NetX Simple Mail Transfer Protocol for Clients (NetX SMTP Client)

User Guide

Express Logic, Inc.
858.613.6640
Toll Free 888.THREADX
FAX 858.521.4259
www.expresslogic.com

Part Number: 000-1054
Revision 5.10

Contents

Chapter 1

Introduction to NetX SMTP Client

The Simple Mail Transfer Protocol (SMTP) is a protocol for transferring mail across networks and the Internet. It utilizes the reliable Transmission Control Protocol (TCP) services to perform its content transfer function.

NetX SMTP Client Requirements

The NetX SMTP Client requires creation of a NetX IP instance and NetX packet pool. The SMTP Client uses a TCP socket to connect to an SMTP Server on the well-known port 25. Therefore, TCP must first be enabled by calling the nx_tcp_enable service on a previously created IP instance.

The SMTP Client create service, *nx_smtp_client_create*, requires a previously created packet pool for transmitting SMTP commands to the Server as well as for sending the actual mail message. Packet payload depends on the anticipated size of the mail contents and must allow for TCP, IP header, and MAC header. (Note that the TCP and IP headers are 20 bytes.)

If the entire mail message cannot fit in one packet, the SMTP Client allocates additional packets to contain the rest of the message.

NetX SMTP Client Constraints

While the NetX SMTP protocol implements the RFC 2821 and 2554 standards, there are some constraints:

1. The NetX SMTP Client supports only LOGIN and PLAIN authentication, but not CRAM-MD5 digest authentication.

2. The NetX SMTP Client messages are limited to one recipient per mail item, and only one mail message per TCP connection with the SMTP server.

3. VRFY, SEND, SOML, EXPN, SAML, ETRN, TURN and SIZE SMTP options are not supported.

4. The SMTP Client is not mail browser (“mail user agent”) which is typically used for creating the mail message. It is a “mail transfer agent” only. It will provide the necessary processing of the mail message body for SMTP transport as specified in RFC 2821. It does not check the contents for correct syntax e.g. the recipient and reverse pathway. There is no restriction what is in the mail buffer e.g. MIME data or clear text messages. Mail message format, specified in RFC 2822 for including headers and message body is beyond the scope of the SMTP Client API.

Commands Supported by NetX SMTP Client

The NetX SMTP Client uses the following commands during a mail session with an SMTP Server.

Command	Meaning
EHLO	The Client would like to initiate a session that includes some or all extension protocol SMTP services available from the SMTP Server. This is the default.
HELO	The Client would like to initiate a session limited to basic SMTP services.
MAIL	The Client would like the Server to receive Client mail.
AUTH	The Client would like to initiate authentication by the Server.
RCPT	The Client would like to submit a mailbox of another host it would like the mail to be delivered to.
DATA	The Client would like to initiate sending mail message data to the Server.
QUIT	The Client would like to terminate the session.

Getting Started

The SMTP Client application creates an IP instance and an enables TCP on that IP instance. It then creates the SMTP Client using the following service:

```
UINT    nx_smtp_client_create(NX_SMTP_CLIENT *client_ptr,
                              NX_IP *ip_ptr, NX_PACKET_POOL
                              *client_packet_pool_ptr,
                              CHAR *username, CHAR *password,
                              CHAR *from_address,
                              CHAR *client_domain,
                              UINT authentication_type,
                              ULONG server_address, UINT port)
```

The *client_packet_pool_ptr* is a pointer to a previously created packet pool the SMTP Client will use to send messages to the SMTP Server.

Note that an application must provide a *from_address* for the local device and a server IP address. All addresses must be fully qualified domain names. A fully qualified domain name contains a local-part and a domain name, separated by an '@' character. Note that the SMTP Client does not check the syntax of the *from_address* or the *recipient_address* in the *nx_smtp_mail_send* service below.

After the SMTP Client is created, the SMTP Client application creates a mail item with a properly formatted SMTP mail message, and uses the following service to transmit the mail item:

```
UINT    nx_smtp_mail_send(NX_SMTP_CLIENT *client_ptr,
                          CHAR *recipient_address, UINT priority,
                          CHAR *subject, CHAR *mail_body,
                          UINT mail_body_length)
```

Note that an application wishing to send mail must provide a recipient address in the *nx_smtp_client_mail* call.

