that setm are
  set by the function.

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

The library offers quite all the bsdsocket.library functions, and some other
functions of a "higher level" (hehe just a couple for now). The library does not
directly support RAW packet, or protocolls different from TCP, UDP, but creating
a memory block and filling it by hands makes use of other protocolls and RAW
packets possible.

1.3 installation

No installation script is given.

To install the library:
- copy rxsocket.library to LIBS:
- write in a shell: rx "call addlib('rxsocket.library',0,-30)

1.4 requirements

The library needs AmigaOS, version >=2, and a TCP/IP stack.
The library is tested on Miami and it works on AmiTCP.
It works on TermiteTCP ver 1.50, but same functions are not avaible
with that stack.

1.5 author

The author is:

    Alfonso Ranieri

His e-mail address is:

    alfier@iol.it

You can find him at:

    - #amichat   dalnet;
    - #amigaita   ircnet;
    - #amyita   ircnet.

You can find the last version of this library at:

    http://users.iol.it/alfier/

1.6 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.

1.7 terms

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

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

- socketfd: the socket descriptor id as numeric value;

- addr or address: the Internet address, in dotted form (".").
  An Intenert 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 addressed are passed/returned in dotted form, i.e. resolve()
  return the dotted form of is argument or -1.
  I wanted to use the original bsdsocket.library API, but ARexx integer
  implementation makes difficult to use intger 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

1.8 bugs

```
There are bugs in our wonderfull kernel.
There are bugs in very expansive commercial products.
Why shouldn't they be here?

There is actually a well known problems:

on Miami after a ReleaseSocket()/ObtainSocket() if the socket obtained
is passed to connect() an enforcer hit cames out.
This is a problem of Miami, not mine.

If you find bugs (and I am sure you'll do) write me asap, please.
```

1.9 structures

```
Actually 4 structures are passed or returned to/from functions.

They 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.SERVSPORT
    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 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)

- struct sockaddr_in
  passed as argument to functions
  - ADDRFAMILY /D (just "INET" for now)
  - ADDRPORT /N
  - ADDRADDR /N
  returned from functions
  - ADDRLEN
  - ADDRFAMILY
  - ADDRPORT
  - ADDRADDR

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" and MUST BE THAT STRING IN THIS VERSION OF THE LIBRAY.
  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.

Hehe just few protocoll will work in this version of the libray, but ...
If you are pretty good in TCP/IP and Arexx programming you can get same mem in  ↩
   ARexx,
create your own packet, i.e. an ICMP packet, filling the right values for IP  ↩
   header
and ICMP header, open a row socket setsockopt() socket as IP_HDRINCL and send your
own packet. Maybe in future versions, general packet creation will be performed.
```

