
smxNet™

smxNet is a TCP/IP network stack, which has been optimized for use in real-time embedded systems. It offers RFC-compliant TCP and conforms to the Berkeley sockets API. smxNet is fully integrated with smx. Several higher-level protocols are also available and described herein.

smxNet works well for small ROM'ed hosts, as well as larger hosts. No disk services are required. It can configure itself after power up using BOOTP. ROM requirements are small and configurable to application requirements — see table below. RAM requirements are tunable to the application and vary from about 35KB (PPP) or 60KB (Ethernet) to about 150KB (including optional packages) for x86. Use of smxNet with 16-bit processors and small memory is feasible. A no copy operating mode improves UDP and TCP performance. smxNet provides extensive error monitoring and reporting.

smxNet ROM Requirements (KB)

Function	x86 rm	PowerPC
1. IP + ARP + UDP	37.4 ¹	48 ²
2. IP + ARP + TCP/UDP	52.3 ¹	73 ²
3. DNS	7.3	5
4. Fragmentation	4.2	4
5. BOOTP	2.6	2
6. ICMP	1.0	1
7. FTP Client	5.7	15
8. FTP Server	15.9	22
9. FTP/TFTP/TELNET ³	29.0	26
10. NFS Client ³	30.1	
11. NFS Server ³	35.2	
12. SNMP V2	35.7	40
13. DHCP Client	8.6	9
14. DHCP Server	6.3	6
15. DHCP Client or Server	14.9	14
16. MicroWeb Server	12.4	23
17. SMTP	11.8	16
18. POP3	7.1	17
19. Drivers		
a. PPP + CHAP	38.3	41
b. PPP (no CHAP)	28.5	35
c. SLIP + CSLIP + MODEM	13.4	10
d. Ethernet	1.3 – 6.0	3

Notes:

1. Includes NE2000 Ethernet Driver
2. Includes 860 Ethernet Driver
3. FTP, TFTP, and NFS require file system support. A virtual file system is included in smxNet. Alternatively, smxFile may be used.

Simultaneous communications over multiple interfaces are supported. Several drivers are included in the package. A simple, table-driven, device driver interface is provided for custom drivers. *smxNet* includes the following protocols: IP, UDP, TCP, ARP, RARP, BOOTP, and ICMP.

smxNet provides a total of 66 services. (See table on pgs 3 & 4.) These are grouped into six categories as follows:

Basic System Services

These are a subset of the Berkeley Sockets API. They initialize *smxNet*, manage interfaces, and perform control functions.

Socket Services

These are the standard Berkeley Socket Services. They permit creating, binding, and connecting sockets, sending and receiving data, and other socket functions. (A *socket* is the combination of an *IP address* and a *port number* at the host.)

High-Speed Services

Nine high-speed UDP and TCP services are provided. These utilize *smx* message facilities to pass messages from layer to layer of the stack without copying from buffer to buffer (as is done by the standard socket services such as *recv()* and *send()*.) This significantly reduces processor loading and improves throughput.

DNS Services

Domain Name System services permit getting IP addresses and domain name information from a domain name server. (An example of a domain name is *www.smxinfo.com*)

Miscellaneous Services

These include getting protocol and service information and sending error information to the application.

Modem Control

Modem control for SLIP, CSLIP, and PPP.

smxNet Services

Basic System Services

xn_abort	Abort a TCP socket and sever any connections
xn_attach	Open an interface for serial communication
xn_bootp	Call a BOOTP server to get this node's IP address
xn_interface_close	Close an interface
xn_interface_info	Return interface information
xn_interface_open	Open an interface for Ethernet or loopback
xn_interface_set_multicast	Install a receive multicast address list in an interface
xn_ping	Ping a host and wait for it to reply
xn_rarp	Call a RARP server to get this node's IP address
xn_rt_add	Add a route to the routing table
xn_rt_del	Delete a route from the routing table
xn_rtip_exit	Perform cleanup before exiting
xn_rtip_init	Initialize TCP/IP run-time environment
xn_rtip_restart	Restart TCPIP runtime environment
xn_set_ip	Set the IP address and IP network mask for an interface
xn_tcp_is_connect	Checks state of socket
xn_tcp_is_read	Checks state of socket
xn_tcp_is_write	Checks state of socket

