ClearPath Enterprise Servers

MCP Sockets Service
Programming Guide

ClearPath MCP Release 8.0

February 2003

Printed in USA
4310 3530–006

ClearPath Enterprise Servers

MCP Sockets Service
Programming Guide

UNISYS

ClearPath MCP Release 8.0

February 2003

Printed in USA
4310 3530–006

Correspondence regarding this publication can be e-mailed to **doc@unisys.com**.

ClearPath Enterprise Servers

MCP Sockets Service

Programming Guide

ClearPath MCP Release 8.0

4310 3530–006

ClearPath Enterprise Servers

MCP Sockets Service

Programming Guide

ClearPath MCP Release 8.0

4310 3530–006

Bend here, peel upwards and apply to spine.

Contents

Figures

Figures

Tables

Tables

Section 1
Introduction

The MCP Sockets Service Programming Guide is a reference for programmers who use the MCP Sockets Service API. This guide is a reference for programs written in ALGOL, COBOL, NEWP, and C, and is intended for MCP environment programmers.

What's New

The following information is new to the MCP 8.0 release of the MCP Sockets Service API:

- In Section 4 a new function, "SockLib_Select (),"is provided that allows you to wait for input, open notification, or change in output window on a group of sockets.
- The MCP 8.0 release supports the Transport Level Security (TLS) Internet standard [RFC 2246].

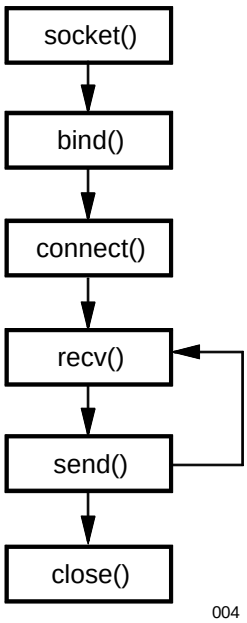
What Is a Socket?

A socket is a TCP/IP network access point, through which data can be read and written. Sockets using the TCP protocol offer a virtual two-way pipe across the network to a specific remote server, while sockets using the UDP (User Datagram Protocol) allow the user to send and receive messages with other hosts in the network. Before sending data, the socket must be "bound" to a local IP address and port number, and, in the case of TCP sockets, must be connected to the remote socket. Data reading and writing to the socket can either be synchronous, in which the operation does not return until the data has been sent or received (also known as blocking), or asynchronous, in which the user has more control over when the operation is returned (also known as non-blocking). MCP sockets are always blocking (non-blocking sockets are not currently supported); but a timeout can be specified in order to restrict the duration of the blocking.

Client software using TCP sockets usually follows the standard client algorithm:

1. Establish a socket.
2. Connect to the required server.
3. Communicate with the server through a well-defined protocol.
4. Close the socket.

Using socket API calls, this algorithm can be described with the following program flow:

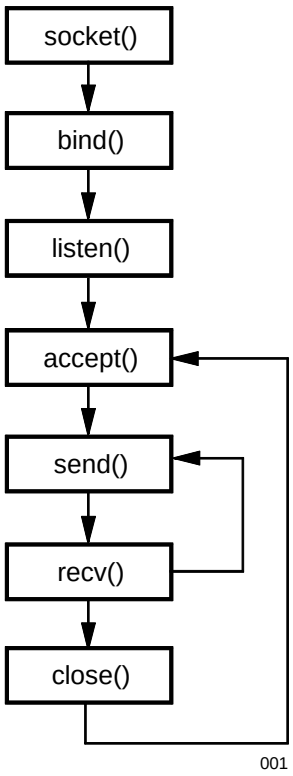


The UDP client algorithm is very similar to the TCP client algorithm.

Server software over TCP sockets operates similarly, but must already be executing and waiting for client requests on a well-known port number. Server software usually follows the following algorithm:

1. Establish a socket.
2. Bind to a well-known TCP port number for this service.
3. Tell the socket to look for incoming requests.
4. Receive client requests for service and process them (permanently).

Using the socket API, this algorithm can be shown as:



The preceding program flow is an example of an iterative TCP server. An iterative server only handles one client at a time. The other type of TCP server is a concurrent server, in which a thread, or process, is started to handle client interaction. After a new request is accepted, control of this socket is transferred to a thread for client communication, while the main thread waits for new client requests and handles any thread management issues.

For more information about the actual routines, refer to Sections 3 and 4.

Port File Differences

Sockets provide a very different type of network interface than MCP programmers familiar with the PORT FILE interface may be accustomed to. The goal of this section is to describe the major differences between PORT FILES and sockets, specifically comparing sockets with PORT FILES configured to use the TCPIPNATIVESERVICE.

- **EVENT Handling** – The socket API has no knowledge of PORT FILE events such as CHANGEEVENT, OUTPUTEVENT, and INPUTEVENT.

- **Client Handling** – In concurrent servers, a thread is used to handle communication with each client. Typically, a thread only waits for input on one socket. If a thread wants to wait for input on more than one socket, it should use select().

- **File Addressing** – Address and port numbers are assigned through a SOCKADDR structure. This structure contains a port number and IPAddress. There is one SOCKADDR structure for the local endpoint (MYNAME, MYIPADDRESS) and another for the remote (YOURNAME, YOURIPADDRESS).
  Local and remote endpoints are only set through IP addresses (sockets do not contain attributes for hostname or domain). Utility routines are supplied to help translate domain names (MYDOMAINNAME and YOURDOMAINNAME) and hostnames (MYHOSTNAME and YOURHOSTNAME) to IP addresses through SYSTEM/RESOLVER. Refer to Section 4, GetHostByName(), and other related routines for more information.

- **File Attributes** – PORT FILE attributes that specify file attributes (such as ACTUALMAXRECSIZE, BLOCKEDTIMEOUT, CURRENTRECORDLENGTH, DIALOGCHECKINTERVAL, FRAMESIZECENSUS, and FRAMESIZE) are not supported since the socket presents a direct TCP connection, not a file interface. The state of the socket (presented in PORT FILES through the FILESTATE attribute) is available as the socket option SO_SocketState. This is an inquiry-only option.

- **Urgent Data** – All urgent data (referred to as Out of Band or OOB data), is presented in the TCP data stream (referred to as OOB_INLINE). A program can inquire if urgent data is present.

- **OPEN processing** – Initiating an OPEN command in TCP sockets is done through the connect() procedure call. Performing a passive OPEN (equivalent to an AWAITOPEN call in PORT FILES) is done through a combination of the listen() procedure call, which puts the socket in the passive state, and the accept() procedure call, which accepts the next client request, blocking until one arrives.

- **Blocking operation** – All calls in sockets are blocking; they do not return control to the calling program until they have finished their processing. If a non-blocking read is required by the program the program should check to see if any data is available, using ioctl() for one socket or select() for multiple sockets, before issuing the recv() procedure call, or specify a small value for the receive timeout (RcvTimeO) option. If a non-blocking accept() is required by the program, the program should check to see if any sockets are waiting for an accept(), using ioctl() for one socket or select() for multiple sockets, before issuing the accept(). Refer to Section 4 for more information on ioctl() and setsockopt().

Socket Communication States

As shown in Figure 1-1, sockets operate in states, reacting to programming calls made to the socket.

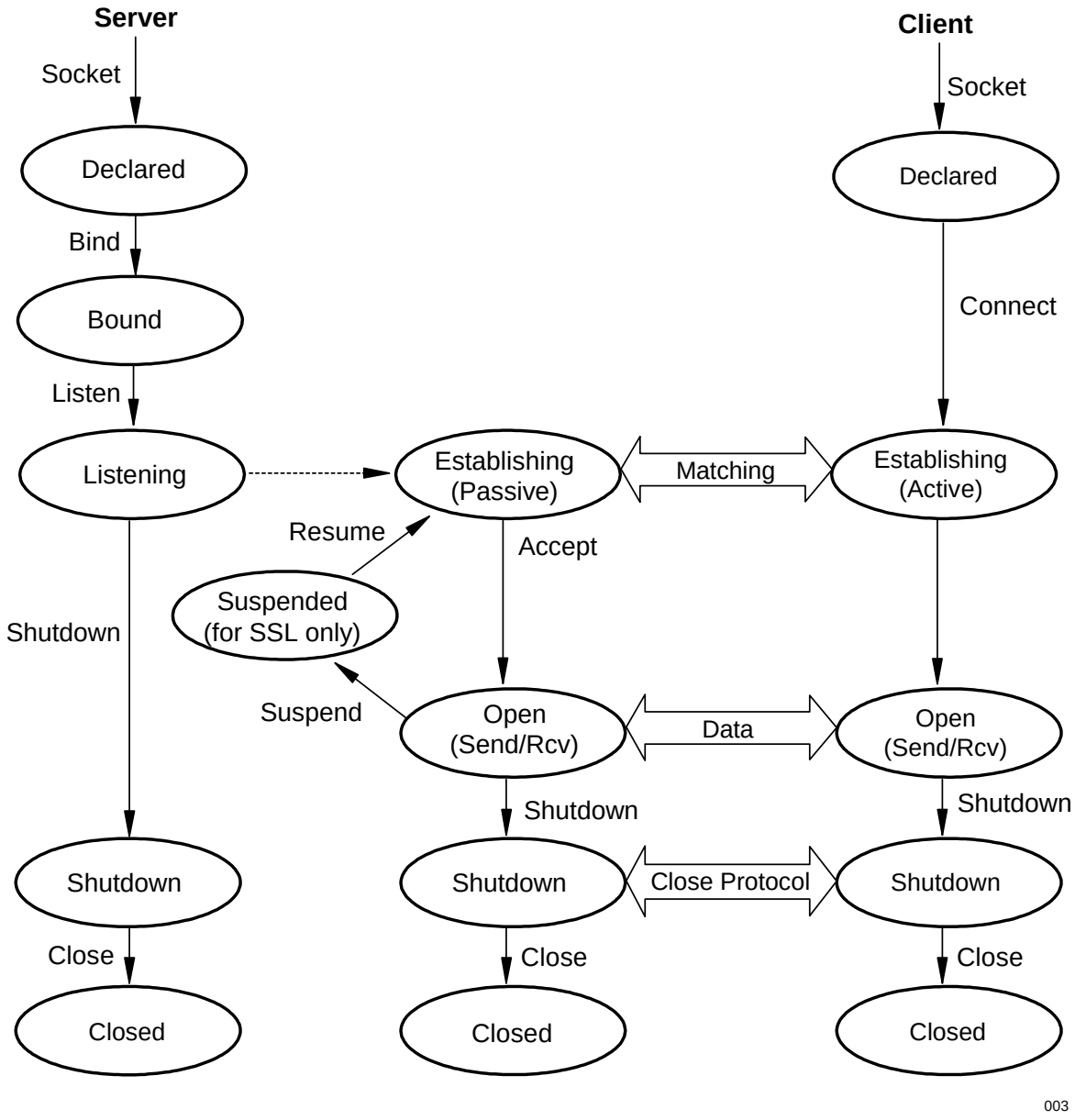


Figure 1–1. Socket Communication Overview

The following list describes the states illustrated in Figure 1-1:

Declared – A virgin socket has been created with a SockLib_Socket call.

Bound – A local IP address has been bound to the socket with a SockLib_Bind call. Typically, the SockLib_Bind call is only required for server sockets. For client sockets, this is an optional call; SockLib_Connect will implicitly assign a local IP address and an unused local port number if SockLib_Bind is not called.

Listening – SockLib_Listen has been called for a server socket.

Establishing – SockLib_Connect has been called (that is, a client socket is in the process of establishing communication with its corresponding server socket), or SockLib_Listen was called (that is, a server socket is preparing to accept—SockLib_Accept—corresponding clients).

