KwikNet®

SMTP Client / Server

User's Guide

Version 3

First Printing: **October 1, 2003**
Last Printing: **September 15, 2005**

Manual Order Number: PN303-9E

KADAK Products Ltd.
206 - 1847 West Broadway Avenue
Vancouver, BC, Canada, V6J 1Y5
Phone: (604) 734-2796
Fax: (604) 734-8114

TECHNICAL SUPPORT

KADAK Products Ltd. is committed to technical support for its software products. Our programs are designed to be easily incorporated in your systems and every effort has been made to eliminate errors.

Engineering Change Notices (ECNs) are provided periodically to repair faults or to improve performance. You will automatically receive these updates during the product's initial support period. For technical support beyond the initial period, you must purchase a Technical Support Subscription. Contact KADAK for details. Please keep us informed of the primary user in your company to whom update notices and other pertinent information should be directed.

Should you require direct technical assistance in your use of this KADAK software product, engineering support is available by telephone, fax or e-mail. KADAK reserves the right to charge for technical support services which it deems to be beyond the normal scope of technical support.

We would be pleased to receive your comments and suggestions concerning this product and its documentation. Your feedback helps in the continuing product evolution.

KADAK Products Ltd.
206 - 1847 West Broadway Avenue
Vancouver, BC, Canada, V6J 1Y5

Phone: (604) 734-2796
Fax: (604) 734-8114
e-mail: amxtech@kadak.com

DISCLAIMER

KADAK Products Ltd. makes no representations or warranties with respect to the contents hereof and specifically disclaims any implied warranties of merchantability and fitness for any particular purpose. Further, KADAK Products Ltd. reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of KADAK Products Ltd. to notify any person of such revision or changes.

TRADEMARKS

AMX in the stylized form and KwikNet are registered trademarks of KADAK Products Ltd.
AMX, AMX/FS, InSight, *KwikLook* and KwikPeg are trademarks of KADAK Products Ltd.
UNIX is a registered trademark of AT&T Bell Laboratories.
Microsoft, MS-DOS and Windows are registered trademarks of Microsoft Corporation.
All other trademarked names are the property of their respective owners.

KwikNet SMTP Client / Server User's Guide

Table of Contents

KwikNet SMTP Client / Server User's Guide
Table of Contents (continued)

KwikNet SMTP Client / Server User's Guide
Table of Figures

1. KwikNet SMTP Overview

1.1 Introduction

The Simple Mail Transfer Protocol (SMTP) is a standard protocol for transferring mail, simple text messages, between networked machines. The KwikNet SMTP option implements this protocol using the TCP services provided by the KwikNet® TCP/IP Stack. KwikNet is a compact, reliable, high performance TCP/IP stack, well suited for use in embedded networking applications.

SMTP provides simple mail transfers over a direct point-to-point connection between two network nodes. The reliability of the connection is determined by the network protocol used to establish the connection. The KwikNet SMTP option uses a TCP connection to establish a highly reliable link for its SMTP mail transfers.

A network computer which incorporates the KwikNet SMTP option can act as a client to send a message to any mailbox to which it has network access. By acting as a dedicated SMTP server, the same network computer can accept a message directed to any of its local mailboxes. In this way, embedded devices can use KwikNet SMTP services to send status and alarm updates and to receive configuration and operational parameters.

The KwikNet SMTP option is best used with a real-time operating system (RTOS) such as KADAK's AMX™ Real-Time Multitasking Kernel. However, the KwikNet SMTP option can also be used in a single threaded environment without an RTOS. The KwikNet Porting Kit User's Guide describes the use of KwikNet with your choice of RT/OS. Note that throughout this manual, the term RT/OS is used to refer to any operating system, be it a multitasking RTOS or a single threaded OS.

You can readily tailor the KwikNet stack to accommodate your SMTP needs by using the KwikNet Configuration Builder, a Windows® utility which makes configuring KwikNet a snap. Your KwikNet stack will only include the SMTP features required by your application.

This manual makes no attempt to describe the Simple Mail Transfer Protocol (SMTP), what it is or how it operates. It is assumed that you have a working knowledge of the SMTP protocol as it applies to your needs. Reference materials are provided in Appendix A of the KwikNet TCP/IP Stack User's Guide.

The purpose of this manual is to provide the system designer and applications programmer with the information required to properly configure and implement a networking system using the KwikNet TCP/IP Stack and SMTP. It is assumed that you are familiar with the architecture of the target processor.

KwikNet and its options are available in C source format to ensure that regardless of your development environment, your ability to use and support KwikNet is uninhibited. The source program may also include code fragments programmed in the assembly language of the target processor to improve execution speed.

The C programming language, commonly used in real-time systems, is used throughout this manual to illustrate the features of KwikNet and its SMTP option.

1.2 General Operation

The SMTP protocol is formally defined by the IETF document STD-10 which was derived from RFC-821. The KwikNet SMTP option is compliant with a subset of that specification as recommended by the more recent RFC-2821, the currently proposed standard for SMTP. RFC-2821 should be consulted for any detailed questions concerning the SMTP protocol. The KwikNet SMTP option implements the subset of SMTP features typically required for use in embedded applications.

SMTP is a peer-to-peer protocol. A mail transfer is initiated between two peers in client-server fashion. One machine, the client, starts a mail transfer session by establishing a connection with another machine, the server. The client initiates the transfer and then identifies itself and one or more mail recipients. If the server grants the request, the mail transfer proceeds with the client assuming the role of sender and the server assuming the role of receiver. Once the client has completed the transmission of its message, it terminates the mail transfer. The client can then initiate another mail transfer if required. When the final mail transfer has been completed, the client severs the connection with the server.

While a mail transfer is being initiated, the SMTP client can end the session prematurely and terminate the connection. However, once the mail data transfer has been started, the client can only abort the transfer by breaking the connection, contrary to the spirit of the SMTP specification.

While a mail transfer is being initiated, the SMTP server can respond with error indications that may force the client to end the session and terminate the connection. However, once the mail data transfer has been started, the server must wait for the transfer to complete before it can respond with an error reply rejecting the mail message. Thus, a server can only abort the data transfer by breaking the connection, contrary to the spirit of the SMTP specification.

The KwikNet SMTP option provides all of the services necessary to implement one or more SMTP clients and servers. Although multiple clients and multiple servers can coexist and operate concurrently, most applications will require only a single SMTP client and/or a single SMTP server.

> Note
>
> The KwikNet SMTP server is **not suitable** for use as an SMTP relay agent or gateway. It simply permits an embedded device to implement one or more mailboxes in which it can receive mail messages.

> Note
>
> The KwikNet SMTP option is a KADAK software product, separate from that offered by Treck Inc. Hence the SMTP application programming interface (API) described in the Treck SMTP Client User's Manual does not apply to the KwikNet SMTP component.

SMTP Connection

The SMTP protocol does not specify the nature of the peer-to-peer connection. The connection must be established before a mail transfer can occur. The KwikNet SMTP option uses a TCP connection to ensure the reliable transfer of each message.

The SMTP server listens on a TCP socket which is bound to the well known SMTP port number 25, waiting for requests from potential clients. The SMTP client can then connect its own TCP socket to the server. This connection is referred to as the SMTP connection. The connection is used to convey commands and the mail message from the client to the server and to return replies from the server back to the client. The connection between the two peers lasts until one or more mail transfers have been completed and the SMTP session has been ended by the client or until the transfer is aborted by either peer.

SMTP Commands and Replies

An SMTP command is an ASCII string which always includes a command directive followed by zero or more command parameters. All commands are terminated by a two character, end of line sequence consisting of a carriage return (ASCII 13, *0x0D*, *'\r'*) followed by a linefeed (ASCII 10, *0x0A*, *'\n'*). This character sequence is referred to as *<CRLF>* in this manual. Figure 1.2-1 lists the SMTP command strings which the KwikNet SMTP client can generate and which the KwikNet SMTP server can interpret.

The SMTP server reply is an ASCII string consisting of a 3 digit decimal response code followed by some explanatory text. Generally, codes in the 200 range indicate success and codes in the 500 range indicate failures. RFC-2821 summarizes these reply codes.

Command	Purpose
HELO Domain	* Client domain name provided to start session
EHLO Domain	* Client domain name provided to start session
MAIL FROM:<Return-Path>	Message from client mailbox "*Return-Path*"
RCPT TO:<Forward-Path>	Message to mailbox "*Forward-Path*"
DATA	Message follows; ends with *<CRLF>.<CRLF>*
RSET	▪ Reset: discard received message information
HELP	▪ Request help
NOOP	▪ No operation
QUIT	Terminate SMTP session and break connection
VRFY user-mailbox	▪ Verify user or mailbox name
EXPN mail-list	! Expand a mailing list
SOML, SAML, TURN	! Deprecated commands

Note * Client sends *EHLO*; server accepts both but offers no extensions
! Not used by client or supported by server
▪ Client sends command literally

Figure 1.2-1 KwikNet SMTP Commands

Mail Data Content

As its name implies, the Simple Mail Transfer Protocol (SMTP) is designed to allow the transfer of simple mail messages between networked machines. However, interpretation of the term "mail message" is left to the application clients and servers which use SMTP. The transferred mail message must be constructed by any KwikNet peer which acts as an SMTP client. The message must be interpreted (parsed) by any KwikNet peer which acts as an SMTP server.

The recommended structure of a mail message is defined by RFC-2822, Internet Message Format. The mail message (often called mail data) consists of a sequence of message headers followed by a message body. The body content is further described by RFC-2045, Multipurpose Internet Mail Extensions, commonly known as MIME.

The SMTP protocol specifies that mail data is transferred as a sequence of lines of text, with each line limited to 1000 characters encoded using the 7-bit ASCII character set. Each line must be terminated by the *<CRLF>* end of line sequence, the same character sequence used to terminate SMTP commands and replies. Hence, each text line is restricted to 998 usable characters.

When formatting mail messages, your SMTP client should adhere to the above specifications. Do not create text lines of length greater than 998 characters. Always terminate lines with the *<CRLF>* sequence. Do not embed single CR or LF characters within the message. It is also recommended that other non-printing characters (ASCII *0x00* to *0x1F*) be excluded from the message.

Although MIME extensions exist which permit the use of 8-bit characters in the body of a mail message, the KwikNet SMTP option does not support the SMTP service extensions which are required to ensure that the SMTP server can accept such messages. Hence, it is advisable that your SMTP client restrict its message content to the 7-bit ASCII character set. Be aware that your SMTP server may receive 8-bit MIME encoded mail data from an SMTP client, even though the KwikNet SMTP server notifies the client that it does not support service extensions.

Your SMTP client can send a complete mail message with one call to service procedure *knsm_data()*. The mail data can also be sent as a sequence of text blocks via a series of calls to *knsm_data()*. An alternate form of service procedure *knsm_data()* can be used to send a series of simple text strings a line at a time.

Your SMTP server will receive the text lines of a mail message in their entirety, one line at a time. Each line will include the terminating *<CRLF>* end of line characters.

> Note
>
> The KwikNet SMTP option will not prevent your SMTP client from sending any particular character sequence within your mail message. Your SMTP server will also receive each mail message without translation.

Timing Requirements

The SMTP protocol imposes very loose timing constraints on the SMTP client and server. Timeout intervals while awaiting SMTP commands or replies vary from 2 to 10 minutes. Thus, once a TCP connection has been established, an SMTP transfer can take a considerable time to complete and still fall well within the timing specifications recommended by RFC-2821. Your SMTP client and server can use service procedure *knsm_ioctl()* to adjust these timeout parameters for any particular SMTP mail transfer.

The SMTP client establishes the pace at which mail data can be transferred to its peer. Once an SMTP connection has been established and the mail data transfer has been started, the mail data provided by your KwikNet SMTP client will be streamed into the TCP socket. Partial lines of mail data are never presented to the TCP socket.

When a KwikNet SMTP server is accepting a mail message, the mail data will be fetched from the TCP socket as it becomes available until a complete line of mail data is available for presentation to your application.

The SMTP connection will be serviced at the SMTP client or server service interval specified in your KwikNet Library configuration (see Chapter 1.3). Hence, a brief pause equal to the client service interval will occur during transmission of a line of mail data if the entire data line cannot be written to the client's TCP socket. Similarly, a brief pause equal to the server service interval can occur during reception of a line of mail data if an entire data line is not immediately available from the TCP socket being used by the server to get mail from its client.

SMTP Error Handling

The SMTP specification defines error response codes that an SMTP server must return to an SMTP client if it is unable to accept and process a particular SMTP command. The RSET command can be used by the client to force the server to flush the message transaction details received prior to the RSET command. Your KwikNet SMTP client can call service procedure *knsm_literal()* to send the RSET command. The client can also use procedure *knsm_ioctl()* to install a client callback function to gain access to the server's error response code and descriptive text.

Unfortunately, these error reporting mechanisms can only be used prior to the actual transfer of the mail message. Once the data transfer request has been made by the client and accepted by the server, the data transfer must proceed to completion. Your SMTP client can only abort the message delivery by calling service procedure *knsm_close()* to break the SMTP connection. Your server application has two options. It can accept but ignore the message data and then generate an error reply once the entire message has been received and discarded. Alternatively, it can reject the mail data, forcing the SMTP connection to be broken, thereby aborting the SMTP client session.

> Note
>
> Aborting an SMTP mail data transfer by breaking the SMTP connection is considered poor practice and should be avoided unless no other solution is available.

SMTP Descriptor Options

KwikNet assigns an SMTP descriptor for each SMTP connection which it establishes. It also allocates a server SMTP descriptor for listening for client mail requests. Many of the attributes and parameters associated with an SMTP descriptor can be accessed or modified by your application using KwikNet service procedure *knsm_ioctl()*.

The following SMTP descriptor options are recognized by KwikNet. The option mnemonics are defined in file *KN_SMTP.H*. A parameter of the specified type must be passed to function *knsm_ioctl()* when manipulating a specific option.