High Speed UDP/TCP Services

xn_pkt_alloc	Allocate a message buffer for the high-speed interface
xn_pkt_data_max	Maximum number of bytes transferable per packet
xn_pkt_data_pointer	Returns the address of the user area in a packet
xn_pkt_data_size	Return count of valid user bytes in a received packet
xn_pkt_free	Free a message buffer for the high-speed interface
xn_pkt_recv	Wait for next UDP packet at a port
xn_pkt_recv_from	Wait for next UDP packet at a port, and retrieve host address
xn_pkt_send	Send UDP data directly to a port (unbuffered)
xn_pkt_send_to	Send UDP data directly to a host

Domain Name Server Services

xn_add_host_table_entry	Add entry to global host table
xn_clear_host_cache	Reinitialize the internal host cache
xn_delete_host_table_entry	Delete entry from global host table
xn_set_server_list	Set global name server list
gethostbyname	Takes name string and return IP address
gethostbyaddr	Take IP address and return name string

***smxNet* Services (Cont.)**

Socket Services

accept	Accept a connection from any remote host (TCP only)
bind	Bind a socket to a local port number and IP address
closesocket	Close a socket and sever any connections
connect	Establish a connection to a remote host
getpeername	Get port # and IP address of remote host connected to a socket
getsockname	Get port number and IP address of a local host
getsockopt	Get value of a socket option
htonl	Convert unsigned long from host to network byte order
htons	Convert unsigned short from host to network byte order
ioctlsocket	Turn I/O blocking on or off
listen	Establish a backlog queue for a socket (TCP only)
ntohl	Convert unsigned long from network to host byte order
htohs	Convert unsigned short from network to host byte order
recv	Receive data at a socket
recvfrom	Receive data at a socket and retrieve host address
select	Block until status (read or write ready or error status)
send	Send data to a socket
sendto	Send data to a host
setsockopt	Set a socket option
shutdown	Disable receives and/or sends on a socket
socket	Allocate a UDP or TCP socket port

Miscellaneous Services

getprotobyname	Get protocol info from protocol name
getprtbobyname	Get protocol info from protocol number
getservbyname	Get service info from server name and protocol name
getservbyport	Get service info from server port and protocol name
aos_out_two	Print status message for FTP
aos_wr_screen_string	Write debug message to screen
ERROR_REPORT	Inform the application of an error
raw_mode_in_char	Processes an input char while in raw mode

Modem Control

xn_autoanswer	Wait for remote login via modem and answer.
xn_autologin	Automate login to remote server over Hayes-compatible modem
xn_hangup	Hang up modem
xn_scriptlogin	Login to remote server using script

***smxNet* Included Protocols**

ARP *Address Resolution Protocol* dynamically binds a high-level IP address to a low-level physical hardware address (e.g. an Ethernet address). This is called an *address mapping*. A local ARP cache is maintained by *smxNet*. If a needed address mapping is not found in the cache, an ARP request is sent out on the network. The ARP reply (returned by the network) is used to update the cache. Information received in ARP requests from other hosts is also used to update the cache.

RARP *Reverse Address Resolution Protocol* allows a host to obtain its own IP address from a RARP server. This is done by sending the host's hardware address. This is useful for ROM'ed hosts. *smxNet* includes both a RARP client and a RARP server.

BOOTP *Bootstrap Protocol* is a better way than RARP to initialize a ROM'ed host. Data returned from the BOOTP server is used to set the host's IP address and network IP mask. Other data may also be returned such as the address of the nearest IP gateway and the address of an available file server.

DNS *Domain Name Server Protocol* obtains IP addresses and names from a domain name server.

ICMP *Internet Control Message Protocol* allows gateways or hosts to send error or control messages to other gateways or hosts. *smxNet* allows initiating ping requests and echoing received ping requests. It also supports standard ICMP functions such as source quench, redirect, etc.

IGMP *Internet Group Management Protocol* supports IP multicast addressing. *smxNet* allows hosts to inform routers which multicast addresses they are listening to.