For authentication, usernames can either be fully qualified domain names, or display user names. This depends on how the Server performs authentication.

The demo in the Small Example section later in this User Guide shows how the message should be formatted. The status if the mail item was successfully sent will be NX_SUCCESS. If an error occurs, whether it is an internal error, a broken TCP connection or receiving a Server reply error code, *nx_smtp_mail_send* will return a non-zero error status.

When sending a mail item, NetX SMTP Client creates a new TCP connection with the SMTP server and begins an SMTP session. In this session, the Client sends a series of commands to the SMTP Server as

part of the SMTP protocol, culminating in sending out the actual mail message. The TCP connection is then terminated, regardless of the outcome of the SMTP session.

After mail transmission, regardless of success or failure, the SMTP Client is returned to the ‘initial’ state, and can be used for another mail transfer session.

NetX SMTP Authentication

Authentication is a way for SMTP Clients to prove their identity to the SMTP Server and have their mail delivered as trusted users. Most commercial SMTP Servers require that Clients be authenticated.

Typically, authentication data consists of the sender's username and password. During an authentication challenge, the Server prompts for this information and the Client responds by sending the requested data in encoded format. The Server decodes the data and attempts to find a match in its user database. If found, the Server indicates the authentication is successful. SMTP authentication is defined in RFC 2554.

There are two flavors of authentication, namely basic and digest. Digest is not supported in the current NetX SMTP Client, and will not be discussed here. Basic authentication is equivalent to the name and password authentication described above. In SMTP basic authentication, the name and passwords are base64 encoded. The advantage of basic authentication is its ease of implementation and widespread use. The main disadvantage of basic authentication is name and password data is transmitted openly in the request.

Plain Authentication

The NetX SMTP Client sends an AUTH command with the PLAIN parameter. If the NetX SMTP Server supports this type of authentication, it will reply with a 334 reply code. The Client replies with a single base64 encoded username and password message to the Server. If the Server determines the Client authentication is successful, it responds with the 235 success code.

Login Authentication

The NetX SMTP Client sends an AUTH command with the LOGIN parameter. If the NetX SMTP Server supports this type of authentication, it will reply with a 334 reply code as the start of the authentication 'challenge'. It sends a base64 encoded prompt back to the Client which is typically "Username". The Client decodes the prompt, and replies with a base64 encoded username. If the Server accepts the Client username, it sends out a base64 encoded prompt for the Client password. The Client responds with a base64 encoded password. If the Server determines the Client authentication is successful, it responds with the 235 success code.

RFCs Supported by NetX SMTP Client

NetX SMTP Client API is compliant with RFC2821 "Simple Mail Transfer Protocol" and RFC 2554 "SMTP Service Extension for Authentication. "

Chapter 2 Installation and Use of NetX SMTP Client

This chapter contains a description of various issues related to installation, setup, and usage of the NetX SMTP Client component.

NetX SMTP Client Installation

The NetX SMTP Client is shipped on a single CD-ROM compatible disk. The package includes the following files:

nx_smtp_client.c C	Source file for NetX SMTP Client API
nx_smtp_client.h C	Header file for NetX SMTP Client API
demo_netx_smtp_client.c	Demo for NetX SMTP Client
nx_smtp_client.pdf	User Guide for NetX SMTP Client API

To use the NetX SMTP Client API, the entire distribution mentioned previously may be copied to the same directory where NetX is installed. For example, if NetX is installed in the directory “c:\myproject” then the *nx_smtp_client.h*, and *nx_smtp_client.c* files should be copied into this directory.

Using NetX SMTP Client

To create the NetX SMTP Client application, it must first build the ThreadX and NetX libraries and include them in the build project. The application must then include *tx_api.h* and *nx_api.h* in its application source code. This will enable ThreadX and NetX services. It must also include *nx_smtp_client.c* and *nx_smtp_client.h* after *tx_api.h* and *nx_api.h* to use SMTP Client services.

These files must be compiled in the same manner as other application files and the object code must be linked along with the files of the application. This is all that is required to create a NetX SMTP Client application.