Option	Parameter Type	Purpose
KN_SMTIOC_GETCBF	*KN_SMCALLBACK **	Get callback function pointer
KN_SMTIOC_SETCBF	*KN_SMCALLBACK **	Set callback function pointer
KN_SMTIOC_GETCBP	*long **	Get private user variable/callback parameter
KN_SMTIOC_SETCBP	*long*	Set private user variable/callback parameter
KN_SMTIOC_GETLADR	*sockaddr_in **	Get local network IPv4 address and port
KN_SMTIOC_GETFADR	*sockaddr_in **	Get foreign network IPv4 address and port
KN_SMTIOC_SETLADR	*sockaddr_in **	Set local server network IPv4 address and port
KN_SMTIOC_SETERRNUM	*long*	Set SMTP error number
KN_SMTIOC_SETERRMSG	*char **	Set SMTP error message
KN_SMTIOC_SETHELP	*char **	Set SMTP help message
KN_SMTIOC_SETNAME	*char **	Set domain name
KN_SMTIOC_SETTMVAL	*long array[2]*	Set interval for one or more timeout values

SMTP User Variable / Callback Parameter

KwikNet allows your SMTP client or server to maintain an application specific variable associated with any KwikNet SMTP descriptor. This user variable is a *long* value whose purpose is dictated by your application. KwikNet function *knsm_ioctl()* can be used to set/get the user variable associated with a particular SMTP descriptor. The SMTP client and server callback functions (see Chapters 1.5 and 1.7 respectively) receive this variable as a callback parameter.

SMTP Logging

KwikNet provides a data logging service which can be used to advantage to observe the progress of stack activity. This service is described in Chapter 1.6 of the KwikNet TCP/IP Stack User's Guide.

The KwikNet SMTP client or server can be configured to record error information on the KwikNet logging device. To enable this feature, edit your KwikNet Network Parameter File and check the client or server Logging option on the SMTP property page.

The KwikNet SMTP server does not log the normal receipt of mail requests from SMTP clients. The SMTP client and server do not log the successful transfer of mail data. Since these operations involve normal TCP transactions, they can be observed using standard KwikNet debug logging and trace facilities.

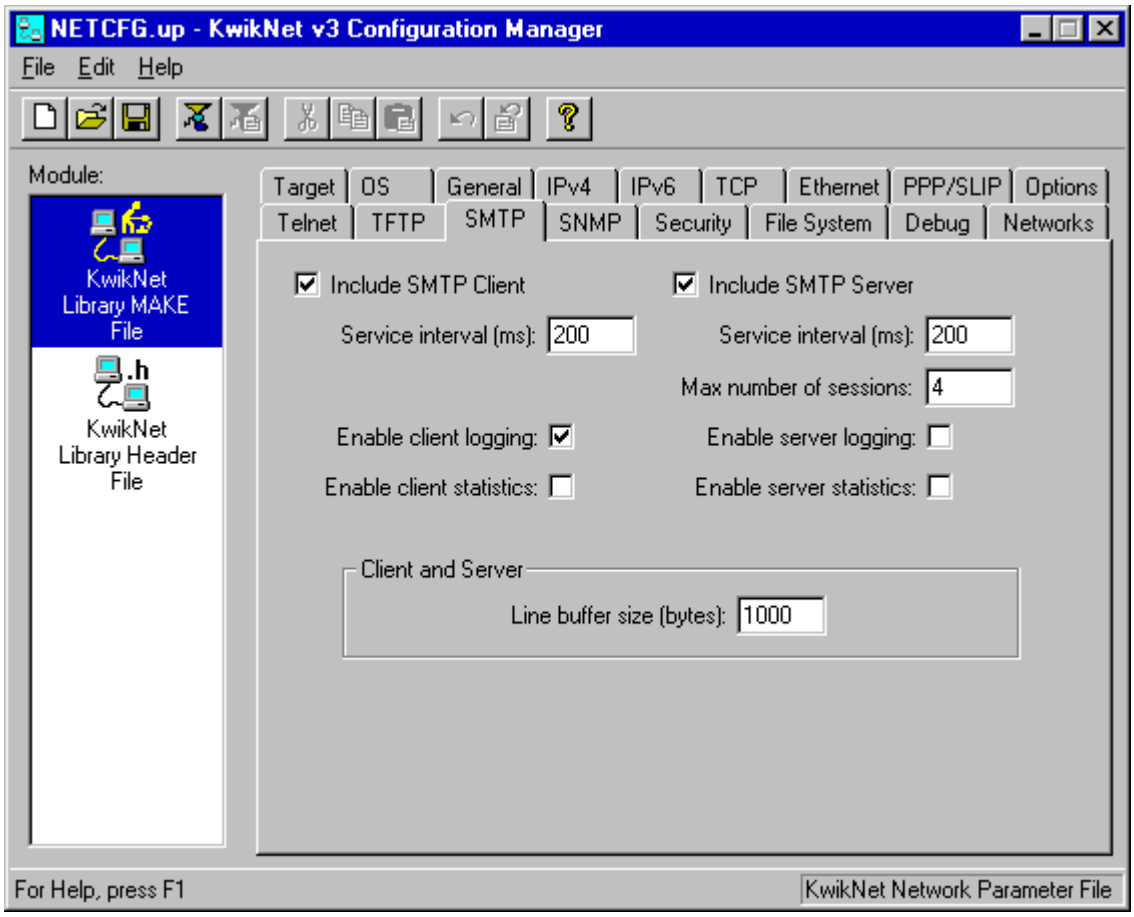
The KwikNet SMTP client logs all SMTP error replies which are received from an SMTP server. The KwikNet SMTP server logs all SMTP error replies which are sent to an SMTP client. Both client and server also log abnormal run-time conditions which usually indicate that KwikNet's private SMTP control structures have been compromised (altered or misused) by your application.

If SMTP statistics logging is enabled, your KwikNet SMTP client or server can call procedure *knsm_status()* to generate an SMTP status log summarizing the current state of a particular SMTP client connection. The status log identifies the SMTP endpoints and describes the operating characteristics of the connection. The log includes a statistics summary showing counts of various events detected while the SMTP connection was being serviced. The server can also generate a brief summary of its current status, including counts of events detected while servicing mail requests from its clients. Each status summary will be logged on the KwikNet logging device. To enable this feature, edit your KwikNet Network Parameter File and check the client or server Statistics option on the SMTP property page.

1.3 KwikNet SMTP Configuration

You can readily tailor the KwikNet stack to accommodate your SMTP needs by using the KwikNet Configuration Builder to edit your KwikNet Network Parameter File. The KwikNet Library parameters are edited on the SMTP property page. The layout of the window is shown below.

Note that the TCP protocol is a prerequisite for using the SMTP protocol with KwikNet.



Include SMTP Client

Check this box if your application will include an SMTP client. Otherwise, leave this box unchecked.

Client Service Interval

Specify the interval in milliseconds at which an SMTP client should service its SMTP connection. This interval determines the period at which the client connection to the server will be serviced by KwikNet whenever the client is waiting to establish a TCP connection or to send or receive using its TCP connection.

SMTP Parameters (continued)

Include SMTP Server

Check this box if your application will include an SMTP server. Otherwise, leave this box unchecked.

Server Service Interval

Specify the interval in milliseconds at which an SMTP server should service all of its SMTP client connections.

Maximum Number of Sessions

This parameter defines the maximum number of SMTP clients which the SMTP server will support at any one time. Once this many sessions are active, the SMTP server will reject further requests for service until one or more of the active sessions terminates.

Enable Client / Server Logging

Check this box if you wish your SMTP client or server to log events as they occur during an SMTP session. All logged messages will be directed to the KwikNet logging device, if one has been provided. Otherwise, leave this box unchecked. Leaving this box unchecked reduces the code size by eliminating all error message generation logic from the SMTP service procedures in the KwikNet Library.

Enable Client / Server Statistics

Check this box if your SMTP client or server will use function *knsm_status()* to generate an SMTP status summary for an SMTP connection. The summary will be directed to the KwikNet logging device, if one has been provided. Otherwise, leave this box unchecked. Leaving this box unchecked reduces the code size by eliminating all status summary generation logic from the KwikNet Library.

Line Buffer Size

This parameter defines the size (in bytes) of the SMTP client or server data buffer used for sending or receiving client commands, server replies and mail data. The recommended value is 1000, the longest line of mail data contemplated by the SMTP specification. Although you can reduce this value, doing so may prevent your SMTP client from sending mail to other servers and your SMTP server from receiving mail from other clients.

1.4 SMTP Client Task

The KwikNet SMTP option includes a set of services for use by one or more SMTP clients. Each SMTP client is an application program which makes use of these services to communicate with an SMTP server. The collection of application procedures which makes up such a program is called an SMTP client task.

Your SMTP client task must call *knsm_create()* to acquire an SMTP descriptor to identify its mail sessions. The client task can alter some of its operating parameters using service procedure *knsm_ioctl()*. For example, you should install a valid client domain name if possible. Otherwise the client will be identified using the dotted decimal IPv4 address of the local network interface used to connect to the SMTP server. You may also wish to revise the default client timeout values assigned by KwikNet to every SMTP client.

Once your SMTP client task is ready, it can call *knsm_open()* to establish a connection with a particular SMTP server and open a mail session. The IPv4 address of the SMTP server and its SMTP port number must be provided by your client task. Once an SMTP connection has been established, the client task can send one or more mail messages.

One mail transaction sequence is required to deliver each mail message. A single mail transaction proceeds as follows. The client task calls *knsm_mailfrom()* to initiate a mail request. If the server acknowledges that it can accept the mail from the sender, the client can proceed with a series of one or more calls to *knsm_sendto()* to identify each mailbox to which the mail message must be delivered. If the server acknowledges that it can accept the mail for delivery to one or more of the recipients, a mail data transfer can be started.

The client task must make a series of one or more calls to *knsm_data()* to send the complete mail message, including all mail headers and the mail body. It is the client task's responsibility to provide the properly formatted mail message in its entirety. The client task terminates the message transaction by calling *knsm_data()* with an end of message indication. The client is then free to initiate another mail transaction starting with another *knsm_mailfrom()* call.

Once all mail messages have been sent, the client task must call *knsm_close()* to end its conversation with the server and break the SMTP connection. Once disconnected, further SMTP mail transfers cannot be performed by the client without first calling *knsm_open()* to establish a new connection. If its services are no longer required, a client task can call *knsm_delete()* to discard its SMTP descriptor.

The client task can call procedure *knsm_literal()* to send any of the SMTP commands marked with ▪ in Figure 1.2-1. These SMTP commands can only be sent by the client task after its *knsm_open()* call and before mail data transmission begins. To assist the client in interpreting the server replies to such commands, the client task can provide a callback function which will be called with each server reply as it is received. The callback function is installed and removed using procedure *knsm_ioctl()*. The client callback function is described in Chapter 1.5.

Once an SMTP connection has been established, the client task can send the VRFY command to determine if a particular mailbox address is known by the server. If a call to *knsm_mailfrom()* or *knsm_sendto()* fails because the server rejects the mailbox address, the client task can send the RSET command to the server and then initiate a new mail transaction with a corrected mailbox address.

Once the client task starts to send mail data, it can only abort the mail transaction by calling *knsm_close()* to break the SMTP connection. However, doing so is contrary to the spirit of the SMTP specification.

The SMTP Sample Program provided with the KwikNet SMTP option illustrates an SMTP client task. The client task uses the KwikNet console driver (see Chapter 1.8 of the KwikNet TCP/IP Stack User's Guide) to record its progress on a simple console device.

Multitasking Operation

When used with a real-time operating system (RTOS) such as KADAK's AMX Real-Time Multitasking Kernel, each SMTP client must be an application task. Although one task can be written to service multiple SMTP connections, it is usual, and conceptually simpler, to consider each SMTP client to be a unique task. Such a task is referred to as an SMTP client task.

An SMTP client task is created and started just like any other application task. Once started, the SMTP client task operates as previously described.

> Note
>
> In multitasking systems, each KwikNet SMTP client task MUST execute at a priority below that of the KwikNet Task.

Single Threaded Operation

When used with a single threaded operating system, the SMTP client operates in the user domain as part of your App-Task as described in Chapter 1.2 of the KwikNet TCP/IP Stack User's Guide. When your App-Task is executing your client code, the App-Task is referred to as an SMTP client task.

While executing as an SMTP client, your App-Task must continue to regularly call KwikNet procedure *kn_yield()* to let the KwikNet TCP/IP Stack continue to operate. Fortunately, when your SMTP client initiates any SMTP transaction that might force the client task to block, KwikNet ensures that the TCP/IP stack continues to operate until the SMTP transaction is complete.

Although KwikNet can support multiple, concurrent SMTP connections, it is up to your SMTP client task to manage the separate SMTP transaction sequences for each of the connections.

1.5 Client Callback Function

Most client tasks will operate without requiring access to the SMTP server's reply strings. The client task will simply connect to the server and then send a sequence of one or more mail messages to the server. The KwikNet client service procedures will decode the server's replies and interact with the client task accordingly. If client logging is enabled, error conditions will be logged. No special action by the client task is required.

However, the KwikNet SMTP option does provide a mechanism for the client task to gain access to the server's replies, should there be such a need. The client task can call procedure *knsm_ioctl()* to install a client callback function. The callback function will be called by KwikNet whenever a reply is received from the SMTP server.

The callback function is prototyped as follows. The callback function prototype is defined using a *typedef* declaration named *KN_SMCALLBACK*. The prototype declaration and definitions for callback parameters are provided in KwikNet header file *KN_SMTP.H*.

```
int sm_callback(KN_SMD smd, int opcode,
                      char *p1, int p2, void *param)
```

Parameter ***smd*** is the SMTP descriptor allocated by KwikNet when the client task calls *knsm_create()* and used to identify a particular SMTP client session. This parameter must be used by the callback function to identify the client session in any calls which it makes to KwikNet's SMTP service procedures.