RIP *Routing Information Protocol* allows sharing route information between directly connected gateways and hosts. *smxNet* supports both versions 1 and 2 of RIP.

SLIP *Serial Line Internet Protocol* permits a host to connect to a TCP/IP network via a telephone line. Includes UART driver and modem control.

CSLIP is SLIP with Van Jacobson compression.

Development Kit

The *smxNet* development kit includes full source code, a comprehensive demo, a simplified demo that *smxNet* works with the *smx* integrated product platform, and a reference manual. *smxNet* supports the same tools as *smx*. It is available in real-mode, 16-bit protected mode, and 32-bit protected mode for x86 processors and also supports all ARM, ColdFire, PowerPC, and SH processors. It may be easily ported to other 16 and 32-bit processors. *smxNet* is sold royalty-free per developed product. One development kit may be shared by all programmers on a single project.

smxNet Drivers

The following drivers are included with *smxNet*:

<u>ISA Cards</u>	
3Com:	3C509 10BaseT Ethernet cards
Intel:	Ether Express Pro Card ISA 10BaseT
Novell:	NE2000-compatible board driver 10BaseT
<u>PCMCIA Cards</u>	
3Com:	Ethercard Plus
Linksys:	Linksys and other NE2000-compatible cards
Ositech:	4 of Diamonds
Xircom:	Realport 10/100BaseT
<u>Bus Mastering PCI Cards</u>	
3Com:	3C905C Etherlink 10/100BaseT
Accton:	Cheetah 10/100BaseT (Realtek 8139)
Intel:	82559-based cards 10/100BaseT
<u>Single Chip Solutions</u>	
<u>ISA</u>	
AMD:	Lance chips 10BaseT Bus Mastering and Shared Memory PCNet-ISA (79C960), PCNET-ISA+ (79C961)
Crystal Semiconductor:	CS8900A and CS8920
National Semiconductor:	8390X and compatibles (Gavicom DM98008, UMC 9008)
SMSC:	SMC91C92/4/6 10BaseT, SMC91C100 10/100BaseT
<u>PCI</u>	
AMD:	10BaseT PCNet-PCI (79C970) 10/100BaseT PCNet-PCI (79C972,73,75)
Intel:	82559ER 10/100BaseT
National Semiconductor:	83815 10/100BaseT
Realtek:	8139 10/100BaseT
<u>System on a Chip Solutions</u>	
Motorola	68EN360 QUICC Ethernet, MPC860 QUICC Ethernet, ColdFire 5272
VAutomation	MAC Core
<u>Asynchronous Device Support</u>	
SLIP and Compressed SLIP	
PPP with compression and CHAP/PAP authentication	
Dial-up modem support answer and initiate modes	
16550/8250/186ES UART	
Motorola:	68EN360 QUICC UART MPC860 QUICC UART

The simple device driver interface for *smxNet* permits adding new device drivers, easily. The device table entry for each driver has four entry points: *open*, *close*, *send*, and *statistics*. Standard services are provided to handle received packets from interrupt service routines. *Please contact us for custom driver quotations.*

OPTIONAL PACKAGES*

TFTP**

TFTP *Trivial File Transfer Protocol* is a simplified file transfer protocol which supports only reading and writing files. It consists of three client services and one server service.

tftpcli_connect	Connect to a TFTP server
tftpcli_recvfile	Retrieve a file from a TFTP server
tftpcli_sendfile	Send a file to a TFTP server
tftp_server_daemon	Start a TFTP server main program

FTP**

FTP *File Transfer Protocol* is a more extensive file transfer protocol. FTP client consists of 10 services. (See the *ftpcli* services, below.) An FTP client demo is included. The FTP server supports 18 commands from clients. If running with *smx*, multiple sessions can be supported by the FTP server.

ftpcli_connect	Connect to an FTP server
ftpcli_cwd	Change the current working directory
ftpcli_dir	Perform a LIST or NLST command
ftpcli_mkdir	Create a directory
ftpcli_pwd	Retrieve the current working directory
ftpcli_quit	Close a connection
ftpcli_rmdir	Remove a directory
ftpcli_type	Set the transfer type to ASCII or Binary
ftpcli_unlink	Remove a file
ftpcli_xfer	Perform GET or PUT command
ftpsrv_main	Start an FTP server