Small Example System

An example of using the NetX SMTP Client is described in Figure 1 that appears below. The packet pool for the IP instance is created using the nx_packet_pool_create service, on line 68 and has a very small packet payload. This is because the IP instance only sends control packets which don't require much payload. The SMTP Client packet pool created on line 84 and is used for transmitting SMTP Client messages to the server and message data. Its packet payload is much larger. The IP instance is created in line 118 using the same packet pool. TCP, required for the SMTP protocol, is enabled on the IP instance in line 130.

In the application thread, the SMTP Client is created using the *nx_smtp_client_create* service, in line 170. Then the mail message is submitted to the SMTP Client for transmission on line 184 using the *nx_smtp_mail_send* service. Note that the subject line with the mail content header is created separately from the message body. Also note that the send mail request accepts only one recipient mail address which is assumed to be syntactically correct.

Then the application terminates the SMTP Client on line 200. The *nx_smtp_client_delete* service checks that the socket connection is closed and the port is unbound. Note that it is up to the SMTP Client application to delete the packet pool if it no longer has use for it.

```
1    /*
2       demo_netx_smtp_client.c
3
4       This is a small demo of the NetX SMTP Client on the high-performance NetX
5       TCP/IP stack.  This demo relies on Thread, NetX and SMTP Client API to
6       perform simple SMTP mail transfers in an SMTP client application to an SMTP mail
7       server.   */
8
9    #include "nx_api.h"
10   #include "nx_ip.h"
11   #include "nx_smtp_client.h"
12
13
14   /* Define the host user name and mail box parameters */
15   #define USERNAME               "myusername"
16   #define PASSWORD               "mypassword"
17   #define FROM_ADDRESS           "my@mycompany.com"
18   #define RECIPIENT_ADDRESS      "your@yourcompany.com"
19   #define LOCAL_DOMAIN           "mycompany.com"
20
21   #define SUBJECT_LINE           "NetX SMTP Client Demo"
22   #define MAIL_BODY              "NetX SMTP client is an SMTP client \r\n" \
                                    "implementation for embedded devices to send  \r\n" \
23                                  "email to SMTP servers. This feature is \r\n" \
24                                  "intended to allow a device to send simple \r\n " \
25                                  "status reports using the most universal \r\n " \
26                                  "Internet application, email.\r\n"
```

```
27
28  /* See the NetX SMTP Client User Guide for how to set the authentication type.
29     The most common authentication type is PLAIN. */
30  #define CLIENT_AUTHENTICATION_TYPE 3
31
32
33  #define CLIENT_IP_ADDRESS  IP_ADDRESS(1,2,3,5)
34  #define SERVER_IP_ADDRESS  IP_ADDRESS(1,2,3,4)
35  #define SERVER_PORT        25
36
37
38  /* Define the NetX and ThreadX structures for the SMTP client appliciation. */
39  NX_PACKET_POOL                  ip_packet_pool;
40  NX_PACKET_POOL                  client_packet_pool;
41  NX_IP                           client_ip;
42  TX_THREAD                       demo_client_thread;
43  static NX_SMTP_CLIENT           demo_client;
44
45
46  void    _nx_ram_network_driver(struct NX_IP_DRIVER_STRUCT *driver_req);
47  void    demo_client_thread_entry(ULONG info);
48
49  /* Define main entry point.  */
50  int main()
51  {
52      /* Enter the ThreadX kernel.  */
53      tx_kernel_enter();
54  }
55
56  /* Define what the initial system looks like.  */
57  void    tx_application_define(void *first_unused_memory)
58  {
59
60  UINT    status;
61  CHAR    *free_memory_pointer;
62
63
64      /* Setup the pointer to unallocated memory.  */
65      free_memory_pointer =  (CHAR *) first_unused_memory;
66
67      /* Create IP default packet pool. */
68      status =  nx_packet_pool_create(&ip_packet_pool, "Default IP Packet Pool",
69                                      128, free_memory_pointer, 2048);
70
71      /* Update pointer to unallocated (free) memory. */
72      free_memory_pointer = free_memory_pointer + 2048;
73
74      /* Create SMTP Client packet pool. This is only for transmitting packets to the
75         server. It need not be a separate packet pool than the IP default packet pool
76         but for more efficient resource use, we use two different packet pools
77         because the CLient SMTP messages generally require more payload than IP
78         control packets.
79