Open – Communication with a corresponding socket has been successfully completed using either a SockLib_Connect (client) or SockLib_Listen/SockLib_Accept (server).

Suspended – Communication with a corresponding socket has been temporarily terminated.

Resumed – Communication with a corresponding socket may be resumed.

Note: *The Suspended and Resumed states are only valid for sockets with SSL enabled.*

Shutdown – Communication with the corresponding socket is being terminated. Shutting down can have various forms based on the input parameter.

SockLib_Shutdown (SD_Send) – The socket user has requested shutdown of the SockLib_Send interface. The underlying TCP connection is closed. Recv data can still arrive and be read.

SockLib_Shutdown (SD_Receive) – The socket user has requested shutdown of the receive interface. No "network" action is taken. If Recv data is waiting or Recv data arrives, the underlying TCP connection is RESET.

SockLib_Shutdown (SD_Both) – Both of the above have occurred.

tcp_TerminationIndication – The underlying TCP connection has been closed, which may have been initiated by the corresponding socket.

Closed – Shutdown of the socket has completed.

Sockets Overview

Installing the MCP Sockets Service

The user can install the MCP Sockets API on an MCP system using Simple Installation. The user can also manually install the Sockets API by issuing the following command:

```
SL SOCKETSUPPORT = SYSTEM/SOCKETSUPPORT: LINKCLASS=1, TRUSTED
```

Where the file name "SYSTEM/SOCKETSUPPORT" occurs, substitute the actual file name and pack where the system software is located.

Samples

There are four files for use by sockets programmers:

SYMBOL/SOCKETS/INCLUDE/SOCKLIB/NEWP is an example of a declaration of the library in NEWP.

SYMBOL/SOCKETS/INCLUDE/SOCKLIB/ALGOL is an example of a declaration of the library in ALGOL.

SYMBOL/SOCKETS/INCLUDE/ENUMERATIONS has the declarations for the library interface.

SYMBOL/SOCKETS/SAMPLE/COBOL is an example of how to call the sockets library in COBOL.

These files do not need to be present for sockets to run.

Verification

Issue the following command to show that the library is established:

```
SL SOCKETSUPPORT
```

Note: *The sockets interface requires that TCPIPSUPPORT be running. Use of the Secure Sockets Layer (SSL) functionality also requires that TCPIPSecurity be running with SSL configured and enabled. See Appendix J of the* ClearPath Enterprise Servers Security Administration Guide *for more detailed information about SSL.*

Section 2
Sockets Library Interface

This section describes the sockets library interface.

C Compiler Sockets.h File

The following diagram illustrates the design to provide a sockets library interface to C programs such as the Java Virtual Machine (JVM), COBOL, ALGOL, and NEWP programs. Specifically, this design provides both TCP and UDP blocking sockets functionality. For details on the use of the sockets.h file, refer to the *ClearPath HMP NX and A Series C Programming Reference Manual.*

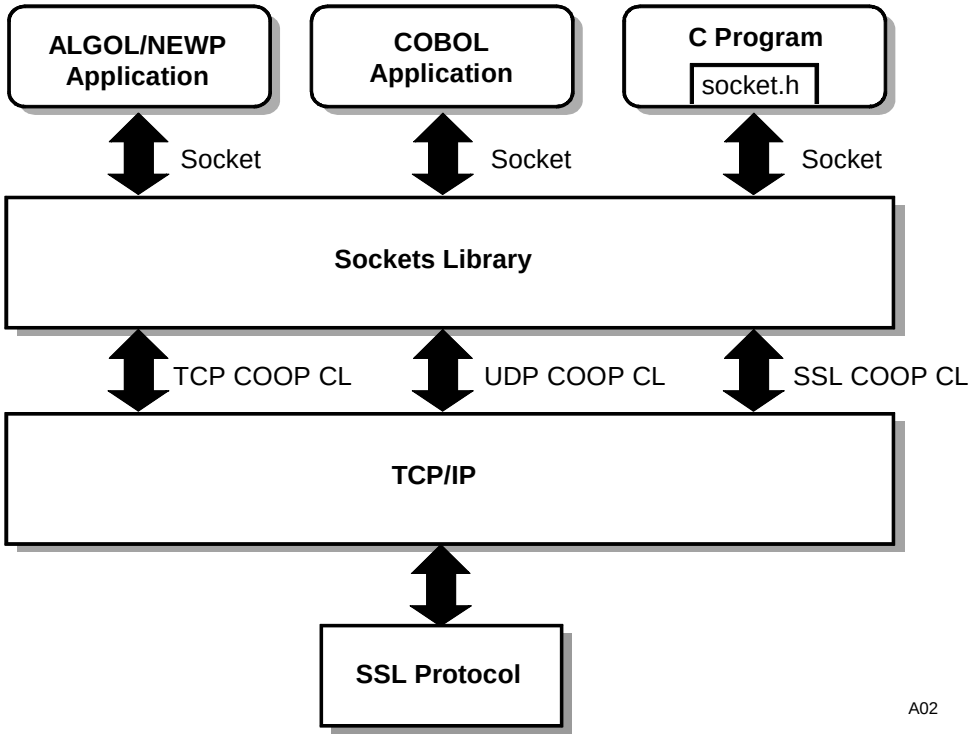


Figure 2–1. Overview of the Sockets Library Interface

SocketSupport Library

The SocketSupport Library is a SHARED library (compiler option SHARING = SHAREDBYALL) that exports the functions that comprise a Berkeley Software Distribution (BSD) style sockets interface. These functions are described in more detail in the following subsections. When this library is initialized, it will attempt to link to the TCPIP COOP connection libraries in the TCPIPSupport library and then FREEZE (temporary). If the links are successful, sockets functionality will be available to the client. If the links are not successful, an error will be returned to the client.

If a user references any SSL attributes, an attempt will be made to link to the SSL COOP reference library in the TCPIPSupport library. If that link is successful, SSL functionality will be available.

Although only blocking operations are supported, a program may have multiple dependent tasks calling the same entry points concurrently for different sockets. A task using one socket will not interfere with a task using a different socket.

Note: *The sockets interface requires that TCPIPSupport be running. Use of the SSL functionality also requires TCPIPSecurity to be running with SSL configured and enabled.*

Purpose of this Library

This library provides sockets functionality directly to programs that are written in COBOL, ALGOL, or NEWP.

This library provides sockets functionality to the C sockets.h file for programs that are written in C.

Functions

Library Startup

The TCP/IP SocketSupport Library is initialized when a C program invokes a socket function from its imported socket.h file, or when a COBOL, ALGOL, or NEWP program performs a LinkLibrary function with the Sockets library or calls one of the sockets API routines.

SockLib_Accept

This function accepts an incoming connection on a previously created and bound server socket and returns the identifier of the newly connected socket. This operation is a blocking operation, meaning it will not complete until either an incoming connection is received or the socket closes.

SockLib_Bind

This function associates a local name (for example, "My Port") to a previously created but not yet named socket.

SockLib_Close

This function destroys a socket. This operation is a blocking operation, meaning it will not complete until the socket is deallocated.

SockLib_Connect

This function initiates a connect on a previously created client socket. This is a blocking operation, meaning it will not complete until either the connect completes and the socket opens, or until the socket closes.

Note: *This operation returns an error if SSL is enabled on the socket. (Only the server side of SSL is supported in the MCP environment.)*

SockLib_GetHostByAddr

This function returns the information about the host with the IP address specified.

SockLib_GetHostByName

This function returns the information about the host specified.

SockLib_GetHostName

This function returns the name of the local host.

SockLib_GetPeerName

This function returns the name of the peer (for example, "Your Port") connected to the specified socket.

SockLib_GetSockName

This function returns the current name (for example, "My Port") of the specified socket.

SockLib_GetSockOpt

This function returns the current value of the specified socket option.

SockLib_IOCtl

This function enables the user to control the mode of the specified socket. The FIONBIO command used by WinSock, which allows the user to enable or disable blocking mode for the socket, is not supported by this implementation. See Section 4 for a list of commands that are supported in this implementation.

SockLib_Listen

This function initiates a listen operation on a previously created and bound socket.

SockLib_Recv

This function initiates a READ with flags operation on a currently connected TCP or UDP socket. This is a blocking operation, meaning it will not complete until either there is data to read, the socket closes, or the SO_RcvTimeout (if any) for the socket has expired.

SockLib_RecvFrom

This function initiates a READ with flags operation on a previously created and bound UDP socket with the specified address, or a TCP socket. This is a blocking operation, meaning it will not complete until either there is data to read, the socket closes, or the SO_RcvTimeout (if any) for the socket has expired.

SockLib_Resume

This function allows an application using an SSL socket to resume a suspended session, opening a new corresponding TCP connection.

SockLib_Select

This function allows an application to wait for data, opens, send window, and exceptions, with a timeout, on multiple sockets.

SockLib_Send

This function initiates a WRITE operation with flags on a currently connected TCP or UDP socket. This is a blocking operation, meaning it will not complete until either the data is accepted for transmission by TCP/IP, the socket closes, or the SO_SndTimeout (if any) for the socket has expired.

SockLib_SendTo

This function initiates a WRITE operation to the specified address with flags on a previously created and bound UDP socket, or a WRITE operation to a TCP socket. This is a blocking operation, meaning it will not complete until either the data is accepted for transmission by TCP/IP, the socket closes, or the SO_SndTimeout (if any) for the socket has expired.

SockLib_SetSockOpt

This function sets the value of the specified socket option. It is used to configure attributes such as debugging, multicast, close linger, and SSL.

SockLib_Shutdown

This function disables sends and/or receives on a socket. The socket is not deallocated until a SockLib_Close is performed.

SockLib_Socket

This function creates a socket endpoint and returns the identifier of the socket.

SockLib_Suspend

This procedure allows an application using an SSL socket to suspend the session, closing the underlying TCP connection. The session is still allocated until an explicit SockLib_Close is performed on the socket.

Accessing the Library

This library is accessed implicitly or explicitly by linking to the library and issuing procedure calls into it.

Common Data Structures

The MCP sockets API functions primarily operate on three types of parameters: integers, pointers, and an address data structure called a "sockaddr." A sockaddr is used to pass a socket "name," where "name" includes both port number and IP address (a socket pair). The sockets.h file will transform the C sockaddr structure to an ALGOL sockaddr structure, which is a real array 3 words long that contains the address family (AF_INET) in word 0, the port number in word 1, and the IP address in word 2. An IP V4 address will be 4 bytes, right-justified in a 48-bit word, with each byte representing a node of the IP V4 address in sequential order left to right. For example, 192.85.16.1 would be represented as 0000C0551001.

Several additional data structures are passed as parameters. These are described with the routines that use them.

Array Parameters

Parameters to the sockets library routines that are arrays can be declared in one of two ways for programs written in ALGOL or NEWP. The first way is to declare them with a "*" lower bound, as in the sample files. The caller then passes an array to the routine, which is accessed beginning at the first element.

For example, in SockLib_Recv, the current declaration in the sample file is:

```
INTEGER PROCEDURE SockLib_Recv (Socket_Id,
                                Buffer,
                                Length,
                                Flags);
VALUE   Socket_Id,
        Length,
        Flags;
INTEGER Socket_Id,
        Length,
        Flags;
EBCDIC ARRAY Buffer [*];
IMPORTED;
```

Calling example:

```
INTEGER Result, Socket, InputLength, InputFlags;
EBCDIC ARRAY InputBuffer [0:2000];

Result := SU.SockLib_Recv (Socket, InputBuffer, InputLength, InputFlags);
```

The second way to declare array parameters is with a "0" lower bound immediately followed by an integer offset. The caller then passes an array and an offset to the routine. The array is accessed beginning at the offset.

