# Table of Contents

**About This Book** ................................................................................................................................. ix

1. Related Information ............................................................................................................................... ix
2. How to Read the Syntax Diagrams ........................................................................................................ ix
3. A Note About Program Examples in this Document ............................................................................. x
4. Getting Help......................................................................................................................................... x
   4.1. The Rexx Language Association Mailing List ............................................................................. xi
   4.2. The Open Object Rexx SourceForge Site ....................................................................................... xi
   4.3. comp.lang.rexx Newsgroup ............................................................................................................ xii

## 1. What is RxSock? ................................................................................................................................. 1

## 2. Installation and Removal ................................................................................................................... 3

## 3. Parameters and Return Values .......................................................................................................... 5
   3.1. Stem Variables ................................................................................................................................ 6

## 4. Special Variables ............................................................................................................................... 9

## 5. Function Reference ........................................................................................................................... 11
   5.1. SockLoadFuncs ................................................................................................................................. 11
   5.2. SockDropFuncs ................................................................................................................................. 12
   5.3. SockVersion .................................................................................................................................. 12
   5.4. SockAccept .................................................................................................................................. 12
   5.5. SockBind ....................................................................................................................................... 14
   5.6. SockClose ..................................................................................................................................... 15
   5.7. SockConnect .................................................................................................................................. 16
   5.8. SockGetHostByAddr ....................................................................................................................... 18
   5.9. SockGetHostByName ....................................................................................................................... 18
   5.10. SockGetHostId ............................................................................................................................... 19
   5.11. SockGetPeerName ......................................................................................................................... 19
   5.12. SockGetSockName ......................................................................................................................... 20
   5.13. SockGetSockOpt ............................................................................................................................. 21
   5.14. SockInit ....................................................................................................................................... 24
   5.15. SockIoctl ..................................................................................................................................... 24
   5.16. SockListen ................................................................................................................................... 25
   5.17. SockPSock_Errno ......................................................................................................................... 26
   5.18. SockRecv ..................................................................................................................................... 26
   5.19. SockRecvFrom ............................................................................................................................... 28
   5.20. SockSelect ................................................................................................................................... 29
   5.21. SockSend ..................................................................................................................................... 31
   5.22. SockSendTo .................................................................................................................................. 32
   5.23. SockSetSockOpt ............................................................................................................................. 34
   5.24. SockShutDown ............................................................................................................................... 36
   5.25. SockPSock_Errno ........................................................................................................................... 37
   5.26. SockSocket .................................................................................................................................. 38
   5.27. SockSoClose ................................................................................................................................ 39
List of Figures

6-1. The Socket Class .........................................................................................................................41
6-2. The InetAddress Class ...............................................................................................................46
6-3. The HostInfo Class ....................................................................................................................48
About This Book

This book describes the Open Object Rexx™ TCP/IP Sockets Function Library.

This book is intended for people who plan to develop applications using Rexx and TCP/IP sockets. Its users range from the novice, who might have experience in some programming language but no Rexx or sockets experience, to the experienced application developer, who might have had some experience with Object Rexx and sockets.

This book is a reference rather than a tutorial. It assumes you are already familiar with object-oriented programming concepts.

Descriptions include the use and syntax of the language and explain how the language processor "interprets" the language as a program is running.

1. Related Information

See also: Open Object Rexx: Reference

2. How to Read the Syntax Diagrams

Throughout this book, syntax is described using the structure defined below.

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line.
  - The >>>--- symbol indicates the beginning of a statement.
  - The ---> symbol indicates that the statement syntax is continued on the next line.
  - The >--- symbol indicates that a statement is continued from the previous line.
  - The <--- symbol indicates the end of a statement.
  - Diagrams of syntactical units other than complete statements start with the >--- symbol and end with the <--- symbol.
- Required items appear on the horizontal line (the main path).
  - >>>STATEMENT--required_item------------------------------------><
- Optional items appear below the main path.
  - >>>STATEMENT--+-required_choice1-+-----------------------------><
- If you can choose from two or more items, they appear vertically, in a stack. If you must choose one of the items, one item of the stack appears on the main path.
  - >>>STATEMENT--+-required_choice1-+-required_choice2--><
- If choosing one of the items is optional, the entire stack appears below the main path.
  - >>>STATEMENT--+-optional_item--><
3. A Note About Program Examples in this Document

The program examples in this document are rendered in a mono-spaced font that is not completely compatible for cut-and-paste functionality. Pasting text into an editor could result in some characters outside of the standard ASCII character set. Specifically, single-quote and double-quote characters are sometimes converted incorrectly when pasted into an editor.

4. Getting Help

The Open Object Rexx Project has a number of methods to obtain help for ooRexx. These methods, in no particular order of preference, are listed below.
4.1. The Rexx Language Association Mailing List

The Rexx Language Association (http://www.rexxla.org/) maintains a mailing list for its members. This mailing list is only available to RexxLA members thus you will need to join RexxLA in order to get on the list. The dues for RexxLA membership are small and are charged on a yearly basis. For details on joining RexxLA please refer to the RexxLA Home Page (http://rexxla.org/) or the RexxLA Membership Application (http://www.rexxla.org/rexxla/join.html) page.

4.2. The Open Object Rexx SourceForge Site

The Open Object Rexx Project (http://www.oorexx.org/) utilizes SourceForge (http://sourceforge.net/) to house the ooRexx Project (http://sourceforge.net/projects/oorexx) source repositories, mailing lists and other project features. Here is a list of some of the most useful facilities.

The ooRexx Forums

The ooRexx project maintains a set of forums that anyone may contribute to or monitor. They are located on the ooRexx Forums (http://sourceforge.net/forum/?group_id=119701) page. There are currently three forums available: Help, Developers and Open Discussion. In addition, you can monitor the forums via email.

The Developer Mailing List

You can subscribe to the oorexx-devel mailing list at ooRexx Mailing List Subscriptions (http://sourceforge.net/mail/?group_id=119701) page. This list is for discussing ooRexx project development activities and future interpreter enhancements. It also supports a historical archive of past messages.

The Users Mailing List

You can subscribe to the oorexx-users mailing list at ooRexx Mailing List Subscriptions (http://sourceforge.net/mail/?group_id=119701) page. This list is for discussing using ooRexx. It also supports a historical archive of past messages.

The Announcements Mailing List

You can subscribe to the oorexx-announce mailing list at ooRexx Mailing List Subscriptions (http://sourceforge.net/mail/?group_id=119701) page. This list is only used to announce significant ooRexx project events.

The Bug Mailing List

You can subscribe to the oorexx-bugs mailing list at ooRexx Mailing List Subscriptions (http://sourceforge.net/mail/?group_id=119701) page. This list is only used for monitoring changes to the ooRexx bug tracking system.

Bug Reports

You can create a bug report at ooRexx Bug Report (http://sourceforge.net/tracker/?group_id=119701&atid=684730) page. Please try to provide as much information in the bug report as possible so that the developers can determine the problem as
About This Book

quickly as possible. Sample programs that can reproduce your problem will make it easier to debug
reported problems.

Request For Enhancement

You can suggest ooRexx features at the ooRexx Feature Requests
(http://sourceforge.net/tracker/?group_id=119701&atid=684733) page.

Patch Reports

If you create an enhancement patch for ooRexx please post the patch using the ooRexx Patch Report
(http://sourceforge.net/tracker/?group_id=119701&atid=684732) page. Please provide as much
information in the patch report as possible so that the developers can evaluate the enhancement as
quickly as possible.

Please do not post bug patches here, instead you should open a bug report and attach the patch to it.

4.3. comp.lang.rexx Newsgroup

The comp.lang.rexx (news:comp.lang.rexx) newsgroup is a good place to obtain help from many
individuals within the Rexx community. You can obtain help on Open Object Rexx or on any number of
other Rexx interpreters and tools.
Chapter 1. What is RxSock?

RxSock is a Rexx function package providing access to the TCP/IP socket APIs available to the C programming environment. Most of the functions described in this reference are similar to the corresponding C functions available in the TCP/IP socket library.

It is assumed that you are familiar with the basic socket APIs and can reference those specific to the system. For more information, refer to the book *Internetworking with TCP/IP, Volume I: Principles, Protocols and Architecture* by Douglas Comer (Prentice Hall PTR).

The RxSock package requires TCP/IP support to be active on your system.
Chapter 1. What is RxSock?
Chapter 2. Installation and Removal

The RxSock package is contained in the file rxsock.dll. This file must be placed in a directory listed in your LIBPATH. To get access to the functions in the RxSock package, execute the following Rexx code:

```rexx
If RxFuncQuery("SockDropFuncs") then
do
    rc = RxFuncAdd("SockLoadFuncs","rxsock","SockLoadFuncs")
    rc = SockLoadFuncs()
end
```

To unload the DLL, call the SockDropFuncs() function and then exit all CMD.EXE shells. After exiting all command shells, the DLL is dropped by the system and can be deleted or replaced.
Chapter 2. Installation and Removal
Chapter 3. Parameters and Return Values

Unless otherwise stated, the return values are the same as for the corresponding C functions. The following standard parameter types are referred to throughout this reference:

socket

is a socket value, which is an integral number.

domain

is a domain value. Currently, only the domain AF_INET is supported.

address

is the stem of a stem variable with the following values:

address.family

must always be AF_INET.

address.port

is a port number.

address.addr

is a dotted decimal address or INADDR_ANY, where appropriate.

