rxsocket_english_guide

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

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

rxsocket_english_guide

1.1 RxSocket 30.0

```
RxSocket 30.0 © 2000, 2001 Alfonso Ranieri

 1. Introduction
 2. Author
 3. WRID
 4. Terms
 5. Bugs
 6. Structures
 7. Functions
 8. Passing sockets
 9. Inetd support
10. Thanks
11. Note
12. Bibliography
```

1.2 RxSocket Introduction

```
Introduction


  RxSocket is a complete bridge to bsdsocket.library, so it is a
  powerful ARexx API for internet communication programming.

  With RxSocket you will be able to create internet applications
  in the same way you would do in any high programming languages.

  The functions of rxsocket.library directly call bsdsocket.library
  ones, named here original functions, but this document
  is a guide to rxsocket.library functions and not an introduction
  to bsdsocket.library. This mean that you must be familiar with
  the commons terms and techniques of the internet programming.


  The environment is macro-private: each macro opens the
  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.

1.3 RxSocket Author

Author

I am Alfonso Ranieri

My e-mail address is alforan@tin.it

My Amiga Home Page is at
http://web.tiscalinet.it/amiga

You can find the last version of RxSocket at
http://web.tiscalinet.it/amiga/english/soft/rxsocket.htm

1.4 RxSocket Warning, Requirements, Introduction and Distribution

Warning, Requirements, Introduction 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 WARRANTY IS MADE.

Requirements

o RxSocket needs:

. AmigaOS version 2 or higher
. a TCP/IP stack
. rmh.library, included in RxSocket archive

o RxSocket was tested with:

. Miami 2.xx 3.xx MiamiDx any beta release
. Genesis (genesis.library 2.xx 3.xx)
. TermiteTCP 1.50: it works, but some functions are not available with it.

Installation

Run the installation script.

Distribution

RxSocket is Freeware

1.5 RxSocket Terms

Terms

The main terms used in this document are:

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

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

o socketfd: the socket descriptor id as an integer value

o addr or address: the Internet address, in dotted form. An Internet address is a 32 bits unsigned long, represented in the "dotted" form as "a.b.c.d" "a.b.c" "a.b" "a" or as a symbolic name. In RxSocket addresses are represented in dotted form, e.g. resolve() returns the dotted form of its argument or -1.

o Types of arguments:

. D data -- any ARexx data
. N numeric /N integral number
. S symbol /S ARexx valid symbol

```
          . V stem name   /V  as S but with length<20
```

1.6 RxSocket Bugs

```
Bugs

  None known.
```

1.7 RxSocket Structures

```
Structures


  The main structures supplied to or set by functions are:

      o sockaddr_in passed to and returned from various functions
        (e.g. connect() and GetPeerName):
          . AddrAddr
          . AddrFamily
          . AddrPort

      o hostent returned by GetHostByName() GetHostByAddr():
          . HostName
          . HostAddrType
          . HostLength
          . HostAliases.num
          . HostAliases.0, ... ,HostAliases.last
            (last = HostaliAses.num-1)
          . HostAddrList.num
          . HostAddrList.0, ... ,HostAddrList.last
            (last = HostaDdrList.num-1)

      o servent returned by GetServByName() GetServByPort():
          . ServName
          . ServPort
          . ServProto
          . ServAliases.num
          . ServAliases.0, ... ,ServAliases.last
            (last = ServAliases.num-1)

      o protoent returned by GetProtoByName() GetProtoByNumber():
          . ProtoName
          . ProtoProto
          . ProtoAliases.num
          . Protoaliases.0, ... ,ProtoAliases.last
            (last = ProtoAliases.num-1)

  Parameters supplied to functions or set in stem fields can be
  specified as number and strings

      o Family as in socket(FAMILY,type,protocol) or the field
        AddrFamily of sockaddr_in has is one of:
```

```
        . Inet

    o Type as in socket(family,TYPE,protocol) is one of:
        . Stream
        . Dgram
        . Raw
        . Rdm
        . SeqPacket

    o Protocol as in socket(family,type,PROTOCOL) if one of:
        . 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
```

1.8 RxSocket Functions

```
Functions


Sockets
  accept                bind                  CloseSocket           connect
  Dup2Socket            FreeLineRead          GetSocketEvents        ↩
     GetSockOpt
  InitLineRead          IoCtlSocket           IsOnSocks             IsSocket
  LastSocket            LineRead              listen                 ↩
     NextRxsReleased
  ObtainSocket          OpenConnection        recv                  RecvFrom
  RecvFromUntil         RecvLine              ReleaseCopyOfSocket    ↩
     ReleaseSocket
  send                  SendTo                SetRxSocketOpt         ↩
     SetSockOpt
  ShutDown              socket                WaitSelect
```

```
Databases
  addr2c                GetHost               GetHostByAddr          ↩
     GetHostByName
  GetHostID             GetHostName           GetPeerName            ↩
     GetProtoByName
  GetProtoByNumber      GetServByName         GetServByPort          ↩
     GetSockName
  InetAddr              InetLNaof             InetMakeAddr           InetNetOf
  InetNtoa              IsDotAddr             IsUp                   ↩
     QueryInterfaces
  resolve

Errors
  errno                 ErrorString           HostErrorno            ↩
     HostErrorString

SocketBase
  GetSocketBase         GetSocketBaseSingle   SetSocketBase          ↩
     SetSocketBaseSingle
  SetSocketSignals

Various
  help                  InetCkSum             IsLibOn                ↩
     CloseRxSocket
  RxsCall               SysLog                SysLogCtl

RxsConsole
  CloseRxsCon           OpenRxsCon            WriteRxsCon
```

1.9 RxSocket Functions - accept

```
accept - accepts connections

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

Function

  The argument 'socketfd' is a socket, created with socket(), bound
  to an address with bind(), and that is listening after a listen().

  The function extracts the first connection request on the queue of
  pending connections and creates a new socket with the same
  properties of socketfd.

  If no connection is present on the queue and the socket is not
  marked as non-blocking, accept() blocks until a connection comes.
  If no connection is present on the queue and the socket is marked
  as non-blocking, accept() returns [EWOULDBLOCK].

  The returned socket can be used to accept more connections.
  The original socket remains opened.

  The function fills 'remote' with the sockaddr_in fields
```

```
    of the connected peer.

    The function must be used with STREAM sockets.

    WaitSelect() can be used to wait for connection, selecting the
    socket for READ.

    The function fails if:

        o [EBADF]       - 'socketfd' is invalid.

        o [EOPNOTSUPP]  - The socket is not of type STREAM.

        o [EWOULDBLOCK] - The socket is marked as non-blocking
                          and no connections are present.

  Inputs
    socketfd - the socket to accept connection on
    remote   - an ARexx stem name

  Result
    sockfd - the new connected socketfd or -1 for failure

 See
   errno()
```

1.10 RxSocket Functions - Addr2C

```
  Addr2C - converts from dotted to packed chars.

  Synopsis
    packetAddr=Addr2C(addr)
    <addr/N>

  Function
    Converts an Internet address, e.g. as returned
    by resolve() to packed chars.

    Useful when you want to export an address into memory.

  Inputs
    addr - a dotted address

  Result
    packedChar - the address as a packed chars string
```

1.11 RxSocket Functions - bind

```
  bind - binds a socket to a name

  Synopsis
    res=bind(socketfd,name)
```

```
    <socketfd/N>,<locale/V>


  Function
    Assigns a name to an unnamed socket.

    When a socket is created it only exists in the named space
    given by the address family.

    bind() gives a name to the socket.

    'name' must be set as sockaddr_in.
    ADDRADDR may be as 0 or not set at all, e.g. for DGRAM sockets.

    The function fails if:

        o [EBADF]         - 'socketfd' is invalid.

        o [EADDRNOTAVAIL] - The specified address is not available
                            from the local machine.

        o [EADDRINUSE]    - The specified address is already in use.

        o [EINVAL]        - The socket is already bound to an
                            address.

  Inputs
    socketfd - the socket to bind to 'name'
    name     - a sockaddr_in to bind the socket to

  Result
    res - 0 for success, -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

  See
    accept() listen() errno()
```

1.12 RxSocket Functions - CloseRxsCon

```
  CloseRxsCon - closes the global RxSocket console
```

```
Synopsis
  res=CloseRxsCon(attempt)
  <attempt/N>

Function
  Closes the global RxSocket console.

  'attempt' is an ARexx boolean:

      o if 'attempt' is omitted it is assumed to be 0

      o if 'attempt' is 0, the function wait till all the console
        users release it.

      o if 'attempt' is not 0, the function doesn't wait and the
        console is closed iff it is not busy.

Inputs
  attempt - the way the console is closed, see above

Result
  res - 0 the console was not closed, 1 it was

See
  OpenRxsCon() WriteRxsCon()
```

1.13 RxSocket Functions - CloseRxSocket

```
CloseRxSocket - closes local structures

Synopsis
  res = CloseRxSocket()
  -

Function
  When the TCP/IP stack is closed, it sends a ctrl_c to all the
  processes that are using its libraries. It means that even if
  your macros closed all the sockets it may receive a ctrl_c
  and so be compelled to exit. To prevent that, you may use
  this function. It will try to close all the libraries bases,
  so that you may go on without any problem. CloseRxSocket()
  will also close all RxLibnet libraries.