For example, in SockLib_Recv, a valid modification of the declaration from the sample file is:

```
INTEGER PROCEDURE SockLib_Recv (Socket_Id,
                                Buffer,
                                ByteOffset,
                                Length,
                                Flags);
VALUE   Socket_Id,
        ByteOffset,
        Length,
        Flags;
INTEGER Socket_Id,
        ByteOffset,
        Length,
        Flags;
EBCDIC ARRAY Buffer [0];
IMPORTED;
```

Calling example:

```
INTEGER Result, InputOffset, Socket, InputLength, InputFlags;
EBCDIC ARRAY InputBuffer [0:2000];

Result := SU.SockLib_Recv (Socket, InputBuffer, InputOffset, InputLength,
InputFlags);
```

Using Secure Sockets Layer (SSL)

The use of Secure Sockets Layer (SSL) is restricted to server programs (Connect() will return an error if SSL is enabled on the socket). In order to use SSL, the name of the key container must be configured using the SetSockOpt() SSL_KEY_CONTAINER option. The user also has the option to set the version of SSL to be used and the list of SSL ciphers suites on the socket.

Note: *For more information about keys and certificates and how to create and install them, refer to Appendix J of the* Security Administration Guide. *For more information on how to set SSL options, refer to Section 4, "SetSockOpt()."*

Socket Call Errors

Calls into the sockets library will generally return a zero or a positive value if there are no errors and a negative value if there are errors. Table 2-1 lists the error values that may be returned and their meaning. See the individual interface descriptions to see which errors a given interface might return.

Note: *An asterisk indicates that the error might be a system error and not a sockets error. See Table 2-2, "System Error Codes," for details on system errors.*

Table 2–1. Error Codes

Error	Error Code	Explanation
EAcces	(–24)	The socket's local address is in the range reserved for system software (1–1024).
EAddrInUse	(–48)	The socket's local address is already in use.
EAddrListLength	(–302)	The address list length is longer than the maximum address list length allowed for an output parameter.
EAddrNotAvail	(–49)	The IP address specified is not available on the system.
EAliasListLength	(–301)	The alias list length is longer than the maximum alias list length allowed for an output parameter.
EConnRefused	(–61)	The peer socket refused the connection.
EFault	(–114)	TCP/IP returned an error.
EHostNameLength	(–300)	The host name is longer than the maximum host name length allowed for an output parameter.
EInProgress	(–100)	A blocking call is in progress.

Table 2–1. Error Codes

Error	Error Code	Explanation
EInval	(–122)	Meaning varies depending on the interface involved.
EIsConn	(–56)	The socket is already connected.
EMsgSize	(–83)	For a DGRAM receive, the message would not fit in the supplied buffer. For a send, the size of the send is greater than 65535 bytes.
ENetReset*	(–52)	TCP/IP is no longer available.
ENoBufs*	(–55)	The system ran out of internal buffer space.
ENoProtoOpt	(–50)	The option is unknown/unsupported.
ENotConn	(–57)	The connection has been reset due to Keep-Alive.
ENotGettable	(–401)	The option specified is not allowed for GetSockOpt.
ENotSettable	(–400)	The option specified is not allowed for SetSockOpt.
ENotSock	(–59)	The socket parameter is not a valid socket id.
EOpNotSupp	(–62)	Meaning varies depending on the interface involved.
EProtoNoSupport	(–63)	The type specified is not supported.
EShutDown	(–58)	An Accept was done on a socket that has been Shutdown.
ESSLAttributeNotAvail	(–227)	The attribute specified has not been set.
ESSLBadRecordMAC	(–232)	A message has been received with an incorrect MAC.
ESSLClientNotSupported	(–241)	SSL clients are not supported in this release.
ESSLConnectionClosed	(–225)	The SSL connection associated with the socket has closed.
ESSLCryptoNotAvailable*	(–217)	The cryptography interface is not available.
ESSLHandshakeFailed	(–216)	An attempt to connect to the socket failed due to a bad handshake attempt.
ESSLInvalidAttribute	(–215)	An invalid attribute id was specified.

Table 2–1. Error Codes

Error	Error Code	Explanation
ESSLInvalidAttributeValue	(–224)	The value associated with an attribute is invalid.
ESSLInvalidHandle	(–202)	The SSL session id associated with this socket is not valid.
ESSLInvalidRecord	(–228)	A message has been received that is formatted incorrectly.
ESSLInvalidState*	(–212)	SSL has been disabled on the system.
ESSLInvalidVersion	(–203)	A message has been received with an unsupported version identifier.
ESSLNoCertificate	(–205)	No certificate was found in the key container specified for the socket.
ESSLNoKeyDefined	(–204)	No key has been defined for the socket.
ESSLNoResources*	(–222)	The maximum number of resources is already in use.
ESSLNotLinked*	(–238)	Support for SSL has terminated.
ESSLProtocolError	(–231)	A message has been received that does not follow the SSL handshake protocol.
ESSLSecurityViolation	(–206)	The application is not allowed to use the specified key container.
ESSLSessionInterrupted*	(–200)	The cryptography interface aborted an operation.
ESSLSessionNotConfigured	(–230)	A message has been received for a session that is not completely configured.
ESSLSessionNotResumable	(–209)	The session has not been configured as resumable.
ESSLSessionSuspended	(–218)	The SSL connection is currently in the suspended state.
ESSLSysErr*	(–201)	The MCP Crypto API encountered an error.
ESSLUnexpectedMsgType	(–229)	A message has been received which is not allowed in the current state of the connection.
ETimedOut	(-60)	A SockLib_Recv or SockLib_RecvFrom operation timed out.
EWouldBlock	(–111)	Linger timeout expires before socket close completes.

System Call Errors

Some of the available error codes indicate a problem with the system, rather than with the sockets application. These codes are described in Table 2–2.

Table 2–2. System Error Codes

Error	Error Code	Explanation
ENetReset	(—52)	TCP/IP is no longer available. The system administrator must issue an NW TCPIP + and the application must be restarted.
ENoBufs	(–55)	The system-wide maximum number of sockets is already in use. The application needs to try again later when some other sockets users have terminated.
ESSLCryptoNotAvailable	(–217)	There are no NT environments with the specified provider loaded. The system administrator must check the cryptography system and ensure that at least one NT server is available. The application must be tried again later when a server is available.
ESSLInvalidState	(–212)	One of the system components required by SSL is not operational. The system administrator must issue an NA MCAPI + and/or an NW TCPIP OPTION + SSL. The application must be tried again later when the components are operational.
ESSLNoResources	(–222)	The TCPIPSecurity library has reached the system-wide maximum number of entries on one of its tables (sessions, connections, etc.). The application must be tried again later when some other SSL users have terminated.
ESSLNotLinked	(–238)	SSL support has terminated. The system administrator must issue an NW TCPIP OPTION + SSL. The application must be tried later when SSL is again available.
ESSLSessionInterrupted	(–200)	MCAPI has interrupted the session for some reason. The system administrator needs to issue an NA MCAPI – and an NA MCAPI + and the applications must be tried again.
ESSLSysErr	(–201)	An internal system error has occurred. The system operator may need to issue an NW TCPIP OPTION – SSL and an NW TCPIP OPTION + SSL. The application must be tried again later when SSL has been restarted.

Section 3
The MCP Sockets API

Purpose of the Sockets Service

The sockets interface provides a channel for interprocess communication between programs that communicate across the Internet. Historically, interprocess communication on an A Series has taken place through the Port File mechanism. When TCP/IP network capability was initially implemented for an MCP system, the interface between applications and the TCP/IP protocol stack was also through the port file mechanism. There is no way to utilize the User Datagram Protocol using the port file interface or to access SSL functionality.

Standards and Conformance

This implementation is modeled after Berkeley Software Distribution (BSD) sockets, not Windows sockets. Differences and restrictions between this implementation and the standard BSD implementation are explained in the following paragraphs.

Blocking sockets and the sending/receiving of multicast datagrams are supported.

Out of Band (Urgent) Data

This implementation is RFC 1122-compatible (the urgent pointer points to the urgent data byte), rather than BSD-compatible (the urgent pointer points to the byte after the urgent data byte).

SockLib_Bind

When a socket is bound using INADDR_ANY, the local name (port and IP address) is assigned to a currently available value. However, there is no guarantee that the value will still be available when a subsequent SockLib_Connect () or SockLib_Listen () is performed.

When a socket is bound using a local name, the uniqueness of the name is verified against those currently in use. However, there is no guarantee that the value will still be available when a SockLib_Connect () or SockLib_Listen () is performed.

Certain addresses are restricted from use by normal applications (1–1023) SockLib_Bind () will not return an error when these addresses are used, but a subsequent call to SockLib_Connect () or SockLib_Accept () will return an error.

SockLib_IOCtl

Use of the following SockLib_IOCtl commands is not supported and will return an error:

FIONBIO

FIOASync

SIOCSHWat

SIOCGWat

SIOCSLWat

SIOCGLWat

SIOCAtMark

SockLib_Select

Since socket ids can be very large numbers, the arrays passed into SockLib_Select are not bit arrays as with some implementations, where each bit corresponds to an id. Instead, they are arrays of socket ids.

The WriteSet array is allowed for compatibility, but it is rare for a socket to not be available for writing. A socket will only be unavailable for writing if another stack has a write in progress and it has been put into flow control.

The timeout parameter is a Real variable containing the number of seconds until the timeout. Note that fractional values are allowed. This differs from other implementations where it is a structure consisting of seconds and milliseconds.

SockLib_Send/SockLib_SendTo

Use of the following SockLib_Send flag is not supported and will return an error:

Msg_DontRoute

SockLib_SetSockOpt/SockLib_GetSockOpt

Use of the following options is not supported and will return an error:

SO_ReuseAddr

SO_DontRoute

SO_Broadcast

SO_UseLoopBack

Use of the following options will not return an error but will have no effect:

SO_Error

TCP_NoDelay

SO_SndBuf

SO_RcvBuf

SockLib_Socket

The following address formats are not supported:

PFFile

PFUnix

The following socket types are not supported:

Sock_Raw

Sock_RDM

Sock_SeqPacket

The following protocols are not supported:

IPProto_ICMP

IPProto_GGP

IPProto_TCP

IPProto_PUP

IPProto_UDP

IPProto_IDP

IPProto_ND

IPProto_RAW

IPProto_MAX

Section 4
Declarations to the Sockets API

This section details declarations to the sockets API for C, ALGOL, COBOL, and NEWP programmers. You can refer to samples in the following directory on your system:

*SYMBOL/SOCKETS/

C programmers should include socket.h in their programs:
    #include <socket.h>

ALGOL programmers should include:
    $$INCLUDE SOCKLIB=*SYMBOL/SOCKETS/INCLUDE/SOCKLIB/ALGOL
    $$INCLUDE SOCKLIB=*SYMBOL/SOCKETS/INCLUDE/ENUMERATIONS

NEWP programmers should include:
    $$INCLUDE SOCKLIB=*SYMBOL/SOCKETS/INCLUDE/SOCKLIB/NEWP
    $$INCLUDE SOCKLIB=*SYMBOL/SOCKETS/INCLUDE/ENUMERATIONS

COBOL programmers should insert the lines of code found in
    *SYMBOL/SOCKETS/SAMPLE/COBOL
into the appropriate places in their program.

Programming Declarations

SockLib_Accept

Declaration in C:

```
int accept(int, sockaddr &, int &);
```

Declaration in ALGOL/NEWP:

```
INTEGER PROCEDURE SOCKLIB_ACCEPT (SOCKET, SOCKADDR, ADDRLENGTH);

VALUE SOCKET;

REFERENCE SOCKADDR, ADDRLENGTH;

INTEGER SOCKET, ADDRLENGTH;

REAL ARRAY SOCKADDR [*];
```

Access in COBOL:

CALL "SOCKLIB_ACCEPT_CBL OF SOCKETSUPPORT"

USING SOCKET, SOCKET-SOCKADDR, SOCKET-ADDRLENGTH

GIVING SOCKET-RESULT.