When this parameter is needed, set it the name of a stem variable for the function to set (or that the function will read from). For example, if you pass the string xxx.! as a parameter, the following variables are set or queried by the function:

"xxx./family"
"xxx./port"
"xxx./addr"

A null address is an address with the family field being AF_INET, the port field being 0, and the addr field being 0.0.0.0.

dotAddress

is the standard dotted decimal address. For example, the string 9.23.19.63 is a valid address.

host

is the stem of a stem variable with the following values:

host.name

is the standard name of the host.

host.alias.0

is the number of aliases for this host.
Chapter 3. Parameters and Return Values

host.alias.1
is the first alias for this host.

host.alias.n
is the nth alias for this host.

host.addrtype
must always be AF_INET.

host.addr
is a dotted decimal address (default address).

host.addr.0
is the number of addresses for this host.

host.addr.1
is the first address for this host.

host.addr.n
is the nth address for this host.

When this parameter is needed, set it the name of a stem variable for the function to set (or that the function will read from). For example, if you pass the string xxx.! as a parameter, the following variables are set or queried by the function:
"xxx.!name"
"xxx.!alias.0", "xxx.!alias.1" ... "xxx.!alias.n"
"xxx.!addrtype"
"xxx.!addr"
"xxx.!addr.0", "xxx.!addr.1" ... "xxx.!addr.n"

3.1. Stem Variables

The address and host type of a parameter are stems of a stem variable. Normally, when you pass a string like addr. as a parameter, you expect the variables addr.family, addr.port, and addr.addr to be set by the function. In the previous examples, however, the stem contained an exclamation mark. This exclamation mark helps prevent the value that follows from getting misused as a normal variable. Example:

```
port = 923
sNew = SockAccept(sOld,"addr.")
say addr.port
```

In this example, you might expect the say statement to write the port number of the accepted socket. Instead, it writes the value of the variable, namely addr.923, because the port variable is set to this value. Because exclamation marks are rarely used in variables, it is unlikely that the variable !port is used in your program.
**Note:** Do not use the characters _, 0, and 1 to prefix tail values. 0 and 1 are difficult to distinguish from O, I, and l.
Chapter 4. Special Variables

The following variables are maintained by the system: errno and h_errno.

Variable errno

The variable errno is set after each RxSock call. It can have one of the following values or any other numeric value:

- EWOULDBLOCK
- EINPROGRESS
- EALREADY
- ENOTSOCK
- EDESTADDRREQ
- EMSGSIZE
- EPROTOTYPE
- ENOPROTOOPT
- EPROTONOSUPPORT
- ESOCKTNOSUPPORT
- EOPNOTSUPP
- EPFNOSUPPORT
- EAFNOSUPPORT
- EADDRINUSE
- EADDRNOTA V AIL
- ENETDOWN
- ENETUNREACH
- ENETRESET
- ECONNABORTED
- ECONNRESET
- ENOBUFS
- EISCONN
- ENOTCONN
- ESHUTDOWN
- ETOOMANYREFS
- ETIMEDOUT
- ECONNREFUSED
- ELOOP
Chapter 4. Special Variables

- ENAMETOOLONG
- EHOSTDOWN
- EHOSTUNREACH
- ENOTEMPTY

**Note:** The value is set even if the function called does not set the variable, in which case the value has no meaning. A value of 0 indicates that no error occurred.

Variable `h_errno`

The variable `h_errno` is set after each RxSock call. It can have one of the following values or any other numeric value:

- HOST_NOT_FOUND
- TRY_AGAIN
- NO_RECOVERY
- NO_ADDRESS

**Note:** The value is set even if the function called does not set the variable, in which case the value has no meaning. A value of 0 indicates that no error occurred.
Chapter 5. Function Reference

The following sections describe how the individual functions contained in RxSock are invoked from the Rexx programming environment:

- SockLoadFuncs
- SockDropFuncs
- SockVersion
- SockAccept
- SockBind
- SockClose
- SockConnect
- SockGetHostByAddr
- SockGetHostByName
- SockGetHostId
- SockGetPeerName
- SockGetSockName
- SockGetSockOpt
- SockInit
- SockIoctl
- SockListen
- SockPSock_Errno
- SockRecv
- SockRecvFrom
- SockSelect
- SockSend
- SockSendTo
- SockSetSockOpt
- SockShutDown
- SockSock_Errno
- SockSocket
- SockSoClose
5.1. SockLoadFuncs

The SockLoadFuncs() call loads all RxSock functions.

Syntax:

```plaintext
>>>--SockLoadFuncs(--+-+-----+)----------------------------------------><
        +--parm--+
```

All parameters that you supply are only used to bypass copyright information.

5.2. SockDropFuncs

The SockDropFuncs call drops all RxSock functions.

Syntax:

```plaintext
SockDropFuncs()
```

To unload the dynamic load library (DLL), first call SockDropFuncs() and then exit all CMD.EXE shells. After exiting all command shells, the DLL is dropped by the system and can be deleted or replaced.

5.3. SockVersion

The SockVersion() call provides the version of RxSock.

Syntax:

```plaintext
>>>--SockVersion()--------------------------------------------------------><
```

Return Values:

The returned value is in the form version.subversion, for example 2.1.

Prior to Version 1.2, this function did not exist. To check if a former version of RxSock is installed, use the following code after loading the function package with SockLoadFuncs():

```plaintext
/* oldVersion is 1 if a version of RxSock < 1.2 is loaded */
oldVersion = (1 = RxFuncQuery("SockVersion"))
```

5.4. SockAccept

The SockAccept() call accepts a connection request from a remote host.

Syntax:

```plaintext
>>>--SockAccept(socket--+-------------+--)--------------------------------><
        +--, address--+
```
where:

socket

is the socket descriptor created with the SockSocket() call. It is bound to an address using the
SockBind() call and must be enabled to accept connections using the SockListen() call.

address

is a stem variable that contains the socket address of the connection client when the SockAccept() call returns. This parameter is optional.

SockAccept() is used by a server in a connection-oriented mode to accept a connection request from a client. The call accepts the first connection on its queue of pending connection requests. It creates a new socket descriptor with the same properties as socket and returns it to the caller. This new socket descriptor cannot be used to accept new connections. Only the original socket can accept more connection requests.

If the queue has no pending connection requests, SockAccept() blocks the caller unless the socket is in nonblocking mode. If no connection requests are queued and the socket is in nonblocking mode, SockAccept() returns a value of -1 and sets the return code to the value EWOULDBLOCK.

You cannot get information on requesters without calling SockAccept(). The application cannot tell the system from which requesters it will accept connections. The caller can close a connection immediately after identifying the requester.

The SockSelect() call can be used to check the socket for incoming connection requests.

**Return Values:**

A positive value indicates successful execution of the call. The value -1 indicates an error. You can get the specific error code by calling SockSock_Errno() or SockPSock_Errno(). Possible values:

ENOTSOCK

socket is not a valid socket descriptor.

EINTR

Interrupted system call.

EINVAL

SockListen() was not called for socket.

EOPNOTSUPP

socket is not connection-oriented.

EWOULDBLOCK

socket is in nonblocking mode, and there are no connection requests queued.

ECONNABORTED

The software caused a connection close.
5.5. SockBind

The SockBind() call binds a local name to the socket.

Syntax:

```c
SockBind(socket, address)
```

where:

socket

is the socket descriptor returned by a previous call to SockSocket().

address

is a stem variable containing the address that is to be bound to socket.

SockBind() binds the unique local name address to the socket with descriptor socket. After calling SockSocket(), a descriptor does not have a name. However, it belongs to a particular address family that you specified when calling SockSocket().

Because socket was created in the AF_INET domain, the fields of the stem address are as follows:

The family field must be set to AF_INET. The port field is set to the port to which the application must bind. If port is set to 0, the caller allows the system to assign an available port. The application can call SockGetSockName() to discover the port number assigned. The addr field is set to the Internet address.

On hosts with more than one network interface (called multihomed hosts), a caller can select the interface with which it is to bind.

Only UDP packets and TCP connection requests from this interface that match the bound name are routed to the application. This is important when a server offers a service to several networks. If addr is set to INADDR_ANY, the caller requests socket be bound to all network interfaces on the host. If you do not specify an address, the server can accept all UDP packets and TCP connection requests made to its port, regardless of the network interface on which the requests arrived.

Return values:

The value 0 indicates successful execution of the call. The value -1 indicates an error. You can get the specific error code by calling SockSock_Errno() or SockPSock_Errno(). Possible values:

EADDRINUSE

address is already in use. See the SO_REUSEADDR option described under SockGetSockOpt() and the SO_REUSEADDR option described under SockSetSockOpt().

EADDRNOTAVAIL

The address specified is not valid on this host. For example, the Internet address does not specify a valid network interface.
Chapter 5. Function Reference

EAFNOSUPPORT
The address family is not supported.

ENOTSOCK
socket is not a valid socket descriptor.

EINVAL
socket is already bound to an address.

ENOBUFS
No buffer space available.

*Note:* SockBind() interfaces with the C function bind().

5.6. SockClose

The SockClose() call shuts down a socket and frees resources allocated to the socket.

**Syntax**

```c
>>SockClose(socket)
```

where:

socket
is the descriptor of the socket to be closed.