80         Packet payload depends on the SMTP Client application requirements.  Size of
81         packet payload must include IP and TCP headers. For IPv6 connections, IP and
82         TCP header data is 60 bytes. For IPv4 IP and TCP header data is 40 bytes (not
83         including TCP options). */
84      status |=  nx_packet_pool_create(&client_packet_pool, "SMTP Client Packet Pool",
85                                       800, free_memory_pointer, (10*800));
86
87      if (status != NX_SUCCESS)
88      {
89          return;
90      }
91
92      /* Update pointer to unallocated (free) memory. */
93      free_memory_pointer = free_memory_pointer + (10*800);
94
95      /* Initialize the NetX system. */
96      nx_system_initialize();
97
```

```
98      /* Create the client thread */
99      status = tx_thread_create(&demo_client_thread, "client_thread",
100                               demo_client_thread_entry, 0, free_memory_pointer,
101                               2048, 16, 16,
102                               TX_NO_TIME_SLICE, TX_DONT_START);
103
104     if (status != NX_SUCCESS)
105     {
106
107         printf("Error creating Client thread. Status 0x%x\r\n", status);
108         return;
109     }
110
111     /* Update pointer to unallocated (free) memory. */
112     free_memory_pointer =  free_memory_pointer + 4096;
113
114
115     /* Create Client IP instance. Remember to replace the generic driver
116        with a real ethernet driver to actually run this demo! */
117
118     status = nx_ip_create(&client_ip, "SMTP Client IP Instance", CLIENT_IP_ADDRESS,
                             0xFFFFFF00UL, &ip_packet_pool, _nx_ram_network_driver,
                             free_memory_pointer, 2048, 1);
119
120
121     free_memory_pointer =  free_memory_pointer + 2048;
122
123     /* Enable ARP and supply ARP cache memory. */
124     status =  nx_arp_enable(&client_ip, (void **) free_memory_pointer, 1040);
125
126     /* Update pointer to unallocated (free) memory. */
127     free_memory_pointer = free_memory_pointer + 1040;
128
129     /* Enable TCP for client. */
130     status =  nx_tcp_enable(&client_ip);
131
132     if (status != NX_SUCCESS)
133     {
134         return;
135     }
136
137     /* Enable ICMP for client. */
138     status =  nx_icmp_enable(&client_ip);
139
140     if (status != NX_SUCCESS)
141     {
142         return;
143     }
144
145     /* Start the client thread. */
146     tx_thread_resume(&demo_client_thread);
147
148     return;
149 }
150
151
152 /* Define the smtp application thread task.   */
153 void    demo_client_thread_entry(ULONG info)
154 {
155
156 UINT        status;
157 UINT        error_counter = 0;
158 ULONG       server_ip_address;
159
160
161     tx_thread_sleep(100);
162
163     /* Set up the server IP address. */
164
165     server_ip_address = SERVER_IP_ADDRESS;
166
```

```
167      /* The demo client username and password is the authentication
168         data used when the server attempts to authentication the client. */
169
170      status =  nx_smtp_client_create(&demo_client, &client_ip, &client_packet_pool,
171                                      USERNAME,
172                                      PASSWORD,
173                                      FROM_ADDRESS,
174                                      LOCAL_DOMAIN, CLIENT_AUTHENTICATION_TYPE,
175                                      server_ip_address, SERVER_PORT);
176
177      if (status != NX_SUCCESS)
178      {
179          printf("Error creating the client. Status: 0x%x.\n\r", status);
180          return;
181      }
182
183      /* Create a mail instance with the above text message and recipient info. */
184      status =  nx_smtp_mail_send(&demo_client, RECIPIENT_ADDRESS,
                                    NX_SMTP_MAIL_PRIORITY_NORMAL,
185                                 SUBJECT_LINE, MAIL_BODY, strlen(MAIL_BODY));
186
187      /* Check for errors. */
188      if (status != NX_SUCCESS)
189      {
190
191          /* Mail item was not sent. Note that we need not delete the client. The
193             error status may be a failed authentication check or a broken connection.
194             We can simply call nx_smtp_mail_send again.  */
195          error_counter++;
196      }
197
198      /* Release resources used by client. Note that the transmit packet
199         pool must be deleted by the application if it no longer has use for it.*/
200      status = nx_smtp_client_delete(&demo_client);
201
202      /* Check for errors. */
203      if (status != NX_SUCCESS)
204      {
205          error_counter++;
206      }
207
208      return;
209  }
210
```