1.10 functions

```
THIS IS NOT A DOC ABOUT BSDSOCKET FUNCTIONS.
READ A GOOD BSDSOCKET FUNCTIONS BOOKS OR A SOCKET.LIBRARY
AUTODOC FOR USE, ARGUMENTS AND RESULTS OF THESE FUNCTIONS.

 @{ "ACCEPT" LINK "ACCEPT"}
 @{ "ADDR2C" LINK "ADDR2C"}
 @{ "BIND" LINK "BIND"}
 @{ "CLOSESOCKET" LINK "CLOSESOCKET"}
 @{ "CONNECT" LINK "CONNECT"}
 @{ "DUP2SOCKET" LINK "DUP2SOCKET"}
 @{ "ERRNO" LINK "ERRNO"}
 @{ "ERRORSTRING" LINK "ERRORSTRING"}
 @{ "GETHOST" LINK "GETHOST"}
 @{ "GETHOSTBYADDR" LINK "GETHOSTBYADDR"}
```

```
 @{ "GETHOSTBYNAME" LINK "GETHOSTBYNAME"}
 @{ "GETHOSTID" LINK "GETHOSTID"}
 @{ "GETHOSTNAME" LINK "GETHOSTNAME"}
 @{ "GETPEERNAME" LINK "GETPEERNAME"}
 @{ "GETPROTOBYNAME" LINK "GETPROTOBYNAME"}
 @{ "GETPROTOBYNUMBER" LINK "GETPROTOBYNUMBER"}
 @{ "GETSERVBYNAME" LINK "GETSERVBYNAME"}
 @{ "GETSERVBYPORT" LINK "GETSERVBYPORT"}
 @{ "GETSOCKETBASE" LINK "GETSOCKETBASE"}
 @{ "GETSOCKETEVENTS" LINK "GETSOCKETEVENTS"}
 @{ "GETSOCKNAME" LINK "GETSOCKNAME"}
 @{ "GETSOCKOPT" LINK "GETSOCKOPT"}
 @{ "HELP" LINK "HELP"}
 @{ "HOSTERRORNO" LINK "HOSTERRORNO"}
 @{ "HOSTERRORSTRING" LINK "HOSTERRORSTRING"}
 @{ "INETCKSUM" LINK "INETCKSUM"}
 @{ "IOCTLSOCKET" LINK "IOCTLSOCKET"}
 @{ "ISDOTADDR" LINK "ISDOTADDR"}
 @{ "ISLIBON" LINK "ISLIBON"}
 @{ "ISSOCKET" LINK "ISSOCKET" }
 @{ "LISTEN" LINK "LISTEN"}
 @{ "MIAMIDISALLOWDNS" LINK "MIAMIDISALLOWDNS"}
 @{ "MIAMIGETPID" LINK "MIAMIGETPID"}
 @{ "MIAMIISONLINE" LINK "MIAMIISONLINE"}
 @{ "MIAMIONOFFLINE" LINK "MIAMIONOFFLINE"}
 @{ "MIAMISUPPORTSIPV6" LINK "MIAMISUPPORTSIPV6"}
 @{ "OBTAINSOCKET" LINK "OBTAINSOCKET"}
 @{ "OPENCONNECTION" LINK "OPENCONNECTION"}
 @{ "RECV" LINK "RECV"}
 @{ "RECVFROM" LINK "RECVFROM"}
 @{ "RECVLINE" LINK "RECVLINE"}
 @{ "RELEASECOPYOFSOCKET" LINK "RELEASECOPYOFSOCKET"}
 @{ "RELEASESOCKET" LINK "RELEASESOCKET"}
 @{ "RESOLVE" LINK "RESOLVE"}
 @{ "SEND" LINK "SEND"}
 @{ "SENDTO" LINK "SENDTO"}
 @{ "SETSOCKETBASE" LINK "SETSOCKETBASE"}
 @{ "SETSOCKETSIGNALS" LINK "SETSOCKETSIGNALS" }
 @{ "SETSOCKOPT" LINK "SETSOCKOPT"}
 @{ "SHUTDOWN" LINK "SHUTDOWN"}
 @{ "SOCKET" LINK "SOCKET"}
 @{ SOCKATMARK LINK SOCKATMARK }
 @{ "SYSLOG" LINK "SYSLOG"}
 @{ "WAITSELECT" LINK "WAITSELECT"}
```

1.11 accept

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

Accepts a connection on socket after a a bind() and 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 as an integer >=0 or -1 for failure.
```

1.12 addr2c

```
ADDR2C
Usage: packetAddr=Addr2C(addr)
<addr/N>

Converts an Internet address, i.e. as returned by resolve(), to packed chars.
Usefull when you have to export an address into memory.
```

1.13 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.

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
```

1.14 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.
```

1.15 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
```

1.16 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.
```

1.17 errno

```
ERRNO
Usage: error=errno()
-

Returns the current error code.
```

1.18 errorstring

```
ERRORSTRING
Usage: errorString=ErrorString(code)
<code/N>

Returns the error string associated with the error code.

No TermiteTCP
```

1.19 gethost

```
Usage: res=GetHost(host,name)
<host/V>,<name>

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

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

1.20 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.
```

1.21 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.
```

1.22 gethostid

```
GETHOSTID
Usage: id=GetHostID()

Returns the unique identifier of current host.
```

1.23 gethostname

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

Fills "name" with the current host name.