Parameter ***opcode*** is an integer value which identifies the purpose of the call.
The following callback codes are defined in SMTP header file *KN_SMTP.H*.

KN_SMCBC_EXTN	Extension replies from server
KN_SMCBC_HELP	Help string from server
KN_SMCBC_REPLY	Normal, error or warning reply from server

Parameter ***p1*** is a pointer to the server's SMTP reply string in the client's line buffer.
The terminating <*CRLF*> is stripped. The string ends with the *'\0'* termination character.

Parameter ***p2*** is the integer value of the reply code in the server's SMTP reply string.

Parameter ***param*** is always the client task's callback parameter. The interpretation of this parameter is up to you. The value received by the callback function is the callback parameter value, if any, established by the client task via procedure *knsm_ioctl()*.

Callback Function Execution

The client callback function executes in the context of the KwikNet SMTP client task which installed the function. The callback function must service the request as quickly as possible. Service of all other SMTP activity by the client task will be delayed until the callback function finishes execution.

Callback Function Return Codes

Your client callback function should always return the value *KN_SMCBR_OK*.
Any other value is ignored by KwikNet.

1.6 SMTP Server Task

The KwikNet SMTP option includes a set of services which can be used to implement an SMTP server. The SMTP server is an application program which makes use of these services to communicate with one or more SMTP clients. The collection of application procedures which makes up such a program is called an SMTP server task.

When ready to begin operation, your SMTP server task simply calls KwikNet procedure *knsm_start()*. The server task must provide a callback function with which the KwikNet SMTP server can interact to service SMTP clients. Once the server is started, it is this callback function which provides the interface to your application. The server callback function is described in Chapter 1.7.

The KwikNet SMTP server allocates an SMTP descriptor and initializes it with the server's default parameters. The server then immediately calls your server callback function (with callback code *KN_SMCBC_PREP*), offering an opportunity to alter some of the server's operating parameters using service procedure *knsm_ioctl()*.

For example, your callback function should install a valid server domain name if possible. Otherwise the server will be identified using the dotted decimal IPv4 address of the local network interface used to connect to the SMTP client. You may also wish to revise the default server timeout values assigned by KwikNet to every SMTP server. It is also recommended that you install a set of help reply strings to be presented to SMTP clients upon their request for help.

By default, the SMTP server will service all SMTP requests directed to any of the IP addresses assigned to the server's network node. If preferred, your server callback function can use procedure *knsm_ioctl()* to provide a specific IPv4 address which clients must use to connect to your SMTP server.

The SMTP server accepts requests directed to the well known SMTP port number 25. However, your server callback function can use procedure *knsm_ioctl()* to alter its configuration to service requests on any port number which you choose. Of course, only clients which know your alternate port number will be able to connect to your server.

Once your callback function returns to the server after handling the *KN_SMCBC_PREP* preparation request, the server establishes a TCP listening socket and awaits requests from potential SMTP clients. Thereafter, the KwikNet SMTP server interacts with your server callback function to service those clients.

The callback function receives notification whenever a client connection is made or broken. The callback function is called to qualify the client's domain name and mailbox address and to confirm your acceptance of each mail recipient specified by the client. SMTP mail data from the client is then passed to your application through the callback function which must process the mail message according to the requirements of your application. In the absence of any client activity, the callback function is called periodically (at the server's service interval) to allow your server to abort the connection if deemed necessary.

When erroneous mail data is detected, your callback function should discard the erroneous mail and all subsequent mail data until the end of message indication is received. At that time, the callback function can reject the mail message. In this case, an error response will be sent to the client notifying it that its mail was rejected. The SMTP client connection will remain open until it is closed by the client.

Although the SMTP specification discourages the practice, the KwikNet SMTP server allows your callback function to break the SMTP connection in order to exit from an otherwise potentially unrecoverable mail transaction. To do so, the callback function simply rejects the mail data with an indication that the mail transaction should be aborted.

The SMTP Sample Program provided with the KwikNet SMTP option illustrates an SMTP server task. The server task uses the KwikNet console driver (see Chapter 1.8 of the KwikNet TCP/IP Stack User's Guide) to record its progress on a simple console device.

Multitasking Operation

When used with a real-time operating system (RTOS) such as KADAK's AMX Real-Time Multitasking Kernel, the SMTP server operates as an application task. Such a task is referred to as an SMTP server task. Although more than one SMTP server task is allowed, rarely is there such a need.

An SMTP server task is created and started just like any other application task. When ready to begin operation, the server task simply calls KwikNet procedure *knsm_start()* to establish an SMTP server session and begin service. There is no return from this procedure until the server is forced to stop.

The SMTP server task will operate until some unrecoverable error condition is detected or until some other application task calls KwikNet service procedure *knsm_stop()* requesting the server to stop. The SMTP server task will abort all of its active SMTP sessions and resume execution following the initial call to *knsm_start()*.

> Note
>
> In multitasking systems, each KwikNet SMTP server task MUST execute at a priority below that of the KwikNet Task.

Single Threaded Operation

When used with a single threaded operating system, the SMTP server operates in the KwikNet domain in the context of the KwikNet Task as described in Chapter 1.2 of the KwikNet TCP/IP Stack User's Guide.

When your App-Task calls KwikNet procedure *knsm_start()*, an SMTP server session is established and the SMTP server is added to the KwikNet server queue. Such a server is referred to as an SMTP server task. Although more than one SMTP server task is allowed, rarely is there such a need.

Once the SMTP server is operational, your App-Task must regularly call KwikNet procedure *kn_yield()* to let the SMTP server operate.

The SMTP server task will operate until some unrecoverable error condition is detected or until your App-Task calls KwikNet service procedure *knsm_stop()* requesting the server to stop. The SMTP server task will abort all of its active SMTP sessions and remove itself from the KwikNet server queue.

1.7 Server Callback Function

Your interface to a KwikNet SMTP server is the server callback function. The server callback function is the application procedure which you must identify when you call procedure *knsm_start()* to create and start the SMTP server. It is this callback function which operates to accept mail messages from any SMTP client. The callback function is called by the KwikNet SMTP server to interpret and handle all mail from clients.

The callback function is prototyped as follows. The callback function prototype is defined using a *typedef* declaration named *KN_SMCALLBACK*. The prototype declaration and definitions for callback parameters are provided in KwikNet header file *KN_SMTP.H*.

```
int sm_callback(KN_SMD smd, int opcode,
                    char *p1, int p2, void *param)
```

Unless otherwise specified, parameter ***smd*** is an SMTP descriptor allocated by KwikNet to identify a particular SMTP client session. This parameter must be used by the callback function to identify the client session in any calls which it makes to KwikNet's SMTP service procedures.

Parameter ***opcode*** is an integer value which identifies the purpose of the call to the server callback function. The following callback codes are defined in SMTP header file *KN_SMTP.H*. The interpretation of parameters *p1* and *p2* is dictated by the value of parameter *opcode*.

KN_SMCBC_PREP	Prepare SMTP server to meet your specific requirements
KN_SMCBC_CONN	New client connection established
KN_SMCBC_DROP	Client disconnected and session terminated
KN_SMCBC_IDLE	SMTP connection is idle or server is discarding mail data
KN_SMCBC_INIT	Client initiation; provides client's domain name
KN_SMCBC_FROM	Mail sender identified by client
KN_SMCBC_TO	Mail recipient identified by client
KN_SMCBC_MAIL	Client is ready to send mail headings and body
KN_SMCBC_DATA	Mail data line with <*CRLF*> received from client
KN_SMCBC_DATX	Mail data without <*CRLF*> received from client
KN_SMCBC_EOM	End of mail indication received from client
KN_SMCBC_RSET	Reset (discard) client's current mail transaction
KN_SMCBC_VRFY	Verify recipient identified by client

Parameter ***param*** is always the application callback parameter. The interpretation of this parameter is up to you. The value received by the callback function is either the server's default callback parameter value or the current callback parameter value.

Server Preparation

As soon as the SMTP server starts, it calls the callback function with callback code *KN_SMCBC_PREP*. In this special callback, parameter *smd* is the server's SMTP descriptor, not a client session descriptor. This callback gives your application an opportunity to alter any of the server's operating parameters which can be adjusted with calls to *knsm_ioctl()*. Parameters *p1* and *p2* are unused. The function must return the value *KN_SMCBR_OK* to allow the server to continue. Any other return code will force the server to shut down with an indication that the server was unable to successfully start.

Callback Parameter

Whenever a new client session is created (*opcode* is *KN_SMCBC_CONN*), the session inherits the server's **default callback parameter value**. Usually the default value is the original callback parameter value specified to procedure *knsm_start()* when the SMTP server was started. However, when your callback function receives the server's *KN_SMCBC_PREP* initialization request, it can call procedure *knsm_ioctl()* to alter the server's default callback parameter.

Once a client session is underway, the callback parameter value inherited from the server becomes the **current callback parameter value**. Hence, after the initial call establishing a new client session, the callback function will always receive the current value. Of course, your callback function can always use procedure *knsm_ioctl()* to alter the current callback parameter value for the client session specified by parameter *smd*.

Callback Function Execution

The server callback function executes in the context of the KwikNet SMTP server task which called it. The callback function must service the request as quickly as possible. Service of all other SMTP activity by the server task will be delayed until the callback function finishes execution.

In a multitasking system, responsibility for time consuming operations such as mail data transfers to files should be passed to other lower priority tasks, if possible.

The server callback function can call any of the KwikNet SMTP server-specific procedures described in Chapter 2 as long as it adheres to the documented restrictions, if any.

Callback Function Return Codes

Your server callback function must return one of the following return codes.

KN_SMCBR_OK	Request handled without error
KN_SMCBR_ERROR	Request denied
KN_SMCBR_ABORT	Request cannot be handled; disconnect from client

Client Connections

Whenever the SMTP server accepts a new TCP socket connection from an SMTP client, it calls the callback function with callback code *KN_SMCBC_CONN*. Parameters *p1* and *p2* are unused. The function must return *KN_SMCBR_OK* to accept the connection and establish a new client session. Any other return code will force the connection to be rejected.

When a new connection is established, you may wish to use procedure *knsm_ioctl()* to alter the value of the callback parameter for the particular client session. For example, your server might assign a unique client management block to each new client session for subsequent use throughout the session. The pointer to the client management block is installed as the current callback parameter value for the client's session.

If a client breaks its TCP socket connection, the SMTP server will call the callback function with callback code *KN_SMCBC_DROP*. The SMTP server will also make this callback if it automatically disconnects the client for any reason. Note that this callback will occur even if the disconnect was initiated by the server to abort a mail data transfer following receipt of return code *KN_SMCBR_ABORT* from the callback function. The callback function must release the resources, if any, which it has been using to service the client. Upon return to the SMTP server, the client session will be terminated. Parameters *p1* and *p2* are unused. The return code is ignored.

Client Initiation

When a client presents its domain name at the start of a client session, the SMTP server calls the callback function with callback code *KN_SMCBC_INIT*. Your function must qualify the client's domain name. Parameter *p1* is a pointer to the client's domain name string. The terminating *<CRLF>* is stripped. The string ends with the *'\0'* termination character. Parameter *p2* is the length of the string. The function must return *KN_SMCBR_OK* if it is willing to converse with the named client or *KN_SMCBR_ERROR* if not. The function can return *KN_SMCBR_ABORT* to abort the mail transaction.

Mail From

When a client requests to begin a new mail transaction, the SMTP server calls the callback function with callback code *KN_SMCBC_FROM*. Your function must qualify the sender's mailbox address. Parameter *p1* is a pointer to the mailbox address string. The terminating *<CRLF>* is stripped. The string ends with the *'\0'* termination character. Parameter *p2* is the length of the string. The function must return *KN_SMCBR_OK* if it is willing to accept mail from the sender or *KN_SMCBR_ERROR* if not. The function can return *KN_SMCBR_ABORT* to abort the mail transaction.

Mail To

When a client identifies a particular mail recipient, the SMTP server calls the callback function with callback code *KN_SMCBC_TO*. Your function must qualify the recipient's mailbox address. Parameter *p1* is a pointer to the mailbox address string. The terminating *<CRLF>* is stripped. The string ends with the *'\0'* termination character. Parameter *p2* is the length of the string. The function must return *KN_SMCBR_OK* if it is willing to accept mail for the identified recipient or *KN_SMCBR_ERROR* if not. The function can return *KN_SMCBR_ABORT* to abort the mail transaction.

Mail Ready

When a client is ready to send its mail message, the SMTP server calls the callback function with callback code *KN_SMCBC_MAIL*. The server will only make this call if the client has identified the mail source, the sender's mailbox and at least one destination mailbox, all of which have been qualified by you. Parameters *p1* and *p2* are unused. The function must return *KN_SMCBR_OK* if it is able to accept the mail data or *KN_SMCBR_ERROR* if not. The function can return *KN_SMCBR_ABORT* to abort the mail transaction.

Mail Data from Client

Whenever the SMTP server receives a complete line of mail data from a connected client, it calls the callback function with callback code *KN_SMCBC_DATA* to present the data for interpretation by your application. Parameter *p1* is a pointer to the data line in the client session's line buffer. Parameter *p2* specifies the number of data characters stored at **p1*. *P2* will always be greater than *0* and will never be greater than the configured line buffer size which, by default, is set to 1000. The count *p2* will include the two characters in the end of line character string *<CRLF>*. There is no *'\0'* termination character following the *<CRLF>* character pair.

If the SMTP server receives a line of mail data which exceeds its configured line buffer size, it calls the callback function with callback code *KN_SMCBC_DATX* to present the data to your application. Parameter *p1* is a pointer to the data fragment in the line buffer. The fragment will not include a *<CRLF>* character pair or a terminating *'\0'* character. Parameter *p2* specifies the number of data characters stored at **p1*. One or more sequential data fragments may be delivered to your callback function in this manner. When the end of the overlength mail data line is detected, the final line fragment will be presented to your callback function with the usual callback code *KN_SMCBC_DATA*.