If the SO_LINGER option of SockSetSockOpt() is enabled, any queued data is sent. If this option is disabled, any queued data is flushed.

**Return values:**

The value 0 indicates successful execution of the call. The value -1 indicates an error. You can get the specific error code by calling SockSock_Errno() or SockPSock_Errno(). Possible values are:

ENOTSOCK
socket is not a valid socket descriptor.

EALREADY
The socket is in nonblocking mode. A previous connection attempt has not completed.

SockClose() is exactly the same as SockSoClose().

*Note:* SockClose() interfaces with the C function soclose() or, in the Windows environments, with closesocket().
Chapter 5. Function Reference

5.7. SockConnect

The SockConnect() socket call requests a connection to a remote host.

**Syntax:**

```c
SockConnect(socket, address)
```

where:

- `socket` is the socket descriptor used to issue the connection request.
- `address` is a stem variable containing the address of the socket to which a connection is to be established.

The SockConnect() call performs the following tasks when called for a stream socket:

1. It completes the binding for a socket, if necessary.
2. It attempts to create a connection between two sockets.

This call is used by the client side of socket-based applications to establish a connection with a server. The remote server must have a passive open pending, which means it must successfully call SockBind() and SockListen(). Otherwise, SockConnect() returns the value -1 and the error value is set to ECONNREFUSED.

In the Internet communication domain, a timeout occurs if a connection to the remote host is not established within 75 seconds.

If the socket is in blocking mode, the SockConnect() call blocks the caller until the connection is established or an error is received. If the socket is in nonblocking mode, SockConnect() returns the value -1 and sets the error value to EINPROGRESS if the connection was successfully initiated. The caller can test the completion of the connection by calling:

- SockSelect(), to test for the ability to write to the socket
- SockGetsockOpt(), with option SO_ERROR, to test if the connection was established

Stream sockets can call SockConnect() only once.

Datagram or raw sockets normally transfer data without being connected to the sender or receiver. However, an application can connect to such a socket by calling SockConnect(). SockConnect() specifies and stores the destination peer address for the socket. The system then knows to which address to send data and the destination peer address does not have to be specified for each datagram sent. The address is kept until the next SockConnect() call. This permits the use of the SockRecv() and SockSend() calls, which are usually reserved for connection-oriented sockets. However, data is still not necessarily delivered, which means the normal features of sockets using connectionless data transfer are maintained. The application can therefore still use the SockSendTo() and SockRecvFrom() calls.

Datagram and raw sockets can call SockConnect() several times. The application can change their destination address by specifying a new address on the SockConnect() call. In addition, the socket can be returned to a connectionless mode by calling SockConnect() with a null destination address. The null
address is created by setting the stem variable address as follows: the family field to AF_INET, the port field to 0, and the addr field to 0.0.0.0.

The call to SockConnect returns the value -1, indicating that the connection to the null address cannot be established. Calling SockSock_Errno() returns the value EADDRNOTAVAIL.

**Return values:**

The value 0 indicates successful execution of the call. The value -1 indicates an error. You can get the specific error code by calling SockSock_Errno() or SockPSock_Errno(). Possible values are:

- **EADDRNOTAVAIL**
  - The calling host cannot reach the specified destination.

- **EAFNOSUPPORT**
  - The address family is not supported.

- **EALREADY**
  - The socket is in nonblocking mode. A previous connection attempt has not completed.

- **ENOTSOCK**
  - The socket is not a valid socket descriptor.

- **ECONNREFUSED**
  - The destination host rejected the connection request.

- **EINPROGRESS**
  - The socket is in nonblocking mode, and the connection cannot be completed immediately.
  - EINPROGRESS does not indicate an error.

- **EINTR**
  - Interrupted system call.

- **EISCONN**
  - The socket is already connected.

- **ENETUNREACH**
  - The network cannot be reached from this host.

- **ETIMEDOUT**
  - Establishing the connection timed out.

- **ENOBUFFS**
  - There is no buffer space available.

- **EOPNOTSUPP**
  - The operation is not supported on socket.
5.8. **SockGetHostByAddr**

The `SockGetHostByAddr()` call retrieves information about a specific host using its address.

**Syntax:**

```plaintext
SockGetHostByAddr(dotAddress, host) +-- domain-->
```

where:

- **dotAddress**
  - is the standard dotted decimal address of the host.
- **host**
  - is a stem variable that is to receive the information on the host.
- **domain**
  - is the domain AF_INET. This parameter is optional.

**Return values:**

The value 1 indicates successful execution of the call. The value 0 indicates an error.

**Note:** `SockGetHostByAdress()` interfaces with the C function `gethostbyaddr()`.

5.9. **SockGetHostByName**

The `SockGetHostByName()` call retrieves host information on a specific host using its name or any alias.

**Syntax:**

```plaintext
SockGetHostByName(nameAddress, host) +-->
```

where:

- **nameAddress**
  - is the name of a host, for example www.ibm.com.
Chapter 5. Function Reference

host

is the name of a stem variable to receive the information on the host.

Return values:
The value 1 indicates successful execution of the call. The value 0 indicates an error.

Note: SockGetHostByName() interfaces with the C function gethostbyname().

5.10. SockGetHostId

The SockGetHostId() call retrieves the dotAddress of the local host.

Syntax:

```text
>>>--SockGetHostId()------------------------------------------------------><
```

The return value is the dotAddress of the local host.

Note: SockGetHostId() interfaces with the C function gethostid().

5.11. SockGetPeerName

The SockGetPeerName() call gets the name of the peer connected to a socket.

Syntax:

```text
>>>--SockGetPeerName(socket, address)-------------------------------------><
```

where:

socket

is the socket descriptor.

address

is a stem variable that will contain the address of the peer connected to socket.

Return values:
The value 0 indicates successful execution of the call. The value -1 indicates an error. You can get the specific error code by calling SockSock_Errno() or SockPSock_Errno(). Possible values are:

ENOTSOCK

socket is not a valid socket descriptor.
ENOTCONN
socket is not connected.

ENOBUFFS
There is no buffer space available.

Note: SockGetPeerName() interfaces with the C function getpeername.

5.12. SockGetSockName

The SockGetSockName() call gets the local socket name.

Syntax:
>>--SockGetSockName(socket, address)-------------------------------------><<

where:

socket
is the socket descriptor.

address
is a stem variable that is to receive the address of the socket returned.

SockGetSockName() returns the address for socket in the stem variable address. If the socket is not
bound to an address, the call returns a null address.

The returned null address is a stem variable with the family field set to AF_INET, the port field set to 0,
and the addr field set to 0.0.0.0.

All sockets are explicitly assigned an address after a successful call to SockBind(). Stream sockets are
implicitly assigned an address after a successful call to SockConnect() or SockAccept() if SockBind()
was not called.

The SockGetSockName() call is often used to identify the port assigned to a socket after the socket has
been implicitly bound to a port. For example, an application can call SockConnect() without previously
calling SockBind(). In this case, the SockConnect() call completes the binding necessary by assigning a
port to the socket.

Return values:
The value 0 indicates successful execution of the call. The value -1 indicates an error. You can get the
specific error code by calling SockSock_Errno() or SockPSock_Errno(). Possible values are:

ENOTSOCK
socket is not a valid socket descriptor.
Chapter 5. Function Reference

ENOBUFS
There is no buffer space available.

Note: SockGetSockName() interfaces with the C function getsockname().

5.13. SockGetSockOpt

The SockGetSockOpt() call gets the socket options associated with a socket.

Syntax:

```c
>>>SockGetSockOpt(socket, level, optName, optVal)----------------------><
```

where:

socket
is the socket descriptor.

level
specifies which option level is queried for the specified optname. The only supported level is
SOL_SOCKET.

optname
is the name of the specified socket option. Only one option can be specified with a call.

optval
is the variable to receive the option values requested. For socket options that are Boolean the option
is enabled if optval is nonzero and disabled if optval is 0.

SockGetSockOpt() returns the value of a socket option at the socket level. It can be requested for sockets
of all domain types. Some options are supported only for specific socket types.

The following options are recognized for SOL_SOCKET:

SO_BROADCAST
returns the information whether datagram sockets are able to broadcast messages. If this option is
enabled, the application can send broadcast messages using datagram socket, if the interface
specified in the destination supports broadcasting of packets.

SO_DEBUG
returns the information whether debug information can be recorded for a socket.
Chapter 5. Function Reference

SO_DONTROUTE
returns the information whether the socket is able to bypass the routing of outgoing messages. If this option is enabled, outgoing messages are directed to the network interface specified in the network portion of the destination address. When enabled, packets can only be sent to directly connected networks.

SO_ERROR
returns any error pending at the socket and clears the error status. It can be used to check for asynchronous errors at connected datagram sockets or for asynchronous errors that are not explicitly returned by one of the socket calls.

SO_KEEPALIVE
returns the information whether stream sockets are able to send keepalive packets. TCP uses a timer called the keepalive timer. This timer monitors idle connections that might have been disconnected because of a peer crash or timeout. If this option is enabled, a keepalive packet is periodically sent to the peer.

This option is mainly used to enable servers to close connections that are no longer active as a result of clients ending connections without properly closing them.