Input Parameters:

Socket

The identifier of the socket returned by the call to socket ().

Output Parameters:

SockAddr

The port number and IP address used by the remote application.

AddrLength

Must be set to 3 (3 words) in ALGOL/NEWP/COBOL.

Must be set to 18 (18 bytes) in C.

Only AF_INET is supported.

Results:

If positive, the procedure result is the identifier of the newly connected socket. If negative, it is an error. Zero is not a valid result for this procedure.

EInval

1. The SockAddr structure is not large enough to hold the remote's address.
2. A SockLib_Listen () has not yet been done on the socket.

ENotSock

The socket id specified is not valid.

EOpNotSupp

1. The socket id is one returned from a SockLib_Accept () call rather than a Socket () call.
2. The socket is not a Sock_Stream socket.

EShutdown

The socket has been shut down.

ENetReset

The TCP/IP is no longer available.

ESSLSessionInterrupted

The cryptography engine aborted the operation.

ESSLCryptoNotAvailable

The cryptography interface is not available.

ESSLNotLinked

Support for the SSL has terminated.

ESSLNoCertificate

No certificate was found in the key container specified for the socket.

ESSLInvalidState

SSL has been disabled on the system.

ESSLHandshakeFailed

An attempt to connect to the socket failed due to a bad handshake attempt.

ESSLSecurityViolation

The application is not allowed to use the specified key container.

Conditions:

The socket specified must be one returned from a socket () call, and a SockLib_Listen () must have been performed on the socket.

Waits for a passive open to complete.

SockLib_Bind

Declaration in C:

```
int bind(int, sockaddr &, int);
```

Declaration in ALGOL/NEWP:

```
INTEGER PROCEDURE SOCKLIB_BIND (SOCKET, SOCKADDR, ADDRLENGTH);

VALUE SOCKET, ADDRLENGTH;
```

```
REFERENCE SOCKADDR;

INTEGER SOCKET, ADDRLENGTH;

REAL ARRAY SOCKADDR [*];
```

Access in COBOL:

```
CALL "SOCKLIB_BIND_CBL OF SOCKETSUPPORT"

USING SOCKET, SOCKET-SOCKADDR, SOCKET-ADDRLENGTH

GIVING SOCKET-RESULT.
```

Input Parameters:

Socket

The identifier of the socket returned by the call to SockLib_Socket ().

SockAddr

The source port number ("MyName") and IP address to be associated with this socket. If the port number in SockAddr is null, the TCPIPSupport library will choose a unique port number for the socket. The port number may become unavailable before the connect or listen is complete, in this case an EInval will be returned from the SockLib_Connect () or SockLib_Listen ().

For stream sockets, if the IP address in SockAddr is null or INADDR_ANY, it will be filled in when the connection is opened. These values may be retrieved later by calling SockLib_GetSockName ().

AddrLength

Must be set to 3 (3 words) in ALGOL/NEWP/COBOL.

Must be set to 18 (18 bytes) in C.

Only AF_INET is supported.

Output Parameters:

None.

Results:

If the SockLib_Bind is successful, the result is 0; if an error occurred, a negative error code is returned.

EAcces

The application does not have permission to use the local address that was bound to the socket.

EAddrInUse

The socket's local address is already in use.

EAddrNotAvail

The IP address specified is not available on the system.

ENotSock

The socket id specified is not valid.

EInval

1. The SockAddr structure or AddrLength is invalid.
2. The socket is not in the correct state for a Bind.

EOpNotSupp

The socket id is one returned from a SockLib_Accept () call rather than a Socket () call.

ENetReset

The TCP/IP is no longer available.

Conditions:

SockLib_Bind () is an optional call for client sockets.

The socket specified must be one returned from a SockLib_Socket () call, not a SockLib_Accept () call.

Only valid before a SockLib_Connect () or SockLib_Listen () has been done.

Certain addresses are restricted from use by normal applications. SockLib_Bind () will return an error when these addresses are used.

If the socket type is Sock_Dgram, an implicit SockLib_Connect will be done to open the socket when a SockLib_Bind is done.

SockLib_Close

Declaration in C:

int closesocket(int);

Declaration in ALGOL/NEWP:

```
INTEGER PROCEDURE SOCKLIB_CLOSE(SOCKET);

VALUE SOCKET;

INTEGER SOCKET;
```

Access in COBOL:

```
CALL "SOCKLIB_CLOSE_CBL OF SOCKETSUPPORT"

USING SOCKET

GIVING SOCKET-RESULT.
```

Input Parameters:

Socket

The identifier of the socket returned by a SockLib_Socket or SockLib_Accept call.

Output Parameters:

None.

Results:

If the SockLib_Close is successful, the result is 0; if an error occurred, a negative error code is returned:

EInProgress

A close is already in progress for the socket.

ENotSock

The socket id specified is not valid.

ENetReset

The TCP/IP is no longer available.

Conditions:

SO_Linger is only applicable to stream sockets. If SO_Linger is not set, the socket is closed immediately but the underlying connection is not closed until all data has been acknowledged by an "ACK" message.

If SO_Linger is set with a zero timeout, the socket is closed immediately, pending data is discarded, and the TCP connection is reset.

If SO_Linger is set with a non-zero timeout, the socket waits for all sent data to be acknowledged by an “ACK” message until the timeout expires. When all data has been acknowledged by an "ACK" message, or after the timeout has expired, the socket is closed.

SockLib_Connect

Declaration in C:

```
int connect(int, sockaddr &, int);
```

Declaration in ALGOL/NEWP:

```
INTEGER PROCEDURE SOCKLIB_CONNECT (SOCKET, SOCKADDR, ADDRLENGTH);

VALUE SOCKET, ADDRLENGTH;

REFERENCE SOCKADDR;

INTEGER SOCKET, ADDRLENGTH;

REAL ARRAY SOCKADDR [*];
```

Access in COBOL:

```
CALL “SOCKLIB_CONNECT_CBL OF SOCKETSUPPORT”

USING SOCKET, SOCKET-SOCKETADDR, SOCKET-ADDRLENGTH

GIVING SOCKET-RESULT.
```

Input Parameters:

Socket

The identifier of the socket returned by a SockLib_Socket call.

SockAddr

The remote port and IP address that the local Socket is to be connected to. The local Socket may or may not have been previously bound; if not, a unique local port/IP address pair is assigned to the Socket and it is considered bound.

AddrLength

Must be set to 3 (3 words) in ALGOL/NEWP/COBOL.

Must be set to 18 (18 bytes) in C.

Only AF_INET is supported.

Output Parameters:

None.

Results:

If the SockLib_Connect is successful, the result is 0; if an error occurred, a negative error code is returned.

EConnRefused

The remote socket refused the connection.

EFault

The TCP/IP returned an error.

EInval

1. The SockAddr structure or AddrLength is invalid.
2. The TCPIP Support library returned an error.
3. A Sock_DGram socket is not in a valid state for a Connect.

EIsConn

The socket is already connected.

ENetReset

TCP/IP is no longer available.

ENotSock

The socket id specified is not valid.

EOptNotSupp

The socket id is one returned from a SockLib_Accept () call rather than a SockLib_Socket () call.

EAcces

The application does not have permission to use the local address that was bound to the socket.

ESSLClientNotSupported

SSL clients are not supported in this release.

Conditions:

The socket must be one returned from a call to SockLib_Socket ().

If the socket is not already bound, it is bound now.

If the socket type is Sock_Stream:

The socket group must not have been previously connected.

An active open is issued and the SockLib_Connect blocks until it is completed.

If the socket type is Sock_DGram:

The port and IP address in SockAddr are stored for use with future SockLib_Send operations, replacing any previously stored values.

SockLib_GetHostByAddr

Declaration in C:

```
int gethostbyaddr(int, int, int,

                  char (&) [ ], int &,

                  char (&) [ ], int &, int &,

                  float (&) [ ], int &, int &, int &, int &);
```

Declaration in ALGOL/NEWP:

```
INTEGER PROCEDURE SOCKLIB_GETHOSTBYADDR (NETADDR, NETADDRLENGTH, NETADDRTYPE,
HOSTNAME, MAXHOSTNAMELENGTH, ALIASLIST, MAXALIASLISTLENGTH, NUMALIASES,
ADDRLIST, MAXADDRLISTLENGTH, NUMADDRS, ADDRTYPE, ADDRLENGTH);

VALUE NETADDR, NETADDRLENGTH, NETADDRTYPE;

REFERENCE HOSTNAME, MAXHOSTNAMELENGTH, ALIASLIST, MAXALIASLISTLENGTH,
NUMALIASES, ADDRLIST, MAXADDRLISTLENGTH, NUMADDRS, ADDRTYPE, ADDRLENGTH;

INTEGER NETADDR, NETADDRLENGTH, NETADDRTYPE, MAXHOSTNAMELENGTH,
AXALIASLISTLENGTH, NUMALIASES, MAXADDRLISTLENGTH, NUMADDRS, ADDRTYPE,
ADDRLENGTH;

REAL ARRAY ADDRLIST [*];

EBCDIC ARRAY HOSTNAME [*], ALIASLIST [*];
```

Access in COBOL:

```
CALL "SOCKLIB_GETHOSTBYADDR_CBL OF SOCKETSUPPORT"

USING SOCKET-NETADDR, SOCKET-NETADDRLENGTH, SOCKET-NETADDRTYPE,

      SOCKET-HOSTNAME, SOCKET-MAXHOSTNAMELENGTH,

      SOCKET-ALIASLIST, SOCKET-MAXALIASLISTLENGTH, SOCKET-NUMALIASES,
```

SOCKET-ADDRLIST, SOCKET-MAXADDRLISTLENGTH, SOCKET-NUMADDRS,

SOCKET-ADDRTYPE, SOCKET-ADDRLENGTH

GIVING SOCKET-RESULT.

Input Parameters:

NetAddr

The IP address of the desired host information, formatted like the IP address in a SockAddr structure.

NetAddrLength

The number of bytes in NetAddr—it must be 6.

NetAddrType

The address type of NetAddr—it must be AF_INET.

Input/Output Parameters:

MaxHostNameLength

The maximum number of characters, including the terminating null (h00) that can be returned in HostName. If EHostNameLength is returned, this parameter returns the number of characters needed for the host name. Note that the returned length is not guaranteed to still be correct for a subsequent call to SockLib_GetHostByAddr.

MaxAliasListLength

The maximum number of characters, including the terminating nulls (h00) that can be returned in AliasList. If EAliasListLength is returned, this parameter returns the number of characters needed for the alias list. Note that the returned length is not guaranteed to still be correct for a subsequent call to SockLib_GetHostByAddr.

MaxAddrListLength

The maximum number of words that can be returned in AddrList. If EAddrListLength is returned, this parameter returns the number of words needed for the address list. Note that the returned length is not guaranteed to still be correct for a subsequent call to SockLib_GetHostByAddr.

Output Parameters:

HostName

The host name associated with NetAddr. This is a string of characters up to MaxHostNameLength long terminated by a null character (h00).

AliasList

A list of aliases for the NetAddr. This is a series of strings of characters, each terminated by a null character (h00), up to MaxAliasListLength.

NumAliases

The number of strings of characters returned in AliasList.

AddrList

A series of words, up to MaxAddrListLength, each containing an IP address associated with NetAddr, formatted like the IP address in a SockAddr structure.