  WARNING: all bsdsocket.library options set are lost.

Result
  res - an ARexx boolean that indicates if all the libraries
        base were closes.
```

1.14 RxSocket Functions - CloseSocket

```
CloseSocket - closes a socket

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

Function
  Closes a socket.

  Sockets are closed when the macro exits, but, because of the
  per-macro max number of opened sockets is limited, it is useful
  to close a socket when it is not needed anymore.

  The function may fail if:

      o [EBADF] - 'socketfd' is invalid

Inputs
  socketfd - a socket descriptor

Result
  res - 0 for success, -1 for failure

See
  ShutDown() errno()
```

1.15 RxSocket Functions - connect

```
connect - connects a socket

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

Function
  Connects the socket to the socketaddr_in as defined in 'remote'.

  If the socket is DGRAM, the function specified the peer with which
  the socket is associated: the address specified is that to which
  datagram are sent and from which datagram are received.

  If the socket is STREAM, the function tries to make a connection
  to another socket as specified in 'remote'.

  STREAM sockets may connect only once.

  DGRAM sockets may connect multiple times to change their
  association and connect to a invalid address to dissolve it.

  The function may fail if:

      o [EBADF]         - 'socketfd' is invalid.

      o [EADDRNOTAVAIL] - The specified address is not available on
```

```
                          this machine.

      o [EAFNOSUPPORT]  - Addresses in the specified address family
                          cannot be used with this socket.

      o [EISCONN]       - The socket is already connected.

      o [ETIMEDOUT]     - Connection timeout.

      o [ECONNREFUSED]  - The attempt to connect was forcefully
                          rejected.

      o [ENETUNREACH]   - The network isn't reachable from this host.

      o [EADDRINUSE]    - The address is already in use.

      o [EINPROGRESS]   - The socket is non-blocking and
                          the connection cannot be completed
                          immediately. It is possible to
                          WaitSelect() for completion by
                          selecting the socket for writing.

      o [EALREADY]      - The socket is non-blocking and a previous
                          connection attempt has not yet been
                          completed.

Inputs
  socketfd - the socket to connect
  remote   - an ARexx stem name set as a sockaddr_in

Result
  res - 0 for success, -1 for failure

Example
  sin.addrFamily = "INET"
  sin.addrPort   = 80
  sin.addrAddr   = addr
  if connect(sockfd,"SIN")<0 then do
      say "connect: error" Errno()
      exit
  end

See
  errno()
```

1.16 RxSocket Functions - Dup2Socket

```
Dup2Socket - duplicates a socket

Synopsis
  sockfd=Dup2Socket(socketfd,newsockfd)
  <socketfd/N>,[newsockfd/N]

Function
  Duplicates an existing socket and returns the new socketfd.
```

```
  A new internal socket resource is allocated.
  If 'newsockfd' is not given, it is assumed to be -1.

  The function may fail if:

      o [EBADF] - 'socketfd' or 'newsockfd' is invalid

Inputs
  socketfd  - the socket to duplicate
  newsockfd - the new socket descriptor

Result
  res - 0 for success, -1 for failure

See
  errno()
```

1.17 RxSocket Functions - errno

```
errno - returns the current bsdsocket.library error.

Synopsis
  error=errno()
  -

Function
  Returns the current error code, one of:

      o EPERM            1      Operation not permitted
      o ENOENT           2      No such file or directory
      o ESRCH            3      No such process
      o EINTR            4      Interrupted system call
      o EIO              5      Input/output error
      o ENXIO            6      Device not configured
      o E2BIG            7      Argument list too long
      o ENOEXEC          8      Exec format error
      o EBADF            9      Bad file descriptor
      o ECHILD          10      No child processes
      o EDEADLK         11      Resource deadlock avoided
      o ENOMEM          12      Cannot allocate memory
      o EACCES          13      Permission denied
      o EFAULT          14      Bad address
      o ENOTBLK         15      Block device required
      o EBUSY           16      Device busy
      o EEXIST          17      File exists
      o EXDEV           18      Cross-device link
      o ENODEV          19      Operation not supported by device
      o ENOTDIR         20      Not a directory
      o EISDIR          21      Is a directory
      o EINVAL          22      Invalid argument
      o ENFILE          23      Too many open files in system
      o EMFILE          24      Too many open files
      o ENOTTY          25      Inappropriate ioctl for device
      o ETXTBSY         26      Text file busy
      o EFBIG           27      File too large
```

- ENOSPC 28 No space left on device
- ESPIPE 29 Illegal seek
- EROFS 30 Read-only file system
- EMLINK 31 Too many links
- EPIPE 32 Broken pipe
- EDOM 33 Numerical argument out of domain
- ERANGE 34 Result too large
- EAGAIN 35 Resource temporarily unavailable
- EWOULDBLOCK 35 Operation would block
- EINPROGRESS 36 Operation now in progress
- EALREADY 37 Operation already in progress
- ENOTSOCK 38 Socket operation on non-socket
- EDESTADDRREQ 39 Destination address required
- EMSGSIZE 40 Message too long
- EPROTOTYPE 41 Protocol wrong type for socket
- ENOPROTOOPT 42 Protocol not available
- EPROTONOSUPPORT 43 Protocol not supported
- ESOCKTNOSUPPORT 44 Socket type not supported
- EOPNOTSUPP 45 Operation not supported
- EPFNOSUPPORT 46 Protocol family not supported
- EAFNOSUPPORT 47 Address family not supported by protocol family
- EADDRINUSE 48 Address already in use
- EADDRNOTAVAIL 49 Can't assign requested address
- ENETDOWN 50 Network is down
- ENETUNREACH 51 Network is unreachable
- ENETRESET 52 Network dropped connection on reset
- ECONNABORTED 53 Software caused connection abort
- ECONNRESET 54 Connection reset by peer
- ENOBUFS 55 No buffer space available
- EISCONN 56 Socket is already connected
- ENOTCONN 57 Socket is not connected
- ESHUTDOWN 58 Can't send after socket shutdown
- ETOOMANYREFS 59 Too many references: can't splice
- ETIMEDOUT 60 Operation timed out
- ECONNREFUSED 61 Connection refused
- ELOOP 62 Too many levels of symbolic links
- ENAMETOOLONG 63 File name too long
- EHOSTDOWN 64 Host is down
- EHOSTUNREACH 65 No route to host
- ENOTEMPTY 66 Directory not empty
- EPROCLIM 67 Too many processes
- EUSERS 68 Too many users
- EDQUOT 69 Disc quota exceeded
- ESTALE 70 Stale NFS file handle
- EREMOTE 71 Too many levels of remote in path
- EBADRPC 72 RPC struct is bad
- ERPCMISMATCH 73 RPC version wrong
- EPROGUNAVAIL 74 RPC prog. not avail
- EPROGMISMATCH 75 Program version wrong
- EPROCUNAVAIL 76 Bad procedure for program
- ENOLCK 77 No locks available
- ENOSYS 78 Function not implemented
- EFTYPE 79 Inappropriate file type or format
- EAUTH 80 Authentication error
- ENEEDAUTH 81 Need authenticator

Inputs

```
  none

Result
  error - the current error code

See
  ErrorString()
```

1.18 RxSocket Functions - ErrorString

```
ErrorString - returns the string associated with an error code.

Synopsis
  errorString=ErrorString(code)
  [code/N]

Function
  Returns the error string associated with 'code'.
  If 'code' is omitted, it is assumed to be the current
  error code.

Inputs
  code - the id of the error

Result
  errorString - the string associated with 'code'

See
  errno()
```

1.19 RxSocket Functions - FreeLineRead

```
FreeLineRead - free a LineRead

Synopsis
  call FreeLineRead(socketfd)
  <socketfd/N>

Function
  Free all resources allocated with InitLineRead().

Inputs
  socketfd - the socket associated with the LineRead to free

See
  InitLineRead() LineRead()
```

1.20 RxSocket Functions - GetHost

```
GetHost - returns host information

Synopsis
  res=GetHost(host,name)
  <host/V>,<name>

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

  HostErrorno() can be used to get the host-lookup
  error code on failure.

Inputs
  host - an ARexx stem name
  name - a host name or a dotted address to find information about

Result
  res - an ARexx boolean

See
  GetHostByAddr() GetHostByName()
```

1.21 RxSocket Functions - GetHostByAddr

```
GetHostByAddr - returns host information

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

Function
  Fills 'host' with a hostent data, host given as
  dotted address.

  HostErrorno() can be used to get the host-lookup
  error code on failure.

Inputs
  host - an ARexx stem name
  addr - a dotted address to find information about

Result
  res - an ARexx boolean

See
  GetHost() GetHostByName()
```

1.22 RxSocket Functions - GetHostByName

```
GetHostByName - returns host information
```

```
Synopsis
  res=GetHostByName(host,hostName)
  <host/V>,<hostName>

Function
  Fills 'host' with a hostent, host given as name.