SO_LINGER
returns the information whether stream sockets are able to linger on close if data is present. If this option is enabled and there is data still to be sent when SockSoClose() is called, the calling application is blocked during the SockSoClose() call until the data is transmitted or the connection has timed out. If this option is disabled, the SockSoClose() call returns without blocking the caller while TCP is trying to send the data. Although the data transfer is usually successful, it cannot be guaranteed because TCP tries to send the data only for a specific amount of time.

SO_OOBINLINE
returns the information whether stream sockets are able to receive out-of-band data. If this option is enabled, out-of-band data is placed in the normal data input queue as it is received. It is then made available to SockRecv() and SockRecvFrom() without the MSG_OOB flag being specified in those calls. If this option is disabled, out-of-band data is placed in the priority data input queue as it is received. It can then only be made available to SockRecv() and SockRecvFrom() by specifying the MSG_OOB flag in those calls.

SO_RCVBUF
returns the buffer size for input.

SO_RCVLOWAT
returns the receive low-water mark.

SO_RCVTIMEO
returns the timeout value for a receive operation.
SO_REUSEADDR
returns the information whether stream and datagram sockets are able to reuse local addresses. If this option is enabled, the local addresses that are already in use can then be bound. This alters the normal algorithm used in the SockBind() call. At connection time, the system checks whether the local addresses and ports differ from foreign addresses and ports. If not, the error value EADDRINUSE is returned.

SO_SNDBUF
returns the size of the send buffer.

SO_SNDLOWAT
returns the send low-water mark. This mark is ignored for nonblocking calls and not used in the Internet domain.

SO_SNDTIMEO
returns the timeout value for a send operation.

SO_TYPE
returns the socket type. The integer pointed to by optval is then set to one of the following: STREAM, DGRAM, RAW, or UNKNOWN.

SO_USELOOPBACK
bypasses hardware where possible.

All option values are integral except for SO_LINGER, which contains the following blank-delimited integers:

• The l_onoff value. It is set to 0 if the SO_LINGER option is disabled.
• The llinger value. It specifies the amount of time, in seconds, to be lingered on close. A value of 0 causes SockSoClose() to wait until disconnection completes.

Return values:
The value 0 indicates successful execution of the call. The value -1 indicates an error. You can get the specific error code by calling SockSock_Errno() or SockPSock_Errno(). Possible values are:

EADDRINUSE
The address is already in use.

ENOTSOCK
socket is not a valid socket descriptor.

ENOPROTOOPT
optname or level is not recognized.

Note: SockGetSockOpt() interfaces with the C function getsockopt().
5.14. SockInit

The SockInit() call initializes the socket data structures and checks whether the TCP/IP network is active.

Syntax:

```
>>--SockInit()-----------------------------------------------------------><
```

SockInit() can be called at the beginning of each program that uses SockSocket(). However, it is not obligatory because each RxSock function is automatically initialized. For this reason, explicit initialization is not available in all system environments.

Return values:

The value 0 indicates successful execution of the call. The value 1 indicates an error.

**Note:** SockInit() interfaces with the C function sock_init().

5.15. SockIoctl

The SockIoctl() call performs special operations on the socket.

Syntax:

```
>>--SockIoctl(socket, ioctlCmd, ioctlData)-------------------------------><
```

where:

- **socket**
  - is the socket descriptor.

- **ioctlCmd**
  - is the ioctl command to be performed.

- **ioctlData**
  - is a variable containing data associated with the particular command. Its format depends on the command requested. Valid commands are:

  - **FIONBIO**
    - sets or clears nonblocking input or output for a socket. This command is an integer. If the integer is 0, nonblocking input or output on the socket is cleared. If the integer is a number other than 0, input or output calls do not block until the call is completed.

  - **FIONREAD**
    - gets the number of immediately readable bytes for the socket. This command is an integer.
**Return values:**

The value 0 indicates successful execution of the call. The value -1 indicates an error. You can get the specific error code SockSock_Errno() or SockPSock_Errno(). Possible values are:

ENOTSOCK
   socket is not a valid socket descriptor.

EINVAL
   The request is not valid or not supported.

EOPNOTSUPP
   The operation is not supported on the socket.

**Note:** Sockioctl() interfaces with the C function ioctl() or, in the Windows environments, with ioctlsocket().

### 5.16. SockListen

The SockListen() call completes the binding necessary for a socket to accept connections and creates a connection request queue for incoming requests.

**Syntax:**

```plaintext
>>>--SockListen(socket, backlog)------------------------------------------><
```

where:

socket
   is the socket descriptor.

backlog
   controls the maximum queue length for pending connections.

SockListen() performs the following tasks:

1. It completes the binding necessary for socket, if SockBind() has not been called for the socket.
2. It creates a connection request queue with a length of backlog to queue incoming connection requests.

When the queue is full, additional connection requests are ignored.

SockListen() can only be called for connection-oriented sockets.

SockListen() is called after allocating a socket with SockSocket() and after binding a name to socket with SockBind(). It must be called before SockAccept().
SockListen() indicates when it is ready to accept client connection requests. It transforms an active socket to a passive socket. After it is called, socket cannot be used as an active socket to initiate connection requests.

If backlog is smaller than 0, SockListen() interprets the backlog to be 0. If it is greater than the maximum value defined by the network system, SockListen() interprets the backlog to be this maximum value.

Return values:
The value 0 indicates successful execution of the call. The value -1 indicates an error. You can get the specific error code SockSock_Errno() or SockPSock_Errno(). Possible values are:

ENOTSOCK
socket is not a valid socket descriptor.

EOPNOTSUPP
socket is not a socket descriptor that supports the SockListen() call.

Note: SockListen() interfaces with the C function listen().

5.17. SockPSock_Errno

The SockPSock_Errno() call writes a short error message to the standard error device. It describes the last error encountered during a call to a socket library function.

Syntax:

```plaintext
>>>SockPSock_Errno(--+----------------+--)---------------------------------->>
    +--error_string--+
```

where:

error_string

is the error string written to the standard error device describing the last error encountered. The string printed is followed by a colon, a space, and then the message. If it is omitted or empty, only the message is printed. The string is optional.

The error code is acquired by calling SockSock_Errno(). It is set when errors occur. Subsequent socket calls do not clear the error code.

Note: SockPSock_Errno() interfaces with the C function psock_errno().
5.18. SockRecv

The SockRecv() call receives data on a connected socket.

Syntax:

```rexx
>>>SockRecv(socket, var, len---------------------)<
     +---, flags---+
```

where:

socket
  is the socket descriptor.

var
  is the name of a Rexx variable to receive the data.

len
  is the maximum amount of data to be read.

flags
  is a blank-delimited list of options:

  MSG_OOB
     reads any out-of-band data on the socket.

  MSG_PEEK
     peeks at the data on the socket. The data is returned but not removed, so the subsequent receive
     operation sees the same data.

SockRecv() receives data on a socket with descriptor socket and stores it in the Rexx variable var. It
applies only to connected sockets. For information on how to use SockRecv() with datagram and raw
sockets, see Datagram or raw sockets.

SockRecv() returns the length of the incoming data. If a datagram is too long to fit the buffer, the
excessive data is discarded. No data is discarded for stream sockets. If data is not available at socket, the
SockRecv() call waits for a message and blocks the caller unless the socket is in nonblocking mode. See
SockIoctl() for a description of how to set the nonblocking mode.

Return values:

If successful, the length of the data in bytes is returned. The value 0 indicates that the connection is
closed. The value -1 indicates an error. You can get the specific error code SockSock_Errno() or
SockPSock_Errno(). Possible values are:

ENOTSOCK
  socket is not a valid socket descriptor.
Chapter 5. Function Reference

EINTR
Interrupted system call.

EINVAL
Invalid argument.

EWOULDBLOCK
socket is in nonblocking mode and no data is available, or the SO_RCVTIMEO option has been set for socket and the timeout expired before any data arrived.

Note: SockRecv() interfaces to the C function recv().

5.19. SockRecvFrom

The SockRecvFrom() call receives data on a socket.

Syntax:

>>--SockRecvFrom(socket, var, len--+-+-----------+, address)-------------><
                  +--+, flags--+

where:

socket
is the socket descriptor.

var
is the name of a Rexx variable to receive the data.

len
is the maximum amount of data to be read.

flags
is a blank delimited list of options:

MSG_OOB
reads any out-of-band data on the socket.

MSG_PEEK
peeks at the data present on the socket. The data is returned but not consumed. The subsequent receive operation thus sees the same data.
address

is a stem variable specifying the address of the sender from which the data is received, unless it is a null address.

SockRecvFrom() receives data on a socket with descriptor socket and stores it in a Rexx variable named var. It applies to any socket type, whether connected or not.

SockRecvFrom() returns the length of the incoming message or data. If a datagram is too long to fit the supplied buffer, the excessive data is discarded. No data is discarded for stream sockets. If data is not available at socket, the SockRecvFrom() call waits for a message to arrive and blocks the caller, unless the socket is in nonblocking mode. See SockIoctl() for a description of how to set the nonblocking mode.

**Return values:**

If successful, the length of the data in bytes is returned. The value -1 indicates an error. You can get the specific error code SockSockErrno() or SockPSockErrno(). Possible values are:

ENOTSOCK

socket is not a valid socket descriptor.

EINVAL

Invalid argument.

EWOULDBLOCK

socket is in nonblocking mode, no data is available, or the SO_RCVTIMEO option has been set for socket and the timeout expired before data arrived.