NumAddrs

The number of words in AddrList that contain an IP address.

AddrType

The type of the addresses in AddrList—it will always be AF_INET.

AddrLength

The length of the addresses in AddrList—it will always be 6.

Results:

If SockLib_GetHostByAddr is successful, the result is 0. If an error occurred, a negative error code is returned and no host information is returned.

EAddrNotAvail

There is no information about the NetAddr specified.

EProtoNoSupport

NetAddrLength or NetAddrType is not supported.

EInval

An array specified for output is not as big as Max…Length implies.

EHostNameLength

The host name is longer than MaxHostNameLength.

EAliasListLength

The alias list length is longer than MaxAliasListLength.

EAddrListLength

The address list length is longer than MaxAddrListLength.

Conditions:

The DNS resolver is invoked for the specified address to obtain the host information.

SockLib_GetHostByName

Declaration in C:

```
int gethostbyname(char (&) [ ], int,

                  char (&) [ ], int &,

                  char (&) [ ], int &, int &,

                  float (&) [ ], int &, int &, int &, int &);
```

Declaration in ALGOL/NEWP:

```
INTEGER PROCEDURE SOCKLIB_GETHOSTBYNAME (NETNAME,NETNAMELENGTH, HOSTNAME,
MAXHOSTNAMELENGTH, ALIASLIST, MAXALIASLISTLENGTH, NUMALIASES, ADDRLIST,
MAXADDRLISTLENGTH, NUMADDRS, ADDRTYPE, ADDRLENGTH);

VALUE NETNAMELENGTH;

REFERENCE NETNAME, HOSTNAME, MAXHOSTNAMELENGTH, ALIASLIST, MAXALIASLISTLENGTH,
NUMALIASES, ADDRLIST, MAXADDRLISTLENGTH, NUMADDRS, ADDRTYPE, ADDRLENGTH;

INTEGER NETNAMELENGTH, MAXHOSTNAMELENGTH, MAXALIASLISTLENGTH, NUMALIASES,
MAXADDRLISTLENGTH, NUMADDRS, ADDRTYPE, ADDRLENGTH);

REAL ARRAY ADDRLIST [*];

EBCDIC ARRAY NETNAME [*], HOSTNAME [*], ALIASLIST [*];
```

Access in COBOL:

```
CALL "SOCKETLIB_GETHOSTBYNAME_CBL OF SOCKETSUPPORT"

USING SOCKET-NETNAME, SOCKET-NETNAMELENGTH,

      SOCKET-HOSTNAME, SOCKET-MAXHOSTNAMELENGTH,

      SOCKET-ALIASLIST, SOCKET-MAXALIASLISTLENGTH, SOCKET-NUMALIASES,

      SOCKET-ADDRLIST, SOCKET-MAXADDRLISTLENGTH, SOCKET-NUMADDRS,

      SOCKET-ADDRTYPE, SOCKET-ADDRLENGTH

GIVING SOCKET RESULT.
```

Input Parameters:

NetName

The name of the desired host information.

NetNameLength

The number of bytes in NetName, including a terminating null character (h00).

Input/Output Parameters:

MaxHostNameLength

The maximum number of characters, including the terminating null (h00) that can be returned in HostName. If EHostNameLength is returned, this parameter returns the number of characters needed for the host name. Note that the returned length is not guaranteed to still be correct for a subsequent call to SockLib_GetHostByName.

MaxAliasListLength

The maximum number of characters, including the terminating nulls (h00) that can be returned in AliasList. If EAliasListLength is returned, this parameter returns the number of characters needed for the alias list. Note that the returned length is not guaranteed to still be correct for a subsequent call to SockLib_GetHostByName.

MaxAddrListLength

The maximum number of words that can be returned in AddrList. If EAddrListLength is returned, this parameter returns the number of words needed for the address list. Note that the returned length is not guaranteed to still be correct for a subsequent call to SockLib_GetHostByName.

Output Parameters:

HostName

The host name associated with NetName. This is a string of characters up to MaxHostNameLength long terminated by a null character (h00).

AliasList

A list of aliases for the NetName. This is a series of strings of characters, each terminated by a null character (h00), up to MaxAliasListLength.

NumAliases

The number of strings of characters returned in AliasList.

AddrList

A list of words, up to MaxAddrListLength, each containing an IP address associated with NetName, formatted like the IP address in a SockAddr structure.

NumAddrs

The number of words in AddrList that contain an IP address.

AddrType

The type of the addresses in AddrList—it will always be AF_INET.

AddrLength

The length of the addresses in AddrList—it will always be 6.

Results:

If SockLib_GetHostByName is successful, the result is 0. If an error occurred, a negative error code is returned and no host information is returned:

ENetReset

TCP/IP is no longer available.

EAddrNotAvail

There is no information about the NetName specified.

EInval

An array specified for output is not as big as Max…Length implies.

EHostNameLength

The host name is longer than MaxHostNameLength.

EAliasListLength

The alias list length is longer than MaxAliasListLength.

EAddrListLength

The address list length is longer than MaxAddrListLength.

Conditions:

The DNS resolver is invoked for the specified address to obtain the host information.

SockLib_GetHostName

Declaration in C:

```
int gethostname(char (&) [ ], int);
```

Declaration in ALGOL/NEWP:

```
INTEGER PROCEDURE SOCKLIB_GETHOSTNAME (HOSTNAME, MAXHOSTNAMELENGTH);

VALUE MAXHOSTNAMELENGTH;

REFERENCE HOSTNAME;

INTEGER MAXHOSTNAMELENGTH;

EBCDIC ARRAY HOSTNAME [*];
```

Access in COBOL:

```
CALL "SOCKLIB_GETHOSTNAME_CBL OF SOCKETSUPPORT"

USING SOCKET-HOSTNAME, SOCKET-MAXHOSTNAMELENGTH

GIVING SOCKET-RESULT.
```

Input/Output Parameters:

MaxHostNameLength

The maximum number of characters, including the terminating null (h00) that can be returned in HostName.

Output Parameters:

HostName

The host name associated with the local host. This is a string of characters up to MaxHostNameLength long terminated by a null character (h00).

Results:

If SockLib_GetHostName is successful, the result is 0. If an error occurred, a negative error code is returned:

EInval

The HostName array specified for output is not as big as MaxHostNameLength implies.

Conditions:

The local host name is returned. If the local host name is longer than MaxHostNameLength, it is truncated to MaxHostNameLength.

SockLib_GetPeerName

Declaration in C:

int getpeername(int, sockaddr &, int &);

Declaration in ALGOL/NEWP:

```
INTEGER PROCEDURE SOCKLIB_GETPEERNAME (SOCKET, SOCKADDR, ADDRLENGTH);

VALUE SOCKET;

REFERENCE SOCKADDR, ADDRLENGTH;

INTEGER SOCKET, ADDRLENGTH;

REAL ARRAY SOCKADDR [*];
```

Access in COBOL:

CALL "SOCKLIB_GETPEERNAME_CBL OF SOCKETSUPPORT"

USING SOCKET, SOCKET-SOCKADDR, SOCKET-ADDRLENGTH

GIVING SOCKET-RESULT.

Input Parameters:

Socket

The identifier of a socket returned by the SockLib_Socket call.

Output Parameters:

SockAddr

The port and IP address of the remote peer.

AddrLength

Must be set to 3 (3 words) in ALGOL/NEWP/COBOL.

Must be set to 18 (18 bytes) in C.

Only AF_INET is supported.

Results:

If the query is successful, the result is 0; if an error occurred, a negative error code is returned:

ENotConn

The socket is not connected.

EInval

The SockAddr structure is not large enough to hold the remote's address.

ENotSock

The socket id specified is not valid.

ENetReset

TCP/IP is no longer available.

Conditions:

The socket must be connected.

SockLib_GetSockName

Declaration in C:

```
int getsockname(int, sockaddr &, int &);
```

Declaration in ALGOL/NEWP:

```
INTEGER PROCEDURE SOCKLIB_GETSOCKNAME (SOCKET, SOCKADDR, ADDRLENGTH);

VALUE SOCKET;

REFERENCE SOCKADDR, ADDRLENGTH;

INTEGER SOCKET, ADDRLENGTH;

REAL ARRAY SOCKADDR [*];
```

Access in COBOL:

```
CALL "SOCKLIB_GETSOCKNAME_CBL OF SOCKET SUPPORT"

USING SOCKET, SOCKET-SOCKADDR, SOCKET-ADDRLENGTH

GIVING SOCKET-RESULT.
```

Input Parameters:

Socket

The identifier of a socket returned by the SockLib_Socket call.

Output Parameters:

SockAddr

The local socket's port number and IP address.

AddrLength

Must be set to 3 (3 words) in ALGOL/NEWP/COBOL.

Must be set to 18 (18 bytes) in C.

Only AF_INET is supported.

Results:

If the query is successful, the result is 0; if an error occurred, a negative error code is returned:

ENotSock

The socket id specified is not valid.

EInval

1. The SockAddr structure is not large enough to hold the remote's address.
2. A SockLib_Bind () has not been done, or a SockLib_Bind (Socket, Any_Addr, Any_Addr) has been done but the socket has not been connected yet.

ENetReset

TCP/IP is no longer available.

Conditions:

The local address for the socket must have been established.

SockLib_GetSockOpt

Declaration in C:

```
int getsockopt(int, int, int, char (&) [ ], int &);
```

Declaration in ALGOL/NEWP:

```
INTEGER PROCEDURE SOCKLIB_GETSOCKOPT (SOCKET, LEVEL, OPTION, OPTVAL, OPTLEN);

VALUE SOCKET, LEVEL, OPTION;

REFERENCE OPTVAL, OPTLEN;
```

```
INTEGER SOCKET, LEVEL, OPTION, OPTLEN;

EBCDIC ARRAY OPTVAL[*];
```

Access in COBOL:

```
CALL "SOCKLIB_GETSOCKOPT_CBL OF SOCKETSUPPORT"

USING SOCKET, SOCKET-LEVEL, SOCKET-OPTION,

        SOCKET-OPTVAL, SOCKET-OPTLEN

GIVING SOCKET-RESULT.
```

Input Parameters:

Socket

The identifier of the socket returned by the SockLib_Socket or SockLib_Accept call.

Level

The level associated with the option

Option

The option whose value is being queried

OptLen

The length of OptVal in bytes

See SockLib_SetSockOpt for an enumeration of the input parameters.

Output Parameters:

OptVal

If OptLen > 0, this structure contains the value of the specified option. If OptLen = 0, then the size of this array was insufficient to contain the option value. If the option is a Boolean or integer type, the value is returned in a 6-byte field.

OptLen

The length in bytes of the returned option value

Results:

If the query is successful, the result is 0; if an error occurred, a negative error code is returned.

See SockLib_SetSockOpt for an enumeration of the error codes returned.

SockLib_IOCtl

Declaration in C:

int ioctlsocket(int, long, unsigned long &);

Declaration in ALGOL/NEWP:

```
INTEGER PROCEDURE SOCKLIB_IOCTL (SOCKET, CMD, ARGP);

VALUE SOCKET, CMD;

REFERENCE ARGP;

INTEGER SOCKET, CMD;

EBCDIC ARRAY ARGP[*];
```

Access in COBOL:

CALL "SOCKLIB_IOCTL_CBL OF SOCKETSUPPORT"

USING SOCKET, SOCKET-CMD, SOCKET-ARGP

GIVING SOCKET-RESULT.

Input Parameters:

Socket

The identifier of the socket returned by the SockLib_Socket call.

Cmd

The command to be processed:

FION_Read (2)

For sockets of type Sock_Stream, returns the total number of bytes available to be read in ArgP.