  HostErrorno() can be used to get the host-lookup
  error code on failure.

Inputs
  host     - an ARexx stem name
  hostName - a host to find information about

Result
  res - an ARexx boolean

See
  GetHost() GetHostByAddr()
```

1.23 RxSocket Functions - GetHostID

```
GetHostID - gets the address of this host

Synopsis
  id=GetHostID()
  -

Function
  Returns the unique identifier of current host.

Inputs
  none

Result
  id - address of this host

Note
  This function is deprecated:
      o to get the ip of an interface use IoctlSocket()
      o to get the ip of the interface a named socket is
        using use GetSocketName()
```

1.24 RxSocket Functions - GetHostName

```
GetHostName - gets this host name

Synopsis
  res=GetHostName(name)
  <name/S>

Function
```

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

Inputs
  name - an ARexx var name, the function will set
         with this host name

Result
  res - an ARexx boolean.

Note
  This function is deprecated:
      o to get the ip of an interface use IoctlSocket()
      o to get the ip of the interface a named socket is
        using use GetSocketName()
```

1.25 RxSocket Functions - GetPeerName

```
GetPeerName - gets connected peer information

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

Function
  Fills 'remote' with a sockaddr_in of the peer
  connected to the socket.

  The function may fails if:

      o [EBADF]    - 'socketfd' is invalid.

      o [ENOTCONN] - The socket is not connected.

      o [ENOBUFS]  - Insufficient resources were available
                     in the system to perform the operation.

Inputs
  socketfd - the socket
  remote   - an ARexx stem name, the function will set
             as a sockaddr_in

Result
  res - an ARexx boolean

Note
  The function does not work with TermiteTCP.

See
  errno()
```

1.26 RxSocket Functions - GetProtoByName

```
GetProtoByName - returns protocol info

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

Function
  Fills 'stem' with the protoent of
  the proto given as name.

Inputs
  stem      - an ARexx stem name, the function will set
              as protoent
  protoName - the name of the protocol (lower case)

Result
  res - an ARexx boolean

See
  GetProtoByNumber()
```

1.27 RxSocket Functions - GetProtoByNumber

```
GetProtoByNumber - returns protocol info

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

Function
  Fills 'stem' with the protoent of
  the proto given as number.

Inputs
  stem    - an ARexx stem name, the function will set
            as protoent
  protoID - the id of the protocol

Result
  res - an ARexx boolean

See
  GetProtoByName()
```

1.28 RxSocket Functions - GetServByName

```
GetServByName - returns service info

Synopsis
  res=GetServByName(stem,serviceName,protoName)
  <stem/V>,<serviceName>,[protoName]
```

```
Function
  Fills 'stem' with the servent of
  the service given as name and protocol.

Inputs
  stem        - an ARexx stem name, the function will
                set as protoent
  serviceName - the name of the service
  protoName   - the name of a protocol (lower case)

Result
  res - an ARexx boolean

Note
  The function does not work with TermiteTCP.

See
  GetServByPort()
```

1.29 RxSocket Functions - GetServByPort

```
GetServByPort - returns service info

Synopsis
  res=GetServByPort(stem,portNumber,protoName)
  <stem/V>,<potNumber/N>,[protoName]

Function
  Fills 'stem' with the servent of
  the service given as port number and protocol.

Inputs
  stem       - an ARexx stem name, the function will set
               as protoent
  portNumber - the number of the port
  protoName  - the name of a protocol (lower case)

Result
  res - an ARexx boolean

Note
  The function does not work with TermiteTCP.

See
  GetServByName()
```

1.30 RxSocket Functions - GetSocketBase

```
GetSocketBase - returns global bsdsocket.library parameters

Synopsis
```

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

Function
  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().

  In 'stem' you must set the fields you want to get.
  The function will fill the fields you set with their current
  value.

  The parameters that may be read are:

      o DTABLESIZE    size of the socket descriptor table.
                      The default is 64.

      o BREAKMASK     exec signal mask which corresponds to the
                      EINTR signal (Ctrl-C), typically 2**12.

      o SIGEVENTMASK  Exec signal mask for asynchronous event
                      notification (see GetSocketEvents()).

      o SIGURGMASK    Exec signal mask for out-of-band data.

Inputs
  stem - an ARexx stem name

Result
    res - 0 for success, -1 for failure

See
  GetSocketBaseSingle() errno()
```

1.31 RxSocket Functions - GetSocketBaseSingle

```
GetSocketBaseSingle - returns one global bsdsocket.library
                      parameter

Synopsis
  res=GetSocketBaseSingle(opt,var)
  <opt>,<var/V>

Function
  Just as GetSocketBase() but get a single option.
  'var' is where the value will be written.

Inputs
  opt - the parameter to read
  var - an ARexx var name

Result
  res - 0 for success, -1 for failure
```

```
See
  GetSocketBase() errno()
```

1.32 RxSocket Functions - GetSocketEvents

```
GetSocketEvents - returns next async event.

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

Function

  Returns the next asynchronous events on sockets an remove it from
  the internal queue. The socket number for which the event occurred
  is returned.

  Application are notified of async event through the event signal
  with code SIGEVENTMASK. This may be specified with a call to
  SetSocketBase().

  Several different events may occur on sockets. To enable reporting
  of a specific event for a socket, use SetSockOpt() with EVENTMASK.

  The function sets the fields:
      o ACCEPT
      o CLOSE
      o CONNECT
      o ERROR
      o OOB
      o READ
      o WRITE
  of 'stem' with an ARexx boolean.

  The fields represent the events that may occur:

      o ACCEPT
        A new connection is waiting to be accepted. The kernel keeps
        track of each pending connection. If more than one
        connection is pending then the kernel immediately generates
        a new ACCEPT event until all pending connections are
        accounted for.

      o CLOSE
        The connection was closed. If it was closed due to an error
        then the ERROR event is set as well.

      o CONNECT
        A pending connection has been established. This event is an
        indication that a non-blocking connect() has been completed.

      o ERROR
        An asynchronous error has occurred.
```

```
      o OOB
        New out-of-band data is available for reading.

      o READ
        New data is available for reading.

      o WRITE
        The socket is able to accept data for writing again without
        blocking (only for non-blocking sockets).

Inputs
  stem - an ARexx stem name

Result
  res - the socket interested in the event or -1 if no socket

Note
  errno() can't be used to get info on failure.
```

1.33 RxSocket Functions - GetSocketName

```
GetSocketName - returns socket local name

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

Function
  Fills 'stem' as a sockaddr_in of the
  current name of the socket.

  The function may fail if:

      o [EBADF]   - socketfd is invalid.

      o [ENOBUFS] - Insufficient resources were available in
                    the system to perform the operation.

Inputs
  socketfd - the socket to read the local name of
  stem     - an ARexx stem name

Result
  res - 0 for success, -1 for failure

See
  errno()
```

1.34 RxSocket Functions - GetSockOpt

```
GetSockOpt - returns socket parameters
```

```
Synopsis
  res=GetSockOpt(socketfd,level,opt,var)
  <socketfd/N>,<level>,<opt>,<var/V>

Function
  Fills 'var' with the value of the option 'opt' associated with
  'socketfd' at level 'level'.

  'level' is one of:
      o SOCKET
      o IP

  Valid options for SOCKET are:

      o DEBUG
        enables debugging in the underlying protocol modules.

      o REUSEADDR
        A local address supplied in a bind() can be reused.

      o REUSEPORT
        A port specified in a bind() can be reused.
        This option permits multiple instances of a program to each
        receive UDP/IP multicast or broadcast datagrams destined for
        the bound port.

      o KEEPALIVE
        Enables the periodic transmission of messages on a connected
        socket. If the connected party fails to respond to these
        messages, the connection is considered broken and processes
        using the socket receive an error indication when attempting
        to send data.

      o DONTROUTE
        Indicates that outgoing messages should bypass the standard
        routing facilities. Instead, messages are directed to the
        appropriate network interface according to the network
        portion of the destination address.

      o LINGER
        Controls the action taken when unsent messages are queued on
        socket and a close() is performed. If the socket promises
        reliable delivery of data and LINGER is set, the system will
        block the process on the close attempt until it is able to
        transmit the data or until it decides it is unable to
        deliver the information (a timeout period, termed the linger
        interval, is specified in seconds in the setsockopt() call
        when LINGER is requested). If LINGER is disabled and a close
        is issued, the system will process the close in a manner
        that allows the process to continue as quickly as possible.
        The fields
          o ONOFF
          o LINGER
        of 'var' are set.