**Note:** SockRecvFrom() interfaces with the C function recvfrom().

### 5.20. SockSelect

The SockSelect() call monitors the activity on a socket with regard to readability, readiness for writing, and pending exceptional conditions.

**Syntax:**

```rexx
>>--SockSelect(reads, writes, excepts--+-------------+--)----------------><
       +--, timeout--+
```

where:

reads

is the number of sockets to be checked for readability.

writes

is the number of sockets to be checked for readiness for writing.
excepts

is the number of sockets to be checked for pending exceptional conditions. For Network Services
sockets, the only pending exceptional condition is out-of-band data in the receive buffer.

timeout

is the maximum number of seconds the system waits for the selection to complete. Set the timeout
parameter to 0 for a blocking operation. If the socket is ready, the return will be immediate.

Each parameter specifying a number of sockets is qualified by a stem variable which is queried and set
by this function. The stem variable has the following format: stem.0 contains the number of sockets,
stem.1 the first socket, and so on. Upon return, the stem variables are reset to the sockets that are ready. If
any of the stem variables are empty (), or no parameter is passed, no sockets for that type are checked.

The timeout value must be integral (no fractional values). Nonnumeric and negative numbers are
considered to be 0. If no timeout value is passed, an empty string () is assumed.

If the timeout value is 0, SockSelect() does not wait before returning. If the timeout value is an empty
string (), SockSelect() does not time out, but returns when a socket becomes ready. If the timeout value is
in seconds, SockSelect() waits for the specified interval before returning. It checks all indicated sockets
at the same time and returns as soon as one of them is ready.

Return values:
The number of ready sockets is returned. The value 0 indicates an expired time limit. In this case, the
stem variables are not modified. The value -1 indicates an error. You can get the specific error code
SockSock_Errno() or SockPSock_Errno(). Possible values are:

ENOTSOCK

socket is not a valid socket descriptor.

EFAULT

The address is not valid.

EINVAL

Invalid argument.

EINTR

Interrupted system call.

Examples:

r.0 = 2          /* specify 2 sockets for read in stem r. */
r.1 = 101
r.2 = 102

/* specify 1 socket for write in stem w. */
w.0 = 1
w.1 = 103

/* no sockets for exceptions in stem e. */
e.0 = 0
rc = SockSelect("r.","w.","e.")
do i = 1 to r.0 /* display sockets ready for read */
say "socket" r.i "is ready for reading."
end

That SockSelect() call can be invoked as:

rc = SockSelect("r.","w.","")

or

rc = SockSelect("r.","w.",)

The function call SockSelect(, , x) results in the program pausing for x seconds.

**Note:** SockSelect() interfaces with the C function select().

### 5.21. SockSend

The SockSend() call sends data to a connected socket.

**Syntax:**

```
>>--SockSend(socket, data--+-flags--+)------------------------------><
```

where:

- socket
  - is the socket descriptor.
- data
  - is the name of a Rexx variable containing the data to be transmitted.
- flags
  - is a blank delimited list of options:

  MSG_OOB
  - sends out-of-band data to sockets that support SOCK_STREAM communication.

  MSG_DONTROUTE
  - turns on the SO_DONTROUTE option for the duration of the send operation. This option is usually only used by diagnostic or routing programs.

SockSend() sends data to a connected socket with descriptor socket. For information on how to use SockSend() with datagram and raw sockets, see Datagram or raw sockets.
If the socket does not have enough buffer space to hold the data to be sent, the SockSend() call blocks unless the socket is placed in nonblocking mode. See SockIoctl() for a description of how to set the nonblocking mode. Use the SockSelect() call to determine when it is possible to send more data.

**Return values:**

If successful, the number of bytes of the socket with descriptor socket that is added to the send buffer is returned. Successful completion does not imply that the data has already been delivered to the receiver. The return value -1 indicates that an error was detected on the sending side of the connection. You can get the specific error code SockSock_Errno() or SockPSock_Errno(). Possible values are:

- **ENOTSOCK**
  - socket is not a valid socket descriptor.

- **EINTR**
  - Interrupted system call.

- **EINVAL**
  - Invalid argument.

- **ENOBUFS**
  - There is no buffer space available to send the message.

- **EWOULDBLOCK**
  - socket is in nonblocking mode, the data cannot be sent without blocking, or the SO_SNDTIMEO option has been set for socket and the timeout expired before any data was sent.

**Note:** SockSend() interfaces with the C function send().

### 5.22. SockSendTo

The SockSendTo() call sends data to a connected or unconnected socket.

**Syntax:**

```c
>>>SockSendTo(socket, data------------------, address)-----------------><
   +--, flags--+
```

**where:**

- **socket**
  - is the socket descriptor.

- **data**
  - is a string of data to be transmitted.
flags
is a blank delimited list of options:

MSG_OOB
sends out-of-band data to sockets that support SOCK_STREAM communication.

MSG_DONTROUTE
turns on the SO_DONTROUTE option for the duration of the send operation. This option is usually only used by diagnostic or routing programs.

address
is a stem variable containing the destination address.
SockSendTo() sends data to a connected or unconnected socket with descriptor socket. For unconnected datagram and raw sockets, it sends data to the specified destination address. For stream sockets, the destination address is ignored.
Datagram sockets are connected by calling SockConnect(). This call identifies the peer to send or receive the datagram. After a datagram socket is connected to a peer, you can still use the SockSendTo() call but you cannot include a destination address.
To change the peer address when using connected datagram sockets, issue SockConnect() with a null address. Specifying a null address removes the peer address specification. You can then issue either a SockSendTo() call and specify a different destination address or a SockConnect() call to connect to a different peer. For more information on connecting datagram sockets and specifying null addresses, see Datagram or raw sockets.

Return values:
If successful, the number of bytes sent is returned. Successful completion does not guarantee that the data is delivered to the receiver. The return value -1 indicates that an error was detected on the sending side. You can get the specific error code SockSock_Errno() or SockPSock_Errno(). Possible values are:

ENOTSOCK
socket is not a valid socket descriptor.

EMSGSIZE
The message data was too big to be sent as a single datagram.

ENOBUFS
There is no buffer space available to send the message.

EWOULDBLOCK
socket is in nonblocking mode, the data cannot be sent without blocking, or the SO_SNDTIMEO option has been set for socket and the timeout expired before any data was sent.
Chapter 5. Function Reference

ENOTCONN
The socket is not connected.

EDESTADDRREQ
Destination address required.

**Note:** SockSendTo() interfaces with the C function sendto().

5.23. **SockSetSockOpt**

The SockSetSockOpt() call sets options associated with a socket.

**Syntax:**

```c
>>>SockSetSockOpt(socket, level, optName, optVal)
```

where:

- **socket**
  - is the socket descriptor.
- **level**
  - specifies which option level is set. The only supported level is SOL_SOCKET.
- **optname**
  - is the name of a specified socket option.
- **optval**
  - is the variable containing the data needed by the set command. It is optional.

SockSetSockOpt() sets options associated with a socket with descriptor socket such as enabling debugging at the socket or protocol level, controlling timeouts, or permitting socket data broadcasting. Options can exist at the socket or the protocol level. They are always present at the highest socket level. When setting socket options, the option level and name must be specified.

For socket options that are toggles, the option is enabled if optval is nonzero and disabled if optval is 0.

The following options are recognized for SOL_SOCKET:

**SO_BROADCAST**

enables datagram sockets to broadcast messages. The application can then send broadcast messages using datagram socket, if the interface specified in the destination supports broadcasting of packets.

**SO_DEBUG**

enables debug information to be recorded for a socket.
SO_DONTROUTE

enables the socket to bypass the routing of outgoing messages. Outgoing messages are then directed to the network interface specified in the network portion of the destination address. When enabled, packets can only be sent to directly connected networks.

SO_KEEPALIVE

enables stream sockets to send keepalive packets, which keep the connection alive. TCP uses a timer called the keepalive timer. This timer monitors idle connections that might have been disconnected because of a peer crash or timeout. If this option is enabled, a keepalive packet is periodically sent to the peer.

This option is mainly used to enable servers to close connections that are no longer active as a result of clients ending connections without properly closing them.

SO_LINGER

enables stream sockets to linger on close if data is present. If this option is enabled and there is data still to be sent when SockSoClose() is called, the calling application is blocked during the SockSoClose() call until the data is transmitted or the connection has timed out. If this option is disabled, the SockSoClose() call returns without blocking the caller while TCP is trying to send the data. Although the data transfer is usually successful, it cannot be guaranteed because TCP tries to send the data only for a specific amount of time.

SO_OOBINLINE

enables stream sockets to receive out-of-band data, which is a logically separate data path using the same connection as the normal data path. If this option is enabled, out-of-band data is placed in the normal data input queue as it is received. It is then made available to SockRecv() and SockRecvFrom() without the MSG_OOB flag being specified in those calls. If this option is disabled, out-of-band data is placed in the priority data input queue as it is received. It can then only be made available to SockRecv() and SockRecvFrom() by specifying the MSG_OOB flag in those calls.

SO_RCVBUF

sets the buffer size for input. This option sets the size of the receive buffer to the value contained in the buffer pointed to by optval. In this way, the buffer size can be tailored for specific application needs, such as increasing the buffer size for high-volume connections.

SO_RCVLOWAT

sets the receive low-water mark.

SO_RCVTIMEO

sets the timeout value for a receive operation.