For sockets of type Sock_DGram, returns the total number of bytes in all of the messages in the input stream, plus 16 bytes per message for addressing space in ArgP.

SIO_Set_QOS (11)

Sets the QOS attributes for the socket.

SIO_Get_QOS (7)

Returns the QOS attributes for the socket.

ArgP

Depending on the value of Cmd, this parameter may or may not contain additional parameters for the command.

For SIO_Set_QOS, ArgP contains 23 words of information in the following format:

```
QOSVersion

SendBucketRate

SendBucketDepth

SendPeakRate

SendLatency

SendDelayVariation

SendServiceType

    ServiceType_BestEffort       (1)

    ServiceType_ControlledLoad   (2)

SendMaxPacketSize

SendMinPolicedUnit

Filler

Filler

RcvBucketRate

RcvBucketDepth

RcvPeakRate

RcvLatency

RcvDelayVariation

RcvServiceType

RcvMaxPacketSize

RcvMinPolicedUnit

Filler

Filler

Filler

Filler
```

Output Parameters:

ArgP

Depending on the value of Cmd, this parameter may or may not be updated with a result value.

For SIO_Get_QOS, ArgP contains 23 words of information; see the preceding format.

For FION_Read, ArgP contains the following:

For Sock_Stream sockets, the number of bytes available to be read.

For Sock_Dgram sockets, the number of bytes in the first datagram.

For a Listening socket, a 1, if there is a pending connection waiting for a SockLib_Accept.

Results:

If the command was successfully completed, the result is 0; if an error occurred, a negative error code is returned:

ENotSock

The socket id specified is not valid.

EInval

One of the parameters has an invalid or not supported value.

ENetReset

TCP/IP is no longer available.

Conditions:

This command enables an application to set or interrogate information about the socket.

SockLib_Listen

Declaration in C:

int listen(int, int);

Declaration in ALGOL/NEWP:

```
INTEGER PROCEDURE SOCKLIB_LISTEN (SOCKET, NUMPASSIVES);

VALUE SOCKET, NUMPASSIVES;

INTEGER SOCKET, NUMPASSIVES;
```

Access in COBOL:

```
CALL "SOCKET_LISTEN_CBL OF SOCKETSUPPORT"

USING SOCKET, SOCKET-NUMPASSIVES

GIVING SOCKET-RESULT.
```

Input Parameters:

Socket

The identifier of the socket returned by the SockLib_Socket call. This socket must have been previously bound.

NumPassives

The number of passives that should be maintained for this socket. The valid range for the NumPassives input parameter to SockLib_Listen is 1–200. If an application specifies a NumPassives value outside this range, the SocketSupport library will set NumPassives to the nearest valid value.

Output Parameters:

None.

Results:

If no error occurred, the procedure returns 0; otherwise, a negative error code is returned.

EInval

A SockLib_Bind () has not been done on the socket.

ENotSock

The socket id specified is not valid.

EOpNotSupp

1. The socket id is one returned from a SockLib_Accept () call rather than a SockLib_Socket () call.

2. The socket is not a Sock_Stream socket.

ENetReset

TCP/IP is no longer available.

ESSLNoKeyDefined

No key has been defined for the socket.

ESSLSessionInterrupted

The cryptography engine has aborted the operation.

ESSLNotLinked

Support for the SSL has terminated.

ESSLCryptoNotAvailable

The cryptography interface is not available.

ESSLSysErr

The MCP Crypto API encountered an error.

Conditions:

Socket must be one returned from a call to SockLib_Socket (), not a call to SockLib_Accept ().

The socket's type must be Sock_Stream.

The socket must be bound.

Issues a passive Open_Request. When the open completes, another passive open is initiated until the backlog for the group has been reached.

SockLib_Recv

Declaration in C:

```
int recv(int, char (&) [ ], int, int);

int readsocket(int, void &, int);
```

Declaration in ALGOL/NEWP:

```
INTEGER PROCEDURE SOCKLIB_RECV (SOCKET, BUF, LEN, FLAGS);

VALUE SOCKET, LEN, FLAGS;

REFERENCE BUF;

INTEGER SOCKET, LEN, FLAGS;

EBCDIC ARRAY BUF[*];
```

Access in COBOL:

```
CALL "SOCKLIB_RECV_CBL OF SOCKET SUPPORT"

USING SOCKET, SOCKET-BUF, SOCKET-LEN, SOCKET-FLAGS

GIVING SOCKET-RESULT.
```

Input Parameters:

Socket

The identifier of the socket returned by the SockLib_Accept or SockLib_Socket call.

Buf

The buffer into which the data is to be copied.

Len

The length of the buffer.

Flags

Options for this specific SockLib_Recv.

Msg_OOB (1)

Must be FALSE since only socket option SO_OOBInline = TRUE is supported.

Msg_Peek (2)

Return data but do not remove it from the input queue.

Output Parameters:

None.

Results:

If positive, the procedure result is the length of data returned. If negative, it is an error.

ENotConn

The socket is not in a valid state for a Recv.

ENotSock

The socket id specified is not valid.

EInval

1. The buffer specified is not large enough to hold the requested amount of data.
2. A receive is already in progress.

EConnAborted

The socket has been closed by the remote side.

EOpNotSupp

The flags parameter is invalid.

ENetReset

TCP/IP is no longer available.

ESSLSessionSuspended

The SSL connection is currently in the suspended state.

ESSLSessionInterrupted

The cryptography engine has aborted the operation.

ESSLNotLinked

Support for the SSL has terminated.

ESSLCryptoNotAvailable

The cryptography interface is not available.

ESSLSysErr

The MCP Crypto API encountered an error.

ETimedOut

No data was available after waiting for the amount of time specified by SO_RcvTimeO.

Conditions:

If the socket has been closed from the remote side and all data has been received, SockLib_Recv will return zero as the size of the message.

If no data is available after the amount of time specified by SO_RcvTimeO, a result of ETimedOut will be returned.

If the socket type is Sock_DGram:

It must be bound.

If there is no data available, SockLib_Recv does not return until a complete message is available.

If it was opened via a SockLib_Connect operation, only messages from the address specified in the SockLib_Connect will be returned. All other messages received for this socket will be discarded.

If the next message in the input queue is larger than the Len supplied and the Msg_Peek flag is not set, Len bytes are returned with no error code, and the remainder of the message is discarded.

If the next message in the input queue is larger than the Len supplied and the Msg_Peek flag is set, Len bytes are returned and the result is the size of the entire message. None of the message is discarded.

If the next message in the input queue is not larger than the Len supplied, the entire message is returned and the result is the size of the message.

If the socket is type Sock_Stream:

If there is no data available, SockLib_Recv does not return until some data is available.

Up to LEN bytes are returned and the result is the number of bytes returned.

If out-of-band data is present in the data stream:

If the out-of-band byte is the next available byte, it will be returned and the result will be one.

If the out-of-band byte is not the next available byte, only data up to but not including the out-of-band byte will be returned and the result will be the number of bytes returned.

If the Msg_Peek flag is set, the data is not removed from the input stream. If the flag is not set the data is removed from the input stream.

SockLib_RecvFrom

Declaration in C:

```
int recvfrom(int, char (&) [ ], int, int, sockaddr &, int &);
```

Declaration in ALGOL/NEWP:

```
INTEGER PROCEDURE SOCKLIB_RECVFROM (SOCKET, BUF, LEN, FLAGS, SOCKADDR,
ADDRLENGTH);

VALUE SOCKET, LEN, FLAGS;

REFERENCE BUF, SOCKADDR, ADDRLENGTH;

EBCDIC ARRAY BUF[*];

INTEGER SOCKET, LEN, FLAGS, ADDRLENGTH;

REAL ARRAY SOCKADDR [*];
```

Access in COBOL:

```
CALL "SOCKLIB_RECVFROM_CBL OF SOCKETSUPPORT"

USING  SOCKET, SOCKET-BUF, SOCKET-LEN, SOCKET-FLAGS,

       SOCKET-SOCKADDR, SOCKET-ADDRLENGTH

GIVING SOCKET-RESULT.
```

Input Parameters:

Socket

The identifier of a socket returned by the SockLib_Accept or SockLib_Socket call.

Buf

The buffer into which the data is to be copied.

Len

The length of Buf.

Flags

Options for this specific SockLib_RecvFrom. See SockLib_Recv for flag definitions.

Output Parameters:

SockAddr

The source port and IP address of the application from which the datagram was received.

AddrLength

Must be set to 3 (3 words) in ALGOL/NEWP/COBOL.

Must be set to 18 (18 bytes) in C.

Only AF_INET is supported.

Results:

If positive, the procedure result is the length of data returned. If negative, it is an error.

EInval

1. The SockAddr structure is not large enough to hold the remote's address.
2. Receive is already in progress for the specified socket.

Note: *See SockLib_Recv for additional error definitions.*

Conditions:

If the socket has been closed from the remote side and all data has been received, SockLib_RecvFrom will return zero as the size of the message.

If no data is available after the amount of time specified by SO_RcvTimeO, a result of ETimedOut will be returned.

If the socket's type is Sock_Stream:

If there is no data available, SockLib_RecvFrom does not return until data is available.

The SockAddr parameter is ignored.

Additional Conditions are the same as for SockLib_Recv ().

If the socket's type is Sock_DGram:

If there is no data available, SockLib_Recv does not return until a complete message is available.

If a SockLib_Connect () call was done, the SockAddr parameter is ignored.

If there is no data available, SockLib_RecvFrom does not return until a complete message is available.

If the next message in the input queue is larger than the Len supplied and the Msg_Peek flag is not set, Len bytes are returned with no error code, and the remainder of the message is discarded.

If the next message in the input queue is larger than the Len supplied and the Msg_Peek flag is set, Len bytes are returned and the result is the size of the entire message. None of the message is discarded.

If the next message in the input queue is not larger than the Len supplied, the entire message is returned and the result is the size of the message.

The network address of the sender is copied into the SockAddr parameter.

SockLib_Resume

Declaration in C:

```
int resume(int);
```

Declaration in ALGOL/NEWP:

```
INTEGER PROCEDURE SOCKLIB_RESUME(SOCKET);

VALUE SOCKET;

INTEGER SOCKET;
```

Access in COBOL:

```
CALL "SOCKLIB_RESUME_CBL OF SOCKETSUPPORT"

USING SOCKET

GIVING SOCKET-RESULT.
```

Input Parameters:

Socket

The identifier of the socket returned by the SockLib_Accept call.

Output Parameters:

None.

Results:

If the SockLib_Resume is successful, the result is 0; if an error has occurred, a negative error code is returned.

ENotSock

The socket id specified is not valid.

ENetReset

TCP/IP is no longer available.

ESSLInvalidState

The SSL connection is not currently in the suspended state or SSL is not available.

ESSLCryptoNotAvailable

The cryptography subsystem was interrupted during the processing of this request.

ESSLSessionInterrupted

The cryptography engine has aborted the operation.

ESSLNotLinked

Support for the SSL has terminated.

ESSLHandshakeFailed

An attempt to connect to the socket failed due to a bad handshake attempt.

ESSLSysErr

The MCP Crypto API encountered an error.

Conditions:

The socket must have been previously suspended.

SockLib_Select

Declaration in C:

Note: *The first parameter (the set size) is not actually passed to the sockets library. Instead, the sockets.h file passes the three size parameters below based on how many socket ids are in each fd_set.*

```
int select(int &, fd_set &, fd_set &, fd_set &, float);
```

Supporting macros in C:

Four macros are provided in C to access the ReadSet, WriteSet, and ExceptSet structures.

void FD_ZERO (fd_set *fdset)

Clears all the entries in fdset

void FD_SET (int SocketId, fd_set *fdset)

Adds SocketId to fdset

void FD_CLR (int SocketId, fd_set *fdset)

Deletes SocketId from fdset

int FD_ISSET (int SocketId, fd_set *fdset)

Returns TRUE if SocketId is in fdset

Also, the constant FD_SETSIZE specifies the default size of an fdset structure.