      o BROADCAST
        Requests permission to send broadcast datagrams on the
```

socket.

- OOBINLINE
  With protocols that support out-of-band data, the option requests that out-of-band data be placed in the normal data input queue as received; it will then be accessible with recv calls without the OOB flag. Some protocols always behave as if this option is set.

- SNDBUF RCVBUF
  They are options to adjust the normal buffer sizes allocated for output and input buffers, respectively. The buffer size may be increased for high-volume connections, or may be decreased to limit the possible backlog of incoming data. The system places an absolute limit on these values.

- SNDLOWAT
  It is an option to set the minimum count for output operations. Most output operations process all of the data supplied by the call, delivering data to the protocol for transmission and blocking as necessary for flow control. Nonblocking output operations will process as much data as permitted subject to flow control without blocking, but will process no data if flow control does not allow the smaller of the low water mark value or the entire request to be processed. A WaitSelect() operation testing the ability to write to a socket will return true only if the low water mark amount could be processed. The default value for SNDLOWAT is set to a convenient size for network efficiency, often 1024.

- RCVLOWAT
  It is an option to set the minimum count for input operations. In general, receive calls will block until any (non-zero) amount of data is received, then return with the smaller of the amount available or the amount requested. The default value for RCVLOWAT is 1.
  If RCVLOWAT is set to a larger value, blocking receive calls normally wait until they have received the smaller of the low water mark value or the requested amount.
  Receive calls may still return less than the low water mark if an error occurs, a signal is caught, or the type of data next in the receive queue is different than that returned.

- SNDTIMEO
  It is an option to set a timeout value for output operations. It accepts a struct timeval parameter with the number of seconds and microseconds used to limit waits for output operations to complete. If a send operation has blocked for this much time, it returns with a partial count or with the error [EWOULDBLOCK] if no data were sent.
  This timer is restarted each time additional data are delivered to the protocol, implying that the limit applies to output portions ranging in size from the low water mark to the high water mark for output.

- RCVTIMEO

```
        It is an option to set a timeout value for input operations.
        It accepts a struct timeval parameter with the number of
        seconds and microseconds used to limit waits for input
        operations to complete. This timer is restarted each time
        additional data are received by the protocol, and thus the
        limit is in effect an inactivity timer. If a receive
        operation has been blocked for this much time without
        receiving additional data, it returns with a short count or
        with the error [EWOULDBLOCK] if no data were received.

      o TYPE
        Returns the type of the socket, such as STREAM; it is useful
        for servers that inherit sockets on startup.

      o ERROR
        Returns any pending error on the socket and clears the error
        status. It may be used to check for asynchronous errors on
        connected datagram sockets or for other asynchronous errors.

  Valid options for IP are:

      o HDRICL
        the packets contain the IP header.

      o IPOPTIONS
        sets if the IP header contains IP options.

      o TTL
        time to live

      o TOS
        type of service

  The function may fail if:
      o [EBADF]       - 'socketfd' is invalid.
      o [ENOPROTOOPT] - 'opt' is unknown at 'level'.

Inputs
  socketfd - the socket to read the parameter of
  level    - the level of 'opt'
  opt      - the option to read
  stem     - an ARexx var name, where to write the value of 'opt'

Result
  res - 0 for success, -1 for failure

See
  errno()
```

1.35 RxSocket Functions - help

```
help - returns RxSocket functions help strings

Synopsis
  help=help(funName)
```

```
    <funName>

  Function
    Returns the arguments mask string of rxsocket.library
    function 'funName'.

  Inputs
    funName - a RxSocket function name

  Result
    help - the help string associated with 'funName'
```

1.36 RxSocket Functions - HostErrorno

```
  HostErrorno - returns current host-lookup error.

  Synopsis
    error=HostErrorno()
    -

  Function
    Returns current host-lookup error, one of:

        o HOST_NOT_FOUND 1
          No such host is known.

        o TRY_AGAIN      2
          This is usually a temporary error and means that the local
          server did not receive a response from an authoritative
          server. A retry at some later time may succeed.

        o NO_RECOVERY    3
          Some unexpected server failure was encountered.
          This is a non-recoverable error.

        o NO_DATA        4
          The requested name is valid but does not have an IP address;
          this is not a temporary error. This means that the name is
          known to the name server but there is no address associated
          with this name. Another type of request to the name server
          using this domain name will result in an answer.

  Inputs
    none

  Result
    error - the current host-lookup error

  See
    HostErrorString()
```

1.37 RxSocket Functions - HostErrorString

```
HostErrorString - returns the string associated with an host-lookup
                  error code.

Synopsis
  errorString=HostErrorString(code)
  [code/N]

Function
  Returns string associated with host-lookup error 'code'.

  If 'code' is omitted, it is assumed to be the current host-lookup
  error code.

Inputs
  code - a host-lookup error code

Result
  errorString - the string associated with 'code'

See
  HostErrorno()
```

1.38 RxSocket Functions - InetAddr

```
InetAddr - converts dotted to integer

Synopsis
  inetAddr=InetAddr(addr)
  <addr/N>

Function
  Converts 'addr' from dotted to integer.

  The result is a STRING of decimal digits and NOT an ARexx number.

Inputs
  addr - the dotted address to convert

Result
  inetAddr - a string of decimal digits or -1 for failure
             (it may fail iff 'addr' is bad)

See
  resolve()
```

1.39 RxSocket Functions - InetCksum

```
InetCksum - computes an internet checksum

Synopsis
  cksum=InetCksum(data,len)
```

```
  <data>,[len/N]

Function
  Computes an Internet checksum on 'data' for 'len' bytes.
  If 'len' is omitted, the checksum is computed on all 'data'.

  The checksum 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.

Inputs
  data - the data to compute the checksum on
  len  - len of 'data'

Result
  cksum - the checksum
```

1.40 RxSocket Functions - InetNTOA

```
InetNTOA - converts from integer to dotted

Synopsis
  addrString=InetNTOA(addr)
  <addr>

Function
  Converts 'addr', a string of decimal digits, to dotted.

Inputs
  addr - a string of decimal digits

Result
  addrString - a dotted address
```

1.41 RxSocket Functions - InitLineRead

```
InitLineRead - init a LineRead

Synopsis
  res = InitLineRead(socketfd,flag)
  <socketfd/N>,[flag]

Function
  Init a LineRead.

  'flag' is one of:
      o NOTREQ
        Newline terminated strings are returned unless
        there is no newlines left in currently buffered
        input. In this case remaining buffer is returned.
```

```
      o REQLF
        If there is no newlines left in currently buffered
        input the remaining input data is copied at the
        start of buffer. Caller is informed that next
        call will fill the buffer (and it may block).
        Lines are always returned with newline at the end
        unless the string is longer than whole buffer.

      o REQNUL
        Like REQLF, but remaining newline is removed.
        Note here that length is one longer that actual
        string length since line that has only one
        newline at the end would return length as 0
        which indicates string incomplete condition.

Inputs
 socketfd - the socket associated with the LineRead to create
 flag - control the line read

Result
  res - 0 if the socket already had a LineRead, 1 otherwise

See
  InitLineRead() LineRead()
```

1.42 RxSocket Functions - IoctlSocket

```
IoctlSocket - manipulates socket parameters or gets interface
              attributes

The function has 3 different form:

  1. IoctlSocket - sets socket attributes.

     Synopsis
       res = IoctlSocket(socketfd,attr,value)
       <socketfd/N>,<attrs>,<value>

     Function
       Sets a socket attribute.

       'attr' is one of:

          o FIOASYNC
            Setting the value to 1 enables asynchronous I/O on the
            socket. Setting the value to 0 disables asynchronous I/O
            on the socket.

          o FIONBIO
            Setting the value to 1 sets the socket to non-blocking
            I/O. Setting the value to 0 sets the socket to blocking
            I/O.

     Inputs
       socketfd - the socket to set an attribute of
```

```
      attr     - the attribute to set
      value    - the value of the attribute

    Result
      res - 0 for success, -1 for failure

    Example
      res = IoctlSocket(sock,"FIONBIO",1)

      sets the socket non blocking
```

2. IoctlSocket - reads socket attributes.

```
    Synopsis
      res = IoctlSocket(socketfd,attr,var)
      <socketfd/N>,<attr>,<var/V>

    Function
      Reads socket attributes.

      'attr' is on of:

         o SIOCATMARK
           The kernel sets the value pointed to to 1 if the read
           pointer for the socket points to a mark in the data
           stream, and to 0 if the read pointer for the socket does
           not point to a mark.

         o FIONREAD
           The kernel sets the value pointed to to the number of
           readable characters on the socket.

    Inputs
      socketfd - the socket to read an attribute of
      attr     - the attribute to read
      var      - an ARexx var name where to write the value
                 of the attribute

    Result
      res - 0 for success, -1 for failure

    Example
      res = IoctlSocket(s,"FIONREAD","A")

      gets the bytes ready to be read and write it in A
```

3. IoctlSocket - Reads interface attributes.

```
    Synopsis
      res = IoctlSocket(socketfd,attr,name,var)
      <socketfd/N>,<attr>,<if>,<var/V>

    Function
      Reads an interface attribute.

      attr is one of:
```

```
          o SIOCGIFADDR
          o SIOCGIFDSTADDR
          o SIOCGIFBRDADDR
          o SIOCGIFNETMASK
          o SIOCGIFFLAGS
          o SIOCGIFMETRIC
          o SIOCGIFMTU
          o SIOCGIFPHYS

      Inputs
        socketfd - an UDP socket
        attr     - the attribute to read
        if       - an interface name
        var      - an ARexx var name where to write the value
                   of the attribute

      Result
        res - 0 for success, -1 for failure

      Example
        res = IoctlSocket(s,"SIOCGIFADDR","mi0","A")

        gets the IFAddr of mi0 (if it exists) and write it in A
        as a dotted addr.