SO_REUSEADDR

enables stream and datagram sockets to reuse local addresses. Local addresses that are already in use can then be bound. This alters the normal algorithm used in the SockBind() call. At connection time, the system checks whether the local addresses and ports differ from foreign addresses and ports. If not, the error value EADDRINUSE is returned.
SO_SNDBUF

Sets the buffer size for output. This option sets the size of the send buffer to the value contained in
the buffer pointed to by optval. In this way, the send buffer size can be tailored for specific
application needs, such as increasing the buffer size for high-volume connections.

SO_SNDLOWAT

sets the send low-water mark. This mark is ignored for nonblocking calls and not used in the
Internet domain.

SO_SNDTIMEO

sets the timeout value for a send operation.

SO_USELOOPBACK

bypasses hardware where possible.

Except for SO_LINGER, all values are integral. SO_LINGER expects two blank delimited integers:

1. The l_onoff value. It is set to 0 if the SO_LINGER option is disabled.
2. the l linger value. The l linger field specifies the amount of time, in seconds, to be lingered on
close. A value of 0 causes SockSoClose() to wait until disconnection completes.

Return values:
The value 0 indicates successful execution of the call. The value -1 indicates an error. You can get the
specific error code SockSock_Errno() or SockPSock_Errno(). Possible values are:

EADDRINUSE

The address is already in use.

ENOTSOCK

socket is not a valid socket descriptor.

ENOPROTOOPT

opname is not recognized.

EINVAL

Invalid argument.

ENOBUFS

There is no buffer space available.

Note: SockSetSockOpt() interfaces with the C function setsockopt().
5.24. **SockShutDown**

The SockShutDown() call shuts down all, or part, of a full duplex connection. This call is optional.

**Syntax:**

```plaintext
SockShutDown(socket, howto)
```

where:

- **socket** is the socket descriptor.
- **howto** is the condition of the shutdown of socket.

Because data flows in different directions are independent of each other, SockShutDown() allows you to independently stop data flows in one direction, or all data flows, with one API call. For example, you can enable yourself to send data but disable other senders to send data to you.

The howto parameter sets the condition for shutting down the connection to socket socket. It can be set to one of the following:

- **0**
  - No more data can be received on socket.

- **1**
  - No more output is allowed on socket.

- **2**
  - No more data can be sent or received on socket.

**Return values:**

The value 0 indicates successful execution of the call. The value -1 indicates an error. You can get the specific error code SockSock_Errno() or SockPSock_Errno(). Possible values are:

- **ENOTSOCK**
  - socket is not a valid socket descriptor.

- **EINVAL**
  - howto was not set to a valid value.

**Note:** SockShutDown() interfaces with the C function shutdown().
5.25. **SockSock_Errno**

The SockSock_Errno() call returns the last error code set by a socket call. Subsequent socket API calls do not reset this error code.

**Syntax:**

```c
SockSock_Errno()```

*Note:* SockSock_Errno() interfaces with the C function sock_errno().

5.26. **SockSocket**

The SockSocket() call creates an end point for communication and returns a socket descriptor representing the end point. Each socket type provides a different communication service.

**Syntax:**

```c
SockSocket(domain, type, protocol)```

where:

- **domain**
  is the communication domain requested. It specifies the protocol family to be used. Currently, only the domain AF_INET is supported, which uses addresses in the Internet address format.

- **type**
  is the type of socket created. The following types are supported:

  **SOCK_STREAM**
  provides sequenced, two-way byte streams that are reliable and connection-oriented. It supports a mechanism for out-of-band data. Stream sockets are supported by the Internet (AF_INET) communication domain.

  **SOCK_DGRAM**
  provides datagrams, which are connectionless messages of a fixed length whose reliability is not guaranteed. Datagrams can be received out of order, lost, or delivered several times. Datagram sockets are supported by the Internet (AF_INET) communication domain.

  **SOCK_RAW**
  provides the interface to internal protocols, such as IP and ICMP. Raw sockets are supported by the Internet (AF_INET) communication domain.
protocol

is the protocol to be used with the socket. It can be IPPROTO_UDP, IPPROTO_TCP, or 0. If it is set to 0, which is the default, the system selects the default protocol number for the domain and socket type requested.

Sockets are deallocated with the SockClose() call.

**Return values:**

A non-negative socket descriptor return value indicates successful execution of the call. The return value -1 indicates an error. You can get the specific error code SockSock_Errno() or SockPSock_Errno(). Possible values are:

- **EMFILE**
  The maximum number of sockets are currently in use.

- **EPROTONOSUPPORT**
  The protocol is not supported in the specified domain or the protocol is not supported for the specified socket type.

- **EPFNOSUPPORT**
  The protocol family is not supported.

- **ESOCKTNOSUPPORT**
  The socket type is not supported.

  **Note:** SockSocket() interfaces with the C function socket().

### 5.27. SockSoClose

The SockSoClose() call shuts down a socket and frees resources allocated to the socket.

**Syntax:**

```込め```SockSoClose(socket)```込め```

where:

- **socket**
  is the socket descriptor of the socket to be closed.

This function is identical to SockClose().
Chapter 5. Function Reference
Chapter 6. Socket Class Reference

The following sections describe the socket class supplied with ooRexx. This class encapsulates the rxsock extranale functions into several classes that improve the functionality if the external function library by extending the error checking and reducing the amount of code needed in an average rxsock program.

6.1. Installation

The Socket class package is contained in the file socket.cls. This file must be placed in a directory listed in your PATH. To get access to the class and methods in the Socket class, include the following statement in your Rexx program:

::requires 'socket.cls'

6.2. The Socket Class

Figure 6-1. The Socket Class
6.2.1. getHostByAddr (class) method

```bash
>>--getHostByAddr(ipaddr)--------------------------------------><
```

This is a class method. It returns an instance of the HostInfo class.

6.2.2. getHostByName (class) method

```bash
>>--getHostByName(hostname)------------------------------------><
```

This is a class method. It returns an instance of the HostInfo class.

6.2.3. getHostId (class) method

```bash
>>--getHostId()------------------------------------------------><
```

This is a class method. It returns the dotted decimal host id of the local machine.

6.2.4. accept method

```bash
>>--accept()---------------------------------------------------><
```

This method returns a new socket class instance that is connected to a remote host that has requested a connection from a server socket.

6.2.5. bind method

```bash
>>--bind(address)----------------------------------------------><
```

This method binds a socket to a particular local ip address specified by an instance of the InetAddress class contained in the address argument.

6.2.6. close method

```bash
>>--close()----------------------------------------------------><
```

This method closes this socket instance.

6.2.7. connect method

```bash
>>--connect(address)-------------------------------------------><
```
Chapter 6. Socket Class Reference

This method connect the socket to a remote address specified by an instance of the InetAddress class contained in the address argument.

6.2.8. getOption method

```c
>>>getOption(option)------------------------------------------><
```

This method returns the value of the options specified by the option argument. The option argument must be one of the following:

- SO_BROADCAST
- SO_DEBUG
- SO_DONTROUTE
- SO_ERROR
- SO_KEEPALIVE
- SO_LINGER
- SO_OOBINLINE
- SO_RCVBUF
- SO_RCVLOWAT
- SO_RCVTIMEO
- SO_REUSEADDR
- SO_SNDBUF
- SO_SNDLOWAT
- SO_SNDTIMEO
- SO_TYPE
- SO_USELOOPBACK

6.2.9. getPeerName method

```c
>>>getPeerName()----------------------------------------------><
```

This method returns the peer name of the remote connection.

6.2.10. getSockName method

```c
>>>getSockName()----------------------------------------------><
```

This method returns an instance of the InetAddress class than is the name information of the remote machine.

6.2.11. new (class) method

```c
>>>new(---+------------------------------------------+---++)---+
   +---domain---+------------------------------------------+---+
   +---, type---+------------------------------------------+---+
   +---, protocol---+
```
This method returns a new instance of the Socket.

**domain**
- If specified, this argument must be AF_INET.

**type**
- If specified, this argument must be SOCK_STREAM, SOCK_DGRAM or SOCK_RAW. SOCK_STREAM is the default.

**protocol**
- If specified, this argument must be 0, IPPROTO_UDP or IPPROTO_TCP. 0 is the default.

### 6.2.12. ioctl method

```plaintext
>>> ioctl(cmd, data)
```

This method sends a special command to the socket. The `cmd` and the `data` are not checked for valid values.

### 6.2.13. listen method

```plaintext
>>> listen(backlog)
```

This method turns the socket into a server listening socket. The `backlog` is the number of connection requests the socket should cache.

### 6.2.14. recv method

```plaintext
>>> recv(length)
```

This method receives data on a socket connection. The `length` is the maximum number of bytes the socket should receive. This method returns the data received, which could be less than the maximum length specified.

### 6.2.15. recvFrom method

```plaintext
>>> recv(length, address)
```

This method receives data on a socket connection from the specified `address`. The `address` must be an instance of the InetAddress class. The `length` is the maximum number of bytes the socket should receive. This method returns the data received, which could be less than the maximum length specified.
6.2.16. select method