Declaration in ALGOL:

```
INTEGER PROCEDURE SOCKLIB_SELECT (READSETSIZE, WRITESETSIZE, EXCEPTSETSIZE,
READSET, WRITESET, EXCEPTSET, TIMEOUT);

VALUE TIMEOUT;

REFERENCE READSETSIZE, WRITESETSIZE, EXCEPTSETSIZE, READSET, WRITESET,
EXCEPTSET;

REAL ARRAY READSET[*], WRITESET[*], EXCEPTSET[*];

INTEGER READSETSIZE, WRITESETSIZE, EXCEPTSETSIZE;

REAL TIMEOUT;
```

Access in COBOL:

```
CALL "SOCKLIB_SELECT_CBL OF SOCKETSUPPORT"

USING  SOCKET-READSETSIZE,

       SOCKET-WRITESETSIZE, SOCKET-EXCEPTSETSIZE, SOCKET-READSET,

       SOCKET WRITESET, SOCKET EXCEPTSET SOCKET-TIMEOUT

GIVING SOCKET-RESULT.
```

Input Parameters:

ReadSetSize

The size of the ReadSet.

WriteSetSize

The size of the WriteSet.

ExceptSetSize

The size of the ExceptSet.

ReadSet

An array of socket ids, each 6 bytes in length.

WriteSet

An array of socket ids, each 6 bytes in length.

ExceptSet

An array of socket ids, each 6 bytes in length.

TimeOut

The number of seconds before a timeout will occur.

Output Parameters:

ReadSetSize

The number of sockets in the ReadSet that satisfy the condition.

WriteSetSize

The number of sockets in the WriteSet that satisfy the condition.

ExceptSetSize

The number of sockets in the ExceptSize that satisfy the condition.

ReadSet

The socket ids with data to receive.

WriteSet

The socket ids that can send.

ExceptSet

The socket ids with urgent data.

Results:

If negative, an error occurred. The socket id of the socket in error remains in the ReadSet, WriteSet, or ExceptSet. All other socket ids will have been zeroed out.

Note: *Only the first socket in error is identified; additional sockets may have additional errors.*

If zero, the select timed out. Parameters have not been changed.

If positive, the number of selected sockets whose condition was satisfied. Output parameters identify the sockets whose condition was satisfied. Any socket whose condition was not satisfied will have been zeroed out.

On return from SockLib_Select, if a time out did not occur, the remaining sockets ids (only one if an error occurred) are packed to the front of the arrays and the set size parameters are adjusted to reflect the number of socket ids remaining in the arrays.

EInProgress

A select is already in progress for one of the specified sockets.

EInval

SetSize is invalid or the socket is in an invalid state for a Select.

ENetReset

TCPIP is not linked.

ENotSock

At least one of the specified sockets ids is invalid.

EShutDown

At least one of the specified sockets is in an incorrect state for a select.

ENoBufs

There are no available select events.

ESSLNotLinked

The select includes an SSL socket, but SSL is not linked.

Conditions:

The Select interface is provided to allow users to wait on multiple sockets for a read, write, open, exception, or timeout.

Since this implementation supports large numbers of sockets, the ReadSet, WriteSet, and ExceptSet structures are arrays of socket ids. This differs from some implementations where they are bit arrays with each bit representing a file descriptor. Note that the use of large set sizes will impact performance. It is recommended that socket ids are packed at the front of the arrays and set sizes are adjusted to reflect only the number of socket ids in use.

On return from SockLib_Select, if a time out did not occur, the remaining socket ids (only one if an error occurred) are packed to the front of the arrays and the set size parameters are adjusted to reflect the number of socket ids remaining in the arrays.

If the same socket id appears in any array more than once it will be counted more than once when its condition is satisfied.

A null socket id has the value of zero. Any non-zero value in the ReadSet, WriteSet, or ExceptSet should be a valid socket id for a socket that is either open or listening.

Wait for read on an open socket uses the SO_RcvLoWat setting to determine how much data must be available before the Select is satisfied. If SO_RcvLoWat is set to zero, the select will be satisfied when any input arrives, including a zero-length datagram.

Wait for read on a listening socket waits until a client has connected (i.e. a SockLib_Accept will not block).

Wait for send is provided, but will only wait if another stack is currently sending data on the socket.

The SO_SndLoWat setting is allowed for compatibility, but is ignored since it has no relevance to the sockets buffering mechanisms.

Wait for exception will wait until Out-Of-Band data is received.

A negative timeout value will disable the timeout feature.

SockLib_Send

Declaration in C:

```
int send(int, const char (&) [ ], int, int);

int writesocket(int, const void &, int);
```

Declaration in ALGOL/NEWP:

```
INTEGER PROCEDURE SOCKLIB_SEND (SOCKET, BUF, LEN, FLAGS);
```

```
VALUE SOCKET, LEN, FLAGS;

REFERENCE BUF;

EBCDIC ARRAY BUF[*];

INTEGER SOCKET, LEN, FLAGS;
```

Access in COBOL:

```
CALL "SOCKLIB_SEND_CBL OF SOCKETSUPPORT"

USING SOCKET, SOCKET-BUF, SOCKET-LEN, SOCKET-FLAGS

GIVING SOCKET-RESULT.
```

Input Parameters:

Socket

The identifier of the socket returned by the SockLib_Accept or SockLib_Socket call

Buf

The buffer from which the data is to be copied

Len

The length of the buffer in bytes

Flags

Options for this specific SockLib_Send

Msg_OOB (1)

Send as out-of-band data.

Output Parameters:

None.

Results:

If positive, the procedure result is the length of data sent. If negative, it is an error.

EMsgSize

The size of the send is greater than 65535 bytes.

ENotConn

The socket is not in a valid state for a send.

ENotSock

The socket id specified is not valid.

EInval

The buffer specified is not large enough to have the amount of data specified.

ENetReset

TCP/IP is no longer available.

EOpNotSupp

The flags specified are not supported.

ESSLSessionSuspended

The SSL connection is currently in the suspended state.

ESSLSessionInterrupted

The cryptography engine has aborted the operation.

ESSLNotLinked

Support for the SSL has terminated.

ESSLCryptoNotAvailable

The cryptography interface is not available.

ESSLSysErr

The MCP Crypto API encountered an error.

Conditions:

The socket must be open.

If Msg_OOB is TRUE, the last byte of the buffer will be marked as the out-of-band data byte.

If SO_SndTimeO has been set and the output operation is put into flow control for longer than the value specified, the send will return with the procedure result indicating the number of bytes sent. Note that some, but not all, of the data may have been sent.

SockLib_SendTo

Declaration in C:

int sendto(int, const char (&) [ ], int, int, sockaddr &, int);

Declaration in ALGOL/NEWP:

```
INTEGER PROCEDURE SOCKLIB_SENDTO (SOCKET, BUF, LEN, FLAGS, SOCKADDR,
ADDRLENGTH);

VALUE SOCKET, LEN, FLAGS, ADDRLENGTH;

REFERENCE BUF, SOCKADDR;

EBCDIC ARRAY BUF[*];

INTEGER SOCKET, LEN, FLAGS, ADDRLENGTH;

REAL ARRAY SOCKADDR [*];
```

Access in COBOL:

CALL "SOCKLIB_SENDTO_CBL OF SOCKET SUPPORT"

USING SOCKET, SOCKET-BUF, SOCKET-LEN, SOCKET-FLAGS,

SOCKET-SOCKADDR, SOCKET-ADDRLENGTH

GIVING SOCKET-RESULT.

Input Parameters:

Socket

The identifier of the socket returned by the SockLib_Accept or SockLib_Socket call.

Buf

The buffer from which the data is to be copied.

Len

The length of the buffer in bytes.

Flags

Options for this specific SockLib_SendTo. See SockLib_Send for flag definitions.

SockAddr

The destination port and IP address to which the datagram should be sent.

AddrLength

Must be set to 3 (3 words) in ALGOL/NEWP/COBOL.

Must be set to 18 (18 bytes) in C.

Only AF_INET is supported.

Output Parameters:

None.

Results:

If positive, the procedure result is the length of data sent. If negative, it is an error.

EInval

The SockAddr structure or AddrLength is invalid.

See SockLib_Send for additional error definitions.

Conditions:

If the socket's type is Sock_Stream:

SockAddr is ignored.

SockLib_SendTo is the equivalent of SockLib_Send.

If the socket's type is Sock_DGram:

SockAddr overrides any previously connected address for this message only.

SockLib_SetSockOpt

Declaration in C:

int setsockopt(int, int, int, const char (&) [ ], int);

Declaration in ALGOL/NEWP:

```
INTEGER PROCEDURE SOCKLIB_SETSOCKOPT (SOCKET, LEVEL, OPTION, OPTVAL, OPTLEN);

VALUE SOCKET, LEVEL, OPTION, OPTLEN;

REFERENCE OPTVAL;

INTEGER SOCKET, LEVEL, OPTION, OPTLEN;

EBCDIC ARRAY OPTVAL[*];
```

Access in COBOL:

```
CALL "SOCKLIB_SETSOCKOPT_CBL OF SOCKETSUPPORT"

USING  SOCKET, SOCKET-LEVEL, SOCKET-OPTION, SOCKET-OPTVAL,

       SOCKET-OPTLEN

GIVING SOCKET-RESULT.
```

Input Parameters:

Note: *The options shown below are for both SockLib_SetSockOpt and SockLib_GetSockOpt. Some options are only valid for one, as noted in their descriptions.*

Socket

The identifier of the socket returned by the SockLib_Socket or SockLib_Accept call.

Level

The level associated with the option:

SOL_Socket (hFFFF)

SOL_TCP(h0002) – There are no supported options for this level.

SOL_IP (h0001)

SOL_SSL (h0100)

IPProto_IP (h0000)

IPProto_TCP (h0006)

The following table shows which options belong to which SockLib_SetSockOpt levels. See the detailed description of each option below for more information.

Level	Option
Level SOL_Socket options:	SO_AcceptConn SO_Debug SO_KeepAlive SO_Linger SO_RcvBuf SO_RcvLoWat SO_RcvTimeO SO_SndBuf SO_SndLoWat SO_SndTimeO SO_Type

Level IPProto_TCP option	TCP_NoDelay
Level SOL_IP option	IP_Priority
Level IPProto_IP options	IP_TOS IP_MCast_TTL IP_Add_Membership IP_Drop_Membership IP_MCast_Loop IP_MCast_IF
Level SOL_SSL options:	SSL_System_Options SSL_Socket_Options SSL_System_Versions SSL_Socket_Version SSL_System_Ciphers SSL_Socket_Ciphers SSL_Key_Container SSL_Resumable

Option

The option whose value is being set or queried:

SO_Debug (h0001)

This option controls recording of debugging information in the underlying protocol modules.

Values:

Option_Debug_Socket_On (1)

Enables debugging for the specified socket.

Option_Debug_Socket_Off (-1)

Disables debugging for the specified socket.

Option_Debug_All_On (2)

Enables debugging on all sockets.

Option_Debug_All_Off (-2)

Disables debugging on all sockets.

SO_AcceptConn (h0002)

This option is only valid for SockLib_GetSockOpt ().

The socket has had SockLib_Listen () performed.

SO_KeepAlive (h0008)

This option can only be set before a SockLib_Connect () has been done.

This option controls whether the underlying protocol should periodically transmit messages on a connected socket. If the peer fails to respond to these messages, the connection is considered broken. The value is an integer where a nonzero value means "yes," and is interpreted as the number of minutes to wait for a peer to respond.