    The function may fail if:

       o [EBADF]  - 'socketfd' is invalid.

       o [EINVAL] - Request is not valid.

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

  See
    errno()
```

1.43 RxSocket Functions - IsDotAddr

```
  IsDotAddr - checks a dotted address

  Synopsis
    res=IsDotAddr(addr)
    <addr>

  Function
    Checks if 'addr' is a "good dotted internet address".

  Inputs
    addr - the dotted address to check

  Result
    res - an ARexx boolean
```

1.44 RxSocket Functions - IsLibON

```
IsLibON - returns the running internet stack

Synopsis
  res=IsLibON(name)
  [name]

Function
  Tests on what stack RxSocket is working on
  or if a library is present.

  'name' is a string made of one or more of the following words,
  separated by space(s):

      o MIAMI       running on Miami
      o AMITCP      running on AmiTCP (~"MIAMI TTCP")
      o TTCP        running on TermiteTCP
      o USERGROUP   a usergroup.library is present
      o GENESIS     Genesis is installed

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

Inputs
  name - the name of the stack to check, see above

Result
  res - an ARexx boolean or the stack running

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

1.45 RxSocket Functions - IsOnSocks

```
IsOnSocks - checks if bsdsocket.library is running under a socks

Synopsis
  res=IsOnSocks(wrapper)
  [wrapper]

Function
  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 wrappers with
  no argument.

  It flushes memory and searches for the wrapper in ExecBase library
  list.

Inputs
  wrapper - the name of the socks wrapper

Result
```

```
  res - an ARexx boolean.
```

1.46 RxSocket Functions - IsUp

```
IsUp - checks if an interface is up

Synopsis
  res=IsUp(if)
  <if>

Function
  Checks if an interface is up.

Inputs
  if - an interface name

Result
  res - an integer number:
          o -1 the specified interface doesn't exist
          o  0 the interface is down
          o  1 the interface is up

See
  IoctlSocket()
```

1.47 RxSocket Functions - IsSocket

```
IsSocket - checks a socketfd

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

Function
  Checks if 'socketfd' is a valid socket descriptor.

Inputs
  socketfd - the socketfd to check

Result
  res - an ARexx boolean.

See
  LastSocket() Passing sockets
```

1.48 RxSocket Functions - LastSocket

```
LastSocket - returns the last socketfd

Synopsis
```

```
  socketfd=LastSocket()
  --

Function
  Returns the last socketfd active in the macro, or -1 if none.

  This function is useful 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.

Inputs
  none

Result
  socketfd - -1 no socket active, >=0 the last socketfd

See
  IsSocket() Passing sockets
```

1.49 RxSocket Functions - listen

```
listen - instructs RxSocket to accept connections

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

Function
  To accept connections, a socket is first created with socket(),
  a willingness to accept incoming connections and a queue limit
  for incoming connections are specified with listen(), and then
  the connections are accepted with accept().
  The listen() call applies only to sockets of type STREAM or
  SEQPACKET.

  'backlog' is the max number of connections accepted.

  The function may fail if:

      o [EBADF]      - 'socketfd' is invalid.

      o [EOPNOTSUPP] - The socket is not of a type that supports
                       the operation listen().

Inputs
  socketfd - the socket
  backlog  - max number of connections

Result
    res - 0 for success, -1 for failure

See
  accept() bind() errno()
```

1.50 RxSocket Functions - LineRead

```
LineRead - reads a line from a LineRead

Synopsis
  res=LineRead(socketfd,stem)
  <socketfd/N>,<stem/V>

Function
  Reads a line from a LineRead. Up to RxSocket 14.5, it
  was very difficult to read a line from a connected socket.
  Now it is simple with the LineRead mechanism. What you have
  to do is to associate a LineRead to a socket via InitLineRead().
  It will create an internal buffer where to store partial results.
  The size of the buffer is 1024, so just up to 1024 max long line
  can be read. To signal a buffer overflow, the var RC is set to an
  ARexx boolean. After you are done, the line read maybe be freed
  via FreeLineRead().

  If the LineRead was not, at this function first call
  is is created as REQLF,

  The best way to use a LineRead is:

      call InitLineRead(s)

      sel.read.0=s
      do forever

          res = WaitSelect("sel")
          if res<0 then exit

          /* no read event on s */
          if ~sel.0.read then iterate

          res = LineRead(s,"Buf")
          /* res=0 just after select() means EOF */
          if res=0 then exit
          do while res>0
              call writech("STDOUT",buf)
              res = LineRead(s,"Buf")
          end
          if res<0 then exit /* error on recv */

      end

Inputs
  socketfd - the socket to associate to a LineRead
  var/V - where to write the buffer

Result
  res - an integer:
        <0 means error on recv()
         0 means empty buffer, WaitSelect() again
        >0 means a line is present
```

```
See
  InitLineRead() LineRead()
```

1.51 RxSocket Functions - NextRxsReleased

```
NextRxsReleased - returns next released socket key

Synopsis
  key=NextRxsReleased()
  -

Function

  Supplies the next socket released in a macro called via RxsCall().

  The functions returns a key to use with ObtainSocket().
  If there are no (more) sockets to get, key is null().

Inputs
  none

Result
  key - a key to use with ObtainSocket() or null()

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

See
  RxsCall() ObtainSocket() Passing sockets
```

1.52 RxSocket Functions - ObtainSocket

```
ObtainSocket - obtains a released socket

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

Function
  The function is needed when you want to pass a socket from a macro
  to another. It obtains a previously released socket.
  'key' is the key returned by ReleaseSocket(), ReleaseCopyOfSocket()
  or NextRxsReleased().

  The way used to handle a safe sockets release-obtain is:

      o when a socket is released it is still linked in a
        list that belongs to the macro where it was created.
```

```
      o if a released socket is not obtained it is freed at the
        exit of the macro where it was created;

      o if a release socket was obtained with ObtainSocket()
        it belongs to the environment of the macro where it
        was obtained;

      o if ObtainSocket(), fails for any reasons, the socket is
        still in the macro where it was created;

      o when a socket is released, it can't be used before it
        is obtained.

      o key is the result of ReleaseSocket(), ReleaseCopyOfSocket()
        or NextRxsReleased () and it consists of a packed char of
        length 8 on success or 4 on failure.

  Key can be tested with a comparison to null() as we usually do
  with messages from an ARexx port. Keys passed to ObtainSocket()
  are checked by the function to test their coherence, anyway,
  don't play with them.

  Usually 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 and proto the socket
  is obtained only if it matches them.

Inputs
  key      - the key returned by ReleaseCopyOfSocket(),
             ReleaseSocket()or NextRxsReleased()
  family   - the family of the socket
  type     - the type of the socket
  protocol - the protocol of the socket

Result
  res - -1 for failure or the socketfd of the obtained socket.

See
  Passing sockets
```

1.53 RxSocket Functions - OpenConnection

```
OpenConnection - opens a connection or binds a socket

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

Function
```

```
The function has different forms:

Terms:
    o 'proto' can be the string "tcp" or "udp"
    o 'port1' and 'port2' are service names or port numbers;
      if and only if they are service names, they are
      resolved with getserbyname()

1. Synopsis
     res=OpenConenction(proto,port1)
     <proto>,<port>

   Function
     Creates a socket and binds it:

        o resolves 'port1' if it is a service name
        o creates a socket
        o binds the socket to 'port1'

   Inputs
     proto - the protocol to use: "tcp" or "udp"
     port1 - a service name or a port number

2. Synopsis
     res=OpenConnection(proto,port1,hostName)
     <proto>,<port1>,<hostName>

   Function
     Creates a socket and connect it:

        o resolves 'port1' if it is a service name
        o resolves 'hostName'
        o creates a socket
        o connects the socket to <hostName:port1>

   Inputs
     proto    - the protocol to use: "tcp" or "udp"
     port1    - a service name or a port number
     hostName - the  host to connect the socket to

3. Synopsis
     res=OpenConnection(proto,port1,hostName,port2)
     <proto>,<port1>,<hostName>,<port2>

   Function
     Creates a socket, binds and connects it:

        o resolves 'port1' if it is a service name
        o resolves 'port2' if it is a service name
        o resolves 'hostName'
        o creates a socket
        o binds the socket to 'port1'
        o connects the socket to <hostName:port2>

   Inputs
     proto    - the protocol to use: "tcp" or "udp"
     port1    - a service name or a port number to bind the
```

```
                  socket to
        port2     - a service name or a port number to connect
                    the socket to
        hostName - the  host to connect the socket to

    If it is supplied and if the socket is connected, 'stem' is filled
    as socketaddr_in.

  Note
    Consider the differences between connecting a socket of type TCP or UDP.

  Returns:
    res - an integer:
            o -5   socket can't be bound, e.g. port already bound
            o -4   'port2' not resolved
            o -3   'port1' not resolved
            o -2   host-lookup failure
            o -1   bsdsocket.library error
            o >=0  socketfd

  See
    errno() HostErrorno()
```