All received characters *should* be 7-bit ASCII characters unless the SMTP client is sending a message body using an 8-bit encoded MIME format, contrary to the KwikNet SMTP server's reply to the client's EHLO command.

The callback function must return *KN_SMCBR_OK* if it accepts the mail data, even if it then discards the data. The function must accept all of the data presented to it or lose it. If the data must be preserved, the callback function must copy it from the session's line buffer. As long as the callback function accepts the mail data, it will continue to be called with each successive line of mail data until the end of the mail message is encountered.

The callback function can return *KN_SMCBR_ERROR* to reject a mail message. Subsequent lines of mail data will be silently discarded by the KwikNet SMTP server until the end of the mail data is detected. No end of mail callback will be made. For this reason, the callback function must release any resources that it controls before returning the *KN_SMCBR_ERROR* code. As each line of mail data is discarded, the callback function is called with callback code *KN_SMCBC_IDLE* and a count of the number of discarded data characters. When all mail data has been received, the KwikNet SMTP server will respond to the client indicating that its mail was rejected. The client's SMTP connection will remain in place unless broken by the client.

End of Mail

When the end of the mail data is reached, the SMTP server calls the callback function with callback code *KN_SMCBC_EOM*. Parameters *p1* and *p2* are unused. The callback function must release any resources that it controls.

The callback function must return *KN_SMCBR_OK* if the mail message is accepted for delivery to the named recipients, even if some of the mail data was discarded by your function as it was received. The KwikNet SMTP server will send a response to the client indicating that the mail was delivered without error.

The callback function can return *KN_SMCBR_ERROR* if the mail message is to be rejected for some reason. If it has not already done so, your callback function can call *knsm_ioctl()* to revise the server's error code and error message to indicate the reason for the rejection. The error code and message will be sent by the KwikNet SMTP server in its SMTP error reply to the client.

Whether the mail was accepted or rejected, the SMTP connection to the client will remain intact. In some cases, the client may resend the mail, with or without corrections. In other cases, the client may send another mail message. The connection will not be broken unless the client terminates the SMTP session.

The callback function can return *KN_SMCBR_ABORT* to abort the mail transaction. In this case, the KwikNet SMTP server will send an error indication to the client and break the SMTP connection to the client.

Idle Notification

If the SMTP server has no client activity to report to your application, it calls the callback function at each service interval with callback code *KN_SMCBC_IDLE* and parameter *p2* set to *0*. If *p2* is *n* and *n* is greater than *0*, then the call is an indication that the server has just silently discarded *n* bytes of mail data. In either case, the call gives your server process an opportunity to abort the client session if necessary. Parameter *p1* is unused. The function can return *KN_SMCBR_ABORT* to abort the mail transaction. Any other return code is ignored.

Reset Command

An SMTP client can send a reset command to prematurely terminate a mail transaction. The reset command must be sent before that mail data transfer begins. The server must discard the sender and recipient mailbox identifiers.

When the KwikNet SMTP server receives a reset command during an active mail transaction, it calls the callback function with callback code *KN_SMCBC_RSET*. Parameters *p1* and *p2* are unused. Your function must reset its sender and recipient mailbox parameters, if any, and release any resources that it controls, in preparation for a new mail transaction. The function can return *KN_SMCBR_ABORT* to abort the mail transaction. Any other return code is ignored.

Verify Command

Whenever the SMTP server receives a verify command, it calls the callback function with callback code *KN_SMCBC_VRFY* to allow your application to qualify the potential recipient. Your function must qualify the recipient's mailbox address. Parameter *p1* is a pointer to the mailbox address string. The terminating *<CRLF>* is stripped. The string ends with the *'\0'* termination character. Parameter *p2* is the length of the string. The function must return *KN_SMCBR_OK* if it is able to accept mail for the identified recipient or *KN_SMCBR_ERROR* if not. The function can return *KN_SMCBR_ABORT* to abort the mail transaction.

Server Replies

The KwikNet SMTP server responds to commands from its SMTP clients by sending reply messages. The reply message includes the 3 digit SMTP numeric reply code which uniquely identifies the success or failure of the action requested by the client. The replies generated by the KwikNet SMTP server are summarized in Figure 1.7-1.

Your SMTP server callback function can call procedure *knsm_ioctl()* to override the server's usual reply. If your callback function installs an SMTP error message, the server will concatenate its numeric reply code with your error string to generate its reply to the client. If your callback function installs an SMTP error number, the server will use that number in its reply to the client. You must ensure that your error number meets the SMTP protocol specification. Note that the KwikNet server will always terminate its reply string with the end of line *<CRLF>* string as required by the SMTP protocol.

If your server callback function returns the value *KN_SMCBR_ABORT*, the KwikNet server will abort its SMTP session and disconnect the client. In this special case, the error number and error message, if any, installed by your callback function will not be used.

Command	Reply	Default Reply String
connect	220	<domain> Service ready
	554	Requested action failed
success	250	OK
failure	550	Request failed (mailbox unavailable)
sequence	503	Command out of sequence
HELO	250, 550	<see above>
EHLO	250, 550	<see above>
VRFY	250, 550	<see above>
MAIL	250, 550	<see above>
RCPT	250, 550	<see above>
DATA	354	OK; send mail
		(Client sends message ending with *<CRLF>.<CRLF>*)
	250	OK
	554	Mail delivery failed
QUIT	221	<domain> Service ends
RSET	250	OK
NOOP	250	OK
HELP	214	<application specific help>
	502	Command not implemented
EXPN	502	Command not implemented
SOML	502	Command not implemented
SAML	502	Command not implemented
TURN	502	Command not implemented

Figure 1.7-1 KwikNet SMTP Server Replies

1.8 SMTP Sample Program

An SMTP Sample Program is provided with the KwikNet SMTP option to illustrate the use of the KwikNet SMTP client and server. The sample program is ready for use with the AMX Real-Time Multitasking Kernel. The sample program can also be used with any of the porting examples provided with the KwikNet Porting Kit.

The sample configuration supports a single network interface. The network uses the KwikNet Ethernet Network Driver. Because the sample must operate on all supported target processors without any specific Ethernet device dependence, KwikNet's Ethernet Loopback Driver is used. Use of this driver allows the SMTP client and server to be tested even if network hardware is not available.

Once the SMTP Sample Program has been tested in loopback fashion, you can replace the Ethernet Loopback Driver with your own network device driver. Then the KwikNet SMTP client and server can be adapted to meet your needs, allowing the client to connect to other SMTP servers and foreign clients to access the server.

The KwikNet TCP/IP Stack requires a clock for proper network timing. All examples provided with the KwikNet Porting Kit illustrate the clock interface. However, the sample program provided for use with AMX has been enhanced to eliminate any dependence on specific target hardware. This sample program includes a very low priority task that can detect if you have added a real AMX clock driver to the sample configuration. If a real hardware clock is not available, this task simulates clock interrupts, thereby providing AMX ticks that meet KwikNet's needs.

The sample includes an SMTP server task and an SMTP client task. In a multitasking system, these tasks are real tasks managed by the RTOS. In a single threaded system, the server is attached to the KwikNet Task's server queue and operates in the KwikNet domain. The client is simply the App-Task executing in the user domain.

The sample SMTP server task establishes itself as an SMTP server and waits for a mail transfer request from an SMTP client. It will only accept mail from one sender with a particular mailbox address. The server will only accept mail for two possible mailboxes.

The SMTP client task opens an SMTP connection and requests to send mail to the server. Upon receiving the request from the client, the server grants the request. The SMTP client task then sends one mail message to two mailboxes and closes the SMTP connection. The server then waits for another mail request.

Next, the SMTP client task establishes a new client session and attempts to send mail from a source which the server does not recognize. The server responds with an error message indicating that the sender has been denied access. The client task then requests to send mail to a non-existent mailbox on the server. The server responds with an error message indicating that no such mailbox exists.

The client proceeds to verify that a particular mailbox is a valid recipient. The server acknowledges that the mailbox address is valid. The client starts to send the message but, "by mistake", specifies the wrong mailbox. Recognizing its "error", the client sends a reset request to the server instructing it to ignore the false start. The client then sends a properly constructed mail message to one of the server's two known mailboxes.

Finally, the client installs its own callback function and asks the server for help. The callback function receives the server's help replies and logs them on the KwikNet logging device. The client then closes the SMTP connection.

The sample uses the KwikNet data logging and message recording services to record messages generated by the SMTP server task, the SMTP client task and the KwikNet TCP/IP Stack. These services are described in Chapters 1.6 and 1.7 of the KwikNet TCP/IP Stack User's Guide. Messages are stored as an array of strings in memory but can be easily echoed to a console terminal (see Chapter 1.8 of the KwikNet TCP/IP Stack User's Guide).

Startup

The manner in which the KwikNet SMTP Sample Program starts and operates is completely dependent upon the underlying operating system with which KwikNet is being used. All sample programs provided with KwikNet and its optional components share a common implementation methodology which is described in Appendix E of the KwikNet TCP/IP Stack User's Guide. Both multitasking and single threaded operation are described.

When used with AMX, the sample program operates as follows. AMX is launched from the *`main()`* program. Restart Procedure *`rrproc()`* starts the print task and calls function *`app_prep()`* which creates and starts the SMTP server and client tasks. A low priority background task is also started to simulate clock interrupts in the absence of a hardware clock.

Once the AMX initialization is complete, the high priority print task executes and waits for the arrival of AMX messages in its private mailbox. Each AMX message includes a pointer to a log buffer containing a KwikNet message to be recorded.

Once the print task is ready and waiting, the server task begins to execute. It immediately suspends itself to wait for the client task to prepare KwikNet for use. The client task finally begins to execute. It starts KwikNet at its entry point *`kn_enter()`*. KwikNet self starts and forces the KwikNet Task to execute. Because the KwikNet Task operates at a priority above all tasks which use its services, it temporarily preempts the client task. The KwikNet Task initializes the network and its associated loopback driver and prepares the IP, TCP and SMTP protocol stacks for use by the sample program.

Once the KwikNet initialization is complete, the SMTP client task resumes execution and removes the server task suspension. The server task, being of higher priority than the client task, immediately resumes execution. When the server reaches a state in which it is waiting for client requests, the lower priority SMTP client task is able to proceed and begin the first of its two client scenarios.

In a single threaded system, the SMTP client task executes in the user domain. The client calls the SMTP server procedure *`server_proc()`* which creates and starts the SMTP server. The server automatically adds itself to the KwikNet server queue so that it will be executed in the KwikNet domain at periodic intervals to service SMTP client requests. Once the server has been started, the client proceeds with the first of its two client scenarios.

SMTP Client Operation

The SMTP client task resumes execution as soon as the server task is ready and waiting to serve clients. The client task calls *knsm_create()* to acquire a client SMTP descriptor. Using its descriptor, the client calls *knsm_ioctl()* to install its arbitrary domain name *sampleclient.com*. Next, the client task again calls *knsm_ioctl()*, this time using the server's SMTP descriptor to fetch the server's IPv4 address and port number. The client then calls *knsm_open()* to connect to the SMTP server to send a mail message to two mailboxes, mboxA and mboxB.

The client calls *knsm_mailfrom()* to inform the server that return mail should be directed to mailbox mboxC. Function *knsm_sendto()* is then called twice to identify mboxA and mboxB as the message recipients. The message headings and body are sent as a sequence of text blocks via calls to *knsm_data()*. The client session is then closed and the SMTP connection is broken with a call to *knsm_close()*.

The SMTP client task proceeds to test its ability to detect error conditions reported by an SMTP server and to recover gracefully when possible. The client calls *knsm_open()* to establish a new connect to the SMTP server. The client calls *knsm_mailfrom()* to inform the server that return mail should be directed to mailbox mboxZ. The request is rejected because the server will not accept mail with return mailbox mboxZ. The client corrects its mistake by calling *knsm_mailfrom()* again, this time identifying mboxC as the mail sender. The server accepts this request.

Next the client calls function *knsm_sendto()* to identify mboxZ as the message recipient. Again, the server declines to accept mail for the unknown mailbox. The client calls *knsm_literal()* to send the VRFY command to confirm that mailbox mboxA is a valid destination. When the server acknowledges that it can deliver to mailbox mboxA, the client calls *knsm_sendto()* again, this time identifying mboxA as the correct recipient.

The client then "realizes" that it is about to make a grave mistake. The message should actually go to mailbox mboxB. To recover, the client calls *knsm_literal()* to send the RSET command to force the server to ignore the information transmitted thus far. The client then repeats its call to *knsm_mailfrom()* identifying mailbox mboxC as the sender and calls *knsm_sendto()* specifying mailbox mboxB as the correct destination.

A series of calls to function *knsm_data()* are then made to send the message headings and body as a sequence of text strings, one line at a time.

Once the second message has been transmitted successfully, the client prepares to fetch any user help that the server has to offer. To gain access to the server's help reply, the client calls *knsm_ioctl()* to install its callback function *ccb_client()*. The client then calls *knsm_literal()* to send the HELP command to the server. The server replies with its series of help strings which are presented to the callback function *ccb_client()*, one line at a time. Each line is logged by the client task on the KwikNet logging device.

Finally, the client session is closed and the SMTP connection is broken with a call to *knsm_close()*. Since the client task is finished, it calls *knsm_delete()* to delete its client SMTP descriptor and initiates a shutdown of the sample application.

SMTP Server Operation

The SMTP client cannot open a connection to an SMTP server unless such a server exists on the network. Unless you have replaced the Ethernet Loopback Driver with a real device driver, the SMTP Sample Program has no direct network connection. Hence no SMTP server is accessible.