SO_Linger (h0080)

This option specifies what should happen when the socket of a type that promises reliable delivery still has messages that have not yet been transmitted when it is closed. The structure is a pair of integers. The first integer is interpreted as a Boolean. If nonzero, close blocks until the data is transmitted or the timeout period has expired. The second integer specifies the timeout period in seconds.

SO_OOBInline (h0100)

This is always set.

Out-of-band data received on the socket is placed in the normal input queue. The value is an integer where a nonzero value means "yes."

SO_RcvLoWat (h1004)

Select receive low water mark. This option sets the number of bytes that must be available before a Select terminates its wait for a socket in the ReadSet. The default value is one byte. This option is available with MCP release 8.0 and later.

SO_RcvTimeO (h1006)

Receive timeout. This option sets the amount of time that a receive will block waiting for data. The value is a six byte real specifying the number of seconds to wait. Setting the value to zero disables the timeout function. The default value is zero. This option is available with MCP release 7.0 SSP1 and later.

SO_SendLoWat (h1003)

Select send low water mark. This option sets the number of bytes that must be able to be written before a Select terminates its wait for a socket in the WriteSet. This option is allowed for compatibility, but is ignored since it has no relevance to the socket's buffering mechanism. The default value is 2048 bytes. This option is available with MCP release 8.0 and later.

SO_SndTimeO (h1005)

Send timeout. This option sets the amount of time that a send will block waiting to be taken out of flow control. The value is a six byte real specifying the number of seconds to wait. Setting the value to zero disables the timeout function. Note that if a send times out, some of the data may have been sent, but not all of it. The default value is zero. This option is available with MCP release 7.0 SSP1 and later.

SO_Type (h1008)

This option can be used with SockLib_GetSockOpt only. It is used to get the socket's communication type. The value is an integer and its value designates a communication type.

SO_SocketState (h00FF)

This option is only valid for SockLib_GetSockOpt().

This option returns the state of the socket. The enumerations of the state (which are described in Section 1) are:

SCBS_Bound =	h0001
SCBS_Declared =	h0002
SCBS_Establishing =	h0003
SCBS_Listening =	h0004
SCBS_NotASocket =	h0005
SCBS_Open =	h0006
SCBS_Shutdown =	h0007
SCBS_ShutdownComplete =	h0008
SCBS_Closed =	h0009
SCBS_Suspended =	h000A
SCBS_Resumed =	h000B

IP_Priority (h0001)

This option sets the IP priority field for IP headers. Accepted values are the numbers 0, 1, and 2.

Note: *SOL_SSL options are used to control a secure socket (a given socket can be SSL or non-SSL, but not both). A socket becomes an SSL socket when any SOL_SSL option is set through SockLib_SetSockOpt or interrogated through SockLib_GetSockOpt. SSL options must be set prior to calling SockLib_Listen, as it is the listen that initiates the opening of sockets. The active security attributes are established at that time.*

The only required call to SockLib_SetSockOpt for an SSL socket is to set the SSL_Key_Container, which establishes a key that has been enabled by the system administrator for the usercode that the socket was created under. All the other SSL options have default values that you can use.

SSL_System_Options (h0001)

This option is only valid for SockLib_GetSockOpt (). It returns a bit map indicating which SSL options are available for use through the sockets interface.

Currently no options are supported.

SSL_Socket_Options (h0002)

This option is used to set or get the SSL options for a socket. The OptVal parameter is a bit map indicating the selected options.

By default, no options are set.

SSL_System_Versions (h0003)

This option is only valid for SockLib_GetSockOpt (). It returns a list of which protocol versions are available for use through the sockets interface. OptVal has a count in the first 6 bytes (one word) followed by a list of protocol versions. Each version is 6 bytes (one word).

SSL_Socket_Version (h0004)

This option is used to set or to get the protocol version for a socket. The version field is 6 bytes. The values are:

SSL 3.0	=	Major version	=	3
	=	Minor version	=	0
TLS	=	Major version	=	3
	=	Minor version	=	1

Note: *As of MCP release 8.0, the default version is TLS (h000031). Previous releases default to SSL 3.0 (h000030).*

SSL_System_Ciphers (h0005)

This option is only valid for SockLib_GetSockOpt (). It returns a list of which cipher suites are available for a specified protocol. It is formatted as a 6-byte (one word) count followed by a list of 2-byte cipher ids.

SSL_Socket_Ciphers (h0006)

This option is used to set the allowed cipher suites (and their order) to be used to negotiate an SSL handshake before the socket is opened, or to get the negotiated cipher suite after the socket is opened. OptVal is formatted as in SSL System Ciphers.

The following cipher suites are supported (the default is that all installed cipher suites are available to the user):

V3 Cipher	Value
SSL_RSA_With_NULL_MD5	h0001
SSL_RSA_With_NULL_SHA	h0002
SSL_RSA_Export_With_RC4_40_MD5	h0003
SSL_RSA_With_RC4_128_MD5	h0004
SSL_RSA_With_RC4_128_SHA	h0005
SSL_RSA_Export_With_RC2_CBC_40_MD5	h0006
SSL_RSA_Export_With_DES_40_CBC_SHA	h0008
SSL_RSA_With_DES_CBC_SHA	h0009
SSL_RSA_With_3DES_EDE_CBC_SHA	h000a
TLS_RSA_Export1024_With_DES_CBC_SHA	h0062
TLS_RSA_Export1024_With_RC4_56_SHA	h0064

SSL_Key_Container (h0007)

This option is only valid for SockLib_SetSockOpt (). It is used to set the key container associated with the socket. This is the only option that is required to be set in order to use an SSL connection. OptVal specifies a string, which is the key container name.

SSL_Resumable (h0008)

This option is set to TRUE (6 bytes with the least bit set) if the SSL session is resumable. By default the session is not resumable. For SetSockOpt, any nonzero value will set the option to TRUE. For GetSockOpt, a zero or one is returned.

OptVal

The value of the specified option. If the option is a Boolean type, any nonzero value is interpreted as TRUE.

OptLen

The length in bytes of the specified option value.

Output Parameters:

None.

Results:

If the query is successful, the result is 0; if an error occurred, a negative error code is returned:

EInVal

The Level parameter and/or the OptVal parameter are invalid.

ENotSock

The socket id specified is not valid.

ENoProtoOpt

The option is unknown or unsupported.

ENotGettable

The option specified is not allowed for GetSockOpt.

ENotSettable

The option specified is not allowed for SetSockOpt.

EIsConn

The socket is not in a valid state for setting options.

ENetReset

TCP/IP is no longer available.

ESSLAttributeNotAvail

The socket has not been configured for SSL.

ESSLSessionInterrupted

The cryptography engine aborted the operation.

ESSLNotLinked

Support for SSL has terminated.

ESSLInvalidAttributeValue

The value in OptVal is invalid.

ESSLCryptoNotAvailable

The cryptography interface is not available.

ESSLSysErr

The MCP Crypto API encountered an error.

ESSLSecurityViolation

The application is not allowed to use the specific key container.

SockLib_Shutdown

Declaration in C:

```
int shutdown(int, int);
```

Declaration in ALGOL/NEWP:

```
INTEGER PROCEDURE SOCKLIB_SHUTDOWN (SOCKET, HOW);

VALUE SOCKET, HOW;

INTEGER SOCKET, HOW;
```

Access in COBOL:

```
CALL "SOCKLIB_SHUTDOWN_CBL OF SOCKETSUPPORT"

USING SOCKET, SOCKET-HOW

GIVING SOCKET-RESULT.
```

Input Parameters:

Socket

The identifier of the socket returned by the SockLib_Socket or SockLib_Accept call.

How

Specifies whether to disable sending, receiving, or both on the socket.

SD_Send (1)

Sending is disabled for the socket.

SD_Receive (0)

Receiving is disabled for the socket.

SD_Both (2)

Sending and receiving are disabled for the socket.

Output Parameters:

None.

Results:

If the operation was successful, 0 is returned; otherwise, a negative error code is returned.

EInval

The How parameter is not valid.

ENotSock

The socket id specified is not valid.

ENetReset

TCP/IP is no longer available.

Conditions:

For sockets of type Sock_Stream:

If How = SD_Send, SockLib_Send () operations are no longer allowed but SockLib_Recv () operations are. When a SockLib_Recv () returns zero data, a SockLib_Close () should be done on the socket.

If How = SD_Receive, SockLib_Recv () operations are no longer allowed. If any data is queued when SockLib_Shutdown is called, or if any data is received after it is called, the data is discarded and the socket is closed.

If How = SD_Both, the socket is closed and any queued data is discarded.

For sockets of type Sock_DGram:

If How = SD_Receive, there is no effect on the use of SockLib_Send () and SockLib_Recv () operations.

How = SD_Send and How = SD_Both are equivalent. SockLib_Send () operations are no longer allowed but SockLib_Recv () operations are.

SockLib_Socket

Declaration in C:

int socket(int, int, int);

Declaration in ALGOL/NEWP:

```
INTEGER PROCEDURE SOCKLIB_SOCKET (ADDRFORMAT, TYPE, PROTOCOL);

VALUE ADDRFORMAT, TYPE, PROTOCOL;

INTEGER ADDRFORMAT, TYPE, PROTOCOL;
```

Access in COBOL:

CALL "SOCKLIB_SOCKET_CBL OF SOCKETSUPPORT"

USING SOCKET-ADDRFORMAT, SOCKET-TYPE, SOCKET-PROTOCOL

GIVING SOCKET-RESULT.

Input Parameters:

AddrFormat

The address format. Only PFInet is supported.

Type

The type of service to be used:

Sock_Stream (1)

Stream sockets use the TCP interface.

Sock_DGram (2)

Datagram sockets use the UDP interface.

Protocol

The protocol to be used. Only IPProto_IP (0) is supported.

Output Parameters:

None.

Results:

If positive, the procedure result is the identifier of the newly created socket. If negative, it is an error. Zero is not a valid result for this procedure.

ENoBufs

The socket cannot be created due to lack of system resources.

EProtoNoSupport

The type or protocol specified is not supported.

ENetReset

TCP/IP is no longer available.

Conditions:

A socket data structure is allocated and its id is returned.

SockLib_Suspend

Declaration in C:

```
int suspend(int);
```

Declaration in ALGOL/NEWP:

```
INTEGER PROCEDURE SOCKLIB_SUSPEND(SOCKET);

VALUE SOCKET;

INTEGER SOCKET;
```

Access in COBOL:

```
INTEGER PROCEDURE SOCKLIB_SUSPEND(SOCKET);

VALUE SOCKET;

INTEGER SOCKET;
```

Input Parameters:

Socket

The identifier of the socket returned by the SockLib_Accept call.

Output Parameters:

None.

Results:

If the SockLib_Suspend is successful, the result is 0; if an error occurred, a negative error code is returned.

ENotSock

The socket id specified is not valid.

ENetReset

TCP/IP is no longer available.

ESSLSessionSuspended

The SSL connection is currently in the suspended state.

ESSLInvalidState

The SSL connection is not opened, or SSL has been disabled on the system.

ESSLSessionInterrupted

The cryptography engine aborted the operation.

ESSLNotLinked

Support for SSL has terminated.

ESSLSysErr

The MCP Crypto API encountered an error.

ESSLSessionNotResumable

The session has not been configured as resumable.

Conditions:

If SO_Linger is not set or is set with a zero timeout, the socket is suspended immediately and any pending data is discarded.

If SO_Linger is set with a nonzero timeout, the socket waits for all sent data to be acknowledged by an "ACK" message until the timeout expires. When all data has been acknowledged by an "ACK" message, or after the timeout has expired, the socket is suspended.

43103530-006