```ruby
>>> select(reads, writes, excepts, timeout)
```

This method monitors activity on a set of sockets. It returns the number of sockets ready for activity. Upon return the input argument arrays will be reset to only the sockets that are ready.

**reads**

An array of socket instances to monitor for read activity.

**writes**

An array of socket instances to monitor for write activity.

**excepts**

An array of socket instances to monitor for exception activity.

**timeout**

The timeout in seconds. This must be a whole number (no fractions allowed).

6.2.17. Send method

```ruby
>>> send(data)
```

This method sends the data on the socket. It returns the number of bytes sent, which could be less than the length of data.

6.2.18. setOption method

```ruby
>>> setOption(name, value)
```

This method sets the option given by name with the data in value. See the method getOption for the list of valid names.

6.2.19. string method

```ruby
>>> string()
```

This method returns the string representing the socket.
6.3. The InetAddress Class

6.3.1. address method

>>>--address()--------------------------------------------------><

This method returns the ip address of the original hostname.

6.3.2. address= method

>>>--address(ipaddress)-----------------------------------------><

6.3.2.1. family method)

>>>--family()--------------------------------------------------><

This method returns the ip address family of the original hostname.

6.3.2.2. family= method

>>>--family(newfamily)------------------------------------------><

This method sets the ip address family of the original hostname.
6.3.2.3. init method

```plaintext
>---init(hostname, port +---, family---)
```

This method creates a new instance of the InetAddress class.

**hostname**

The ip address or host name of the host machine.

**port**

The port number of the connection.

**family**

The address family. The only valid value is AF_INET.

6.3.2.4. makeStem method

```plaintext
>---makeStem()
```

This method returns a stem variable set to the current values of the instance. This method has limited usefulness to the programmer.

6.3.2.5. port method

```plaintext
>---port()
```

This method returns port number of the original hostname.

6.3.2.6. port= method

```plaintext
>---port(newport)
```

This method sets the port number of the original hostname.
6.3.3. The HostInfo Class

Figure 6-3. The HostInfo Class

```
+------------------+
| Object           |
+------------------+
| HostInfo         |
| addr             |
| address          |
| alias            |
| name             |
| init             |
| makeStem         |
```

6.3.3.1. addr method

```inherited from Socket```

This method returns an array of ip addresses of the host.

6.3.3.2. address method

```inherited from Socket```

This method returns the main ip address of the host.

6.3.3.3. alias method

```inherited from Socket```

This method returns an array of alias host name of the host.

6.3.3.4. name method

```inherited from Socket```

This method returns the main host name of the host.
6.3.3.5. init method

>>>--init(hostname)---------------------------------------------><

This method creates an instance of the HostInfo class and sets all the attribute methods of the instance. The *hostname* can be either a valid DNS host name or an ip address.

6.3.3.6. makeStem method

>>>--makeStem()-------------------------------------------------><

This method returns a stem variable set to the current values of the instance. This method has limited usefulness to the programmer.

6.4. Socket Class Example