Figure 1. Example of SMTP Client use with NetX

Client Configuration Options

There are several configuration options with the NetX SMTP Client API. Following is a list of all options described in detail:

Define	Meaning
NX_SMTP_CLIENT_TCP_WINDOW_SIZE	This option sets the size of the Client TCP receive window. This should be set to below the MTU size of the underlying Ethernet hardware and allow room for IP and TCP headers. The default NetX SMTP Client TCP window size is 1460.
NX_SMTP_CLIENT_PACKET_TIMEOUT	This option sets the timeout on NetX packet allocation. The default NetX SMTP Client packet timeout is 2 seconds.
NX_SMTP_CLIENT_CONNECTION_TIMEOUT	This option sets the Client TCP socket connect timeout. The default NetX SMTP Client connect timeout is 10 seconds.
NX_SMTP_CLIENT_DISCONNECT_TIMEOUT	This option sets the Client TCP socket disconnect timeout. The default NetX SMTP Client disconnect timeout is 5 seconds. Note that if the SMTP Client encounters an internal error such as a broken connection it may terminate the connection with a zero wait timeout.

NX_SMTP_GREETING_TIMEOUT

This option sets the timeout for the Client to receive the Server reply to its greeting. The default NetX SMTP Client value is 10 seconds.

NX_SMTP_ENVELOPE_TIMEOUT

This option sets the timeout for the Client to receive the Server reply to a Client command. The default NetX SMTP Client value is 10 seconds.

NX_SMTP_MESSAGE_TIMEOUT

This option sets the timeout for the Client to receive the Server reply to receiving the mail message data. The default NetX SMTP Client value is 30 seconds.

NX_SMTP_CLIENT_SEND_TIMEOUT

This option defines the wait option of the buffer to store the user password during SMTP authentication with the Server. The default value is 20 bytes.

NX_SMTP_SERVER_CHALLENGE_MAX_STRING

This option defines the size of the buffer for extracting the Server challenge during SMTP authentication. The default value is 200 bytes. For LOGIN and PLAIN authentication, the SMTP Client can probably use a smaller buffer.

NX_SMTP_CLIENT_MAX_PASSWORD

This option defines the size of the buffer to store the user password during SMTP authentication with the Server. The default value is 20 bytes.

NX_SMTP_CLIENT_MAX_USERNAME

This option defines the size of the buffer to store the host username during SMTP authentication with the Server. The default value is 40 bytes.

Chapter 3 Client Description of SMTP Client Services

This chapter contains a description of all NetX SMTP Client services (listed below) in order of usage in a typical SMTP Client application.

In the “Return Values” section in the following API descriptions, values in **BOLD** are not affected by the *NX_DISABLE_ERROR_CHECKING* define that is used to disable API error checking, while non-bold values are completely disabled.

Services for Client Session and Mail Setup

nx_smtp_client_create
: Create an SMTP Client Instance

nx_smtp_client_delete
: Delete an SMTP Client instance

nx_smtp_mail_send
: Create and send an SMTP Mail item

nx_smtp_client_create

Create an SMTP Client Instance

Prototype

```
UINT    nx_smtp_client_create(NX_SMTP_CLIENT *client_ptr,
                              NX_IP *ip_ptr, NX_PACKET_POOL
                              *client_packet_pool_ptr,
                              CHAR *username, CHAR *password,
                              CHAR *from_address,
                              CHAR *client_domain,
                              UINT authentication_type,
                              ULONG server_address,
                              UINT port)
```

Description

This service creates an SMTP Client instance on the specified IP instance.

Input Parameters

client_ptr	Pointer to SMTP Client control block;
ip_ptr	Pointer to IP instance;
packet_pool_ptr	Pointer to Client packet pool;
username	NULL-terminated Username for uthentication;
password	NULL-terminated password for authentication;
from_address	NULL-terminated sender's address;
client_domain	NULL-terminated domain name;

authentication_type	Client authentication type
	Supported types are:
	`NX_SMTP_CLIENT_AUTH_LOGIN`
	`NX_SMTP_CLIENT_AUTH_PLAIN`

server_address	SMTP Server IP address
server_port	SMTP Server TCP port

Return Values

NX_SUCCESS	(0x00)	SMTP Client successfully created TCP socket creation status
NX_SMTP_INVALID_PARAM	(0xA5)	Invalid non pointer input
NX_IP_ADDRES_ERROR	(0x21)	Invalid IP address type
NX_PTR_ERROR	(0x07)	Invalid input pointer parameter