FTP Server Commands Requested by a Client

CWD/CDUP/XCWD	Change a directory
DELE	Delete a file
HELP	Display help
LIST	Detailed directory list
MKD/XMKD	Create a directory
MODE	Select mode (ignored)
NLST	Directory list files only
NOOP	No operation
PASS	Password
PASV	Set server to passive mode
PORT	Reassign data port
PWD/XPWD	Query working directory
QUIT	Quit the session
RETR	Upload a file
RMD/XRMD	Delete a directory
SIZE	Return file size
STOR	Download a file
SYST	System type
TYPE	Set file transfer type
USER	Log in with user name

* All optional packages require *smxNet*.

** *smxFile*, DOS, or the *smxNet* Virtual File System is supported for file servers.

Telnet

Telnet The Telnet client permits connecting to a Telnet server in another host and transferring characters to or from it. The Telnet server can process multiple connections in an *smx* environment.

telnet_client	Start a Telnet client
telnet_server_daemon	Start a Telnet server
telnet_server_int	Initialize Telnet server daemon
telnet_kill_server_daemon	Kill Telnet server daemon
telnet_server_process_one	Process one Telnet server session

NFS

NFS *Network File System* client provides a comprehensive API for accessing and manipulating a file system on a remote host. From the user's perspective, NFS client is almost invisible. He can use arbitrary files for input and output. The file names, themselves, do not show whether the files are local or remote.

The NFS client API is reentrant, so multiple tasks may access a remote file system simultaneously. Drives a: through *: (user specified) are reserved for *smxFile*. Drives above *: are reserved for NFS. The NFS client API provides 30 calls such as file open, read and write, directory create and delete, disk open and close, etc. NFS server performs client commands. It can be configured to use *smxFile*, DOS, or the *smxNet* Virtual File System.

PPP

PPP *Point to Point Protocol* permits TCP/IP network access via a serial link. It is like SLIP (which is included with *smxNet*). However, PPP offers more features and better throughput. It can be used to interface with an ISP (Internet Service Provider) via a modem. PPP has the ability to negotiate several parameters such as compression and security with another PPP host. Also, it provides better security than SLIP. Both client and server modes are supported and memory usage is configurable.

In opening a PPP connection, PPP progresses through five phases: It is initially in the DEAD phase and remains there until an *open LCP (Link Control Protocol)* command is performed. PPP then transitions to the LCP phase. During the LCP phase, LCP options are negotiated. After this, if an authentication protocol was negotiated, PPP transitions to the AUTHENTICATION phase. If no authentication was negotiated, or after authentication has been performed, PPP transitions to the READY phase where IPCP (IP Control Protocol) options are negotiated. After this, the connection is OPEN and PPP IP packets may be transferred over the interface. PPP returns to the DEAD phase after a close command is performed. The PPP API provides 20 calls to open or close a PPP connection and to set negotiation parameters.

PPP supports two types of authentication: PAP (Password Authentication Protocol) and CHAP (Challenge-Handshake Authentication Protocol). Either, or both, may be configured. PPP includes a UART and modem control. PPP is compliant with RFC's 1661, 1172, 1332 & 4.

DHCP

DHCP *Dynamic Host Configuration Protocol*, is a client server protocol for obtaining network parameters needed by a host. It is an extension of the BOOTP protocol, which allows clients to obtain network configuration parameters and an IP address. DHCP adds the capability to associate with an IP address a “lease”, which specifies the amount of time a client is entitled to use the IP address before it becomes invalid. The specification for DHCP is set forth in RFC 1541, with additional information in RFC’s 1533 and 1534.