1.54 RxSocket Functions - OpenRxsCon

```
  OpenRxsCon - open the RxSOcket console

  Synopsis
    res=OpenRxsCon()
    -

  Function
    Tries to open the global RxSocket console.

    The global RxSocket console ("console") is a console to be used
    for debugging porpouses.

    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 flushed
    until it is closed via CloseRxsCon() .

  Inputs
    none

  Result
    res - an ARexx boolean

  See
    CloseRxsCon() WriteRxsCon()
```

1.55 RxSocket Functions - QueryInterfaces

```
QueryInterfaces - returns interfaces parameters

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

Function
  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.

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

  The attributes are:
      o Name
      o Family    the family of the interface as integer
      o AFAddr    the address of this interface
      o PPaddr
      o BAddr
      o NMask
      o Metric
      o MTU
      o IFWire
      o Flags     the decimal value of flags, and also:
          * up
          * broadcast
          * debug
          * loopback
          * pointtopoint
          * notrailers
          * running
          * noarp
          * promisc
          * allmulti
          * oactive
          * simplex
          * link0
          * link1
          * link2
          * multicast

Inputs
  stem - an ARexx stem name

Result
  res - an integer:
          o <0 bsdsocket.library error
          o  0 no interface found
          o >0 number of the found interfaces

Example
  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"

See
  IoctlSocket() errno()
```

1.56 RxSocket Functions - recv

```
recv - receives data from a connected socket

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

Function
  Receives data from a connected socket.

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

  If 'len' is omitted it is assumed to be 256.

  If no messages are available, recv() waits for a message to
  arrive, unless the socket is nonblocking (see IoctlSocket()) in
  which case the value -1 is returned and the function Errno()
  returns [EAGAIN].  The receive calls normally return any data
  available, up to the requested amount, rather than waiting for
  receipt of the full amount requested; this behavior is affected by
  the socket-level options RCVLOWAT and RCVTIMEO described in
  GetSockOpt()

  Waitselect() may be used to determine when more data
  arrive.

  'flags' is one or more of:

      o OOB
        requests receipt of out-of-band data that would not be
        received in the normal data stream. Some protocols place
        expedited data at the head of the normal data queue, and
        thus this flag cannot be used with such protocols.

      o PEEK
        causes the receive operation to return data from the
        beginning of the receive queue without removing that data
        from the queue. Thus, a subsequent receive call will return
        the same data.

      o WAITALL
        requests that the operation block until the full request is
```

```
          satisfied. However, the call may still return less data than
          requested if a signal is caught, an error or disconnect
          occurs, or the next data to be received is of a different
          type than that returned.

    The function may fail if:

        o [EBADF]    - socketfd is invalid.

        o [ENOTCONN] - The socket is not connected.

        o [EAGAIN]   - The socket is marked non-blocking, and the
                       receive operation would block, or a receive
                       timeout had been set, and the timeout expired
                       before data were received.

        o [EINTR]    - The receive was interrupted by delivery of a
                       signal before any data were available.

  Inputs
    socketfd - the socket to receive data from
    buff     - an ARexx var name where to write the received data
    len      - number of bytes to receive
    flags    - see above

  Result
    res - an integer:
            o <0 error
            o  0 eof
            o >0 number of bytes received

  See
    RecvFrom() RecvFromUntil() RecvLine() errno()
```

1.57 RxSocket Functions - RecvFrom

```
  RecvFrom - receives data from a socket

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

  Function
    Receives data from a socket.

    If 'remote' is supplied and the socket is not connection oriented,
    it will be set as a sockaddr_in of the source address.

    See recv() for a full description.

  Inputs
    socketfd - the socket to receive data from
    buff     - an ARexx var name where to write the received data
    len      - number of bytes to receive
    flags    - flags
```

```
  remote   - an ARexx stem name

Result
  res - an integer:
          o <0 error
          o  0 eof
          o >0 number of bytes received

See
  recv() RecvFromUntil() RecvLine() errno()
```

1.58 RxSocket Functions - RecvFromUntil

```
RecvFromUntil - receives data from a socket until a string is
                reached

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

Function
  Receives data from a socket until 'stopData' occurs.

  The function waits for a string to occur and then returns the data
  received till that string (but without that string). Next receive
  call will return data AFTER the stop string.

  If 'remote' is supplied and the socket is not connection oriented,
  it will be set as a sockaddr_in of the source address.

  See recv() for a full description.

Inputs
  socketfd - the socket to receive data from
  buff     - an ARexx var name where to write the received data
  len      - number of bytes to receive
  stopData - string to wait
  flags    - see above
  remote   - an ARexx stem name

Result
  res - an integer:
          o <0 error
          o 0  eof
          o >0 number of bytes received

See
  recv() RecvFrom() RecvLine() errno()


Note
  This function is deprecated. To read a line, please use LineRead().
```

1.59 RxSocket Functions - RecvLine

```
RecvLine - receives a line from a socket

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

Function
  Receives a line from a socket.
  If remote is supplied and the socket is not connection oriented,
  it will be set as a sockaddr_in of the source address.

  See recv() for a full description.

Inputs
  socketfd - the socket to receive data from
  buff     - an ARexx var name where to write the received data
  len      - number of bytes to receive
  flags    - flags
  remote   - an ARexx stem name

Result
  res - an integer:
          o <0 error
          o  0 eof
          o >0 number of bytes received

Note
  This is really a 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().
  If you are using this function with a STREAM socket don't supply
  'remote'.

See
  recv() RecvFrom() RecvFromUntil() errno()


Note
  This function is deprecated. Please use LineRead().
```

1.60 RxSocket Functions - ReleaseCopyOfSocket

```
ReleaseCopyOfSocket - releases a copy of a socket

Synopsis
  key=ReleaseCopyOfSocket(socketfd)
  <socketfd/N>

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

```
Inputs
  socketfd - the socket to releases

Result
  key - a key to use with ObtainSocket() or null()

See
  ObtainSocket() ReleaseSocket Passing sockets
```

1.61 RxSocket Functions - ReleaseSocket

```
ReleaseSocket - releases a socket

Synopsis
  key=ReleaseSocket(socketfd)
  <socketfd/N>

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

Inputs
  socketfd - the socket to releases

Result
  key - a key to use with ObtainSocket() or null()

See
  ObtainSocket() ReleaseCopyOfSocket Passing sockets
```

1.62 RxSocket Functions - resolve

```
resolve - resolves a name

Synopsis
  addr=resolve(host)
  <host>

Function
  Converts 'host' to dotted form.

  The functions first tries inet_addr() and then GetHosByName()

Inputs
  host - the name to resolve

Result
  addr - the dotted address of host, or -1 if failure

See
  HostErrorno()
```

1.63 RxSocket Functions - RxsCall

```
RxsCall - call an ARexx macro

Synopsis
  res = RxsCall(macro,socketfd,flags,host,ext,input,output)
  <macro>,[socketfd/N],[flags],[host],[ext],[input],[output]

Function
  Starts a macro and creates a socket by releasing (a copy of)
  socketfd.

  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 or it is omitted, no socket is passed.

  That means the function may be used for general macro
  calling.

  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' is one or more of:

      o 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;

      o STRING
        child is a macro-string rather than a macro file name;

      o RESULT
        a result is expected from child; SYNC is forced;

      o 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;

      o 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.

      o NOIO
        sets noio flags in child macro.

      o NTCOPY
```

```
          releases socket without copying it. Please, use this
          only if you are sure you are runnning with Miami; Genesis
          fails to release sockets obtained via GetSocketEvent().
          This limits the use of this options only to non-async
          sockets.

        o OPENCON
          if child has no stdin/stdout, forces the global RxSocket
          console to be opened, so it will be the stdin/stdout of the
          macro

        o PROC
          the result of the function id the id of the macro
          it has sense only if child is called async, otherwise
          the process does'nt exist when the function returns
          the process id is returned as packed char

        o NOREPORT
          don't bother me with requester about child macro errors

        o ERR
          'output' is also the stderr


Inputs
  macro    - the macro to call
  socketfd - socket to pass to 'macro'
  flags    - see above
  host     - default host for the macro,
             otherwise inherit from calling macro
  ext      - default file extension for the macro,
             otherwise inherit from calling macro
  input    - stdin for the macro, otherwise it is:

                  o if SYNC
                      . STDIN logical file of calling macro
                      . stdin of calling macro
                      . RxSocket log file
                      . NIL:

                  o if not SYNC
                      . RxSocket log file
                      . NIL:
  output   - stdout for the macro, otherwise it is:

                  o if SYNC
                      . STDOUT logical file of calling macro
                      . stdout of calling macro
                      . RxSocket log file
                      . CONSOLE:

                  o if not SYNC
                      . RxSocket log file
                      . CONSOLE:


Result:
```

```
  res - an integer (but see PROC):
          o a result from child if RESULT was specified
          o 0 if child is called async


Note
  If NTCOPY is suppleid, socketfd is duplicate, so if parent
  does not need it it should immediately be closed, especially
  if it is of type STREAM.