```plaintext
host = '127.0.0.1'
port = 8080
srvr = .server~new(host, port)
call syssleep(1) -- just to let the server get started
call client host, port, 'This is test 1'
call client host, port, 'This is test 2'
call client host, port, 'stop'
return
::requires 'socket.cls'
::routine client
use strict arg host, port, message
-- get a new socket
s = .socket~new()
-- set the server address/port to connection information
addr = .inetaddress~new(host, port)
-- connect to the server
retc = s~connect(addr)
if retc <> 0 then do
    say 'Error' s~errno() 'connecting to server socket.'
    return
end
-- send the command
retc = s~send(message)
-- receive the command back
say s~recv(4096)
-- close the socket
s~close()
return
::class server
```

Chapter 6. Socket Class Reference
::method init
use strict arg host, port
-- get a new socket
s = .socket~new()
if s = -1 then do
    say 'Error' s~errno() 'creating server socket'
    return
end
-- set the socket to reuse the addresses assigned to it
retc = s~setoption('SO_REUSEADDR', 1)
if retc = -1 then do
    say 'Error' s~errno() 'setting socket option'
    return
end
-- bind the socket to an address/port
addr = .inetaddress~new(host, port)
retc = s~bind(addr)
if retc = -1 then do
    say 'Error' s~errno() 'binding socket'
    return
end
-- mark it as a listening socket
retc = s~listen(3)
if retc = -1 then do
    say 'Error' s~errno() 'making the socket a listening socket'
    return
end
say 'Server starting'
reply
stop = .false
do while \
    -- accept a client connection socket
    cs = s~accept()
    if cs = .nil then do
        say 'Error accepting new socket'
        iterate
    end
    -- receive the command from the client
    cmd = cs~recv(4096)
    -- echo the command back to the client
    cs~send(cmd)
    -- close the client connection socket
    cs~close()
    -- if the command was stop then stop the server
    if cmd~upper() = 'STOP' then do
        stop = .true
    end
end
-- close the socket
s~close()
return
Appendix A. Notices

Any reference to a non-open source product, program, or service is not intended to state or imply that only non-open source product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any Rexx Language Association (RexxLA) intellectual property right may be used instead. However, it is the user’s responsibility to evaluate and verify the operation of any non-open source product, program, or service.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-open source products was obtained from the suppliers of those products, their published announcements or other publicly available sources. RexxLA has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-RexxLA packages. Questions on the capabilities of non-RexxLA packages should be addressed to the suppliers of those products.

All statements regarding RexxLA's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

A.1. Trademarks

Open Object Rexx™ and ooRexx™ are trademarks of the Rexx Language Association.

The following terms are trademarks of the IBM Corporation in the United States, other countries, or both:

1-2-3
AIX
IBM
Lotus
OS/2
S/390
VisualAge

AMD is a trademark of Advance Micro Devices, Inc.

Intel, Intel Inside (logos), MMX and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in
Appendix A. Notices

the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.
Other company, product, or service names may be trademarks or service marks of others.

A.2. Source Code For This Document

The source code for this document is available under the terms of the Common Public License v1.0 which accompanies this distribution and is available in the appendix Common Public License Version 1.0. The source code itself is available at http://sourceforge.net/project/showfiles.php?group_id=119701.

The source code for this document is maintained in DocBook SGML/XML format.
Appendix B. Common Public License Version 1.0

THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS COMMON PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT’S ACCEPTANCE OF THIS AGREEMENT.

B.1. Definitions

"Contribution" means:

1. in the case of the initial Contributor, the initial code and documentation distributed under this Agreement, and
2. in the case of each subsequent Contributor:
   a. changes to the Program, and
   b. additions to the Program;

where such changes and/or additions to the Program originate from and are distributed by that particular Contributor. A Contribution 'originates' from a Contributor if it was added to the Program by such Contributor itself or anyone acting on such Contributor’s behalf. Contributions do not include additions to the Program which: (i) are separate modules of software distributed in conjunction with the Program under their own license agreement, and (ii) are not derivative works of the Program.

"Contributor" means any person or entity that distributes the Program.

"Licensed Patents " mean patent claims licensable by a Contributor which are necessarily infringed by the use or sale of its Contribution alone or when combined with the Program.

"Program" means the Contributions distributed in accordance with this Agreement.

"Recipient" means anyone who receives the Program under this Agreement, including all Contributors.

B.2. Grant of Rights

1. Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free copyright license to reproduce, prepare derivative works of, publicly display, publicly perform, distribute and sublicense the Contribution of such Contributor, if any, and such derivative works, in source code and object code form.

2. Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free patent license under Licensed Patents to make, use, sell, offer to sell, import and otherwise transfer the Contribution of such Contributor, if any, in source code and object code form. This patent license shall apply to the combination of the Contribution and the Program if, at the time the Contribution is added by the Contributor, such addition of the Contribution causes such
Appendix B. Common Public License Version 1.0

combination to be covered by the Licensed Patents. The patent license shall not apply to any other combinations which include the Contribution. No hardware per se is licensed hereunder.

3. Recipient understands that although each Contributor grants the licenses to its Contributions set forth herein, no assurances are provided by any Contributor that the Program does not infringe the patent or other intellectual property rights of any other entity. Each Contributor disclaims any liability to Recipient for claims brought by any other entity based on infringement of intellectual property rights or otherwise. As a condition to exercising the rights and licenses granted hereunder, each Recipient hereby assumes sole responsibility to secure any other intellectual property rights needed, if any. For example, if a third party patent license is required to allow Recipient to distribute the Program, it is Recipient’s responsibility to acquire that license before distributing the Program.

4. Each Contributor represents that to its knowledge it has sufficient copyright rights in its Contribution, if any, to grant the copyright license set forth in this Agreement.

B.3. Requirements

A Contributor may choose to distribute the Program in object code form under its own license agreement, provided that:

1. it complies with the terms and conditions of this Agreement; and

2. its license agreement:
   a. effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose;
   b. effectively excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits;
   c. states that any provisions which differ from this Agreement are offered by that Contributor alone and not by any other party; and
   d. states that source code for the Program is available from such Contributor, and informs licensees how to obtain it in a reasonable manner on or through a medium customarily used for software exchange.

When the Program is made available in source code form:

1. it must be made available under this Agreement; and

2. a copy of this Agreement must be included with each copy of the Program.

Contributors may not remove or alter any copyright notices contained within the Program.

Each Contributor must identify itself as the originator of its Contribution, if any, in a manner that reasonably allows subsequent Recipients to identify the originator of the Contribution.
B.4. Commercial Distribution

Commercial distributors of software may accept certain responsibilities with respect to end users, business partners and the like. While this license is intended to facilitate the commercial use of the Program, the Contributor who includes the Program in a commercial product offering should do so in a manner which does not create potential liability for other Contributors. Therefore, if a Contributor includes the Program in a commercial product offering, such Contributor ("Commercial Contributor") hereby agrees to defend and indemnify every other Contributor ("Indemnified Contributor") against any losses, damages and costs (collectively "Losses") arising from claims, lawsuits and other legal actions brought by a third party against the Indemnified Contributor to the extent caused by the acts or omissions of such Commercial Contributor in connection with its distribution of the Program in a commercial product offering. The obligations in this section do not apply to any claims or Losses relating to any actual or alleged intellectual property infringement. In order to qualify, an Indemnified Contributor must:

a) promptly notify the Commercial Contributor in writing of such claim, and b) allow the Commercial Contributor to control, and cooperate with the Commercial Contributor in, the defense and any related settlement negotiations. The Indemnified Contributor may participate in any such claim at its own expense.

For example, a Contributor might include the Program in a commercial product offering, Product X. That Contributor is then a Commercial Contributor. If that Commercial Contributor then makes performance claims, or offers warranties related to Product X, those performance claims and warranties are such Commercial Contributor’s responsibility alone. Under this section, the Commercial Contributor would have to defend claims against the other Contributors related to those performance claims and warranties, and if a court requires any other Contributor to pay any damages as a result, the Commercial Contributor must pay those damages.

B.5. No Warranty

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this Agreement, including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

B.6. Disclaimer of Liability

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE
Appendix B. Common Public License Version 1.0

PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

B.7. General

If any provision of this Agreement is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this Agreement, and without further action by the parties hereto, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

If Recipient institutes patent litigation against a Contributor with respect to a patent applicable to software (including a cross-claim or counterclaim in a lawsuit), then any patent licenses granted by that Contributor to such Recipient under this Agreement shall terminate as of the date such litigation is filed. In addition, if Recipient institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Program itself (excluding combinations of the Program with other software or hardware) infringes such Recipient’s patent(s), then such Recipient’s rights granted under Section 2(b) shall terminate as of the date such litigation is filed.

All Recipient’s rights under this Agreement shall terminate if it fails to comply with any of the material terms or conditions of this Agreement and does not cure such failure in a reasonable period of time after becoming aware of such noncompliance. If all Recipient’s rights under this Agreement terminate, Recipient agrees to cease use and distribution of the Program as soon as reasonably practicable. However, Recipient’s obligations under this Agreement and any licenses granted by Recipient relating to the Program shall continue and survive.

Everyone is permitted to copy and distribute copies of this Agreement, but in order to avoid inconsistency the Agreement is copyrighted and may only be modified in the following manner. The Agreement Steward reserves the right to publish new versions (including revisions) of this Agreement from time to time. No one other than the Agreement Steward has the right to modify this Agreement. IBM is the initial Agreement Steward. IBM may assign the responsibility to serve as the Agreement Steward to a suitable separate entity. Each new version of the Agreement will be given a distinguishing version number. The Program (including Contributions) may always be distributed subject to the version of the Agreement under which it was received. In addition, after a new version of the Agreement is published, Contributor may elect to distribute the Program (including its Contributions) under the new version. Except as expressly stated in Sections 2(a) and 2(b) above, Recipient receives no rights or licenses to the intellectual property of any Contributor under this Agreement, whether expressly, by implication, estoppel or otherwise. All rights in the Program not expressly granted under this Agreement are reserved.

This Agreement is governed by the laws of the State of New York and the intellectual property laws of the United States of America. No party to this Agreement will bring a legal action under this Agreement more than one year after the cause of action arose. Each party waives its rights to a jury trial in any resulting litigation.
Index

A
accept method
of Socket class, 42
accepting a socket connection, 12
addr method
of HostInfo class, 48
address method
of HostInfo class, 48
of InetAddress class, 46
address= method
of InetAddress class, 46
alias method
of HostInfo class, 48

B
bind method
of Socket class, 42
binding to a port, 14

C
class method
new
of Socket class, 43
class, socket, 41
close method
of Socket class, 42
closing a socket, 15, 39
Common Public License, 53
cconnect
of Socket class, 42
connect to a host, 16
connection, accepting a socket, 12
connection, listen on a socket for a, 25
CPL, 53

data on socket, receive, 27, 28
data on socket, send, 31, 32
definition of a socket, 1
description, RxSock, 1
dropping functions, 12
error messages, writing, 26
error, get last socket, 38
equivalent
of Socket class, 49
family method
of InetAddress class, 46
family= method
of InetAddress class, 46
function
SockAccept, 12
SockBind, 14
SockClose, 15
SockConnect, 16
Sock DROP Funcs, 12
Sock GetHost By Addr, 18
Sock GetHost By Id, 19
Sock GetHost By Name, 18
Sock GetPeer Name, 19
Sock GetSock Name, 20
Sock GetSock Opt, 21
Sock Init, 24
Sock I O C T L , 24
Sock Listen, 25
Sock Load Funcs, 12
Sock PSock, 26
Sock Recv, 27
Sock Recv From, 28
Sock Select, 29
Sock Send, 31
Sock Send To, 32
Sock SetSock Opt, 34
Sock Shut Down, 37
Sock Socket, 38
SockSock_Errno, 38
SockSoClose, 39
SockVersion, 12
function parameters, 5
function return values, 5
functions, dropping, 12
functions, list of, 11
functions, loading, 12
functions, version of, 12

g
get a new socket, 38
get last socket error, 38
getHostByAddr class method
  of Socket class, 42
getHostByName class method
  of Socket class, 42
getAddress class method
  of Socket class, 42
_addr method
  of HostInfo class, 48
_address method
  of InetAddress class, 46
_address= method
  of InetAddress class, 46
_alias method
  of HostInfo class, 48
bind method
  of Socket class, 42
close method
  of Socket class, 42
connect
  of Socket class, 42

c
license, Common Public, 53
license, Open Object Rexx, 53
list of functions, 11
listen
  of Socket class, 44
listen on a socket for a connection, 25
loading functions, 12
lookup host information, 18, 18
lookup local host ip address, 19
lookup local socket name, 20
lookup remote connected host name, 19

m
makeStem method
  of HostInfo class, 49
  of InetAddress class, 47
method
  accept method
    of Socket class, 42
  addr method
    of HostInfo class, 48
  address method
    of HostInfo class, 48
  address= method
    of InetAddress class, 46
  alias method
    of HostInfo class, 48
  bind method
    of Socket class, 42
  close method
    of Socket class, 42
  connect
    of Socket class, 42
  family method
    of InetAddress class, 46
  family= method
    of InetAddress class, 46
  getAddress method
    of Socket class, 42
  getHostByAddr class method
    of Socket class, 42
  getHostByName class method
    of Socket class, 42

getHostId class method
 of Socket class, 42
getOption
 of Socket class, 43
getPeerName
 of Socket class, 43
getSockName
 of Socket class, 43
init method
 of HostInfo class, 49
 of InetAddress class, 47
ioctl
 of Socket class, 44
listen
 of Socket class, 44
makeStem method
 of HostInfo class, 49
 of InetAddress class, 47
name method
 of HostInfo class, 48
port method
 of InetAddress class, 47
port= method
 of InetAddress class, 47
recv
 of Socket class, 44
recvFrom
 of Socket class, 44
select
 of Socket class, 45
send
 of Socket class, 45
setOption
 of Socket class, 45
string
 of Socket class, 45
monitor a socket, 29

name method
 of HostInfo class, 48
new
 of Socket class, 43
new socket, get a, 38
Notices, 51

ooRexx License, 53
Open Object Rexx License, 53
option of a socket, set an, 34
parameters, function, 5
port method
 of InetAddress class, 47
port, binding to a, 14
port= method
 of InetAddress class, 47
receive data on socket, 27, 28
recv
 of Socket class, 44
recvFrom
 of Socket class, 44
registering functions, 12
retrieve socket options, 21
return values, function, 5
RxSock description, 1
RxSock installation, 3
select
 of Socket class, 45
send
 of Socket class, 45
send data on socket, 31, 32
set an option of a socket, 34
setOption
 of Socket class, 45
shutdown a socket, 37
SockAccept, 12
SockBind, 14
SockClose, 15
SockConnect, 16
SockDropFuncs, 12
socket class, 41
Socket class example, 49
socket class installation, 41
socket error, get last, 38
socket options, retrieve, 21
socket special operations, 24
socket, definition of a, 1
socket, get a new, 38
socket, initialize a, 24
socket, monitor a, 29
socket, shutdown a, 37
socket, status of a, 29
SockGetHostAddr, 18
SockGetHostId, 19
SockGetHostName, 18
SockGetPeerName, 19
SockGetSockName, 20
SockGetSockOpt, 21
SockInit, 24
SockIoctl, 24
SockListen, 25
SockLoadFuncs, 12
SockPSock, 26
SockRecv, 27
SockRecvFrom, 28
SockSelect, 29
SockSend, 31
SockSendTo, 32
SockSetSockOpt, 34
SockShutDown, 37
SockSocket, 38
SockSock_Errno, 38
SockSoClose, 39
SockVersion, 12
special operations, socket, 24
special variables, 9
status of a socket, 29
stem variables, usage of, 6
string
  of Socket class, 45

T

TCP/IP, 1