SNMP Agent v1, v2 and v3

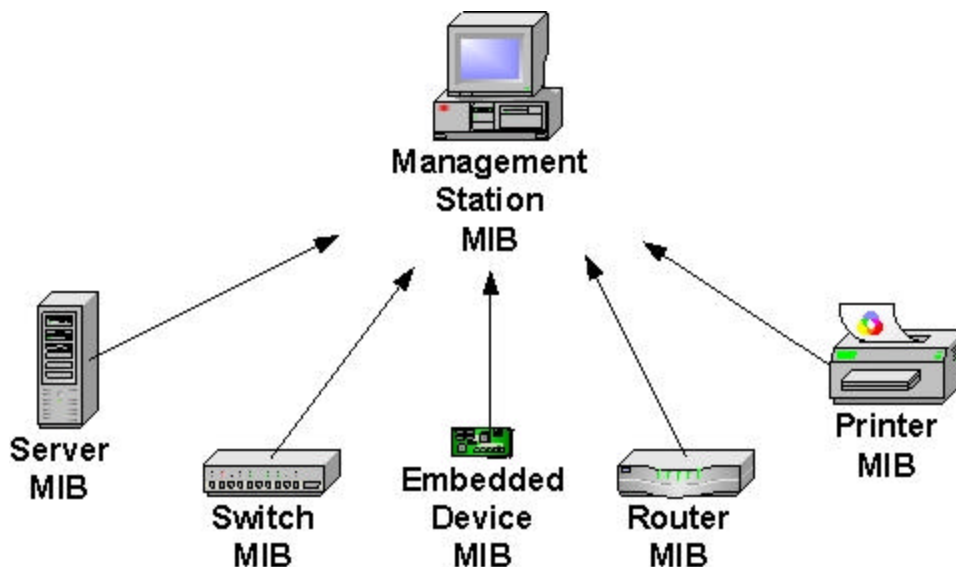
SNMP, *Simple Network Management Protocol*, provides for communication of information from *managed devices* to a centralized *management station*, as shown in the diagram. It supports communication in the form of *queries* (information requested by the management station) or *traps* (information provided by the managed device in response to pre-defined events). At the management station, software like *HPOpenView* (not included) requests, accepts, deciphers, and displays the information. SNMP can be used to monitor and control embedded devices, as well as network devices.

SNMP v1 agent accepts the following commands from the management station:

GetRequest	Retrieve an <i>object</i>
GetNextRequest	Retrieve the next <i>object</i>
SetRequest	Set an <i>object</i> to the specified value

It sends the following back:

Response	Send value in response to a received command
Trap	Send value due to occurrence of a pre-defined event



An *object*, also known as a *managed object*, may be a variable or a table in the device. Objects are defined by a MIB (Management Information Base). The management station and the device have equivalent copies

of the MIB. The MIB in the management information station allows identifying objects for processing of information sent by the device either by *responses* or by *traps*. The MIB in the device relates subroutine addresses to objects in order to read or write data from or to variables or tables in the device.

SNMP v1 agent comes with all standard network objects defined. These comply with the MIB-II standard. In order to add device-specific objects to the MIB, it is necessary to use a MIB compiler to produce the management station MIB and to define the device MIB using C statements. Examples are provided doing this. SNMP v1 agent complies with RFC's: 1155, 1157, 1213, 1215, 1317, 1650, and 2089.

SNMP v2 agent supports both SNMPv2c (RFC's 1902-08) and SNMPv2u (RFC's 1909&10). The first adds the following command from the management station:

GetBulkRequest Retrieve a large number of objects

The second provides for message authentication and privacy using a user-based model. It is recommended only for secure (i.e. private) networks.

SNMP v3 agent adds strong security features including encryption. These features are adequate for use over an insecure network, like the Internet. This is done through the use of a shared password key which is used to encrypt the data. With SNMPv3, secure device monitoring over the Internet is possible. SNMP v3 agent complies with: RFC's 2570-75. It adds the following command back to the management station:

Report Synchronizes with a remote management station

MicroWeb Server

MicroWeb Server is an embedded web server for use with *smxNet*. This server relies on a set of user-defined web pages (written in HTML language) to provide HTML network output. It communicates with a remote web browser using the HTTP protocol.