Returns an ARexx boolean.
```

1.24 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
```

1.25 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.
```

1.26 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.
```

1.27 getservbyname

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

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

Returns an ARexx boolean.

No TermiteTCP
```

1.28 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
```

1.29 getsocketbase

```
GETSOCKETBASE
Usage: res=getSocketBase(stem)
<stem/V>

Gets same 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"
```

1.30 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.
```

1.31 getsockname

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

Sets stem as a sockaddr_in of the socket.

Returns -1 for failure.
```

1.32 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.
```

1.33 help

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

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

1.34 hosterrorno

```
HOSTERRORNO
Usage: error=HostErrorno()
-
Returns current host-lookup error.
```

1.35 hosterrorstring

```
HOSTERRORSTRING
Usage: errorString=HostErrorString(code)
<code/N>

Returns string associated with host-lookup error code.
```

1.36 inetaddr

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

Converts IP from dotted form (XXX.XXX.XXX.XXX) to integer addr.
Returns -1 on error (bad addr).
```

1.37 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.
```

1.38 inetntoa

```
INETNTOA
Usage: addrString=InetNTOA(hostName)
<addr>

Converts IP address from integer addr to dotted form (XXX.XXX.XXX.XXX).
```

1.39 ioctlsocket

```
IOCTLSOCKET
Usage: res=IOCtlSocket(socketfd,parm,value)
<socketfd/N>,<parm>,value/N>

Controls socket parameters.

Actual parm values are:
- "FIOASYNC"
- "FIONBIO"
- "FIONREAD"
- "SIOCATMARK"
(Take a look at bsdsocket.library/IoctlSocket)

Returns -1 for failure.
```

1.40 isdotaddr

```
Usage: res=IsDotAddr(addr)
<addr>

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

Returns an ARexx boolean.
```

1.41 islibon

```
ISLIBON
Usage: res=IsLibON(name)
<name>

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

Value for name are:

- "MIAMI"       running on Miami
- "TTCP"        running on TCP
- "USERGROUP"   (not yet usefull)
```

1.42 issocket

```
Uso: res=IsSocket(socketfd)
<socketfd/N>

Tests if a socket exists.

Returns an ARexx boolean.
```

1.43 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.
A backlog 5 means max number as defined elsewhere in stack.

Returns -1 for failure.
```

1.44 miamidisallowdns

```
Usage: MiamiDisallowDNS(status)
[status/N]

MIAMIONLY FUNCTION

Disabled the extern DNS lookup. Default for status is 0.

Returns always 1.
```

1.45 miamigetpid

```
Usage pid = MiamiGetPid()
-

MIAMIONLY FUNCTION

Returns the pid of the process as packed chars.
```

1.46 miamionoffline

```
Usage: MiamiOnOffline(interface,status)
<interface>,[statu/N]

MIAMIONLY FUNCTION

Switch the status of the interface. Deault value for status is 0.
Interface must be one of:
- "mi0"
- "lo0"
The functions doen't wait for the switching to complete and returns
always 1.
```

1.47 miamiisonline

```
Usage: res = MiamiIsOnline(interface)
<interface>

MIAMIONLY FUNCTION

Tests if the given interface is online.
Interface must be one of:
- "mi0"
- "lo0"

Returns an ARexx boolean.
```

1.48 miamisupportsipv6

```
Usage: res = MiamiSupportsIPV6()
-

MIAMIONLY FUNCTION

Tests if the current version of Miami supports the IPV6 protocol.

Returns an ARexx boolean.
```

1.49 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 ARexx macro released socket 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 in the private environment of 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() or ReleaseCopyOfSocket() 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
(i.e. with an ARexx message on an ARexx port).

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

1.50 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 connect it.