Allowed From

Application Code

Example

```
/* Create the SMTP Client instance. */
NX_PACKET_POOL                client_packet_pool;
NX_IP                         client_ip;
NX_SMTP_CLIENT                demo_client;

#define USERNAME     "myusername"
#define PASSWORD     "mypassword"
#define FROM_ADDRESS "myname@mycompany.com"
#define LOCAL_DOMAIN "mycompany.com"

#define SERVER_PORT  25

/* Define client authentication type as LOGIN. If not specified or
   unknown the SMTP Client will set it to PLAIN. */

#define CLIENT_AUTHENTICATION_TYPE NX_SMTP_CLIENT_AUTH_LOGIN

ULONG server_ip_address;

server_ip_address = SERVER_IP_ADDRESS;

status =  nx_smtp_client_create(&demo_client, &client_ip,
                                &client_packet_pool,
                                USERNAME,
                                PASSWORD,
                                FROM_ADDRESS,
                                LOCAL_DOMAIN,
                                CLIENT_AUTHENTICATION_TYPE,
                                server_ip_address, SERVER_PORT);

/* If an SMTP Client instance was successfully created, status =
NX_SUCCESS. */
```

nx_smtp_client_delete

Delete an SMTP Client Instance

Prototype

```
UINT nx_smtp_client_delete(NX_SMTP_CLIENT *client_ptr)
```

Description

This service deletes a previously created SMTP Client instance.

Input Parameters

client_ptr	Pointer to SMTP Client instance.

Return Values

NX_SUCCESS	(0x00)	Client successfully deleted
NX_PTR_ERROR	(0x07)	Invalid input pointer parameter

Allowed From

Threads

Example

```
/* Delete the SMTP Client instance "my_client." */
NX_SMTP_CLIENT  demo_client;

status = nx_smtp_client_delete(&demo_client);

/* If an SMTP Client instance was successfully deleted, status =
    NX_SUCCESS. */
```

nx_smtp_mail_send

Create and send an SMTP mail item

Prototype

```
UINT    nx_smtp_mail_send(NX_SMTP_CLIENT *client_ptr,
                          CHAR *recipient_address,
                          UINT priority, CHAR *subject,
                          CHAR *mail_body,
                          UINT mail_body_length)
```

Description

This service creates and sends an SMTP mail item. The SMTP Client establishes a TCP connection with the SMTP Server and sends a series of SMTP commands. If no errors are encountered, it will transmit the mail message to the Server. Regardless if the mail is sent successfully it will terminate the TCP connection and return a status indicating outcome of the mail transmission. The application may call this service for as many mail messages as it needs to send without limit.

Input Parameters

client_ptr	Pointer to SMTP Client
recipient_address	NULL-terminated recipient address.
subject	NULL-terminated subject line text;.
priority	Priority level at which mail is delivered
mail_body	Pointer to mail message
mail_body_length	Size of mail message

Return Values

NX_SUCCESS	(0x00)	Mail successfully sent
NX_SMTP_CLIENT_NOT_INITIALIZED	(0xB2)	SMTP Client instance not initialized for SMTP session
status		Outcome of SMTP session
NX_PTR_ERROR	(0x07)	Invalid pointer parameter
NX_SMTP_INVALID_PARAM	(0xA5)	Invalid non pointer input
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
/* Create and send a Client mail item. */

#define RECIPIENT_ADDRESS "your@yourcompany.com"
#define SUBJECT    "NetX SMTP Client Demo"
#define MAIL_BODY  "NetX SMTP client is an SMTP client " \
                   "implementation for embedded devices \r\n" \
                   "to send email to SMTP servers.\r\n"

 status =  nx_smtp_mail_send(&demo_client, RECIPIENT_ADDRESS,
                             NX_SMTP_MAIL_PRIORITY_NORMAL,
                             SUBJECT_LINE, MAIL_BODY,
                             strlen(MAIL_BODY));

 /* Return status being NX_SUCCESS indicates the mail has been
    successfully sent.  */
```