  The function write in RC the secundary result returned from
  child. Obviously, it has sense only if child is called sync.
  Note that, if RC is 1, child couldn't be found.
```

1.64 RxSocket Functions - send

```
send - sends data to a connected socket

Synopsis
  res=send(socketfd,data,flags)
  <socketfd/N>,<data>,[flags]

Function
  Sends data to a connected socket.

  If the message is too long to pass atomically through the
  underlying protocol, the error [EMSGSIZE] is returned, and
  the message is not transmitted.

  No indication of failure to deliver is implicit in a send().
  Locally detected errors are indicated by a return value of -1.

  If no messages space is available at the socket to hold the
  message to be transmitted, then send() normally blocks, unless
  the socket has been placed in non-blocking I/O mode.
  WaitSelect() may be used to determine when it is possible to
  send more data.

  Flags is one or more of:

      o OOB
        is used to send ``out-of-band'' data on sockets that support
        this notion (e.g.  STREAM); the underlying protocol must
        also support ``out-of-band'' data.

      o DONTROUTE
        is usually used only by diagnostic or routing programs.

  The function may fail if:

      o [EBADF]    - socketfd is invalid

      o [EMSGSIZE] - The socket requires that message be sent
                     atomically, and the size of the message
                     to be sent made this impossible.
```

```
      o [EAGAIN]  - The socket is marked non-blocking and the
                    requested operation would block.

      o [ENOBUFS] - The system was unable to allocate an internal
                    buffer. The operation may succeed when buffers
                    become available.

      o [ENOBUFS]  - The output queue for a network interface was
                     full. This generally indicates that the
                     interface has stopped sending, but may
                     be caused by transient congestion.

Inputs
  socketfd - the socket
  data     - the data to send
  flags    - see above

Result
  res - an integer
          o  <0 error
          o >=0 number of bytes sent

See
  SendTo() errno()
```

1.65 RxSocket Functions - SendTo

```
SendTo - sends data to a socket

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

Function
  Sends data to a socket.

  'remote' must be set with the address of the target.

  See send() for a full description.

Inputs
  socketfd - the socket
  data     - the data to send
  flags    - see above
  remote   - the address of the target

Result
  res - an integer
          o  <0 error
          o >=0 number of bytes sent

See
  send() errno()
```

1.66 RxSocket Functions - SetRxSocketOpt

```
SetRxSocketOpt - sets RxSocket options

Synopsis
  call SetRxSocketOpt(options)
  <options>

Function
  Sets local parameters in rxsocket.library

  options is one or more of:

      o 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

      o NOHALT
        set HALT by "hi" OFF

Inputs
  options - the options to set

Result
  none
```

1.67 RxSocket Functions - SetSocketBase

```
SetSocketBase - sets bsdsocket.library options

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

Function
  Sets parameters 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:

      o DTABLESIZE
        size of the socket descriptor table.
        The default is 64.

      o BREAKMASK
        exec signal mask which corresponds to the [EINTR]
        signal (Ctrl-C), typically 2**12.
```

```
      o SIGEVENTMASK
        Exec signal mask for asynchronous event notification
        (see GetSocketEvents()).

      o SIGURGMASK
        Exec signal mask for out-of-band data.

Inputs
  stem - an ARexx stem name

Result
  res - 0 for success, -1 for failure

See
  SetSocketBaseSingle() errno()
```

1.68 RxSocket Functions - SetSocketBaseSingle

```
SetSocketBaseSingle - sets a bsdsocket.library option

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

Function
  Just as SetSocketBase() but sets only one opt.

  'opt' is the option name

  'value' is the value to set, only numeric for now.

Inputs
  opt   - the option to set
  value - the value

Result
  res - 0 for success, -1 for failure

See
  SetSocketBase() errno()
```

1.69 RxSocket Functions - SetSocketSignals

```
SetSocketSignals - sets socket signals

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

Function
    Tells bsdsocket.library which signals to use for
```

```
    SIGINT, SIGIO and SIGURG.

Inputs
  intrMask - the signal to use for Ctrl-C
  ioMask   - the signal to use for event notifications
  urgMask  - the signal to use for out-of-band data

Result
  none

Note
  Please, use SetSocketBase() instead.
```

1.70 RxSocket Functions - SetSockOpt

```
SetSockOpt - sets socket parameters

Synopsis
  res=SetSockOpt(socketfd,level,opt,value,value2)
  <socketfd/N>,<level>,<opt>,<value>,[value2/N]

Function
  Sets the option 'opt' associated with 'socketfd' at level 'level'
  to 'value'.

  'level' is one of:
      o SOCKET
      o IP
      o TCP

  Valid options for SOCKET are:

      o DEBUG/N
        enables debugging in the underlying protocol modules.

      o REUSEADDR/N
        A local address supplied in a bind() can be reused.

      o REUSEPORT/N
        A port specified in a bind() can be reused.
        This option permits multiple instances of a program to each
        receive UDP/IP multicast or broadcast datagrams destined for
        the bound port.

      o KEEPALIVE/N
        Enables the periodic transmission of messages on a connected
        socket. If the connected party fails to respond to these
        messages, the connection is considered broken and processes
        using the socket receive an error indication when attempting
        to send data.

      o DONTROUTE/N
        Indicates that outgoing messages should bypass the standard
        routing facilities. Instead, messages are directed to the
        appropriate network interface according to the network
```

portion of the destination address.

- LINGER/N
  Controls the action taken when unsent messages are queued on socket and a close() is performed. If the socket promises reliable delivery of data and LINGER is set, the system will block the process on the close attempt until it is able to transmit the data or until it decides it is unable to deliver the information (a timeout period, termed the linger interval, is specified in seconds in the setsockopt() call when LINGER is requested). If LINGER is disabled and a close is issued, the system will process the close in a manner that allows the process to continue as quickly as possible. 'value2' may be passes - default 0.

- BROADCAST/N
  Requests permission to send broadcast datagrams on the socket.

- OOBINLINE/N
  With protocols that support out-of-band data, the option requests that out-of-band data be placed in the normal data input queue as received; it will then be accessible with recv calls without the OOB flag. Some protocols always behave as if this option is set.

- SNDBUF/N RCVBUF/N
  They are options to adjust the normal buffer sizes allocated for output and input buffers, respectively. The buffer size may be increased for high-volume connections, or may be decreased to limit the possible backlog of incoming data. The system places an absolute limit on these values.

- SNDLOWAT/N
  It is an option to set the minimum count for output operations. Most output operations process all of the data supplied by the call, delivering data to the protocol for transmission and blocking as necessary for flow control. Nonblocking output operations will process as much data as permitted subject to flow control without blocking, but will process no data if flow control does not allow the smaller of the low water mark value or the entire request to be processed. A select() operation testing the ability to write to a socket will return true only if the low water mark amount could be processed. The default value for SNDLOWAT is set to a convenient size for network efficiency, often 1024.

- RCVLOWAT/N
  It is an option to set the minimum count for input operations. In general, receive calls will block until any (non-zero) amount of data is received, then return with the smaller of the amount available or the amount requested. The default value for RCVLOWAT is 1.
  If RCVLOWAT is set to a larger value, blocking receive calls normally wait until they have received the smaller of the low water mark value or the requested amount.
  Receive calls may still return less than the low water mark

if an error occurs, a signal is caught, or the type of data next in the receive queue is different than that returned.

- o SNDTIMEO/N
  It is an option to set a timeout value for output operations. It accepts a struct timeval parameter with the number of seconds and microseconds used to limit waits for output operations to complete. If a send operation has blocked for this much time, it returns with a partial count or with the error [EWOULDBLOCK] if no data were sent. This timer is restarted each time additional data are delivered to the protocol, implying that the limit applies to output portions ranging in size from the low water mark to the high water mark for output.

- o RCVTIMEO/N
  It is an option to set a timeout value for input operations. It accepts a struct timeval parameter with the number of seconds and microseconds used to limit waits for input operations to complete. This timer is restarted each time additional data are received by the protocol, and thus the limit is in effect an inactivity timer. If a receive operation has been blocked for this much time without receiving additional data, it returns with a short count or with the error [EWOULDBLOCK] if no data were received.

- o EVENTMASK/D
  defines the bitmask of asynchronous events which are supposed to trigger a notification and can later be retrieved by calling GetSocketEvents.
  value is one or more of:
    - . ACCEPT
    - . CLOSE
    - . CONNECT
    - . ERROR
    - . OOB
    - . READ
    - . WRITE

  e.g. "CONNECT ERROR".

Valid option for IP are:

- o HDRINCL/N
  The packets sent contains the ip header as well.

- o TTL/N
  Time to live.

- o TOS/N
  Type of service.

Valid options for TCP are:

- o NODELAY N
- o MAXSEG N
- o NOPUSH N
- o NOOPT N

```
Inputs
  socketfd - the socket
  level    - the level at which 'opt' is
  opt      - the option to set
  value    - the value of the option
  value2   - some option needs a secondary value

Result
  res - 0 for success, -1 for failure

See
  GetSockOpt() errno()
```

1.71 RxSocket Functions - ShutDown

```
ShutDown - closes a sockets

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

Function
  Causes all or part of a full-duplex connection on the socket to be
  shut down.