The Sample Program overcomes this limitation by implementing a KwikNet SMTP server to provide a very restricted set of SMTP services. The KwikNet SMTP client can therefore converse directly with the KwikNet SMTP server *across the network* even though both are executing on the same processor.

When the SMTP Sample Program begins, it automatically starts the KwikNet SMTP server task which immediately executes since it is of higher priority than the client task. The server task calls *`knsm_start()`* to create and start the KwikNet SMTP server. The server callback function *`scb_server()`* is called upon to prepare the server to meet the application's requirements.

The callback function calls *`knsm_ioctl()`* to install the server's domain name *`sampleserver.com`*. It then makes another call to *`knsm_ioctl()`* to install a help message for presentation to the server's clients. Unless a help message is provided in this fashion, the KwikNet SMTP server will simply inform clients asking for help that the feature is unavailable.

Once the callback function finishes handling the initial preparation request, the SMTP server task is ready to serve potential clients. Thereafter, callback function *`scb_server()`* interacts with the KwikNet SMTP server to service its clients until the server is stopped.

The server will only accept mail requests from domain *`sampleclient.com`*. It will only accept mail from a client which uses *`<mboxC@sampleclient.com>`* as its return mail address. The server will only accept mail addressed to *`<mboxA@sampleserver.com>`* or *`<mboxB@sampleserver.com>`*.

The sample server does not actually interpret or store received mail. The server simply logs the information which it receives from the client on the KwikNet logging device. The mail sender's domain name and return mailbox address, the mail recipients and the mail data are all logged. The server also logs errors which it detects and reports to the client.

When the SMTP client has finished its final mail transaction, it signals the SMTP server task to shut down. The server generates an SMTP server status summary on the KwikNet logging device, releases its server SMTP descriptor and ceases operation.

Shutdown

When the sample is finished, the SMTP client task initiates a controlled system shutdown. First, the client task calls *knsm_stop()* to signal to the SMTP server task that it must shut down. The client task then waits until it detects that the server has been shut down. The client's call to *knsm_stop()* illustrates how a callback function can be used by any application task to detect whether or not the server successfully stops.

Once the SMTP server task has stopped, the client task calls procedure *kn_exit()* to stop operation of the KwikNet TCP/IP Stack.

Finally, the client initiates a shutdown of the underlying operating system (if possible) and a return to the *main()* procedure.

KwikNet and SMTP Logging

The SMTP Sample Program uses the simple KwikNet message recording service to log text messages. The recorder saves the recorded text strings in a 30,000 byte memory buffer until either 500 strings have been recorded or the memory buffer capacity is reached.

The SMTP Sample Program directs messages to this recorder by calling the KwikNet log procedure *kn_dprintf()*. This procedure operates similarly to the C *printf()* function except that an extra integer parameter of value *0* must precede the format string. The SMTP client task uses this feature to record its progress, connection activity and the final shutdown message. The SMTP server task uses this feature to record connection activity, all mail transactions and errors as they are detected.

KwikNet formats the message into a log buffer and passes the buffer to an application log function for printing. Log function *sam_record()* in the KwikNet Application OS Interface serves this purpose.

In a multitasking system the log buffer is delivered as part of an RTOS dependent message to a print task. The print task calls *kn_logmsg()* in the KwikNet message recording module to record the message and release the log buffer.

In a single threaded system, the log function *sam_record()* can usually call *kn_logmsg()* to record the message and release the log buffer. However, if the message is being generated while executing in the interrupt domain, the log buffer must be passed to the KwikNet Task to be logged. The sample programs provided with the KwikNet Porting Kit illustrate this process.

Since the recorded strings are just stored in memory, they are not readily visible. If a debugger is used to control execution of the SMTP Sample Program, the program can be stopped and the strings can be viewed in text form in a display window by viewing the array variable *kn_recordlist[]* in module *KNRECORD.C*.

Running the Sample Program

The KwikNet SMTP Sample Program is built as described in Chapter 1.9. The result is an executable load module suitable for testing with a debugger.

Since the KwikNet SMTP Sample Program has no visible output unless operated with a console terminal, its operation can only be confirmed using your debugger. Since the program has no hardware dependence, it can readily be used with a target processor simulator, if one is available.

KwikNet includes a number of debug features (see Chapter 1.9 of the KwikNet TCP/IP Stack User's Guide) which will assist you in running the SMTP Sample Program. With KwikNet's debug features enabled, you can place a breakpoint on procedure *kn_bphit()* to trap all errors detected by KwikNet. Of course, if you are using AMX, it is always wise to execute with a breakpoint on the AMX fatal exit procedure *cjksfatal* (*ajfatl* for AMX 86).

If you breakpoint at the end of the *main()* program, you can examine the messages recorded in memory. The messages are stored sequentially in a character array called *kn_records[]*. Variable *kn_recordlist[]* is an array of string pointers referencing the individual recorded messages. Most debuggers will allow you to dump the strings referenced in *kn_recordlist[]* in text form in a display window. The list of string pointers is terminated with a *NULL* string pointer.

If you are connected to the target processor by a serial link, do not be surprised if the debugger takes quite some time to access and display all of the strings referenced by *kn_recordlist[]*. You may be able to improve the response by limiting the display to the actual number of strings in the array as defined by variable *kn_recordindex*.

Once you are confident that the KwikNet SMTP Sample Program is operating properly, you may wish to breakpoint your way through the SMTP client and server, monitoring the recorded messages as you go.

1.9 Making the SMTP Sample Program

The sheer volume of documentation provided with the SMTP option may at first be daunting. However, constructing the KwikNet SMTP Sample Program is actually a fairly simple process made even simpler by the KwikNet Configuration Manager, a Windows® utility provided with KwikNet.

The SMTP Sample Program includes all of the components needed to build the sample application for a particular target processor. You can take these components and, with minor modifications, adapt them for your particular target processor and development environment.

> Note
>
> The KwikNet SMTP Sample Program for a particular target processor family is provided ready for use on one of the development boards used at KADAK for testing.

SMTP Sample Program Directories

When KwikNet and its SMTP option are installed, the following subdirectories on which the sample program construction process depends are created within directory *KNTnnn*.

TCPIP	KwikNet header and source files, Ethernet Network Driver Ethernet and Serial Loopback Drivers
SMTP	SMTP protocol
CFGBLDW	KwikNet Configuration Builder; template files
ERR	Construction error summary
MAKE	KwikNet Library make directory
TOOLXXX	Toolset specific files
TOOLXXX\DRIVERS	KwikNet device drivers and board driver
TOOLXXX\LIB	Toolset specific KwikNet Library will be built here
TOOLXXX\SAM_MAKE	Sample program make directory
TOOLXXX\SAM_SMTP	KwikNet SMTP Sample Program directory
TOOLXXX\SAM_COMN	Common sample program source files

One or more toolset specific directories *TOOLXXX* will be present. There will be one such directory for each of the software development toolsets that KADAK supports. Each toolset vendor is identified by a unique two or three character mnemonic, *XXX*. The mnemonic *UU* identifies the toolset vendor used with the KwikNet Porting Kit.

SMTP Sample Program Files

To build the KwikNet SMTP Sample Program using make file *KNSMTSAM.MAK*, each of the following source files must be present in the indicated destination directory.

Source File	Destination Directory	File Purpose	
.	*CFGBLDW*	KwikNet Configuration Builder; template files	
	KwikNet source directories containing:		
KN_API.H	*TCPIP*	KwikNet Application Interface definitions	
KN_OSIF.H	*TCPIP*	KwikNet OS Interface definitions	
KN_SOCK.H	*TCPIP*	KwikNet Socket Interface definitions	
KN_SMTP.H	*SMTP*	KwikNet SMTP definitions	
	Toolset root directory containing:		
KN_OSIF.INC	*TOOLXXX*	OS Interface Make Specification	
KNZZZCC.INC	*TOOLXXX*	Tailoring File (for use with make utility)	
KNZZZCC.H	*TOOLXXX*	Compiler Configuration Header File	
	KwikNet SMTP Sample Program directory containing:		
KNSMTSAM.MAK	*TOOLXXX\SAM_SMTP*	SMTP Sample Program make file	
KNSMTSAM.C	*TOOLXXX\SAM_SMTP*	SMTP Sample Program	
KNZZZAPP.H	*TOOLXXX\SAM_SMTP*	SMTP Sample Program Application Header	
KNSMTLIB.UP	*TOOLXXX\SAM_SMTP*	Network Parameter File	
KNSMTSAM.LKS	*TOOLXXX\SAM_SMTP*	Link Specification File	(toolset dependent)
		Other toolset dependent files may be present.	
KNSMTSCF.UP	*TOOLXXX\SAM_SMTP*	User Parameter File	(for use with AMX)
KNSMTTCF.UP	*TOOLXXX\SAM_SMTP*	Target Parameter File	(for use with AMX)
	Common sample program source files:		
KNSAMOS.C	*TOOLXXX\SAM_COMN*	Application OS Interface	
KNSAMOS.H	*TOOLXXX\SAM_COMN*	Application OS Interface header file	
KNRECORD.C	*TOOLXXX\SAM_COMN*	Message recording services	
KNCONSOL.C	*TOOLXXX\SAM_COMN*	Console driver	
KNCONSOL.H	*TOOLXXX\SAM_COMN*	Console driver header	
		Console driver serial I/O support:	
KN8250S.C	*TOOLXXX\SAM_COMN*	INS8250 (NS16550) UART driver	
KN_BOARD.C	*TOOLXXX\DRIVERS*	Board driver for target hardware	

SMTP Sample Program Parameter File

The Network Parameter File *KNSMTLIB.UP* describes the KwikNet and SMTP options and features illustrated by the sample program. This file is used to construct the KwikNet Library for the SMTP Sample Program.

The Network Parameter File *KNSMTLIB.UP* also describes the network interfaces and the associated device drivers that the sample program needs to operate.

SMTP Sample Program KwikNet Library

Before you can construct the KwikNet SMTP Sample Program, you must first build the associated KwikNet Library.

Use the KwikNet Configuration Builder to edit the sample program Network Parameter File *KNSMTLIB.UP*. Use the Configuration Builder to generate the Network Library Make File *KNSMTLIB.MAK*.

Look for any KwikNet Library Header File *KN_LIB.H* in your toolset library directory *TOOLXXX\LIB*. If the file exists, delete it to ensure that the KwikNet Library is rebuilt to match the needs of the SMTP Sample Program.

Then copy files *KNSMTLIB.UP* and *KNSMTLIB.MAK* into the *MAKE* directory in the KwikNet installation directory *KNTnnn*. Use the Microsoft make utility and your C compiler and librarian to generate the KwikNet Library. Follow the guidelines presented in Chapter 3.2 of the KwikNet TCP/IP Stack User's Guide.

> Note
>
> The KwikNet Library must be built before the SMTP Sample Program can be made. If file *KN_LIB.H* exists in your toolset library directory *TOOLXXX\LIB*, delete it to force the make process to rebuild the KwikNet Library.

The SMTP Sample Program Make Process

Each KwikNet sample program must be constructed from within its own directory in the KwikNet toolset directory. Hence, the KwikNet SMTP Sample Program must be built in directory *TOOLXXX\SAM_SMTP*.

All of the compilers and librarians used at KADAK were tested on a Windows® workstation running Windows NT, 2000 and XP. However, you can build each KwikNet sample program using any recent version of Windows, provided that your software development tools operate on that platform.

To create the KwikNet SMTP Sample Program, proceed as follows. From the Windows Start menu, choose the MS-DOS Command Prompt from the Programs folder. Make the KwikNet toolset *TOOLXXX\SAM_SMTP* directory the current directory.

To use Microsoft's *NMAKE* utility, issue the following command.

```
NMAKE -fKNSMTSAM.MAK "TOOLSET=XXX" "TRKPATH=treckpath"
    "OSPATH=yourospath" "TPATH=toolpath"
```

The make symbol *TOOLSET* is defined to be the toolset mnemonic *XXX* used by KADAK to identify the software tools which you are using.

The symbol *TRKPATH* is defined to be the string *treckpath*, the full path (or the path relative to directory *TOOLXXX\SAM_SMTP*) to your Turbo Treck TCP/IP installation directory.

The symbol *OSPATH* is defined to be the string *yourospath*, the full path (or the path relative to directory *TOOLXXX\SAM_SMTP*) to the directory containing your RT/OS components (header files, libraries and/or object modules). When using AMX, string *yourospath* is the path to your AMX installation directory.

The symbol *TPATH* is defined to be the string *toolpath*, the full path to the directory in which your software development tools have been installed. For some toolsets, *TPATH* is not required. The symbol is only required if it is referenced in file *KNZZZCC.INC*.

The KwikNet SMTP Sample Program load module *KNSMTSAM.xxx* is created in toolset directory *TOOLXXX\SAM_SMTP*. The file extension of the load module will be dictated by the toolset you are using. The extension, such as *OMF*, *ABS*, *EXE*, *EXP* or *HEX*, will match the definition of macro *XEXT* in the tailoring file.

The final step is to use your debugger to load and execute the KwikNet SMTP Sample Program load module *KNSMTSAM.xxx*.

1.10 Adding SMTP to Your Application

Before you can add the SMTP protocol to your application, there are a number of prerequisites that your application must include. You must have a working KwikNet TCP/IP stack operating with your RT/OS. It is imperative that you start with a tested TCP/IP stack with functioning device drivers before you add SMTP. If these components are not operational, the KwikNet SMTP option cannot operate correctly.

KwikNet Library

Begin by deciding whether you need an SMTP client or server or both. Rarely are both required. Then decide which SMTP features must be supported. Review the SMTP property page described in Chapter 1.3.

Use the KwikNet Configuration Manager to edit your application's KwikNet Network Parameter File to include the SMTP protocol. Then rebuild your KwikNet Library. The library extension may be *.A* or *.LIB* or some other extension dictated by the toolset which you are using.