Let's see the different forms (proto can be the string "TCP" or "UDP"):

res=OpenConnection(proto,4050)
  create a socket
  bind the socket to port 4050

res=OpenConnection(proto,"funnyService")
  serach for a local serv entry with the name "funnyService"
  create a socket INET STREAM TCP or INET DGRAM UDP
  bind the socket on the service port
```

```
res=OpenConnection("TCP","echo","www.nasa.org")
  search for a local serv entry with the name "echo"
  resolve "www.nasa.org"
  create a socket INET STREAM TCP
  connect the socket to www.nasa.org:echo

res=OpenConnection("UDP","echo","www.nasa.org")
  ARexx Error 18

res=OpenConnection("TCP",4000,"www.nasa.org")
  Arexx Error 18

res=OpenConnection("UDP",4000,"www.nasa.org","echo")
  search for a local serv entry with the name "echo"
  resolve "ww.nasa.org"
  create a socket INET DGRAM UDP
  bind the socket to port 4000
  connect the socket to www.nasa.org:echo

Did you understand?

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

Have fun!

Returns:
- -3   server entry serch failure
- -2   host lookup failure
- -1   bsdbsdsocket.library error
- >=0  socket number id

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

No TermiteTCP
```

1.51 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.

Flags is one or more of:
- "OOB"
- "PEEK"
- "WAITALL"
i.e. "OOB PEEK".

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

1.52 recvfrom

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

Receives data from a socket. I 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"
- "WAITALL"
i.e. "OOB PEEK".

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

1.53 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. I receives max len bytes and fills buff with the  ↩
    data
received.

If present stem must be set as sockaddr_in.

Flags is one or more of:
- "OOB"
- "PEEK"
- "WAITALL"
i.e. "OOB PEEK".

Returns -1 for failure or bytes read length.

This is a relly bad non buffered readline. Don't use it so much!
```

1.54 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().
```

1.55 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().
```

1.56 resolve

```
RESOLVE
Usage: addr=resolve(host)
<host>

Converts IP address from name to integer.
The functions tries first inet_addr() and then a gethosbyname().

Returns -1 or address of host.
```

1.57 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"
i.e. "OOB PEEK".

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

1.58 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"
i.e. "OOB PEEK".

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

1.59 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.
```

1.60 setsocketsignals

```
Uso: 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()
The use of this function is deprecated in Miami autodoc.

Returns always 1.
```

1.61 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"
i.e. "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.
```

1.62 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.
```

1.63 syslog

```
SYSLOG
Usage: res=SysLog(message)
<message>

Writes a message to syslog.
```

1.64 socket

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

Creates an endpoint for communication and returns a descriptor.
It also 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
"socket" argument.

Returns -1 for failure.
```

1.65 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 */
```

Returns -1 for failure.

1.66 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;

- thans to poing for his help;

- thans to Amiga "May it leave for ever!";

- thanks to Kruse for his wonderfull Miami "Hey man I really hope those rumours about backdoors are not true at all";

1.67 bibliography

- Quite all rfc (<ds.internic.net:21> anonymous ftp);

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

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

1.68 todo

- first of all a serious debugging :-);

- init_inetd() function;

- hehe a pretty good reacvline();

- direct creation of RAW packets, icpm packet and so on;

- miami.library and usergroup.library functions;

- higher lever functions.

1.69 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.

1.70 changes

Changes from version < 1.9:

- in old versions Internet addresses were rappresented as ARexx integer;
  That was source of many problems, so I decided to rappresent Internet addresses in their dotted form.
  The result is that the functions InetAddr() and InetNtoA() was eliminated, the function IsDotted() was added and any function that returns or sets an Internet addresses makes it in dotted form.

Changes from version < 2.0:

- rmh.functions are back to rmh.library, part of the rxsocket.lha archive

1.71 inetdsupport

The support for inetd of AmiTCP and Miami is made as:

- as service name in inetd db you must use the name of a program that calls the macro;

- because of inetd use the pr_ExitCode and pr_ExitData of the Process to pass the arguments
  to obtain a socket, it is not possible to use rx as the program to run the macro ;
  a specila rx version called rxs is inclued in the archive;
  the macro must be the argument of rxs;

- if the macro exists, the socket it must handle is socket 0.