  If 'how' is
      o 0 further receives will be disallowed.
      o 1 further sends will be disallowed.
      o 2 further sends and receives will be disallowed.

Inputs
  socketfd - the socket to shut down
  how      - see above

Result
  res - 0 for success, -1 for failure

See
  CloseSocket() errno()
```

1.72 RxSocket Functions - SysLog

```
SysLog - logs a message

Synopsis
  res=SysLog(message,level,facility,closeRXS)
  <message>,[level],[facility],[closeRXS/N]

Function
  Writes a message to syslog.
```

```
  'message' is a string that can't contain %c if c is different from
      o m  %m is the string related to the current errno
      o %  %% is a %

  The function checks for other form and generates ARexx error 18 if
  it find them.

  'level' is on of:
      o EMERG
      o ALERT
      o CRIT
      o ERR
      o WARNING
      o NOTICE
      o INFO
      o DEBUG

  'facility' is on of:
      o KERN
      o USER
      o MAIL
      o DAEMON
      o AUTH
      o SYSLOG
      o LPR
      o NEWS
      o UUCP
      o CRON
      o AUTHPRIV
      o FTP

Inputs
  message  - the message to log
  level    - see above
  facility - see above
  closeRXS - if 1, the functions tries to close bsdsocket.library

Result
  none

See
  SysLogCtl()
```

1.73 RxSocket Functions - SysLogCtl

```
SysLogCtl - controls SysLog

Synopsis
  res=SysLogCtl(logpointer,logmask,facility,opts,closeRXS)
  [logpointer],[logmask],[facility],[opts],[closeRXS/N]

Function
  Controls SysLog().

  'logpointer' is a string (copied) to be a tag for the messages
```

```
  that will be logged with SysLog().

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

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

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

Inputs
  logpointer - see above
  logmask    - see above
  facility   - see above
  opts       - see above
  closeRXS   - if 1, the functions tries to close bsdsocket.library

Result
  res - an ARexx boolean

See
  SysLog()
```

1.74 RxSocket Functions - socket

```
socket - creates a socket

Synopsis
  sockfd=socket(family,type,protocol)
  <family>,<type>,<protocol>
```

```
Function
  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 that can be used in every function which needs
  a "socketfd" argument.

  The function may fail if:

      o [EPROTONOSUPPORT] - The protocol type or the specified
                            protocol is not supported within this
                            domain.

      o [EMFILE]          - The per-process descriptor table is
                            full.

      o [EACCESS]         - Permission to create a socket of the
                            specified type and/or protocol is
                            denied.

      o [ENOBUFS]         - Insufficient buffer space is available.
                            The socket cannot be created until
                            sufficient resources are freed.

Inputs
  family   - the family of the socket (INET)
  type     - the type of the socket
  protocol - the protocol of the socket

Result
  sockfd - the socket descriptor or -1 for failure

See
  errno()
```

1.75 RxSocket Functions - WaitSelect

```
WaitSelect -  synchronous socket I/O multiplexing

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

Function
  The function examines the socketfd that are supplied in 'stem' to
  see if some of them are ready for reading, are ready for writing,
  or have an exceptional condition pending.

  The descriptors to check are supplied in the fields
      o stem.READ.n
      o stem.WRITE.n
      o stem.EX.n

  The function checks the socketfd supplied iff they are different
```

```
   from -1 and until stem.READ.n stem.WRITE.n stem.EX.n exists.

   If 'secs' or micro is supplied and it is not 0, it specifies a
   maximum interval to wait for the selection to complete. If they
   are omitted or they are both 0, the function blocks indefinitely.

   If 'signals' is supplied and it is not 0, it specifies a signals
   mask to wait for.

   The functions returns the number of the ready sockets or 0 on
   timeout or if a signal of 'signals' arrived.

   The function sets:
       o stem.n.READ
       o stem.n.WRITE
       o stem.n.EX

   to an ARexx boolean, so that
       o stem.0.READ   means socket 0 is ready to be read
       o stem.0.WRITE  means socket 0 can be written
       o stem.0.EX     means socket 0 has an exceptional condition
                       pending

   The functions also sets stem.SIGNALS to the received signals.

   An example will help.

   Let's suppose we have 2 sockets, sf1 and sf2, and we want to
   control 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:
           o < 0  error
           o = 0  no events on sockets
           o > 0  number of ready sockets

         To test which 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 */

 Inputs
   stem    - an ARexx stem name, see above
   secs    - seconds to wait
   micro   - microseconds to wait
```

```
    signals - signals to wait

  Result
    res - an integer:
            o -1 an error occurred
            o  0 timeout or signal
            o >0 number of ready socket for *all* the READ,
              WRITE, EX.

  See
    errno()
```

1.76 RxSocket Functions - WriteRxsCon

```
  WriteRxsCon - writes to the RxSocket console

  Synopsis
    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.

  Inputs
    ms - the message to write

  Result
    res - an ARexx boolean

  See
    OpenRxsCon() CloseRxsCon()
```

1.77 RxSocket Passing sockets

```
  Passing sockets


    Passing sockets means:
        o exporting sockets to another macro
        o importing sockets from another macro


    The general mechanism release/obtain can be used to manage
    import/export:

        1. macro A
               o create socket s
               o release socket s via ReleaseSocket()
```

```
            o send to macro B a message containing the key
              returned by ReleaseSocket() (or call macro B
              with the key as an argument)

      2. macro B
            o wait for a message from macro A containing
              the key to pass to ObtainSocket() (or wait
              to be started from macro A with the key as
              an argument)
            o tries to obtain the socket via ObtainSocket()
            o reply the message with the result of the
              operation (or in same way tells A it obtained
              the socket, e.g. via a signal)

      3. macro A
            o wait for answer from macro B
            o test the result: if failure, re-obtain the
              socket and handles it


   A simpler mechanism to export one socket is to use RxsCall
   function:

      1. macro A
            o create socket s
            o call macro B via RxsCall(B,s)
            o close socket s if needed

      2. macro B :
            o get the socket via LastSocket() function


   To import sockets from a macro:

      1. macro A
            o call macro B via RxsCall(B,,"OBTAIN")

      2. macro B
            o create its sockets
            o release them via ReleaseSocket()

      3. macro A
            o 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 RxSocket Inetd support

```
  Inetd support


    With rxsocket.library you can easily create full functional inetd
    servers.
```

```
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 the macro can
obtain the socket from inetd.

In inetd database you must

    o use rxs (complete path) as "Service"
    o use rxs as "Name"
    o specify the name of the macro (and its arguments) as "Args"

rxs template is:

    CON=CONSOLE/S,NR=NOREPORT/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 purposes. This switch forces the global rxsocket
    console to be opened, if it is not. See OpenRxsCon().

    Two little programs are supplied to control the console from
    a shell:
        o rxsco opens the console
        o rxscc close the console.

NOREPORT
    don't show requester about macro errors

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.

Note that from version 18, rxs may be used not only from inetd.
rxs is able to pass to the macro local variables defined in the
calling macro.
```

1.79 RxSocket Thanks

```
Thanks
```

Thanks goes to:

- shido for his gift <<Hi shido! A lot of ovetti for you :-)>>;
- [X_MaN] who introduced me into the irc world and Internet in general;

1.80 RxSocket Bibliography

Bibliography

I suggest to read the following:

- Quite all rfc
- "Unix Network Programming" - W. Richard Stevens PTR Prentice Hall
- socket.library autodoc from MiamiSDK, AmiTCPSDK and TermiteTCPSDK

1.81 RxSocket Note

Note

1. When writing ARexx libraries, two big problems are:
   a. The interpreter opens/closes the library at any use.
   b. The interpreter has no method to tell the world the macro exited.

   To solve these problems we need:
   a. A safe place where to store our data (e.g. bsdoskcet.library base and our sockets).
   b. A mechanism that let use dispose all our data when the macro exits.

   The only way I found to solve both of them is:
   a. The library can't be flushed until some macro is using it.
   b. A structure called "header" is created for any macro that uses the library.
   c. A headers list is saved in the library base.
   d. Anytime a header is created, this happens:

```
...
struct Process *me = (struct Process *)FindTask(NULL);
...
header->prevCode    = me->pr_ExitCode;
header->prevData    = me->pr_ExitData;
...
me->pr_ExitCode = (APTR)exitCode;
```

```
 me->pr_ExitData = (LONG)header;
 ...

I preserve the process ExitCode and ExitData
in the header and I replace the ExitCode with
a custom routine and the ExitData with the
header itself.

The core is now the ExitCode routine:

  static void SAVEDS
  exitCode ( void )
  {
      register struct Process *me;
      register struct header  *header;

      me     = (struct Process *)FindTask(NULL);
      header = (struct header *)me->pr_ExitData;

      me->pr_ExitCode = header->prevCode;
      me->pr_ExitData = header->prevData;

      if (header->prevCode) (*((prevType *)header->prevCode))();

      freeHeader(header);

      Forbid();
      rexxLibBase->use--;
  }

This is the real ExitCode that I use:
it simply call a chain of pr_ExitCode(pr_ExitData)
so that other libraries may use the same system.
```

2. 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.

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