Memory Allocation

An SMTP client or server requires approximately 1200 bytes of memory for each SMTP descriptor which it uses to handle an SMTP client session. An SMTP descriptor of approximately 300 bytes is required by each KwikNet SMTP server to manage the services it offers to SMTP clients. A client SMTP descriptor is allocated when your SMTP client task calls service procedure *knsm_create()*. A server SMTP descriptor is allocated when your SMTP server task calls *knsm_start()*. An additional client session SMTP descriptor is allocated by the SMTP server task each time it accepts an SMTP connection with a client.

To meet these demands, you may have to edit your KwikNet Network Parameter File to increase the memory available for allocation.

SMTP Client and Server Tasks

You must provide one task for each SMTP client and server that you wish to incorporate into your application. Usually one SMTP client task or one SMTP server task is required. Rarely are both needed. Even more rarely are two or more clients or servers required.

If necessary, you can create multiple SMTP client tasks which execute concurrently, each one servicing a single SMTP connection. A single KwikNet SMTP server task is able to service multiple concurrent SMTP client connections.

In a multitasking system, you may have to increase the total number of tasks allowed by your RTOS in order to add the SMTP tasks.

A stack size of 4K to 8K bytes is considered adequate for most SMTP client or server tasks. The stack size can be trimmed after your SMTP tasks have been tested and actual stack usage observed using your debugger.

In a multitasking system, all SMTP tasks must be of lower execution priority than the KwikNet Task. If both SMTP server and client tasks exist, it is usual to make SMTP server tasks of higher priority than SMTP client tasks.

If you are incorporating an SMTP client, you must create an SMTP client task procedure which performs the SMTP operations required by your application. Only you can define such a procedure. All of the KwikNet SMTP client-specific procedures listed in Chapter 2 are at your disposal. You can use the SMTP client task in the SMTP Sample Program as a guideline for proper form.

If you are incorporating an SMTP server, then you may have a significant coding responsibility. You must create an SMTP server task procedure which creates and starts a KwikNet SMTP server. You must also provide the server callback function to interpret and handle all client requests for service. Only you can define these requirements. It is this callback function which provides access by an SMTP client to your application services. All of the KwikNet SMTP server-specific procedures listed in Chapter 2 are at your disposal. You can use the SMTP server task in the SMTP Sample Program as a guideline for proper form.

The SMTP client and server task C source modules must be compiled just like any other KwikNet application module. However, your compiler will also require access to SMTP header file *KN_SMTP.H* in the Treck installation directory, say *C:\TRECK\INCLUDE*. This header file is copied to the Treck directory from the KwikNet *SMTP* installation directory when the KwikNet Library is created. The compilation procedure is described in Chapter 3.4 of the KwikNet TCP/IP Stack User's Guide.

Reconstructing Your KwikNet Application

Since you are adding SMTP to an existing KwikNet application, there is little to be done.

To meet the memory demands of your SMTP client and server, you may have to edit your KwikNet Network Parameter File to increase the memory available for allocation. If you do so, you must then rebuild your KwikNet Library.

Your application link and/or locate specification files must include the KwikNet Library which you built with support for SMTP. The object modules for your SMTP client and server tasks and any support modules that they might require must also be included in your link specification together with your other application object modules.

With these changes in place, you can link and create an updated KwikNet application with SMTP support included.

AMX Considerations

When reconstructing a KwikNet application that uses the AMX Real-Time Multitasking Kernel, adapt the procedure just described to include the following considerations.

You may have to edit your AMX User Parameter File to increase the maximum number of tasks allowed in order to add SMTP client and server tasks.

SMTP client and server tasks can be predefined in your AMX User Parameter File or they can be created dynamically at run-time as is done in the KwikNet SMTP Sample Program. These are simple AMX trigger tasks without message queues.

A stack size of 4K to 8K bytes is considered adequate for use with most device drivers. It should also suffice even if you are using the AMX/FS File System. The stack size can be trimmed after your SMTP tasks have been tested and actual stack usage observed using your debugger.

The SMTP task priorities must be lower than that of the KwikNet Task. If both SMTP server and client tasks exist, it is usual to make SMTP server tasks of higher priority than SMTP client tasks. If you are using AMX 86 to access MS-DOS® file services, the PC Supervisor Task should be below all SMTP client and server tasks in priority.

If you edit your AMX User Parameter File, you must then rebuild and compile your AMX System Configuration Module. If you are using the AMX/FS File System, you should also rebuild and compile your AMX/FS File System Configuration Module.

No changes to your AMX Target Configuration Module are required to support SMTP unless your SMTP client task requires special device support that is not already part of your application.

Performance and Timing Considerations

A meaningful discussion of all of the issues which affect the performance of an SMTP server or client are beyond the scope of this document. Factors affecting the performance of the KwikNet SMTP client and server include the following:

processor speed
memory access speed and caching effects
file system performance and disk access times (if used by your client/server)
competing disk accesses for different users
network type (Ethernet, SLIP, PPP)
network device driver implementation (buffering, polling, DMA support, etc.)
TCP protocol effects (window size adaptations)
IP packet fragmentation
network hops required for connection
operation of the remote (foreign) connected client or server
KwikNet TCP/IP Stack configuration (clock, memory availability, sockets, etc.)
KwikNet SMTP configuration (service/timeout intervals)

Of all these factors, only the last two can be easily adjusted. Increasing the fundamental clock rate for the KwikNet TCP/IP Stack beyond 50Hz will have little effect and will adversely affect systems with slow processors or memory. Increasing the memory available for use by the TCP/IP stack will help if high speed Ethernet devices are in use and the processor is fast enough to keep up.

It is recommended that the SMTP service interval be set to match the TCP/IP stack clock frequency. Although faster SMTP service may improve throughput, it will also introduce further burden on the processor.

The SMTP specification provides specific timeout values for most of the phases of an SMTP transaction. The default KwikNet SMTP timeout intervals are set to match the recommendations offered in RFC-2821. Your client or server can use procedure *`knsm_ioctl()`* to override any of these timeout intervals.

2. KwikNet SMTP Services

2.1 Introduction to SMTP Services

The KwikNet SMTP option provides a full set of SMTP services for use by your SMTP client and server. These service procedures reside in the KwikNet Library which you must link with your application.

A description of these KwikNet SMTP service procedures is provided in Chapter 2.2. The descriptions are ordered alphabetically for easy reference.

Italics are used to distinguish programming examples. Procedure names and variable names that appear in narrative text are also italicized. Occasionally a lower case procedure name or variable name may appear capitalized if it occurs as the first word in a sentence.

Vertical ellipses are used in program examples to indicate that a portion of the program code is missing. Most frequently this will occur in examples where fragments of application dependent code are missing.

```
:
: /* Continue processing */
:
```

Capitals are used for all defined KwikNet file names, constants and error codes. All KwikNet procedure, structure and constant names can be readily identified according to the nomenclature introduced in Chapter 1.3 of the KwikNet TCP/IP Stack User's Guide.

KwikNet Procedure Descriptions

A consistent style has been adopted for the description of the KwikNet SMTP service procedures. The procedure name is presented at the extreme top right and left as in a dictionary. This method of presentation has been chosen to make it easy to find procedures since they are ordered alphabetically.

Purpose A one-line statement of purpose is always provided.

Used by □ Client Task □ Client Callback ■ Server Task □ Server Callback □ ISP □ Timer Procedure

This block is used to indicate which application procedures can call the KwikNet procedure. A filled in box indicates that the procedure is allowed to call the KwikNet procedure. In the above example, only your application server task would be allowed to call the procedure.

For AMX users, this block is used to indicate which of your AMX application procedures can call the KwikNet procedure. You are reminded that the term ISP refers to the Interrupt Handler of a conforming ISP. AMX Timer Procedures, Restart Procedures and Exit Procedures must not call the KwikNet SMTP service procedures unless documented otherwise.

...more

Used by ☐ Client Task ☐ Client Callback ■ Server Task ☐ Server Callback ☐ ISP ☐ Timer Procedure

For other multitasking systems, the client or server task is an application task executing at a priority below that of the KwikNet Task. A timer procedure is any function executed by a task of higher priority than the KwikNet Task. An ISP is a KwikNet device driver interrupt handler called from an RTOS compatible interrupt service routine.

For a single threaded system, your App-Task (see glossary in Appendix A of the KwikNet TCP/IP Stack User's Guide) is the only task. The client task executes as part of your App-Task. The server task is executed by the KwikNet Task in the KwikNet domain. An ISP is a KwikNet device driver interrupt handler called from an interrupt service routine. Timer procedures do not exist.

Setup The prototype of the KwikNet procedure is shown.
The KwikNet header file in which the prototype is located is identified.
Include KwikNet header files *KN_LIB.H* and *KN_SMTP.H* for compilation.

File *KN_LIB.H* is the KwikNet include file that corresponds to the KwikNet Library that your application uses. This file is created for you by the KwikNet Configuration Manager when you create your KwikNet Library. File *KN_LIB.H* automatically includes the correct subset of the KwikNet header files for a particular target processor.

File *KN_SMTP.H* is the KwikNet include file which you must include if your application uses SMTP client or server services. This file is located in KwikNet installation directory *SMTP*. It is copied to the Treck installation directory, say *C:\TRECK\INCLUDE*, when you build the KwikNet Library.

Description Defines all input parameters to the procedure and expands upon the purpose or method if required.

Returns The outputs, if any, produced by the procedure are always defined. Most KwikNet procedures return an integer error status.

Restrictions If any restrictions on the use of the procedure exist, they are described.

Note Special notes, suggestions or warnings are offered where necessary.

See Also A cross-reference to other related KwikNet procedures is always provided if applicable.

2.2 SMTP Service Procedures

KwikNet provides a collection of service procedures for use by your SMTP client or server. These service procedures reside in the KwikNet Library which you must link with your application.

The following list summarizes these KwikNet SMTP service procedures. They are grouped functionally for easy reference.

Client Operations

knsm_create	Create an instance of a KwikNet SMTP client
knsm_delete	Delete an instance of a KwikNet SMTP client
knsm_open	Open a connection to an SMTP server
knsm_close	Close the connection to the SMTP server
knsm_mailfrom	Initiate mail transfer from a particular mailbox
knsm_sendto	Identify a particular mailbox to which mail is to be delivered
knsm_data	Send the mail message, including header and body
knsm_literal	Send an RSET, VRFY, HELP or NOOP command

Server Operations

knsm_start	Create and start a KwikNet SMTP server
knsm_stop	Stop a KwikNet SMTP server

Common Client and/or Server Operations

knsm_ioctl	Read or modify SMTP client, server or session parameters: Network addresses Callback functions and parameters Timing parameters Server reply code and reply string Server help response Client or server domain name
knsm_status	Generate a status log for a KwikNet SMTP client or server or for a client session managed by a KwikNet server

Purpose **Close the Connection to the SMTP Server**

Used by ■ Client Task □ Client Callback □ Server Task □ Server Callback □ ISP □ Timer Procedure

Setup Prototype is in file *KN_SMTP.H*.

```
#include "KN_LIB.H"
#include "KN_SMTP.H"
int knsm_close(KN_SMD smd);
```

Description *Smd* is the SMTP descriptor identifying the SMTP client session with an open SMTP connection which is to be closed. A KwikNet SMTP client task calls this function to end its session with its SMTP server.

Returns If successful, a value of *0* is returned. The SMTP session with the SMTP server is terminated and the SMTP connection to the server is broken.

The SMTP descriptor remains valid and can be used to open another SMTP connection.

On failure, one of the following error codes is returned.

KN_ERSMND	The SMTP descriptor *smd* is invalid.
KN_ERSMNOSESS	A client session has not been opened.
KN_ERSMNET	Network error (socket or TCP) encountered.
KN_ERSMTMREPLY	Timed out waiting for reply from server.

See Also *knsm_open()*

Purpose **Create an Instance of an SMTP Client**

Used by ■ Client Task □ Client Callback □ Server Task □ Server Callback □ ISP □ Timer Procedure

Setup Prototype is in file *KN_SMTP.H*.

```
#include "KN_LIB.H"
#include "KN_SMTP.H"
int knsm_create(KN_SMD *smdp);
```

Description *Smdp* is a pointer to storage for the SMTP descriptor which will be assigned by KwikNet to identify the SMTP client.

Returns If successful, a value of *0* is returned. A valid SMTP descriptor is stored at **smdp*. This descriptor must be used in all subsequent calls by the SMTP client task to identify the particular SMTP client.

On failure, one of the following error codes is returned.

KN_ERPARAM	Parameter *smdp* is *NULL*.
KN_ERNOMEM	Memory is not available for a new SMTP descriptor.

See Also *knsm_open()*, *knsm_delete()*

Purpose **Send Mail Data to the SMTP Server**

Used by ■ Client Task □ Client Callback □ Server Task □ Server Callback □ ISP □ Timer Procedure

Setup Prototype is in file *KN_SMTP.H*.

```
#include "KN_LIB.H"
#include "KN_SMTP.H"
int knsm_data(KN_SMD smd, const char *bufp, int len);
```

Description *Smd* is an SMTP descriptor identifying the SMTP client session which is sending mail. A KwikNet SMTP client task calls this function to send mail data to its server.

Bufp is a pointer to the buffer of application mail data to be sent. When all mail data has been sent, call *knsm_data()* with parameter *bufp* set to *NULL* as an end of message indicator.

Len is the buffer size, measured in bytes, if data is being presented in stream mode. *Len* is 0 if data is being presented in string mode. When *bufp* is *NULL*, parameter *len* is ignored.

When sending mail data in **stream mode**, each buffer specified by *bufp* contains *len* bytes of mail data for transmission. The data stream must contain lines of mail text with each line terminated by the *<CRLF>* character sequence (ASCII character *0x0D* followed by *0x0A*). Text lines are **not** separated by the C end of string character *'\0'*. A buffer of mail data can end with a partial mail line as long as the text at the head of the next buffer completes the mail line.