The following HTTP requests from a remote WEB browser are supported:

GET: Get a file (i.e. a web page)
POST: Execute a POST function
PUT: Put a file

smxFile is used to retrieve web pages which are requested by GET commands from remote web browsers. If *smxFile* is not available, virtual files may be used, as described below.

Virtual File System

A virtual file system is provided to obtain and save web pages when a *smxFile* is not present. When a GET command from a web browser is received for a web page, if the file name is defined in the virtual file table, it will be read from the virtual file system (i.e. the character array will be returned). No HTML compression is supported.

Server Side Includes (CGI)

Keywords can be included in a web page which, when the web page is requested, will cause the web server to execute the functions specified by the keywords. These functions can send data directly to the remote browser as part of the requested document. Using this feature, realtime data, such as the current time or temperature from a sensor, can be included directly into a transmitted HTML document.

Form Action Posts

A *form action post* consists of a keyword in the web page which instructs the remote browser to send a POST command back to the specified IP address (usually this web server's address). The command sent back is based upon input on the browser side. Upon receipt of the POST command, the web server will call a function which parses it and performs the desired action. This feature is useful if there are commands or specific settings that need to be sent to the server from the browser.

Server Push

Server push provides a means for MicroWeb server to continuously send data to a client without needing explicit refresh requests from the client. Server push relies on a variant of the MIME message format called "multipart/mixed" or "multipart/mixed-replace". With server push a *MicroWeb* Server can keep the HTTP connection open indefinitely. Server push can be used for animation and dynamic updating of field data.

Security

Two methods for security are provided: (1) Authentication based upon the IP address of requestor, and (2) authentication based upon user name and password (RFC1945) are supported.

Java

Java is useful for developing interpreted "applets" that may be downloaded from a web server to a web browser and executed locally by the browser. Java is useful for developing graphic, windowed and network-enabled applications.

Some examples of how Java might be used:

1. A device running *smxNet* and *MicroWeb* Server downloads a Java applet to a browser and then continuously sends data such as temperature to the applet. The applet graphically displays the data as a temperature gauge.
2. A device running *smxNet* and *MicroWeb* Server downloads a Java applet to a browser. The browser then displays controls such as sliders and radio buttons which may be used to control hardware managed by the embedded system running *MicroWeb* Server.

Java can be easily used with *MicroWeb* Server. To do so, simply develop the applet and reference it in the appropriate MicroWeb HTML page via the <applet> tag. The applet may reside on disk or may be compiled into a virtual file.

SSL

SSL *Secure Sockets Layer* provides secure point-to-point communications for application protocols such as http or ftp. SSL takes advantage of public or private Public Key Infrastructure and Certificate Authority to secure specified protocols at the application layer. *smxNet* supports SSLv2, SSLv3, and TLSv1.

SMTP Client

SMTP *Simple Mail Transfer Protocol* allows an SMTP client to send an email message to an SMTP server. The client connects to TCP port 25 (the SMTP port) and sends a greeting of the form “*HELO domain*”, where *domain* is the domain name of the client. The server response consists of a result code followed by any optional text and ending with a <CRLF> pair. The client then uses the “*MAIL FROM:*” command to specify the author of the message, and the “*RCPT TO:*” command to specify the recipients. Note that if the message has multiple recipients, this command may be used more than once for a single message. The client then sends the *DATA* command, which informs the server that what follows is the mail message. The client then sends the message, terminated by “<CRLF> <CRLF>”. The *QUIT* command is used to close the connection.

POP3 Client

POP3, *Post Office Protocol* allows a POP client to retrieve email messages from a POP server. The POP client establishes a TCP connection to TCP port 110 (the POP port) and the POP server sends back a greeting. The POP Client then sends login information via the *USER* and *PASS* commands. The POP server responds starting with “+OK” if the login was valid or “-ERR” if the login was invalid. The POP Client may then send the *STAT* command, in response to which the POP server will send a reply of the form “+OK *n m*” where the *n* is the number of messages and *m* is the size of the mail file. The POP Client may then send the *LIST* command, which lists the size of each mail message individually. Following this, commands of the form “*RETR n*” and “*DELE n*” may be used to retrieve and delete messages, respectively, where *n* is the number of the message.