If *len* is *0*, mail data is transferred in **string mode**. Parameter *bufp* is a pointer to a *'\0'* terminated text string. The text string can provide multiple lines of mail data as long as the *<CRLF>* end of line characters are used to separate the mail lines within the string. The *<CRLF>* string will be appended to the string referenced by *bufp* if not already present.

In stream mode, the maximum length of any one mail line in the data stream must not exceed the SMTP limit of 1000 characters. In string mode, the maximum length of the string referenced by *bufp*, including the extra *<CRLF>* string if required, must not exceed 1000 characters.

...more

Returns If successful, a value of *0* is returned.

On failure, one of the following error codes is returned.

KN_ERSMND	The SMTP descriptor *smd* is invalid.
KN_ERSMNOSESS	A client session has not been opened.
KN_ERSMSEQ	The data request is out of sequence. You must identify the sender and recipients before sending mail.
KN_ERSMNET	Network error (socket or TCP) encountered.
KN_ERSMSIZE	The length of a text line (including <*CRLF*>) exceeds the 1000 character SMTP limit.
KN_ERSMMODE	String mode is invalid: partial line not yet sent.
KN_ERSMREJECT	The server declined the initial request to send data.
KN_ERSMTM_DATA	Timed out waiting for initial reply from server.
KN_ERSMTM_SEND	Timed out trying to send mail data.
KN_ERSMTM_EOM	Timed out waiting for final reply from server.

Note When you first call this function to start sending mail data, the SMTP server is notified. If the server does not accept the request to start data transmission, your client task will see the *KN_ERSMREJECT* error code.

Note The mail data text lines are always transmitted as complete lines of text properly terminated by the <*CRLF*> character pair as required by the SMTP protocol. When mail data is presented in stream mode, partial lines of text are never transmitted. If you attempt to switch from stream mode to string mode with a partial mail line still untransmitted, your call will be rejected with the *KN_ERSMMODE* error code.

See Also *knsm_open()*, *knsm_mailfrom()*, *knsm_sendto()*

Purpose **Delete an Instance of a KwikNet SMTP Client**

Used by ■ Client Task □ Client Callback □ Server Task □ Server Callback □ ISP □ Timer Procedure

Setup Prototype is in file *KN_SMTP.H*.

```
#include "KN_LIB.H"
#include "KN_SMTP.H"
int knsm_delete(KN_SMD smd);
```

Description *Smd* is the SMTP descriptor identifying the SMTP descriptor which is being discarded by a KwikNet SMTP client task.

Returns If successful, a value of *0* is returned. The SMTP client task's SMTP descriptor is deleted and all resources associated with it are released.

On failure, one of the following error codes is returned.

KN_ERSMND	The SMTP descriptor *smd* is invalid.
KN_ERSMINSESS	The SMTP client still has an active SMTP session.

Restrictions An SMTP client's session must be closed before the client's SMTP descriptor can be deleted.

Note It is recommended that only a client task delete its instance of a KwikNet SMTP client.

See Also *knsm_create()*

Purpose **Read or Modify SMTP Network Parameters**

Used by ■ Client Task ■ Client Callback □ Server Task ■ Server Callback □ ISP □ Timer Procedure

Setup Prototype is in file *KN_SMTP.H*.

```
#include "KN_LIB.H"
#include "KN_SMTP.H"
#include "KN_SOCK.H"
int knsm_ioctl(KN_SMD smd, int opcode, void *addrp);
```

Description *Smd* is the SMTP descriptor identifying the SMTP entity of interest. SMTP entities include a KwikNet SMTP client or server or any client session which the server manages.

Opcode is an operation code identifying the network parameter to be read or modified. The allowable operations are specified by the following ioctl values which are defined in header file *KN_SMTP.H*.

KN_SMTIOC_GETLADR	Get local network IPv4 address and port
KN_SMTIOC_GETFADR	Get foreign network IPv4 address and port
KN_SMTIOC_SETLADR	Set local server network IPv4 address and port

Addrp is a pointer to a socket address structure used to specify the IPv4 address and port number for one end of a socket connection. The socket address structure *sockaddr_in* is defined in the Treck sockets header file *TRSOCKET.H* located in the *TRECK\INCLUDE* directory.

When setting the network address, the structure must be initialized to contain the IPv4 address in structure member *addrp->sin_addr* and the port number in member *addrp->sin_port*. When reading the network address, these structure members will be filled upon return. Both IP address and port number are specified in net endian form. All other structure members are unused by this procedure.

Returns If successful, a value of *0* is returned.

On failure, one of the following error codes is returned.

KN_ERSMND	The SMTP descriptor *smd* is invalid.
KN_ERPARAM	*Opcode* is invalid **or** *addrp* is *NULL*.
KN_ERSMREFUSE	Operation requested by caller is prohibited.
KN_ERSMNOCONN	There is no SMTP connection.

Restriction An SMTP server callback function can set the server's local network address while handling the server's *KN_SMCBC_PREP* request. Any other request to modify the local network address specified in the SMTP descriptor will be refused with a *KN_ERSMREFUSE* error return.

Purpose **Read or Modify SMTP Callback Settings**

Used by ■ Client Task ■ Client Callback □ Server Task ■ Server Callback □ ISP □ Timer Procedure

Setup Prototype is in file *KN_SMTP.H*.

```
#include "KN_LIB.H"
#include "KN_SMTP.H"
int knsm_ioctl(KN_SMD smd, int opcode, void *cbvarp);
```

Description *Smd* is the SMTP descriptor identifying the SMTP entity of interest. SMTP entities include a KwikNet SMTP client or server or any client session which the server manages.

Opcode is an operation code identifying the callback setting to be read or modified. The allowable operations are specified by the following ioctl values which are defined in header file *KN_SMTP.H*.

KN_SMTIOC_GETCBF	Get callback function pointer
KN_SMTIOC_SETCBF	Set callback function pointer
KN_SMTIOC_GETCBP	Get callback parameter
KN_SMTIOC_SETCBP	Set callback parameter

Cbvarp is a pointer to a callback variable. The callback variable provides the new value or storage for a copy of the current value of one of the callback settings.

When reading or installing the pointer to a callback function, parameter *cbvarp* is a pointer to a callback variable of type *KN_SMCALLBACK*. The callback variable is therefore a pointer to a callback function.

When reading or installing a callback parameter, *cbvarp* is a pointer to an actual callback parameter, which, by definition, is a pointer to *void*.

Returns If successful, a value of *0* is returned.

On failure, one of the following error codes is returned.

KN_ERSMND	The SMTP descriptor *smd* is invalid.
KN_ERPARAM	*Opcode* is invalid **or** *cbvarp* is *NULL*.
KN_ERSMREFUSE	An SMTP server cannot remove its callback function.

Note An SMTP client does not normally have a callback function. A client can use this procedure to install a callback function in order to gain access to the reply strings from the SMTP server. The client can subsequently remove its the callback function by setting the callback function pointer to *(KN_SMCALLBACK)0L*.

Restriction You cannot remove the callback function for an SMTP server or for any client session which it is handling.

knsm_ioctl
(timing parameters)

Purpose **Modify SMTP Timing Parameters**

Used by ■ Client Task ■ Client Callback □ Server Task ■ Server Callback □ ISP □ Timer Procedure

Setup Prototype is in file *KN_SMTP.H*.

```
#include "KN_LIB.H"
#include "KN_SMTP.H"
int knsm_ioctl(KN_SMD smd, int opcode, void *timep);
```

Description This procedure is used to modify the SMTP timeout parameters used by an SMTP client or server to meet the SMTP protocol timing specifications.

Smd is the SMTP descriptor identifying the SMTP entity of interest. SMTP entities include a KwikNet SMTP client or server or any client session which the server manages.

Opcode is the operation code *KN_SMTIOC_SETTMVAL* used to indicate that the caller wishes to modify one or more SMTP timing parameters.

Timep is a pointer to an array of two parameters of type *long*. The first long value is a bit mask which identifies the particular timeout parameters to be adjusted. The second long value is the timeout interval measured in seconds.

The timeout parameters are specified by ORing together one or more of the following bit mask values which are defined in header file *KN_SMTP.H*. The default timeout values are shown in brackets. Any bit mask value not specified in this list is ignored.

KN_SMTTMR_CONNECT	Server reply to connect request	(300 sec)
KN_SMTTMR_MAIL	Server reply to MAIL command	(300 sec)
KN_SMTTMR_RCPT	Server reply to RCPT command	(300 sec)
KN_SMTTMR_DATA	Server reply to permit data transfer	(120 sec)
KN_SMTTMR_SEND	Network accepts data for transfer	(180 sec)
KN_SMTTMR_EOM	Server reply to end data transfer	(600 sec)
KN_SMTTMR_CMD	Server waits for client command Client waits for server reply to commands for which unique timeouts are not defined.	(300 sec)

Returns If successful, a value of *0* is returned.

On failure, one of the following error codes is returned.

KN_ERSMND	The SMTP descriptor *smd* is invalid.
KN_ERPARAM	*Opcode* is invalid **or** *timep* is *NULL*.

Purpose **Modify SMTP Server Reply Message**

Used by □ Client Task □ Client Callback □ Server Task ■ Server Callback □ ISP □ Timer Procedure

Setup Prototype is in file *KN_SMTP.H*.

```
#include "KN_LIB.H"
#include "KN_SMTP.H"
int knsm_ioctl(KN_SMD smd, int opcode, void *errp);
```

Description This procedure can be used by your server callback function to modify the SMTP reply message which the KwikNet SMTP server will send to the client once your callback function indicates the success or failure of the action requested by the client.

Smd is the SMTP descriptor identifying the SMTP server's client session.

Opcode is an operation code which indicates how the server's reply is to be modified. The allowable operations are specified by the following ioctl values which are defined in header file *KN_SMTP.H*.

KN_SMTIOC_SETERRNUM	Set SMTP error number
KN_SMTIOC_SETERRMSG	Set SMTP error message

If opcode is *KN_SMTIOC_SETERRNUM*, *errp* is a pointer to an *unsigned int* which provides the 3 digit numeric reply code to be sent to the client in the reply message.

If opcode is *KN_SMTIOC_SETERRMSG*, *errp* is a pointer to a *'\0'* terminated string to be concatenated with the current reply code to create the reply message to be sent to the client.

Returns If successful, a value of *0* is returned.

On failure, one of the following error codes is returned.

KN_ERSMND	The SMTP descriptor *smd* is invalid.
KN_ERPARAM	*Opcode* is invalid **or** *errp* is *NULL*.
KN_ERSMREFUSE	Operation requested by caller is prohibited. *smd* is not a descriptor for a server's client session.

Note If your server callback function modifies the error message and/or error number, your modified values will be used in the server's reply to its client. Your modifications must constitute a valid server reply for the SMTP command processed by your callback function.

If your callback function returns *KN_SMCBR_OK* to the server task, your server reply must indicate that the SMTP command was handled successfully. Otherwise it must explain the reason for the failure of the client's SMTP command.

knsm_ioctl
(help response)

Purpose **Install a Help Reply for the SMTP Server**

Used by □ Client Task □ Client Callback □ Server Task ■ Server Callback □ ISP □ Timer Procedure

Setup Prototype is in file *KN_SMTP.H*.

```
#include "KN_LIB.H"
#include "KN_SMTP.H"
int knsm_ioctl(KN_SMD smd, int opcode, void *paramp);
```

Description *Smd* is the SMTP descriptor identifying a KwikNet SMTP server task or any client session which such a server manages.

Opcode is the operation code *KN_SMTIOC_SETHELP* used to indicate that the caller wishes to modify the SMTP server's help reply message.

Paramp is a pointer to a **constant**, *'\0'* terminated help string. The string must consist of a properly formatted SMTP server reply. The reply code must be present in the string. The string must end with the SMTP *<CRLF>* end of line character sequence (ASCII character *0x0D* followed by *0x0A*). The string can consist of multiple lines of text, provided that lines are separated by the *<CRLF>* character pair and all but the last line include the continuation character *'-'* after the reply code.

If *paramp* is *NULL*, the server will subsequently respond to a client's help request with an error reply indicating that the HELP command is not supported.

The following are examples of acceptable help reply strings:

```
const char help1[] = "214 KwikNet SMTP Server\015\012";

const char help2[] = "214-KwikNet SMTP Server\015\012"
    "214-Commands supported:\015\012"
    "214-  HELO, EHLO (no service extensions),\015\012"
    "214-  VRFY, MAIL, RCPT, DATA, QUIT,\015\012"
    "214   RSET, NOOP\015\012";
```

Returns If successful, a value of *0* is returned.

On failure, one of the following error codes is returned.

KN_ERSMND	The SMTP descriptor *smd* is invalid.
KN_ERPARAM	*Opcode* is invalid.
KN_ERSMREFUSE	Operation requested by caller is prohibited. *smd* is not a descriptor for an SMTP server or for a server's client session.

Restriction Your help string is not copied for use by the server. The string referenced by *paramp* must remain valid until the help reply string is replaced with another call to *knsm_ioctl()* or until the server is stopped.

Purpose **Install an SMTP Client or Server Domain Name**

Used by ■ Client Task □ Client Callback □ Server Task ■ Server Callback □ ISP □ Timer Procedure

Setup Prototype is in file *KN_SMTP.H*.

```
#include "KN_LIB.H"
#include "KN_SMTP.H"
int knsm_ioctl(KN_SMD smd, int opcode, void *paramp);
```

Description *Smd* is the SMTP descriptor identifying the KwikNet SMTP client or server task whose domain name is to be revised.

Opcode is the operation code *KN_SMTIOC_SETNAME* used to indicate that the caller wishes to modify the domain name to be used to identify a particular SMTP client or server task.

Paramp is a pointer to a **constant**, *'\0'* terminated domain name string. The string must consist of a properly formatted domain name which specifies a local network interface by which the client or server can be identified, usually via a Domain Name Server.

If *paramp* is *NULL*, the previously installed domain name, if any, will be discarded.

Returns If successful, a value of *0* is returned.

On failure, one of the following error codes is returned.

KN_ERSMND	The SMTP descriptor *smd* is invalid.
KN_ERPARAM	*Opcode* is invalid **or** **param* references an empty or overlength string.

Restriction Your domain name string is not copied for use by the client or server. The string referenced by *paramp* must remain valid until the domain name string is replaced with another call to *knsm_ioctl()* or until the client or server ceases to operate.

Restriction A domain name string must fit within an SMTP command or reply line.

Note If a domain name is never installed or is removed, a domain name for your client or server will be derived from the IPv4 address of the network interface being used for the SMTP connection. The domain name will be the IPv4 address expressed using the dotted decimal notation.

Purpose **Send a Literal Command to an SMTP Server**

Used by ■ Client Task □ Client Callback □ Server Task □ Server Callback □ ISP □ Timer Procedure

Setup Prototype is in file *KN_SMTP.H*.

```
#include "KN_LIB.H"
#include "KN_SMTP.H"
int knsm_literal(KN_SMD smd, int cmd, long param);
```

Description *Smd* is an SMTP descriptor identifying the SMTP client session which is sending an SMTP command. A KwikNet SMTP client task calls this function to send a particular SMTP command to its server.

Cmd identifies the SMTP command to be sent to the server. The following commands defined in header file *KN_SMTP.H* are supported:

KN_SMTCMD_RSET	Reset
KN_SMTCMD_HELP	Ask for server's help message
KN_SMTCMD_NOOP	No operation
KN_SMTCMD_VRFY	Verify a user or mailbox name

Param is a command specific parameter.

For command *KN_SMTCMD_VRFY*, *param* is a pointer to a *'\0'* terminated text string. The string is the name of the user or mailbox whose accessibility is to be verified. The name must be of the format *<local@domain>*.

For all other commands, *param* is unused.

Returns If successful, a value of *0* is returned.

On failure, one of the following error codes is returned.

KN_ERSMND	The SMTP descriptor *smd* is invalid.
KN_ERPARAM	Invalid command **or** *param* is *NULL* when a pointer is required **or** **param* references an empty or overlength string.
KN_ERSMNOSESS	A client session has not been opened.
KN_ERSMSEQ	The command cannot be issued while sending data.
KN_ERSMNET	Network error (socket or TCP) encountered.
KN_ERSMREJECT	The server rejected the user or mailbox name identified by *param* in a verification request.
KN_ERSMTMREPLY	Timed out waiting for reply from server.

Note If your client task sends the HELP command without first installing a callback function, it will not have access to the lines of text returned by the SMTP server in its help reply.

Purpose **Identify the Mail Sender**

Used by ■ Client Task □ Client Callback □ Server Task □ Server Callback □ ISP □ Timer Procedure

Setup Prototype is in file *KN_SMTP.H*.

```
#include "KN_LIB.H"
#include "KN_SMTP.H"
int knsm_mailfrom(KN_SMD smd, const char *srcname);
```

Description *Smd* is an SMTP descriptor identifying the SMTP client session which is initiating a new mail transaction. A KwikNet SMTP client task calls this function to start sending a mail message to its server.

Srcname is a pointer to a *'\0'* terminated text string which identifies the mail sender. The string is the name of a mailbox to which the server or mail recipients can send return mail. The name must be of the format *<local@domain>*.

Returns If successful, a value of *0* is returned.

On failure, one of the following error codes is returned.

KN_ERSMND	The SMTP descriptor *smd* is invalid.
KN_ERPARAM	*Srcname* is *NULL* **or** **srcname* references an empty or overlength string.
KN_ERSMNOSESS	A client session has not been opened.
KN_ERSMSEQ	The request is out of sequence (see restriction).
KN_ERSMNET	Network error (socket or TCP) encountered.
KN_ERSMREJECT	The server refused to accept mail from the sender identified by *srcname*.
KN_ERSMTMREPLY	Timed out waiting for reply from server.

Restriction This function can only be used to start a new mail transaction. You must not call this function once a mail transaction is in progress. You can use *knsm_literal()* to send a RSET command to the server to end an active mail transaction. You can then call *knsm_mailfrom()* to start a new mail transaction.

See Also *knsm_sendto()*

Purpose **Open a Connection to an SMTP Server**

Used by ■ Client Task □ Client Callback □ Server Task □ Server Callback □ ISP □ Timer Procedure

Setup Prototype is in file *KN_SMTP.H*.

```
#include "KN_LIB.H"
#include "KN_SMTP.H"
int knsm_open(KN_SMD smd, struct in_addr *inadrp, int port);
```

Description *Smd* is the SMTP descriptor identifying the SMTP client task wishing to connect to its SMTP server and open an SMTP client session.

Inadrp is a pointer to a structure containing the IPv4 address, in net endian form, of the SMTP server with whom an SMTP session is to be established. The BSD structure *in_addr* is defined as follows in Treck header file *TRSOCKET.H* located in the *TRECK\INCLUDE* directory:

```
struct in_addr {
  u_long s_addr;              /* IPv4 address (net endian)*/
  };
```

Port is the port number for the SMTP server. If parameter *port* is *0*, the well known SMTP port number 25 will be used.

Returns If successful, a value of *0* is returned.

On failure, one of the following error codes is returned.

KN_ERSMND	The SMTP descriptor *smd* is invalid.
KN_ERSMINSESS	Client session is already in progress.
KN_ERSMSOCK	Cannot allocate socket for connection.
KN_ERSMNOCONN	The server has refused the connection.
KN_ERSMNOSESS	The server has refused to establish a client session.
KN_ERSMTMCONN	Timed out waiting for connection reply from server.
KN_ERSMTMREPLY	Timed out waiting for session reply from server.
KN_ERSMNET	Network error (socket or TCP) encountered.

Note If an SMTP client already has a connection to a server, any attempt to open another connection without first closing the existing connection will be rejected with a *KN_ERSMINSESS* error indication.

Note A TCP/IP connection will be established with the specified SMTP server. The local and foreign IPv4 addresses and port numbers associated with the connection can be read using procedure *knsm_ioctl()* (network parameters).

See Also *knsm_close()*

Purpose **Identify a Mail Recipient**

Used by ■ Client Task □ Client Callback □ Server Task □ Server Callback □ ISP □ Timer Procedure

Setup Prototype is in file *KN_SMTP.H*.

```
#include "KN_LIB.H"
#include "KN_SMTP.H"
int knsm_sendto(KN_SMD smd, const char *dstname);
```

Description *Smd* is an SMTP descriptor identifying the SMTP client session which is sending mail to a specific mailbox. A KwikNet SMTP client task calls this function to identify the mailbox to which the server must deliver the mail message.

Dstname is a pointer to a *'\0'* terminated text string which identifies a particular mail recipient. The string is the name of a mailbox to which the server must deliver or forward the mail. The name must be of the format <*local@domain*>.

Returns If successful, a value of *0* is returned.

On failure, one of the following error codes is returned.

KN_ERSMND	The SMTP descriptor *smd* is invalid.
KN_ERPARAM	*Dstname* is *NULL* **or** **dstname* references an empty or overlength string.
KN_ERSMNOSESS	A client session has not been opened.
KN_ERSMSEQ	The request is out of sequence (see restriction).
KN_ERSMNET	Network error (socket or TCP) encountered.
KN_ERSMREJECT	The server refused to accept mail destined for the mailbox identified by *dstname*.
KN_ERSMTMREPLY	Timed out waiting for reply from server.

Restriction This function can only be used after a mail transaction has been started and before the mail data transmission has been initiated. To specify multiple mail recipients, you must call this function repeatedly until you have identified each of the destination mailbox addresses. You will not be permitted to send mail data unless the SMTP server has accepted at least one of your intended recipients.

See Also *knsm_data()*, *knsm_mailfrom()*

Purpose **Create and Start a KwikNet SMTP Server**

Used by □ Client Task □ Client Callback ■ Server Task □ Server Callback □ ISP □ Timer Procedure

Setup Prototype is in file *KN_SMTP.H*.

```
#include "KN_LIB.H"
#include "KN_SMTP.H"
int knsm_start(KN_SMD *smdp,
                  int (*cbfn)(KN_SMD smd, int opcode,
                        char *p1, int p2, void *param),
                  void *param)
```

Description *Smdp* is a pointer to storage for the SMTP descriptor which will be assigned by KwikNet to identify the SMTP server.

Cbfn is a pointer to a server callback function which the SMTP server will call to report SMTP session activity. This function must be coded to operate as described in Chapter 1.7.

Param is a parameter which the caller wishes to pass to the callback function *cbfn()*. If no parameter is required, set *param* to *NULL*.

Note Before the KwikNet SMTP server is started, you must identify the network IPv4 address and port number which it is to serve. When a KwikNet SMTP server is created, it is assigned the IP address *INADDR_ANY* and the well known SMTP port number 25. Hence, clients will be able to connect to the SMTP server using the IP address of any network interface in the computer on which the SMTP server is operating. If you wish, your server callback function can use service procedure *knsm_ioctl()* (network parameters) to alter the server's IPv4 address and port number when it receives the server's *KN_SMCBC_PREP* request.

...more

Returns (Multitasking Operation)

This procedure must only be called by the application task which will assume the role of SMTP server. All KwikNet SMTP server operations will be performed in the context of this task.

If the server is successfully started, there will be no return from this procedure until the KwikNet SMTP server is requested to stop. At that time there will be a return to the *knsm_start()* caller.

A value of *0* is returned if, and only if, the KwikNet SMTP server starts successfully and eventually stops without error.

On failure, one of the following error codes is returned.

KN_ERPARAM	Parameter *smdp* or *cbfn* is *NULL*.
KN_ERSMFAIL	Server failed to start or stop successfully.

Failure code *KN_ERSMFAIL* indicates that the KwikNet SMTP server cannot be started, was forced to abort because of a serious, unrecoverable fault or terminated with an error condition when requested to stop.

Returns (Single Threaded Operation)

This procedure must be called from your App-Task. The SMTP server will be added to the KwikNet server queue. Thereafter, all KwikNet SMTP server operations will be performed in the KwikNet domain in the context of the KwikNet Task.

A value of *0* is returned if the KwikNet SMTP server is successfully started.

On failure, one of the following error codes is returned.

KN_ERPARAM	Parameter *smdp* or *cbfn* is *NULL*.
KN_ERSMFAIL	Server cannot be started.

See Also *knsm_status()*, *knsm_stop()*

Purpose **Generate a Status Log for an SMTP Client, Server or Session**

Used by ■ Client Task □ Client Callback □ Server Task ■ Server Callback □ ISP □ Timer Procedure

Setup Prototype is in file *KN_SMTP.H*.

```
#include "KN_LIB.H"
#include "KN_SMTP.H"
int knsm_status(KN_SMD smd);
```

Description *Smd* is the SMTP descriptor identifying the SMTP server or client session for which a status summary is to be generated. The client session can be that of a KwikNet SMTP client or any of those being serviced by a KwikNet SMTP server.

Returns If successful, a value of *0* is returned and an SMTP status summary for descriptor *smd* will be generated on the KwikNet logging device.

On failure, one of the following error codes is returned.

KN_ERSMND	The SMTP descriptor *smd* is invalid.
KN_ERSMREFUSE	Operation prohibited from a client callback function.
KN_ERSMNOLOG	SMTP client (server) statistics are not enabled.

Note If SMTP client statistics are not enabled in the KwikNet Library, an SMTP status summary for a KwikNet client will not be generated.

If SMTP client statistics are enabled, the KwikNet client's status summary will be generated as a sequence of messages of class *KN_PA_C_SMTP*.

If SMTP server statistics are not enabled in the KwikNet Library, an SMTP status summary for a KwikNet server or for a client session being handled by the server will not be generated.

If SMTP server statistics are enabled, the status summary for a KwikNet server or any of its client sessions will be generated as a sequence of messages of class *KN_PA_S_SMTP*.

Restriction If a server callback function initiates a client session status summary, the log will be generated by the SMTP server at the first idle opportunity.

Purpose **Stop a KwikNet SMTP Server**

Used by □ Client Task □ Client Callback □ Server Task ■ Server Callback □ ISP □ Timer Procedure ■ Application Task

Setup Prototype is in file *KN_SMTP.H*.

```
#include "KN_LIB.H"
#include "KN_SMTP.H"
int knsm_stop(KN_SMD smd,
              void (*shutdownfn)(int, unsigned long),
              unsigned long param);
```

Description *Smd* is the SMTP descriptor of a KwikNet SMTP server which is to be stopped.

Shutdownfn is a pointer to a stoppage function which the SMTP server will call when it has shut down and is about to return to the point at which a call to *knsm_start()* started the server. The stoppage function will execute in the context of the stopped SMTP server.

If you do not require notification when the SMTP server shutdown is complete, set *shutdownfn* to *(void (*)(int, unsigned long))0L*.

Param is a parameter which will be passed to your stoppage function *shutdownfn()*. If a stoppage function is not provided, set *param* to *0L*. The call to the stoppage function is of the form:

```
(*shutdownfn)(error, param);
```

Parameter *error* reflects the success or failure of the KwikNet SMTP server as it stopped. *Error* will be *0* if the server stopped successfully. *Error* will be *KN_ERSMFAIL* if the server encountered error conditions while trying to stop.

Returns A value of *0* is returned if, and only if, the KwikNet SMTP server accepts the request, thereby acknowledging that the SMTP server is willing to shut down. The value *0* will be returned even if the KwikNet server eventually terminates with an error condition.

On failure, one of the following error codes is returned.

KN_ERSMND	The SMTP descriptor *smd* is invalid.
KN_ERSMREFUSE	Operation prohibited; server has not been started.

Note If one or more SMTP client connections are open at the time a server is stopped, all such client sessions will be aborted before the server stops.

See Also *knsm_start()*, *knsm_status()*